The 26th International Conference on Automated Planning and Scheduling

Proceedings of the 4th Workshop on

Planning and Robotics (PlanRob)

Edited by:

Alberto Finzi, Erez Karpas

London, UK, 13-14/06/2016

Organising Committee

Alberto Finzi

DIETI - Università di Napoli "Federico II", Italy

Erez Karpas

Technion - Israel Institute of Technology, Israel

Program Committee

Rachid Alami (LAAS-CNRS, France)
Sara Bernardini (King's College, UK)
Amedeo Cesta (ISTC-CNR, Italy)
Marcello Cirillo (Orebro University, Sweden)
Patrick Doherty (Linkoping University, Sweden)
Erez Karpas (Technion, Israel)
Sven Koenig (University of Southern California, USA)
Alberto Finzi (Naples University, Italy)
Robert Fitch (University of Sydney, Australia)
Malik Ghallab (LAAS-CNRS, France)
Joachim Hertzberg (Osnabrück University, Germany)
Felix Ingrand (LAAS-CNRS, France)
Luca Iocchi (University of Rome "La Sapienza", Italy)
Matteo Leonetti (University of Leeds)
Daniele Magazzeni (King's College, UK)
Karen Myers (SRI International, USA)
Daniele Nardi (University of Rome "La Sapienza", Italy)
Goldie Nejat (University of Toronto, Canada)
Andrea Orlandini (ISTC-CNR, Italy)
Frederic Py (Independent)
Enrico Scala (ANU, Australia)
David Smith (NASA Ames, USA)
Tiago Stegun Vaquero (MIT, USA)

Foreword

Robotics is one of the most appealing and natural applicative area for the Planning and Scheduling (P&S) research activity, however such a natural interest seems not reflected in an equally important research production for the Robotics community. In this perspective, the aim of the PlanRob workshop is twofold. On the one hand, this workshop would constitute a fresh impulse for the ICAPS community to develop its interests and efforts towards this challenging research area. On the other hand, it aims at attracting representatives from the Robotics community to discuss their challenges related to planning for autonomous robots (deliberative, reactive, continuous planning and execution etc.) as well as their expectations from the P&S community.

The PlanRob workshop aims at constituting a stable, long-term forum on relevant topics concerned with the interactions between the Robotics and P&S communities where researchers could discuss the opportunities and challenges of P&S when applied to Robotics. Started during ICAPS 2013 in Rome (Italy) and followed by a second edition at ICAPS 2014 in Portsmouth (NH, USA) and a third one at ICAPS 2015 in Jerusalem (Israel), the PlanRob WS series (http://pst.istc.cnr.it/planrob/) has gathered very good feedback from the P&S community which is also confirmed by the organisation of a specific Robotics Track from ICAPS 2014.

This fourth edition of the PlanRob workshop has been proposed in synergy with the Robotics Track to further enforce its original goals and to maintain a more informal forum where more preliminary/visionary works can be discussed. PlanRob 16 succeeded in achieving these objectives providing a rich and articulated program. Indeed, 23 papers have been accepted for oral presentation covering many relevant topics in Planning and Robotics such as high-level task planning, task and motion planning, planning and execution for robots, planning and learning, human-robot interaction, real applications and case studies. The workshop program is completed by the invited talk by Prof. Manuela Veloso (Carnegie Mellon University - CMU) on "The Multiple Facets of Planning in Robot Autonomy".

The varieties of research topics and results collected in these proceedings reflect a stimulating and intense research activity along with a growing interest for a forum where the Planning and Robotics communities can find a common ground.

Among the numerous people that contribute to the success of PlanRob 2016, we would first of all thank the ones that submitted their research papers to the workshop and attended the event. Moreover, we sincerely thank the program committee for the important work on the reviewing process.

Alberto Finzi, Erez Karpas

The PlanRob 2016 Chairs

PlanRob 2016 is partially supported by the SHERPA project (EU FP7 under the grant agreement ICT-600958).

Contents

From videogames to autonomous trucks: A new algorithm for lattice-based motion planning

Marcello Cirillo
Scania Technical Centre, Södertälje, Sweden
marcello.cirillo@scania.com

Abstract

Autonomous navigation in real-world environments is still a challenging task in many respects. One of the key open challenges is fast planning of physically executable complex maneuvers under non-holonomic constraints. In recent years, lattice-based motion planners have been successfully used to generate kinematically and kinodynamically feasible motions for non-holonomic vehicles. However, it is not clear yet what algorithms are best to efficiently explore the lattice state space, while at the same time ensuring real-time performance. In this paper, we show how motion planning can greatly benefit from tapping into the latest results in path planning on grids, and we present a new version of *Time-Bounded A**. Our version is designed to work for high-dimensional motion planning problems in real-world robotic applications. We demonstrate our algorithm both in simulation and on a full-size autonomous truck.

Introduction

In recent years, the interest for autonomously driving vehicles has steadily increased. Many big actors in the car industry, as well as competitive outsiders have joined the race to provide the world with the first fully autonomous cars, trucks or buses (Ross 2014). Thanks to this interest, great resources have been allocated worldwide to develop the new algorithms and techniques necessary to reach the ambitious goal, and the by-product of the race has been the commercialization of new advanced safety systems. When it comes to industrial tasks, such as in mining or intra-logistic scenarios, solutions which totally or partially rely on autonomous vehicles have been available for many years (Thrybom et al. 2015; Andreasson et al. 2015a). This is because some of the most challenging problems that must be addressed in urban environments are muted when autonomous vehicles operate in special, enclosed areas. However, industrial solutions can still greatly benefit from recent advancements.

The industry standard approach to motion planning for autonomous vehicles still relies largely on fixed paths (Marshall, Barfoot, and Larsson 2008), which have been either previously driven by a human operator or manually drawn during system deployment. Both approaches, although effective, present the major drawback that even small modifications to the environment require defining new paths, which can be a costly and cumbersome procedure. Moreover, vehicles on pre-defined paths can only deal with unexpected obstacles by reducing their velocity or by employing very simple avoidance strategies.

In this paper, we address the problem of motion planning for non-holonomic vehicles in unstructured environments, that is, in possibly large areas where a vehicle cannot follow roads and needs to perform complex maneuvers. More specifically, we focus on heavy transport vehicles, as our work is driven by the goal of introducing a new level of autonomy in environments such as open and underground mines, or construction sites. Although these environments typically contain a low number of other agents, trucks operating there require advanced motion planning algorithms: The transportation tasks effectively change the landscape of the areas, loading and unloading locations change over time, and the maneuvering spaces can be quite narrow. As a practical example, consider the area in Figure 1. Here, the truck arrives on the maneuvering area through a narrow road, and it needs to get to a loading place whose location can change over time and can be precisely identified only by sensor readings. Also, other trucks and loaders may have moved material around, thus creating new unexpected obstacles. Although motion planning has been the focus of extensive studies in the past decade, and many solutions have been proposed to deal with non-holonomic vehicles (LaValle 2006; Pivtoraiko, Knepper, and Kelly 2009; Thrun et al. 2006), a definitive solution for the problem described above does not exist as yet, and the existing ones can still be greatly improved.

Our main contribution is to show how motion planning can be further improved by tapping into the wealth of algorithms which have been developed for graph search outside the robotics community. We describe why and how we adapted the popular algorithm *Time-Bounded A** (Björnsson, Bulitko, and Sturtevant 2009) for working with robotic systems, and we demonstrate its effectiveness in simulation and on an autonomous truck.

In the following, we first present recent related work. We then briefly describe our planning framework and the original algorithm, we present our new version of the algorithm, adapted for high-dimensional spaces and robotic applications, and we empirically demonstrate the capabilities of our planner. Finally, we discuss our results and highlight interesting avenues for future research.

Figure 1: The autonomous truck used in our tests and an area were the truck is required to maneuver.

Related Work

Motion planning under differential constraints has been extensively studied in the past decades. Whenever differential constraints and obstacles are considered, combinatorial methods and analytical solutions are of limited use (LaValle 2006). The former are not well suited in the presence of differential constraints, while the latter cannot effectively cope with obstacles. When it comes to planning motions for car-like vehicles on roads, several assumptions can be made that led to the development of specialized planners (Madas et al. 2013; Levinson et al. 2011). Most of these results, however, are not suitable for maneuvering in unstructured environments. In these environments, sampling-based methods have been proven to be effective: Probabilistic Roadmaps (PRMs) (Kavraki et al. 1996), Rapidly-exploring Random Trees (RRTs) (LaValle 1998) and lattice-based motion planners (Pivtoraiko and Kelly 2009), all of which can work in high-dimensional configuration spaces. PRMs have two major drawbacks: First, before running the algorithm, several parameters must be selected (e.g., the duration of the learning phase); and, second, a new roadmap has to be built every time the environment is subject to substantial changes. After their initial introduction (LaValle and Kuffner 2001), RRTs have been extensively studied, and many variants of the original algorithm have been proposed (Karaman and Frazzoli 2011; Kuwata et al. 2009; Karaman and Frazzoli 2013). RRTs do not guarantee convergence (termination is usually implemented with a timeout) and, unless the space is analyzed beforehand, they cannot verify whether a problem offers no solution. Lattice-based motion planners combine the strengths of the previous approaches with classical AI graph-search algorithms, such as A^*, ARA^* (Likhachev, Gordon, and Thrun 2003) and D^*Lite (Koenig and Likhachev 2002). Differential constraints are incorporated into the state space by means of precomputed motion primitives which trap the motions onto a regular lattice. The state space is then explored using graph-search algorithms.

Lattice-based planners have proven to be particularly effective for quickly calculating accurate, complex maneuvers (e.g., three-point turns) in environments cluttered with obstacles (Andreasson et al. 2015b). Existing planners, however, generally use only a very restricted subset of the graph search algorithms developed in recent years. More important still, most of the algorithms currently used need to find a complete solution before starting execution. This can be computationally expensive and even useless, as the environment around a mobile vehicle is usually observable only as far as its sensors' range allows. Therefore, the solutions found often need to be corrected or completely re-calculated as new information is acquired.

Parallel to the research on motion planning under differential constraints, there have been very interesting developments in the area of path finding on grids (Sturtevant et al. 2015). Videogames have very stringent requirements on the amount of time that can be allotted to path finding for each character, especially when the number of agents to be moved is high and the map only partially observable (Sturtevant 2015). The algorithms employed must be effective and must work in real time, providing also partial solutions in the little time available. Although pathfinding algorithms on grids have to consider only a few possible alternative moves at each state (depending if a 4- or an 8-connected grid is used), they can be easily adapted to work on lattices. There is a wealth of algorithms (Koenig and Likhachev 2006; Björnsson, Bulitko, and Sturtevant 2009) that can be used to improve lattice-based motion planners, and in this paper we show how.

Motion planning framework

Given a model of vehicle maneuverability, the intuition behind lattice-based motion planning is to sample the state space in a regular fashion and to constrain the motions of the vehicle to a lattice graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, that is, a graph embedded in a Euclidean space $\mathbb{R}^n$ which forms a regular tiling (Pivtoraiko, Knepper, and Kelly 2009; Cirillo, Uras, and Koenig 2014). Each vertex $v \in \mathcal{V}$ represents a state, or pose of the vehicle, while each edge $e \in \mathcal{E}$ encodes a motion which respects the non-holonomic constraints of the vehicle. Here, we focus only on kinematic constraints, as in our system the generation of a velocity profile is decoupled from the maneuver calculated by the motion planner. The reason for this division is two-fold: First, removing the velocity at planning time we effectively reduce the size of the state space, thus speeding up the search for solutions. Also, the speed profiling can then be better tailored to the requirements of the low-level controller. Second, calculating the velocity using a dedicated module, we can easily take into account dynamic obstacles and avoid them by adapting the cruising speed without re-planning.

In a motion planning problem, a vehicle is fully specified by its *model*. A model encodes the geometric measurements of the vehicle, the discretization of the dimensions of the lattice on which the vehicle moves and a set of *motion primitives* P. The discretization of the lattice defines what states the vehicle can reach. A valid state for a model is represented by a four-dimensional vector $s = \langle x, y, \theta, \phi \rangle$: (x, y) lies on a grid of resolution r, $\theta \in \Theta$ and $\phi \in \Phi$, where Θ and Φ are a finite set of allowed orientations and of allowed steering angles, respectively. The set of motion primitives

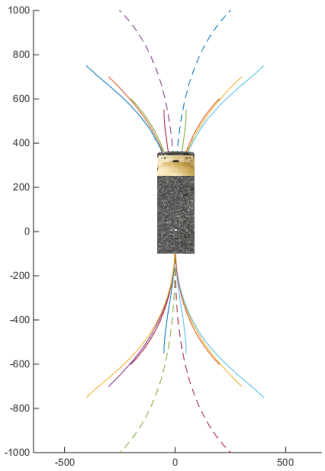


Figure 2: The motion primitives in the state $s = \langle 0, 0, \pi/2, 0 \rangle$. In this model, $r = 50$ cm, $|\Theta| = 16$ and $|\Phi| = 1$.

P captures the mobility of the vehicle while intrinsically taking into account its kinematic constraints. Under the assumption of even terrain, we can design P to be position-invariant.[1] Every $p \in P$ is calculated by using a *boundary value problem* (BVP) solver to connect a set of initial states $s = \langle 0, 0, \theta, \phi \rangle$ to a set of neighboring states in a discrete, bounded neighborhood in free space. The BVP solver guarantees that the motions respect the kinematic constraints of the vehicle, while the position-invariant property ensures that the primitives are translatable to other states. P can then be reduced for efficiency using the techniques described in (Pivtoraiko and Kelly 2011), by removing those primitives that can be decomposed into other primitives in P, without affecting the reachability of the state space of the vehicle when obstacles are not considered. Finally, a cost $g(p)$ is associated with each $p \in P$. In our implementation, $g(p)$ is calculated by multiplying the distance covered by p by a cost factor which penalizes backwards and turning motions. An example of motion primitives for the vehicle model of a truck can be seen in Figure 2, where $r = 50$ cm, $|\Theta| = 16$ and $|\Phi| = 1$. The figure represents all the primitives applicable in the starting state $s = \langle 0, 0, \pi/2, 0 \rangle$.

A planning problem is defined by a starting state $start$, a goal state $goal$ and a world representation $\mathcal{W}$, in which are included all known obstacles. A *valid solution* is a sequence of collision-free primitives $(p_0, \ldots, p_n)$ connecting $start$ to $goal$. Given the set of all valid solutions to a problem, an optimal solution is the one with minimum cost.

Time-Bounded A*

Time-Bounded A^* (TBA^*) was first introduced in (Björnsson, Bulitko, and Sturtevant 2009) and was designed for efficient path finding on grids. The most prominent features of the algorithm (described below in pseudo-code as in the original publication) are that it achieves real-time operation, it allows to interleave search periods with action execution, and it avoids many unnecessary state re-expansions compared to other solutions. These characteristics, combined with the fact that TBA^* maintains completeness, make the algorithm a great choice for gaming applications.

Procedure TBA^* (*start,goal,*$\mathcal{W}$)

```
1  solutionFound ← false
2  solutionFoundAndTraced ← false
3  traceDone ← false
4  loc ← start
5  while loc ≠ goal do
6      if (¬solutionFound) then
7          solutionFound ← A*(lists, start, goal, W, N_E)
8      if (¬solutionFoundAndTraced) then
9          if (doneTrace) then
10             pathNew ← lists.mostPromisingState()
11         doneTrace ← traceBack(pathNew, loc, N_T)
12         if (doneTrace) then
13             pathFollow ← pathNew
14             if (pathFollow.back() = goal) then
15                 solutionFoundAndTraced ← true
16     if (pathFollow.contains(loc)) then
17         loc ← pathFollow.popFront()
18     else
19         if (loc ≠ start) then
20             loc ← lists.stepBack(loc)
21         else
22             loc ← loc_last
23     loc_last ← loc
24     move agent to loc
```

Given an initial state $start$, a desired final state $goal$ and a representation of the world $\mathcal{W}$, TBA^* searches the state space as A^* would do. However, while the latter would continue until it finds a complete path from *start* to *goal*, TBA^* stops the search after a finite number N_E of expansions (line 7), while retaining the open and the closed lists. In case a solution has not yet been found after N_E expansions, the algorithm extracts from the open list the most promising state, tracing it back ($pathNew$) either to *start*, or to the current location of the agent loc, in case the agent is already on $pathNew$. Note that also the tracing operation is done in a time sliced manner, and only N_T steps are traced at each iteration (line 11). When tracing is done, $pathNew$ becomes the path to follow ($pathFollow$, line 13) and the algorithm executes sequentially its actions (line 17). However, the agent might not be on $pathFollow$, but on another path that was extracted as most promising during a previous iteration. In such case it will have to backtrack its steps (line 20) to reach the state where the two paths meet (*start*, in the worst case). Move actions (lines 24) are performed one per iteration, while expansion and tracing steps (N_E and N_T) are fixed in number at each algorithm's iteration. The authors also considered the special case in which the agent has reached *start*, but no new path is available. Here, the agent is forced to act, and it moves back to the state it came from

[1] This assumption can be relaxed if the low-level controller of the vehicle can absorb minor perturbations or by means of a post-processing step.

(loc_last line 22)

Conceptually, it would be straightforward to modify the algorithm to explore a lattice, rather than a grid. However, robotic systems are not equivalent to videogame agents, and domain-specific adaptations are required.

Lattice Time-Bounded A*

Moving from videogames to autonomous vehicles, there are many aspects of TBA^* which require adjusting. The resulting new algorithm, *Lattice Time-Bounded* A^* ($LTBA^*$), is summarized in pseudo-code below. $LTBA^*$ maintains the operational principles of TBA^* and still works in a time-sliced manner, where A^* is repeatedly called at each iteration with the same lists (line 12). However, this new algorithm must take into account the fact that we are planning for a physical system, which cannot be safely steered on a new path without considering its velocity and inertia, and which should not exhibit erratic behaviours. Moreover, the algorithm must take into account that new goals could arrive during execution, and that other systems (such as the low-level controller) may fail, or steer the vehicle out of its intended path. Finally, new sensor data arrive at every iteration, and new obstacles may be perceived. In the following, we detail and motivate all the major differences between the original algorithm and its adaptation for robotic systems.

Procedure $LTBA^*$ (*goal,* $\mathcal{W}$)

```
1  loc ← getSensorData()
2  start ← createState(loc)
3  lastState, committedState ← start
4  t ← now()
5  while loc ≠ goal do
6      (currentGoal, loc, W) ← getSensorData()
7      if (currentGoal ≠ goal) ∨ outOfPath(loc) then
8          return
9      (lastState, committedState) ← trackLoc(loc)
10     if committedState ≠ startState then
11         updateLists(committedState)
12     solutionFound ← A*(lists, start, goal, W, ΔT − t)
13     t ← now()
14     pathNew ← lists.mostPromisingState()
15     if pathNew.size() > 0 then
16         sendPath(pathNew)
17     else
18         sendPath(lastState)
```

Continuous sensor update The first difference between $LTBA^*$ and TBA^* lies in the distinction between *states*, that are a discrete representation used for exploring the lattice space, and the status of the vehicle, contained in the variable loc. loc does not only contain the position of the vehicle in the continuous space, but it also includes information such as current speed and mass.[2] Because of this difference, the starting state of the vehicle is not passed as an argument, but it is inferred directly from sensor data, so that $start$ reflects where the vehicle is expected to be by the end of the first planning cycle (lines 1-2).

Sensor updates are repeated at each iteration of the algorithm (line 6). New readings are processed for updating the representation of the world $\mathcal{W}$ (e.g., the position of detected obstacles), the location of the truck loc and to verify if the current goal has changed. If, for any reason, the truck is outside its designed path, or in case a new $goal$ is received, the procedure terminates and a new one is immediately instantiated with a new $start$ (lines 7-8).

Procedure *mostPromisingState*

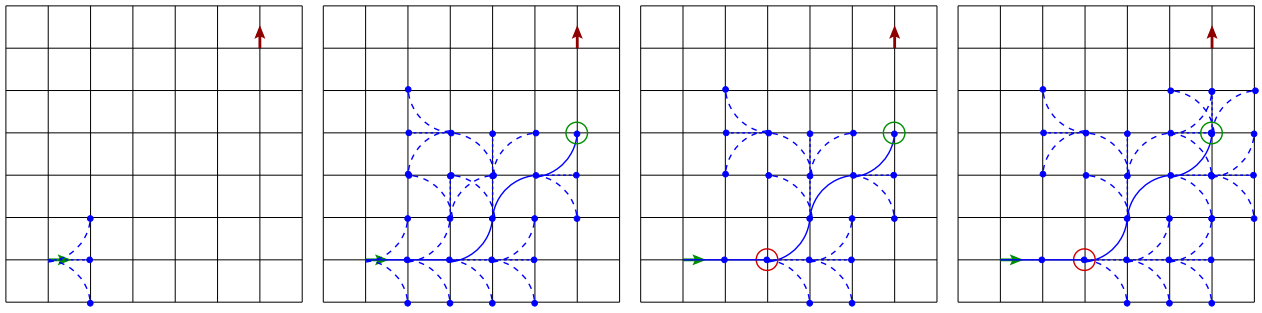
```
1  if solutionFound then
2      pathNew ← solution
3  else
4      pathNew ← openList.pop()
5  while ¬ collFree(pathNew) ∧ openList.size() > 0 do
6      removeCollisionStates(lists, pathNew)
7      pathNew ← openList.pop()
8  if collFree(pathNew) then
9      return(pathNew)
10 else
11     pathNew ← {}
12     return(pathNew)
```

Path selection and path commitment After each planning iteration (line 12), $pathNew$ is updated as it was the case in TBA^*. However, here, the procedure which selects the most promising state is more complex and it is detailed in pseudo code in Procedure *mostPromisingState*.

As it is customary in robotic applications, the planner works under the free world assumption, which means that the area outside the range of the truck's sensors, with the exception of fixed infrastructure, is considered as obstacle-free. Hence, it may happen that the extracted $pathNew$ is invalidated once a new obstacle enters within the sensors' range (see example in Figure 4).

The selection of $pathNew$ works as follows: First, the algorithm checks if A^* during the last search iteration has reached a solution, which would obviously become the first candidate for $pathNew$. In case a solution does not exist, the most promising node is extracted from the $openList$ (Procedure *mostPromisingState*, lines 1-4). Once a candidate has been selected, it is checked for obstacles. In case the result of the check is positive, the edge of the lattice graph on which the first collision point occurs is identified and removed, along with all its successors. Such states, now unreachable, are deleted from the lists (line 6). New candidates are then evaluated until one of two possibilities occurs: Either a candidate $pathNew$ is collision-free, or the $openList$ is empty. In the latter case, an empty $pathNew$ is returned. $pathNew$ is then sent for execution ($LTBA^*$, line 16). In the unlikely case that $pathNew$ is empty, the algorithm sends the position contained in $lastState$ for execution. If the truck is moving, this would cause an emergency braking procedure. Note that this mechanism was never triggered in the course of our experiments, both simulated and with the real truck, but it is nevertheless necessary to ensure

[2] The information about the mass could be included into the model of the vehicle. However, here we are targeting a transportation domain, where the truck's load varies.



(a) (b) (c) (d)

Figure 3: A simplified example of how the explored lattice is pruned after the algorithm has selected a $committedState$ on the current $pathNew$. (a): A vehicle, captured by a simple vehicle model with three motion primitives, must move from $start$ (green arrow, bottom left corner) to $goal$ (red arrow, top right); (b): After the first search iteration, the lattice has been partially explored, and the algorithm selects a $pathNew$, represented by a continuous line and terminating at the state circled in green, which is sent for execution; (c): Before the next search iteration, the algorithm commits to a $committedState$ (circled in red), where the truck is going to be after the search episode. The lattice space is then pruned, so that no further exploration would be possible between $start$ and $committedState$; (d): The next search iteration further explores the lattice starting from the states expanded in previous searches.

overall system safety.

Finally, dealing with a heavy vehicle traveling at considerable speed entails that sudden stops and change of directions are not acceptable. Nor we can allow for the truck to trace back a dismissed path in reverse. Therefore, the $traceBack$ function in TBA^* is replaced by a mechanism to ensure that such occurrences never happen. More specifically, at each cycle we use the information about position and speed of the truck on the $pathNew$ to calculate both the $lastState$ visited and the $committedState$ (line 9). The $committedState$ is the state that lies on $pathNew$ before which the truck, at its current speed, is not allowed to deviate from $pathNew$. The calculation of $committedState$ takes into account two factors: First, the position at which the truck is predicted to be at the end of the current planning cycle; Second, the momentum of the truck, so as to avoid infeasible maneuvers.

Once $committedState$ has been calculated, it becomes the new root node of the search. This is done by removing from all the lists the states that branch from $start$ to $committedState$, thus forcing the search to continue from $committedState$ onwards. A simplified example of this procedure can be seen in Figure 3, while a real test case is shown in Figure 4. Note that the discarded states can be reached again, but only by first passing through $committedState$. This last step ensures that the next $pathNew$ will share the first segment of the previous one.

Real-time execution Maintaining real-time performance in $LTBA^*$ is of the utmost importance. As the algorithm needs to send a collision-free $pathNew$ for execution every fixed ΔT (in our implementation, $\Delta T = 0.5$ seconds), we need to make sure that the time allotted for search takes into account other possibly time-consuming procedures, such as the collision checking and the selection of $pathNew$. Also, when dealing with lattice state, the time required for each expansion may greatly vary, as collision checking is more complex than on a grid. For this reason, we prefer to put a hard limit on the *execution time* of the search phase, rather than on the number of expansions as originally done in TBA^*.

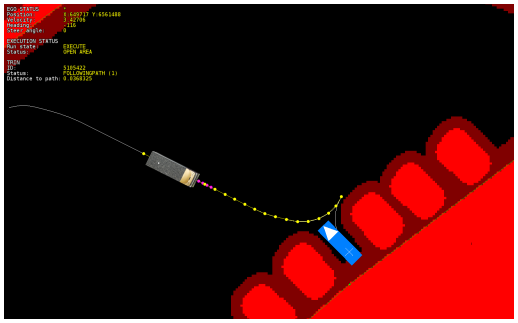


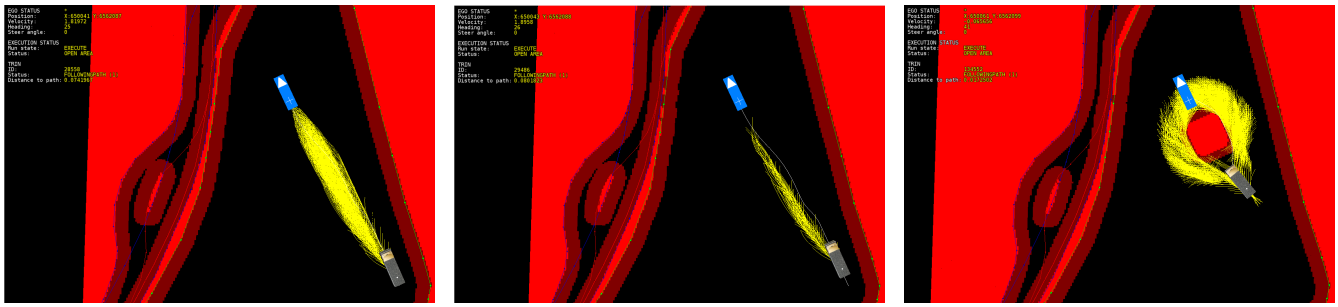
Figure 6: The planner quickly calculates parking maneuvers.

Thus, before the main loop begins (line 4) and right after the search on the lattice space (line 13), a time t variable is reset. A^* is called as the last operation in each time cycle, so that we can calculate exactly how much time is available for exploring the lattice.

Two possible alternatives to maintain real-time execution would be to either start a new search at every planning cycle and select $pathNew$ each time from scratch, or to wait for a complete solution before moving and then safely stop and start a new search if the current solution is invalidated. However, $LTBA^*$ shows better performance: by never discarding the lists, it avoids useless re-expansions. Furthermore, it can exploit moments in which the truck is proceeding slowly or has stopped to continue state expansion.

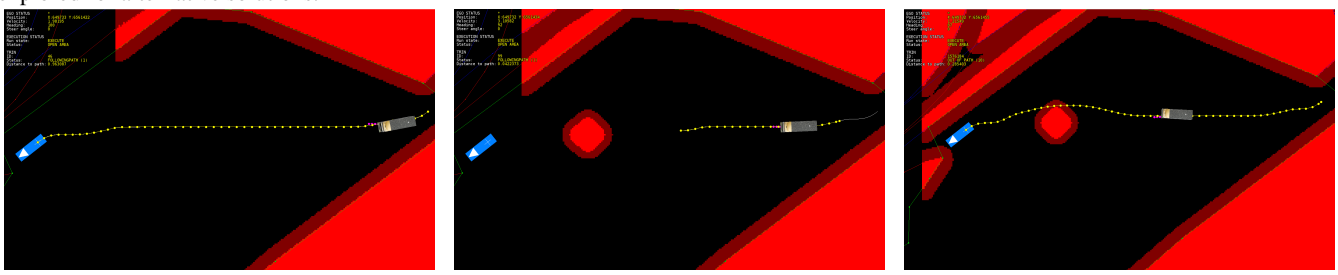
Notes on completeness and complexity

Lattice-based motion planners can be complete with respect to the discretization of the state lattice and the selection of the motion primitives (Cirillo et al. 2014), provided that the algorithm used to search the lattice is complete. $LTBA^*$ relies on a time-sliced A^* search, which is per se complete. However, contrarily to TBA^*, the new algorithm does not expand all the nodes as A^* would do, as some branches are pruned away because of the mechanism of the $committedState$. To ensure completeness, it would be required to modify the algorithm in two ways: (1) whenever a solution is not found within a number of search cycles,



(a) (b) (c)

Figure 4: In this simulation, the truck moves from its current position (lower right) to *goal* (blue). The lattice is explored (a), and a $pathNew$ selected. In successive search iterations, $LTBA^*$ keeps exploring the lattice, finds a complete solution (in white), and discards those branches that stem from behind the $committedState$ (b). When an obstacle is detected (c), the current $pathNew$ is abandoned, and the lattice is explored for alternative solutions.



(a) (b) (c)

Figure 5: TBA^* can efficiently cope with new obstacles detected along the path. As the initially calculated path (a) is not viable any longer (b), the algorithm resume the exploration of the lattice, to find an alternative, collision-free route (c).

the truck is required to reduce speed or even to stop, so as to give enough time to complete a full A^* search from the current $committedState$; and (2) the set P of motion primitives should be designed to have symmetric forward and backward maneuvers.

The memory complexity of $LTBA^*$ is the same of A^*: in the worst-case, the algorithm would require to explore the entire state space. It can be readily understood that the state space of the lattices used for motion planning can be very large. Therefore, we rely on two heuristic functions to direct the search: A simple euclidean distance and a state-to-state heuristic table with exact costs in free space. This last heuristic has been already successfully used in (Pivtoraiko, Knepper, and Kelly 2009) and it is calculated by running Dijkstra's algorithm starting from a few selected starting states. Both heuristics are consistent, and we can combine them so that the resulting function is also consistent.

Experimental Results

We tested our planner both in simulation and on our robotic platform (Figure 1). The simulation environment duplicates all the sub-systems of the real platform, from behaviour selection to the low level controller, and it is effective in testing all the modules deployed on the autonomous truck. All the components of the simulation run on a virtual machine with as Ubuntu OS, which in turns is running on a normal laptop equipped with an Intel Core i7-4810QM CPU @ 2.80GHz and 16 GB RAM (8 GB available to the virtual machine). The computer running the components on the robotic platform is Debian-based, and with comparable hardware specifications. The tests were carried out with two truck models: one with $|\Theta| = 16$, $|\Phi| = 1$ and $|P| = 192$, the second with $|\Theta| = 32$, $|\Phi| = 1$ and $|P| = 1312$ ($r = 50$ cm in both cases). The second model allows for more precise maneuvering, but its state space is obviously much larger. Both models were used with a heuristic table for speeding up close-quarters maneuvering.

The first tests included parking scenarios (Figure 6), point-to-point movements and evasive maneuvers when new obstacles appear on the map unexpectedly during execution (Figure 4). All tests were successful: the low level controller was able to effectively execute all the maneuvers planned and the obstacles were correctly avoided.

The second series of tests were carried out to demonstrate that $LTBA^*$ can cope with new obstacles better than a simple A^*. We designed 5 similar scenarios, in which the planner was invoked with fixed $start$ and $goal$. After 10 seconds, an obstacle would appear between the truck and its destination, as shown in Figure 5. We run the scenarios first using $LTBA^*$ and then A^*. In the second case, the planner was forced to find a complete solution before committing to a path, and to start from scratch when the current solution was invalidated. The model used in these tests was the more complex one ($|\Theta| = 32$ and $|P| = 1312$). Over the 5 runs, not only our algorithm expanded less nodes than A^* ($LTBA^*$: avg 6017 [max 8505] ; A^*: avg 10473 [max 11850]), but it never required abrupt braking for calculating a new path.

Conclusions and Future Work

In this paper we have shown how motion planning for robotic systems can greatly benefit from the latest findings in the area of path finding on grids. We adapted TBA^*, an algorithm designed for path finding in videogames, to lattice-based motion planning. The new algorithm, $LTBA^*$, was described, analyzed and tested in simulation and on an autonomous truck. Future work include a comparison of the new algorithm with other state-of-the-art search algorithms and the extension to multi-robot systems.

References

Andreasson, H.; Bouguerra, A.; Cirillo, M.; Dimitrov, D.; Driankov, D.; Karlsson, L.; Lilienthal, A. J.; Pecora, F.; Saarinen, J.; and Sherikov, A. 2015a. Autonomous transport vehicles: where we are and what is missing. *Robotics & Automation Magazine, IEEE* 22(1):64–75.

Andreasson, H.; Saarinen, J.; Cirillo, M.; Stoyanov, T.; and Lilienthal, A. J. 2015b. Fast, continuous state path smoothing to improve navigation accuracy. In *IEEE Int. Conf. on Robotics and Autom. (ICRA)*.

Björnsson, Y.; Bulitko, V.; and Sturtevant, N. R. 2009. TBA*: Time-bounded A*. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Cirillo, M.; Uras, T.; Koenig, S.; Andreasson, H.; and Pecora, F. 2014. Integrated motion planning and coordination for industrial vehicles. In *Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Cirillo, M.; Uras, T.; and Koenig, S. 2014. A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Karaman, S., and Frazzoli, M. 2011. Sampling-based algorithms for optimal motion planning. *The Int. Journal of Robotics Research* 30(7):846–894.

Karaman, S., and Frazzoli, M. 2013. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Proc. of the IEEE Int. Conf. on Robotics and Autom. (ICRA)*.

Kavraki, L.; Svestka, P.; Latombe, J.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Autom.* 12(4):566–580.

Koenig, S., and Likhachev, M. 2002. D* lite. In *Proc. of the National Conf. on Artificial Antelligence (AAAI)*.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive A*. In *Proc. of the 5th Int. Joint Conf. on Auton. Agents and Multiagent Systems (AAMAS)*.

Kuwata, Y.; Karaman, S.; Teo, J.; Frazzoli, M.; How, J. P.; and Fiore, G. 2009. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology* 17(5):1105–1118.

LaValle, S. M., and Kuffner, J. J. 2001. Rapidly-exploring random trees: Progress and prospects. In Donald, B. R.; Lynch, K. M.; and Rus, D., eds., *Algorithmic and Computational Robotics: New Directions*. Wellesley, MA: A K Peters. 293–308.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press.

Levinson, J.; Askeland, J.; Becker, J.; Dolson, J.; Held, D.; Kammel, S.; Kolter, J. Z.; Langer, D.; Pink, O.; Pratt, V.; Sokolsky, M.; Stanek, G.; Stavens, D.; Teichman, A.; Werling, M.; and Thrun, S. 2011. Towards fully autonomous driving: systems and algorithms. In *Proc. of the IEEE Intelligent Vehicles Symposium (IV)*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems* 16.

Madas, D.; Nosratinia, M.; Keshavarz, M.; Sundstrom, P.; Philippsen, R.; Eidehall, A.; and Dahlen, K.-M. 2013. On path planning methods for automotive collision avoidance. In *Proc. of the IEEE Intelligent Vehicles Symposium (IV)*.

Marshall, J.; Barfoot, T.; and Larsson, J. 2008. Autonomous underground tramming for center-articulated vehicles. *Journal of Field Robotics* 25(6-7):400–421.

Pivtoraiko, M., and Kelly, A. 2009. Fast and feasible deliberative motion planner for dynamic environments. In *Proc. of the ICRA Workshop on Safe Navigation in Open and Dynamic Environments: Application to Autonomous Vehicles*.

Pivtoraiko, M., and Kelly, A. 2011. Kinodynamic motion planning with state lattice motion primitives. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Pivtoraiko, M.; Knepper, R. A.; and Kelly, A. 2009. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics* 26(3):308–333.

Ross, P. 2014. Robot, you can drive my car. *IEEE Spectrum* 51(6):60–90.

Sturtevant, N.; Traish, J.; Tulip, J.; Uras, T.; Koenig, S.; Strasser, B.; Botea, A.; Harabor, D.; and Rabin, S. 2015. The grid-based path planning competition: 2014 entries and results. In *Eighth Annual Symposium on Combinatorial Search*.

Sturtevant, N. 2015. An introduction to search for games. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*. CRC Press.

Thrun, S.; Montemerlo, M.; Dahlkamp, H.; Stavens, D.; Aron, A.; Diebel, J.; Fong, P.; Gale, J.; Halpenny, M.; Hoffmann, G.; et al. 2006. Stanley: The robot that won the DARPA grand challenge. *Journal of Field Robotics* 23(9):661–692.

Thrybom, L.; Neander, J.; Hansen, E.; and Landernas, K. 2015. Future challenges of positioning in underground mines. *IFAC-PapersOnLine* 48(10):222–226.

Using a Model of Ocean Currents to Control the Position of Vertically Profiling Marine Floats

Martina Troesch, Steve Chien
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Yi Chao, John Farrara
Remote Sensing Solutions
248 East Foothill Blvd
Monrovia, CA 91016
ychao@remotesensingsolutions.com
jfarrara@remotesensingsolutions.com

Abstract

We describe a methodology for control of vertically profiling floats that uses an imperfect predictive model of ocean currents. In this approach, the floats have control only over their depth. We combine this control authority with an imperfect model of ocean currents to force the floats to maintain position. First, we study the impact of model accuracy on this ability to station keep (e.g. maintain X-Y position) using simulated planning and nature models. In this study, we examine the impact of batch versus continuous planning. In batch planning the float depth plan is derived for an extended period of time and then executed open loop. In continuous planning the depth plan is updated with the actual position and the remainder of the plan re-planned based on the new information. In these simulation results, we show that (a) active control can significantly improve station keeping with even an imperfect predictive model and (b) continuous planning can mitigate the impact of model inaccuracy. Second, we study the effect of using heuristic path completion estimators in search. In general, using a more conservative estimator increases search quality but commensurately increases the amount of search and therefore computation time. Third, we discuss results from an April 2015 deployment int he Pacific Ocean and compare model accuracy and float control performance.

Introduction

The state of the ocean affects the environment and climate, thus affecting food production, defense, and leisure. As such, ocean dynamics is an important area of study that currently uses a variety of different techniques to measure ocean conditions. One technique involves the use of robotic marine vehicles such as floats, gliders, and autonomous underwater vehicles (AUV) to measure conditions such as currents, salinity, and temperature in a dynamic way. Another technique uses moored buoys, which allow scientists to collect data at a fixed location over time. However, a couple of drawbacks to physically mooring a buoy are that it involves significant financial investment and the location cannot be changed after installation.

As an alternative, a *virtual mooring* is proposed in which a dynamically controlled vehicle uses a control policy in order to maintain its position. Specifically, one proposal is to deploy a vertical profiling float to the location of desired data collection and to use predictive ocean models to plan a control sequence for changing depths that best keeps the float near the same latitudinal and longitudinal location using the ocean currents. A vertical profiling float can change depths, but does not have any lateral propulsive power, meaning that the float is carried solely by the ocean current in the latitudinal and longitudinal directions. By purposefully changing depths it is possible to harness the motion of the ocean to direct the float. This method has multiple benefits over using a physical mooring. First, the float could be retrieved and redeployed when desired. Second, there is more flexibility since the float could be programmed to track a moving target or to drift to facilitate deployment or retrieval. Third, the deployment would be less expensive than building a physical anchor location.

Asset	Control	Speed	Longevity	Cost
Floater	None	None	Weeks	$100's
Vertical Profiling Float	Vertical	∼0.1 m/s	Years	$10K's
Seaglider	Horizontal	∼0.5 m/s	Months	$50's - $100'sK
AUV	Horizontal & Vertical	∼2.5 m/s	1 Day - Weeks	$100K - $M

Table 1: Characteristics and costs for different families of marine vehicles.

Although using an AUV would provide better control to remain at a fixed location, more capable vehicles are more expensive. Table 1 shows approximate costs for families of marine vehicles (Woods Hole Oceanographic Institution ; Sanford et al. 2005; Eriksen et al. 2001; YSI Systems ; OceanServer Technology, Inc. ; Bluefin Robotics Corporation ; Kongsberg Mairtime AS).

Scientists studying the characteristics of the ocean would ideally like to be able to collect data at all depths and all times at a particular location. Obviously, a single float cannot be at all depths at all times and therefore must profile to collect data across the depths. Using a predictive ocean model it is possible to generate a control sequence for the float to change depths in a way that keeps it as close to the desired location of data collection as possible.

To analyze the benefit of planning a path for a float to act as a virtual mooring, compared to allowing the float to continuously profile, an Electromagnetic Autonomous Profiling Explorer (EM-APEX) (Sanford et al. 2005) vertical profiling float is modeled. During a deployment, each time that the EM-APEX float surfaces, it transmits its data and can be commanded to profile to a different depth. This allows for two possible control strategies. First, in *batch planning*, the float plans once for the deployment based on the best model of the ocean currents. In *continuous planning*, at each float surfacing, we re-plan the control sequence of the float during each surfacing. This enables the planning to incorporate: (1) the most up to date information about the location of the float and (2) the most up to date ocean current model. [1] We believe that using this opportunity to re-plan the control sequence using the best information can improve the path when there is information gain in the model.

Since directing the float relies solely on the ocean currents, success is based heavily on the planning process, and thus on the ocean model used for planning. Modeling ocean currents is a tremendously challenging problem - as a chaotic system it is not feasible to model the ocean perfectly and producing even modestly accurate predictions is quite challenging. As we investigate the use of predictive ocean current models to plan underwater vehicle paths, predictive accuracy can dramatically affect the utility of our approach. Therefore studying the impact of model accuracy on path planning is of great import. Additionally, methods of measuring predictive model accuracy and correlating these to planning utility is of great interest.

To study this relationship between model accuracy and planning utility, we use the Regional Ocean Modeling System (ROMS) (Chao et al. 2009; Li et al. 2006; Farrara et al. 2015). Specifically we artificially create models with varying degrees of fidelity. Because it is very expensive to perform a physical deployment in the ocean, we mimic a deployment. In a deployment we plan in a model and we execute in the physical world. In ROMS, we create one or more planning models of varying fidelity to a nature model. We then construct plans in a planning model and execute in the nature model. The planning models were five other ROMS models with decreasing fidelity. In order to evaluate the models, we use the models for path planning of an Electromagnetic Autonomous Profiling Explorer (EM-APEX) (Sanford et al. 2005) vertical profiling float attempting to act as a virtual mooring.

The paths were planned in all of the models and compared to the execution in the nature model. The paths were also re-planned using continuous planning during execution using each of the models to compare to the results without re-planning. We show that using current model to plan a path for a vertical profiling float to act as a virtual mooring can improve its station keeping compared to a naive control strategy and that re-planning the path during execution is an effective technique. Furthermore, this paper aims to show how the information in a model affects the benefit of planning a path as well as the efficacy of re-planning during execution. These ideas have also previously been explored in (Troesch et al. 2016).

The remainder of this paper is organized as follows. First, we describe the ROMS modeling framework that we use as an imperfect predictive model of ocean currents. Second, we describe the batch and continuous float planning algorithms. Third, we describe our results in simulation - highlighting both the effect of model accuracy and heuristic path completion estimation on algorithm performance. Fourth, we describe results from an April 2015 field deployment off the coast of California. Fifth, we describe related and future work, and conclusions.

Ocean Models

A number of ocean models have been developed including the Harvard Ocean Prediction System (HOPS) (Robinson 1999), the Princeton Ocean Model (POM) (Mellor 2004), the Hybrid Coordinate Ocean Model (HYCOM) (Chassignet et al. 2007), and ROMS (Chao et al. 2009). As described in the Introduction, for our experiments we use the ROMS model. However, our techniques naturally extend to any cell-based, predictive model with information about ocean currents over multiple depths and an extended period of time, and thus any of these models could be used for the path planning. Indeed, when multiple modes are available it is also possible to use them in as an ensemble to further enhance results (Wang et al. 2013).

ROMS is a discrete, cell-based, predictive model of the ocean. We used the California coast configuration near the Monterey Bay, which is a grid of 3 km by 3 km in the latitudinal and longitudinal directions, 1 hour in the time dimension up to 72 hours long, and fourteen depths from 0 m to 1000 m in non-uniform intervals. The currents in the grid vary over space and time. At deeper depths, the currents tend to be slow and uniform, conversely, the surface currents are faster and more variable.

As previously stated, it is not feasible to perform an ocean deployment of the planned paths for this experiment, so an approximation using a ROMS model for the ocean is used instead. This model is the best possible representation from ROMS and is referred to as the nature model. Five different planning models were used for this experiment. The difference between the models is the number of days advanced prediction that is used in generating the model. Fewer days of advanced prediction means a higher fidelity model and thus means that the model is closer to the nature model. We used 2, 4, 6, 8, and 10 days of advanced prediction for the planning models. A summary of the inputs for the ROMS models is shown in Table 2.

To show how the model information decreases with more advanced prediction of the model, the correlation coefficient of the currents between the nature model and each of the planning models was calculated. All of the velocities in a 41 grid by 41 grid subsection of the ROMS model across all depths and times were used for the calculation. In

[1] While in our deployment scenarios the model does not change significantly on the timescale of our plan execution so that majority of the gain is from (1), other operational scenarios might exist where (2) may provide significant value

	Planning Models	Nature Model
Archiving, Validation and Interpretation of Satellite Oceanographic (AVISO) sea surface height data	x	x
Advanced Very High Resolution Radiometer (AVHRR) sea surface temperatures	x	x
Moderate Resolution Infrared Spectroradiometer (MODIS) sea surface temperatures	x	x
GOES satellite sea surface temperatures	x	x
High Frequency (HF) radar surface current data		x
Monterey Bay Aquarium Research Institute (MBARI) M1 mooring vertical profiles of temperature and salinity	x	x
Ship sea surface temperatures	x	x
Number of days advanced prediction	2, 4, 6, 8, 10	1

Table 2: ROMS inputs for the planning and nature models.

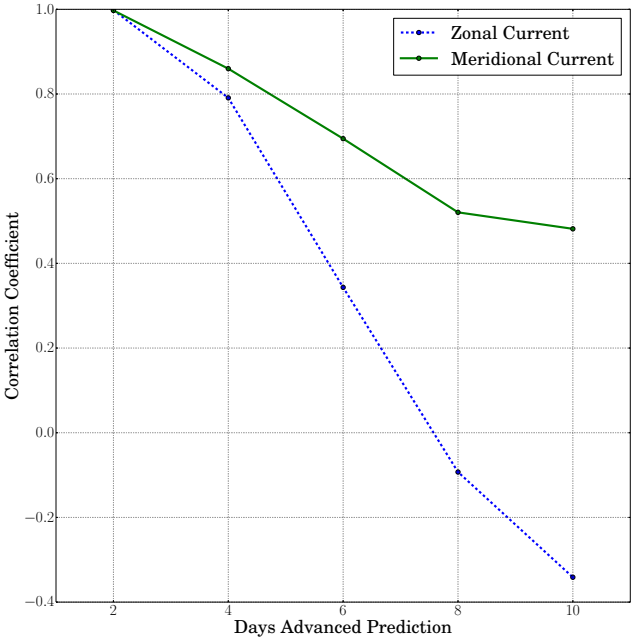


Figure 1: The correlation coefficients of the zonal and meridional currents between the different planning models and the nature model over a 41 by 41 3km x 3km cell subsection of the model averaged over all depths and times.

other words, Z_{nature} is a vector of the zonal (west-east) currents in the selected subregion of the nature ROMS model at all depths and times. The vector M_{nature} contains the corresponding meridional (north-south) currents. Similarly, Z_{plan_x} and M_{plan_x} contain the zonal and meridional currents of the same data in the planning model using x days of advanced prediction, respectively. Using these vectors, the zonal correlation coefficients, ρ_{Z_x}, and the meridional correlation coefficients, ρ_{M_x}, could be calculated with the following equations

$$\rho_{Z_x} = \frac{\mathrm{Cov}(Z_{plan_x}, Z_{nature})}{\sigma_{Z_{plan_x}} \sigma_{Z_{nature}}}$$

$$\rho_{M_x} = \frac{\mathrm{Cov}(M_{plan_x}, M_{nature})}{\sigma_{M_{plan_x}} \sigma_{M_{nature}}}$$

where Cov is the covariance function and $\sigma_{Z_{plan_x}}$, $\sigma_{Z_{nature}}$, $\sigma_{M_{plan_x}}$, and $\sigma_{M_{nature}}$ are the standard deviations of the referenced vector.

Figure 1 summarizes the values of the correlation coefficients and shows how the information in the model becomes less correlated to the nature model as the number of advanced prediction days increases. In fact, the correlation coefficient for the zonal currents using 8 and 10 days of advanced prediction has a negative value, indicating that the

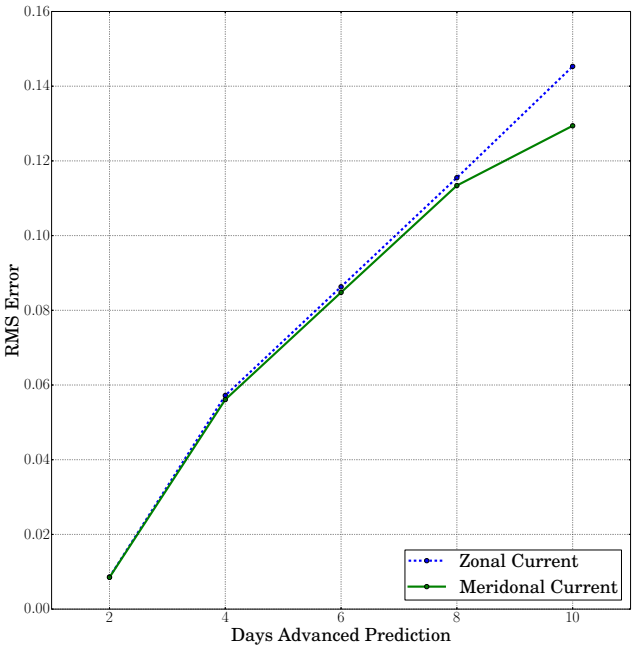


Figure 2: The root mean square error of the zonal and meridional currents between the different planning models and the nature model over a grid that encompasses the entire search space over all locations at all depths and times.

Days Advanced Prediction	RMS Error
2	0.00855
4	0.05665
6	0.08555
8	0.11445
10	0.13735

Table 3: Root mean square error used for each model with the specified number of days advanced prediction.

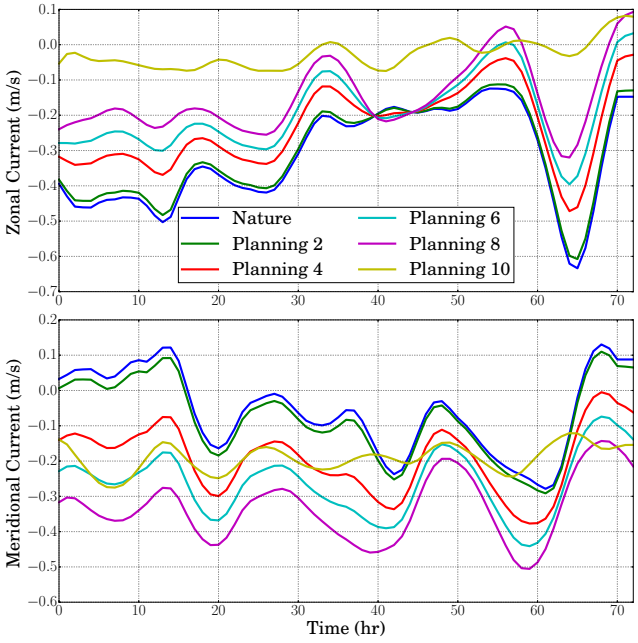


Figure 3: The currents at a surface point over 72 hours in all of the planning models and the nature model.

currents are inversely correlated to the currents in the nature model, which causes the value of planning a path to decrease.

Just as the correlation coefficient decreases with the increase in number of advanced prediction days, the root mean square (RMS) error increases. Figure 2 shows the RMS error between each planning model and the nature model using the currents over the search space that encompasses all of the goal locations using all depths and times. The average RMS error between the zonal and meridional currents for each of the planning models will be used in the rest of this paper as a measure of the fidelity of the model, which in turn could be used as a comparison and reference point in physical deployments to analyze the fidelity of the model being used. The values of RMS error used for each model are shown in Table 3.

Another way to visualize how the currents vary across the different models is by looking at the currents at one location over time across all models. Figure 3 shows the zonal and meridional currents for the same location in all models over time. The currents of the planning model with 2 days of prediction are very close to the nature model, but the other planning models diverge.

Furthermore, Fig. 4 demonstrates how the exact same control sequence executed over 24 hours in the different models results in very different path positions.

From Fig. 1, Fig. 2, Fig. 3, and Fig. 4 it is clear that there is different value in the information provided by the different planning models. Specifically that with further advanced prediction there is a decrease in the predictive accuracy of the model as indicated by the decrease in correlation coefficient and increase in RMS error. Furthermore, that these varying information models will predict significantly varying paths and consequently planing in the different models can produce significantly different paths.

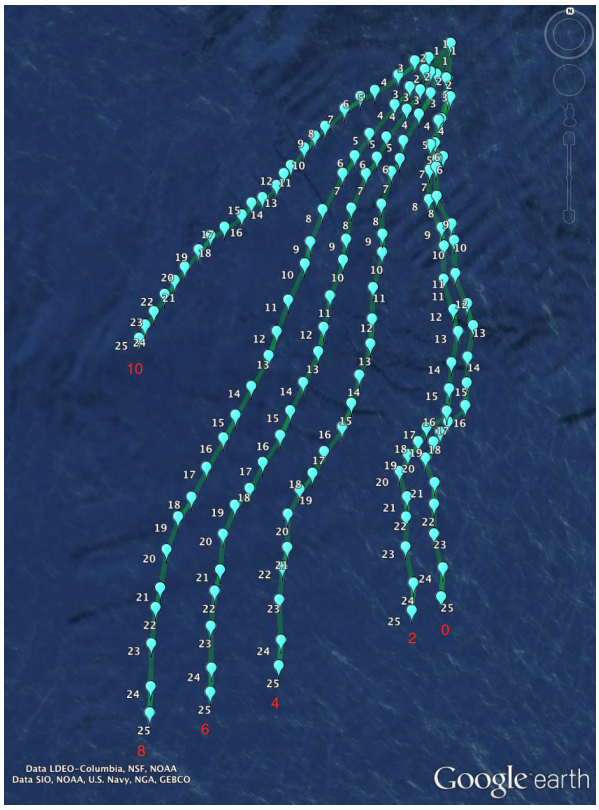


Figure 4: The same executed control sequence in all of the planning models and the nature model starting at the same location over 24 hours. The number 0 labels the path for the nature model and the other numbers indicate the number of planning days in the model used for that path.

Batch Planning

The path planner for this experiment generates a control sequence for a vertical profiling float to act as a virtual mooring. As described in the introduction, a vertical profiling float can change depths both to gather data and to make use of the currents to stay near a desired location. The float can be programmed to move between the surface and the profiling depth, to remain at the surface, or to remain at the profiling depth. Stopping and staying at a depth is restricted to the surface and profiling depth in order to best mimic the behavior of the EM-APEX floats. Depending on the needs of the scientist, it may be more important to stay at a fixed location or to gather more data by profiling. In this way a trade-off can be made between staying at the desired location and performing more profiles.

Even though the problem space is continuous, to make the search tractable, the control sequence of the float is determined in a discrete manner. At each decision point, the float can remain at the depth that it is at, which must be

the surface or the profiling depth, or it can move between the surface and the profiling depth. If the float remains at a depth, the duration is equivalent to the time it takes to move between the depths. To mimic the behavior of a deployed EM-APEX float, each time that the float returns to the surface from the profiling depth, it must remain at the surface to upload the collected data. The duration of the upload is 35 minutes. If the float decides to remain at the surface, it must once again upload the collected data for 35 minutes after the duration of remaining at the surface.

The value of the currents used for determining the motion of the float is based on the position of the float as well as the time. Every approximately 42 seconds (the amount of time that it takes the float to vertically move half of the smallest depth step), the position is updated and the current information is interpolated among the eight closest grid points. This current is used to determine the motion of the float until the next interpolation step or the allotted time step for the node has been reached.

The algorithm that is used to perform the search is an A* algorithm. The objective function that is used to score the paths was designed to make a trade-off between remaining close to a desired location and performing more profiles, depending on the desires of the user. The equation is

$$g(n) = \sum_{n} \sum_{d} (w_T \Delta T_d + w_D \Delta D_d)$$

where n are the nodes in the path, d are the possible depth choices (in this case, the surface or the profiling depth), w_T and w_D are weighting terms, ΔT_d is the time in seconds since the float was at depth d, and Δ_D is the distance in kilometers that the float was from the goal location when it was last at depth d. In other words, each time a node is added to the path, the most recent node at the surface and the profiling depth is used to calculate $(w_T \Delta T_d + w_D \Delta D_d)$ and this sum is added to the score. Since ΔT_d measures the time since the float was at the other depth, it is a good metric for determining the time since the last profile was performed. Therefore, a higher w_T to w_D ratio favors less time between profiles and thus favors more profiles. Similarly, a lower w_T to w_D ratio favors control sequences that keep the float as close as possible to the desired location.

The heuristic function used in the A* algorithm simply assumes that the score will increase at the same rate for the remainder of the path. Therefore, the equation is

$$h(n) = \left(\frac{g(n)}{T} \right) (\text{mission duration} - T)$$

where T is the time since the beginning of the mission when the float is at node n.

The equation used by the A* algorithm is thus

$$f(n) = g(n) + w * h(n)$$

where w is a weighting given to the heuristic function.

A path is considered complete once its duration reaches that of the desired mission length.

The execution of the algorithm is summarized in the following pseudocode for Algorithm 1.

Algorithm 1 Batch Planning A* Algorithm

function BATCHPLANNER($startPath$)
(***Note:** Model = Planning Model)
  $Q \leftarrow startPath$
  while Q not empty **do**
    $curPath \leftarrow$ lowest f score path in Q
    if $curPath$ needs to upload **then**
      $newPath0 \leftarrow curPath$ + node at surface
    else
      $newPath1 \leftarrow curPath$ + node at surface
      $newPath2 \leftarrow curPath$ + node at profiling depth
    end if
    if any $newPath$ duration $>$ mission duration **then**
      return $curPath$
    end if
    Q.push all $newPath$s
  end while
end function

Continuous Planning

During every data upload when the float resurfaces, there is an opportunity to re-plan the path of the float based on the best information of its location. Each time the float performs an upload, the location at the true position of the path so far in the execution can be used in Algorithm 1 to plan the rest of the path using the planning model. The next part of the control sequence is executed according to the re-planned path. The process of re-planning is repeated each time that the float re-surfaces until the duration of the mission has been completed. The control sequence is always executed in the nature model and planned in the planning model. A summary of the re-planning algorithm is shown in Algorithm 2.

Algorithm 2 Continuous Planning Algorithm

function CONTINUOUSPLANNER($startNode$)
(***Note:** Model = Nature Model)
  $path \leftarrow startNode$
  $controlSeq \leftarrow$ BATCHPLANNER($path$)
  while $controlSeq$ duration $>$ $path$ duration **do**
    while $path$ does not need to upload **do**
      $nextNode \leftarrow controlSequence$.next
      $depth \leftarrow$ depth of $nextNode$
      $path \leftarrow path$ + node at $depth$
    end while
    $controlSeq \leftarrow$ BATCHPLANNER($path$)
  end while
  return $path$
end function

Experimental Procedure

The experiment was performed over 100 goal locations in a 10 by 10 grid, which are shown in Fig. 5. At each location, the five different planning models were used to plan a control sequence using Algorithm 1, which were then executed

in the nature model. Each of these control sequences were then also re-planned with continuous planning using Algorithm 2. This process was repeated over all of the locations using 3 different trade-offs in the objective function, specifically a w_T to w_D ratio of 1.6, 4.8, and 8.0 were used, as well as 6 different weights for the heuristic function from 0 to 1 in steps of 0.2.

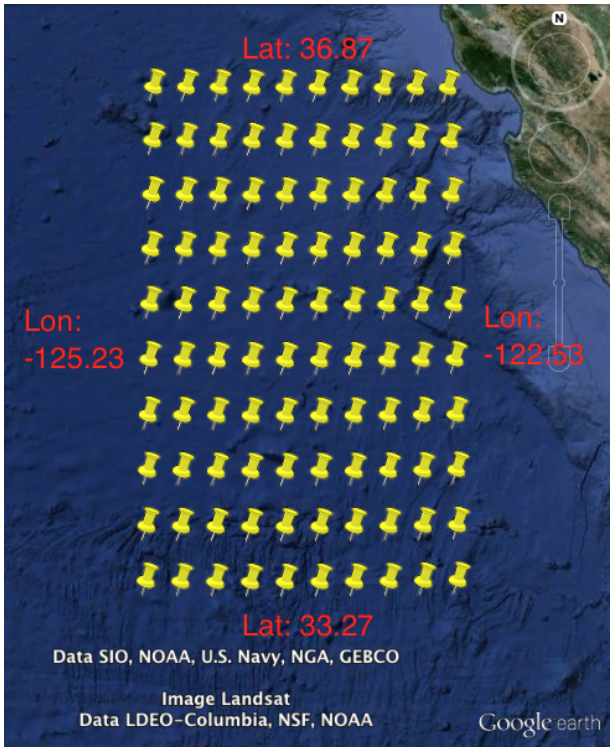


Figure 5: The 100 starting and goal locations used for the experiment. The latitudes and longitudes labeled indicate the boundaries of the grid.

The start location of the float was always set to be the same as the goal location. The profiling depth was 500 m and the vertical speed of the float was 0.12 m/s, resulting in a step time of approximately 69 minutes. As previously stated, the time required for the data upload was 35 minutes. The total duration of the mission was 24 hours. Given these inputs, the maximum number of profiles that can be achieved by continuously profiling is 8.

In order to evaluate the results of the planned paths, baselines were developed that evenly space the profiles, from 0 to the maximum of 8 within the mission duration. In order to attempt to prevent bias in the baselines, three different ways of evenly spacing the profiles were used. The first, referred to as the surface baseline, evenly spaces the profiles by remaining at the surface between profiles that consist of going to the profiling depth and resurfacing immediately. The second baseline, referred to as the even baseline, evenly spaces the profiles by remaining at the surface and at the bottom of the profiles at even intervals. The final baseline, referred to as the deep baseline, remains at the profiling depth during each profile and only surfaces in between profiles for the amount of time required to upload the data. As an example, the depths of the different baseline paths for 3 profiles is shown in Fig. 6.

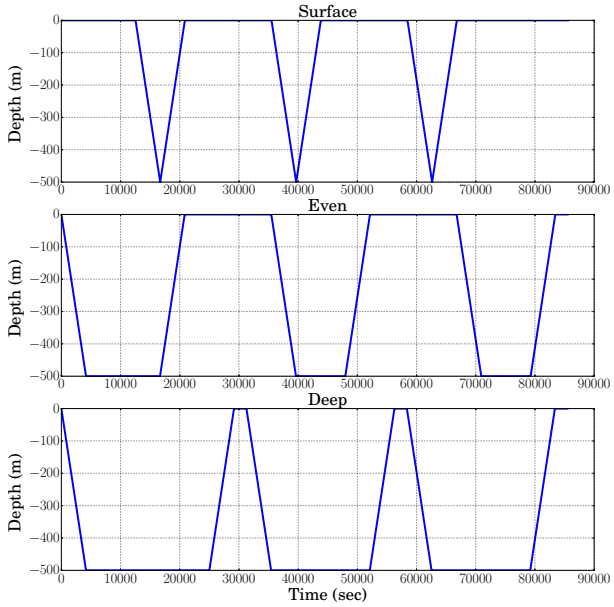


Figure 6: The control sequences for 3 profiles using the surface, even, and deep baselines.

Evaluation of the Heuristic

In order to evaluate the heuristic function and the effect of the chosen weight, an analysis was done on all of the paths in the planning models. Since the scores are heavily dependent on the w_T to w_D ratio in the objective function, the analysis was split among the three executed ratios. The average score and number of node expansions to find a complete path across all locations and planning models was found for each w_T to w_D ratio, which can be seen in Fig. 7 and Fig. 8.

As more weight is given to the heuristic function, the strength of the path decreases, which can be seen by the increasing scores in Fig. 7. However, at the same time, the number of nodes expanded decreases, which can be seen in Fig. 8, meaning that the time to find a complete path is improved.

Empirical Evaluation in Simulation

Using a heuristic weight of 0 in order to ensure using the best planned path, for each w_T to w_D ratio, each ROMS model, and each location, the executed path in the nature model was found for both batch planning and continuous planning. As an example to show how the control sequence and the score changes from batch planning to continuous planning, Fig. 9 shows the depth and score over time using 6 days of advanced planning and a w_T to w_D ratio of 4.8 at location (35.67° lat, -123.73° lon).

The average scores across all of the locations for each w_T to w_D ratio and each ROMS model were then calculated for

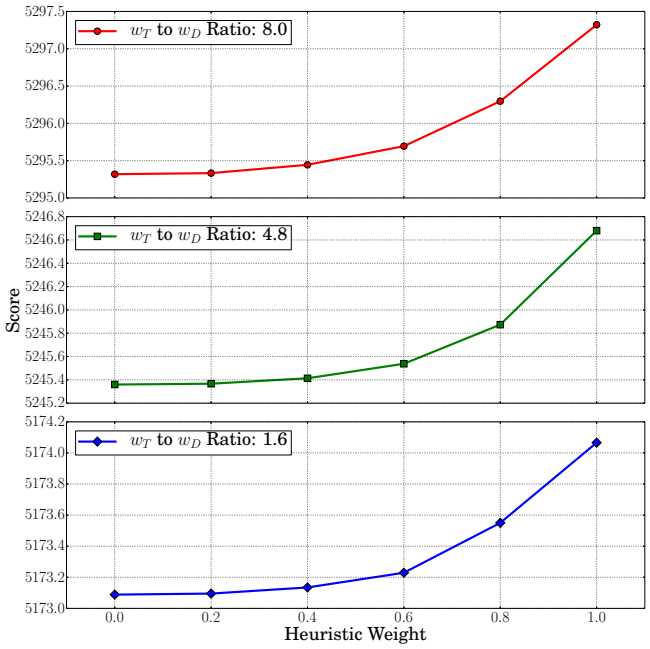


Figure 7: Average score across all planning models and locations for different heuristic weights.

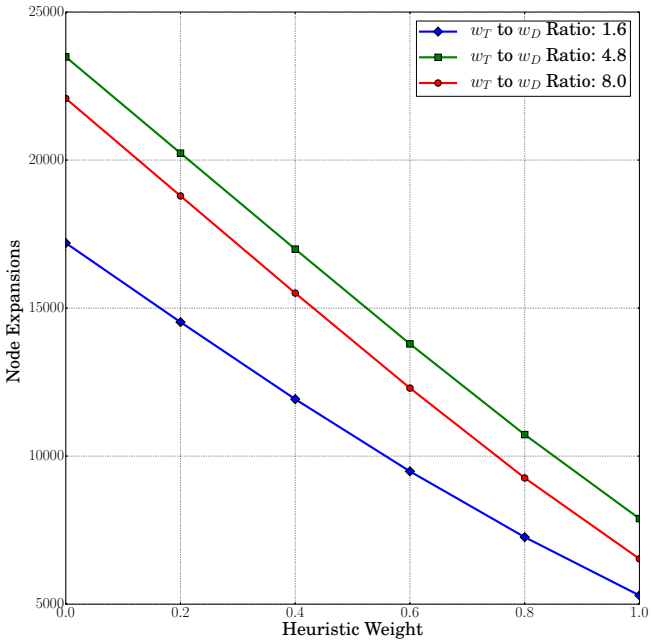


Figure 8: Average number of node expansions required across all planning models and locations for different heuristic weights.

both batch planning and continuous planning, separately. In order to give a fair comparison to the baselines, the average number of profiles for each w_T to w_D ratio and ROMS model combination was found from both batch planning and continuous planning. This number of profiles was then used for the baselines with those same inputs. Since the average number of profiles for each w_T to w_D ratio was not the same for each ROMS model, the baseline scores are not constant across a single ratio. Comparing the baselines to the different planning methods revealed that the surface baseline performed too poorly (with scores over 100,000) to be represented on the same scale as the other results, therefore the surface baselines are not presented on the graphs showing the results. The comparison of the scores using the different planning models can be seen in Fig. 10 for the w_T to w_D ratio of 1.6, in Fig. 11 for the w_T to w_D ratio of 4.8, and in Fig. 12 for the w_T to w_D ratio of 8.0.

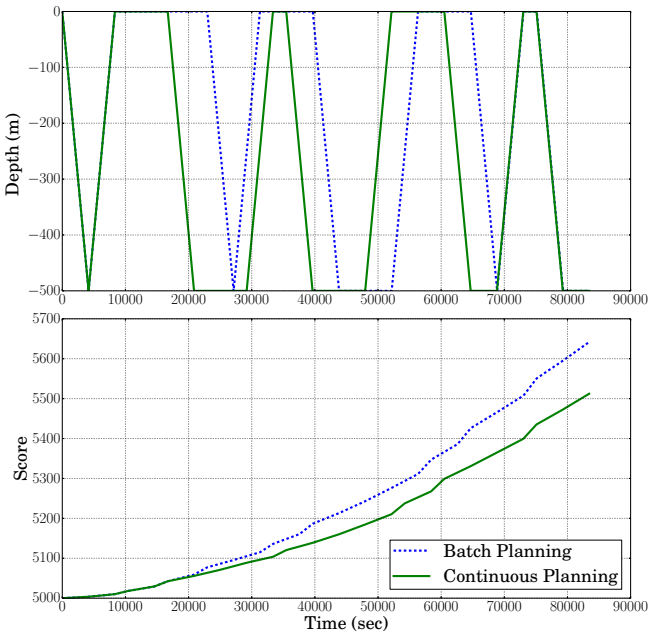


Figure 9: Batch and continuous planning using 6 days of advanced planning and a w_T to w_D ratio of 4.8 at location (35.67° lat, -123.73° lon).

From these figures, it is clear that on average planning a control sequence using any of the models performs better than simply using evenly spaced out profiles. Furthermore, they all show that the benefits of planning are increased when the planning model has better knowledge of the currents in the nature model. As the RMS error in the planning models increases, the benefits of planning a control sequence over the baseline decreases.

When considering continuous planning, the benefit is also related to the amount of knowledge in the planning model. Obviously, when planning using the nature model, there is no increase in benefit from continuous planning, since the position was already correctly known during the planning process and thus the continuous plan and the batch plan are identical and there is no added benefit to continuous planning. However, as the planning model and the nature model diverge, the updated information at each surfacing provides a benefit to the planner.

Looking at Fig. 13, the planning model with just a value

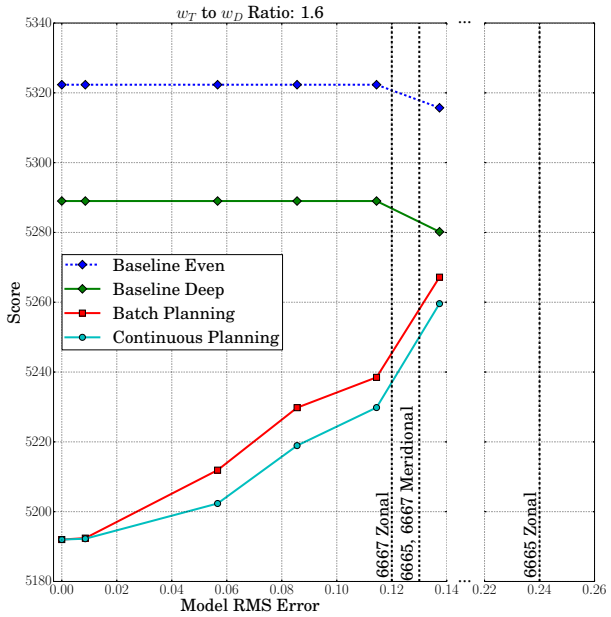


Figure 10: The average batch and continuous planning scores when the w_T to w_D ratio was 1.6 using each planning model compared to the average baseline scores of the same average number of profiles. The RMS errors for the zonal and meridional currents for two deployed floats discussed in the next section are also shown.

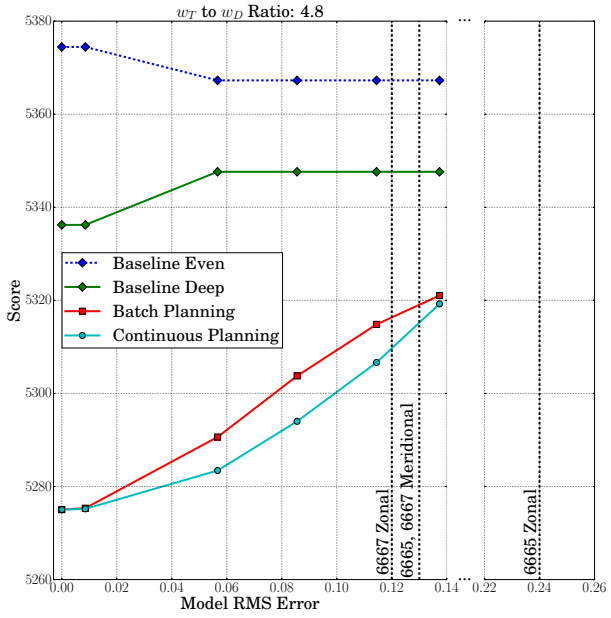


Figure 11: The average batch and continuous planning scores when the w_T to w_D ratio was 4.8 using each planning model compared to the average baseline scores of the same average number of profiles. The RMS errors for the zonal and meridional currents for two deployed floats discussed in the next section are also shown.

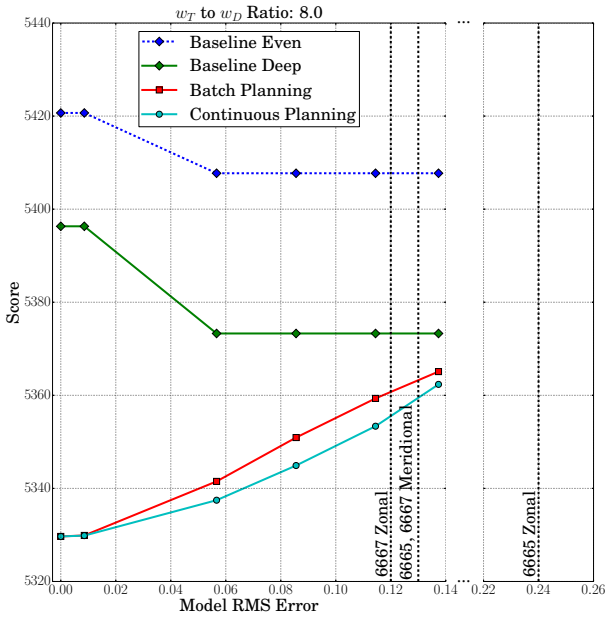


Figure 12: The average batch and continuous planning scores when the w_T to w_D ratio was 8.0 using each planning model compared to the average baseline scores of the same average number of profiles. The RMS errors for the zonal and meridional currents for two deployed floats discussed in the next section are also shown.

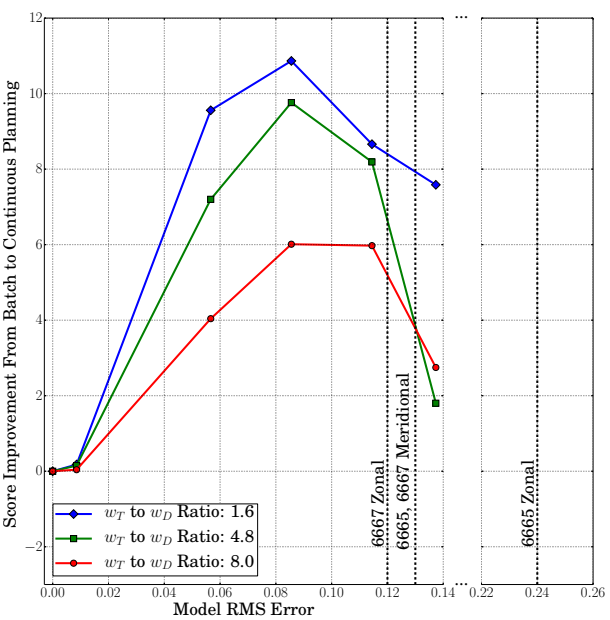


Figure 13: The average batch and continuous planning scores when the w_T to w_D ratio was 8.0 using each planning model compared to the average baseline scores of the same average number of profiles. The RMS errors for the zonal and meridional currents for two deployed floats discussed in the next section are also shown.

of 0.0085 RMS error is very similar to the nature model, therefore although there is a benefit to continuous planning, the added benefit is not significant since the executed path would not have strayed far from the planned path. Looking at the next three higher values of RMS error, continuous planning has a consistent advantage over batch planning. With the models with the highest values of RMS error, the benefit of continuous planning decreases. This could be due to the fact that starting at such a large value of RMS error, the model has such poor information gain that even though the location is updated at every surfacing, the model is not good enough to make a significantly better plan.

Empirical Results in Deployment

A prior version of this software was deployed to control EM-APEX floats during an April 2015 deployment in support of an AirSWOT (Jet Propulsion Laboratory a; b) field experiment in the coast off of Monterey Bay, California. In this field experiment, the goal was to keep EM-APEX floats near features of interest identified manually by scientists. The overall AirSWOT deployment goals are representative of the intended scientific use case for these planning tools.

The overall AirSWOT deployment was to test out an Airborne science instrument by providing corroborative data over interesting science features using in-situ instrumentation (floats, ships) and remote sensing data (from overflying spacecraft). The AirSWOT instruments were scheduled to fly in a coverage pattern over specific areas chosen to overlap satellite overflights.

Three EM-APEX floats were to be deployed to be near satellite overflights and airborne overflights. The float planning tool was used to evaluate potential deployment locations by predicting the projected drift path of the floats.

Figure 14 shows the variability of the expected float drift based on the deployment location. The blue paddle indicates the start location and the green path shows where the float was projected to drift over time.

Sites for each of the three float deployments were screened for projected stability and hand selected by the experiment team.

Additionally, two of the three EM-APEX floats were allowed to be controlled dynamically from shore by an earlier version of the float planning software. This prior float planning software received the satellite phone updated location each time the target float surfaced. Because of connectivity issues, the float planning software could not receive this data rapidly enough to generate a new plan for transmission to the float during this surface cycle as the float was only on the surface approximately 30 minutes each cycle. Instead the planner could only assert a plan with a 1 surface cycle lag. Therefore the plan communicated to the float to be executed after surface cycle n was only based on the actual position from cycle $n-1$ plus the projected drift from cycle $n-1$ to cycle n.

The EM-APEX float tracks planned and executed are shown for floats 6665 and 6667 in Fig. 15. The yellow point indicates the start location of the float. The actual location of

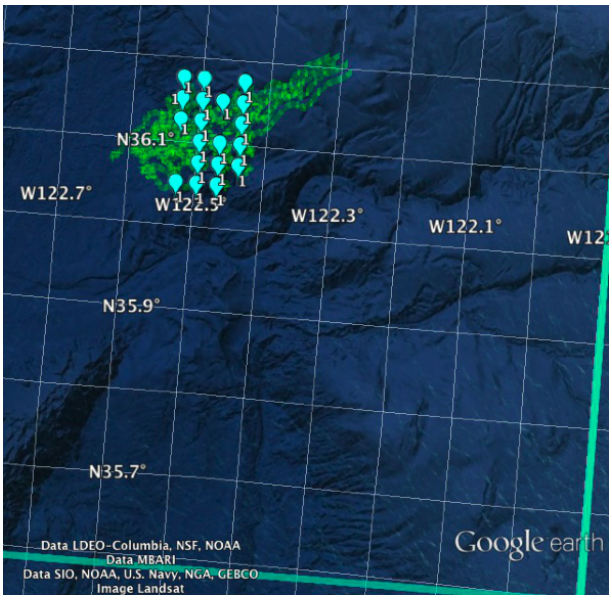


Figure 14: Expected float drift based on starting location. The blue paddle indicates the starting location and the green paths shows the drift.

the float at each surfacing is shown in blue with the arrows indicating the direction. The red points show where, at each step of re-planning, the float was predicted to travel. Since the re-planning was performed two cycles ahead, two surfacing locations are displayed. As shown, the expected control for neither of the floats performed very well.

This poor performance is not surprising as the current velocities in the ROMS model in the area near both floats was not very accurate, as can be seen in Fig. 16 and Fig. 17. Because EM-APEX is designed to get velocity data, it provides a good opportunity to compare collected data to the ROMS model. The plots show the zonal and meridional currents of the interpolated point in the ROMS model at each location where velocity data was collected by the float. In fact, when comparing these values to the values used in the simulated experiment in Table 3, the RMS errors experienced in the deployment are in the range of the worst models. The RMS errors for the meridional current for both floats were similar to the worst model, the RMS error for the zonal current for float 6667 was similar to the second to worst model, and the RMS error for the zonal current of float 6665 was almost twice as much as the worst model. This can be seen by the lines indicating the RMS errors of the currents for the deployed floats compared to the simulated models in Fig. 10, Fig. 11, Fig. 12, and Fig. 13. Starting in this range the benefits of continuous planning started to decrease and even batch planning scores started to approach those of the deep baseline control sequence.

The poor correlation between the ROMS and the float collected velocities is most likely due to the front that was coming in during the deployment that even caused the deployment to be cut short.

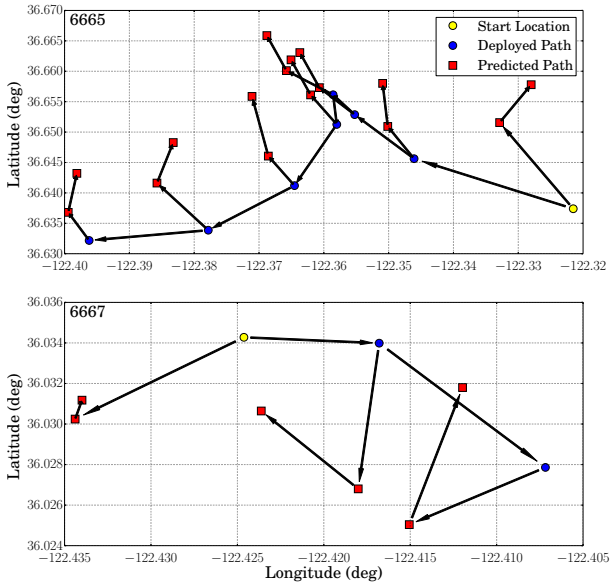


Figure 15: The deployed path and predicted path at each replanned step in the deployed path for floats 6665 and 6667.

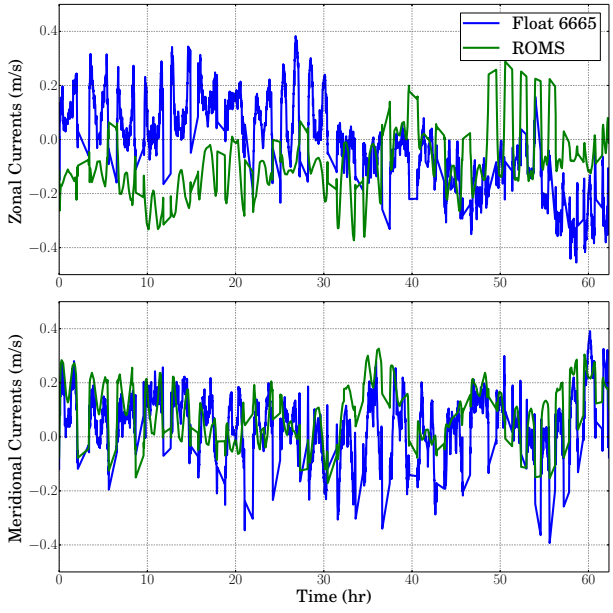


Figure 16: Zonal and Meridional currents found along the path for float 6665 in the ROMS model and actually experienced in the deployment.

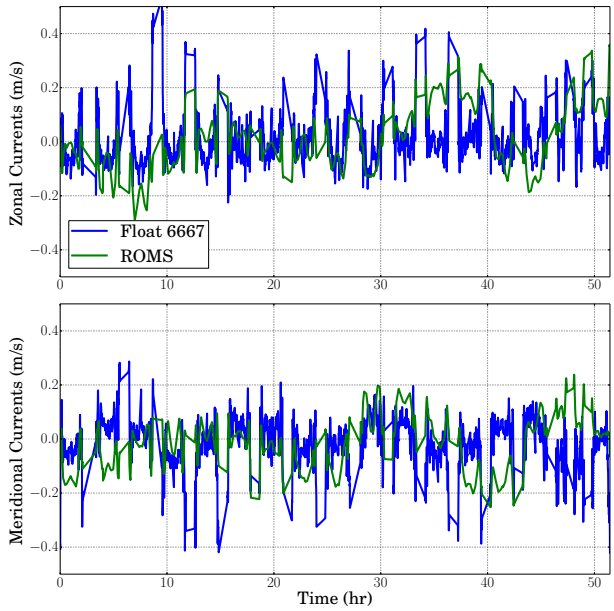


Figure 17: Zonal and Meridional currents found along the path for float 6667 in the ROMS model and actually experienced in the deployment.

Table 4 summarizes the RMS errors and correlation coefficients from the data presented in Fig. 16 and Fig. 17.

6665	**RMS Error**	**Corr Coef**	**Min Vel (m/s)**	**Max Vel (m/s)**
Zonal	0.24	-0.28	-0.46	0.38
Meridional	0.13	0.42	-0.39	0.39
6667	**RMS Error**	**Corr Coef**	**Min Vel (m/s)**	**Max Vel (m/s)**
Zonal	0.12	0.33	-0.23	0.53
Meridional	0.13	0.05	-0.42	0.29

Table 4: Root mean square error and correlation coefficient for the deployed velocities collected by the EM-APEX floats compared to the ROMS velocities. The minimum and maximum velocities collected by the floats is also shown.

The results from the April 2015 deployment reinforce the thesis of this paper. In cases where the current model provides significant information, the model information can be used improve to float control. In cases where the model provides no or bad information, performance will be poor and even in some cases worse than open loop algorithms such as constant profiling.

Related Work

Path planning for underwater vehicles has been widely studied; however, most of this work focuses on marine vehicles with greater control such as Sea gliders and Autonomous Underwater Vehicles (AUVs). A notable exception is (Dahl et al. 2011) which examined the problem of optimizing coverage across the oceans for a large number of floats, but only

considered a constant depth and a greedy algorithm. Much more research has been done on glider planning, where there is some control for choosing a direction of motion, but it is less than the current velocity. (Thompson et al. 2010) also uses the ROMS model, but calculates reachability envelopes using wavefront propagation for glider path planning. The work in (Eriksen et al. 2001) describes Seaglider, a glider that is manually controlled from the shore, and is sometimes controlled to maintain position. No ocean model similar to ROMS was used. (Alvarez, Garau, and Caiti 2007) also does not use an ocean model, but instead uses synthetic data with general algorithms to control a set of floats and gliders. Like the work in this paper, (Rao and Williams 2009) uses an A* graph search algorithm; however, that work assumes that currents change slowly with time and compute the path across many nodes in a single time step, whereas we have many time steps within a single cell. Instead of trying to remain near a specific location, (Pereira et al. 2013) focuses on gliders that are attempting to avoid surfacing in dangerous areas, such as shipping lanes. (Grasso et al. 2010) focuses on the prediction of the glider location, analyzes the accuracy of the predictive model, and uses a physics based control model. Using an asset with more control, (Cashmore et al. 2014) explores the problem of autonomously maneuvering near a site for inspection using an AUV with probabilistic modeling for uncertainty. Autonomous marine vehicles have even been proposed to explore Titan, a moon in the Saturnian system (Pedersen et al. 2015; ESA/NASA 2009; Stofan et al. 2009). (Leonard et al. 2010) present a controls-based methods to guide a set of gliders along coordinated paths. Their approach does not use a model of currents, but does adaptively guides assets back on to paths and desired spacing them along racetracks if the vehicles are perturbed by currents. Therefore this approach would counter-act currents but does not use a projective forward model as our approach does.

Continuous planning has become more prevalent in recent years and the evolution of this planning technique, with respect to multiple assets, is clearly described in (Durfee et al. 1999). (Myers 1999) describes a Continuous Planning and Execution Framework (CPEF), which integrates planning and execution through plan generation, monitoring, execution, and repair. Using an iterative repair process, as well as user interaction, CPEF is able to plan in unpredictable and dynamic environments, which is shown through tests in a simulation of an air-campaign for dominance. (Chien et al. 2000) presents Continuous Activity Scheduling Planning Execution and Replanning (CASPER), which also uses iterative repair as part of continuous planning, specifically for autonomous spacecraft control. (Branch et al. 2016) uses continuous planning to control AUVs, Seagliders, and Wave Gliders, also using different fidelity ROMS models, to follow short distance patterns.

Future Work

There are many different extensions that are possible from this work. A different objective function that guides the float along a line or a moving point could be developed. Allowing the float to move to any depth at any time, instead of requiring full profiles to the same depth could be tested. Deployment and retrieval cost could be included in the objective. Different assets could be included, such as a glider or AUV, to give more flexibility in the control. Multiple assets of different types could be controlled simultaneously with different objectives and goal locations. New methods for understanding the information gain in the different models could be created as well as using the information about the predictive accuracy in the model to change how the planning is performed. For example, if it is known that the model performs poorly after a particular time, that information could be used to adjust the planning algorithm.

Conclusions

When performing predictive path planning for underwater vehicles, the model used to represent the ocean always has some limitations in terms of predictive accuracy. This experiment has shown that the amount of knowledge in the planning model used to generate the control sequence for a vertical profiling float attempting to perform virtual mooring affects the benefits of performing planning over simply evenly spacing the profiles of the float. Specifically, as the predictive accuracy of the planning model decreases, represented by the RMS error of the planning model to the nature model, the benefits of planning also decrease.

One method to counteract the poor information in the planning model is to perform continuous planning. We have shown that continuous planning is beneficial when the planning model does not match the nature model, but there is still some valid information in the model used for planning. The RMS error of the model can be used to determine if continuous planning is worthwhile.

Acknowledgments

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Alvarez, A.; Garau, B.; and Caiti, A. 2007. Combining networks of drifting profiling floats and gliders for adaptive sampling of the ocean. In *IEEE Intl Conf Robotics and Automation, 2007*, 157–162. IEEE.

Bluefin Robotics Corporation. "Vehicles, Batteries & Services". `http://www.bluefinrobotics.com/vehicles-batteries-and-services/`. Accessed Feb, 2016.

Branch, A.; Troesch, M.; Chu, S.; Chien, S.; Chao, Y.; Farrara, J.; and Thompson, A. 2016. Evaluating scientific coverage strategies for a heterogeneous fleet of marine assets using a predictive model of ocean currents. In *Scheduling and Planning Applications Workshop, International Conference on Automated Planning and Scheduling 2016*. ICAPS.

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. Auv mission control via temporal planning. In *IEEE International Conference on Robotics and Automation (ICRA), 2014*, 6535–6541. IEEE.

Chao, Y.; Li, Z.; Farrara, J.; McWilliams, J. C.; Bellingham, J.; Capet, X.; Chavez, F.; Choi, J.-K.; Davis, R.; Doyle, J.; et al. 2009. Development, implementation and evaluation of a data-assimilative ocean forecasting system off the central california coast. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):100–126.

Chassignet, E. P.; Hurlburt, H. E.; Smedstad, O. M.; Halliwell, G. R.; Hogan, P. J.; Wallcraft, A. J.; Baraille, R.; and Bleck, R. 2007. The hycom (hybrid coordinate ocean model) data assimilative system. *J Marine Systems* 65(1):60–83.

Chien, S. A.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc Artificial Intelligence Planning Scheduling (AIPS)*, 300–307. AAAI.

Dahl, K. P.; Thompson, D. R.; McLaren, D.; Chao, Y.; and Chien, S. 2011. Current-sensitive path planning for an underactuated free-floating ocean sensorweb. In *IEEE/RSJ Intl Conf on Intelligent Robots and Systems (IROS), 2011*, 3140–3146. IEEE.

Durfee, E. H.; Ortiz Jr, C. L.; Wolverton, M. J.; et al. 1999. A survey of research in distributed, continual planning. *Ai magazine* 20(4):13–22.

Eriksen, C. C.; Osse, T. J.; Light, R. D.; Wen, T.; Lehman, T. W.; Sabin, P. L.; Ballard, J. W.; and Chiodi, A. M. 2001. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. *Oceanic Engineering, IEEE Journal of* 26(4):424–436.

ESA/NASA. 2009.

Farrara, J. D.; Chao, Y.; Zhang, H.; Seegers, B. N.; Teel, E. N.; Caron, D. A.; Howard, M.; Jones, B. H.; Robertson, G.; Rogowski, P.; and Terrill, E. 2015. Oceanographic conditions during the orange county sanitation district diversion experiment as revealed by observations and model simulations. Estuarine, Coastal and Shelf Science.

Grasso, R.; Cecchi, D.; Cococcioni, M.; Trees, C.; Rixen, M.; Alvarez, A.; and Strode, C. 2010. Model based decision support for underwater glider operation monitoring. In *OCEANS 2010*, 1–8. IEEE.

Jet Propulsion Laboratory. "AirSWOT". `https://swot.jpl.nasa.gov/airswot/`. Accessed Feb, 2016.

Jet Propulsion Laboratory. "Earth Science Airborne Program". `http://airbornescience.jpl.nasa.gov/instruments/airswot`. Accessed Feb, 2016.

Kongsberg Mairtime AS. "Autonomous Underwater Vehicles - AUV". `http://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/D5682F98CBFBC05AC1257497002976E4?OpenDocument`. Accessed Feb, 2016.

Leonard, N. E.; Paley, D. A.; Davis, R. E.; Fratantoni, D. M.; Lekien, F.; and Zhang, F. 2010. Coordinated control of an underwater glider fleet in an adaptive ocean sampling field experiment in monterey bay. *J Field Robotics* 27(6):718–740.

Li, P.; Chao, Y.; Vu, Q.; Li, Z.; Farrara, J.; Zhang, H.; and Wang, X. 2006. Ourocean-an integrated solution to ocean monitoring and forecasting. In *OCEANS 2006*, 1–6. IEEE.

Mellor, G. L. 2004. *Users guide for a three dimensional, primitive equation, numerical ocean model*. Princeton, NJ: Princeton University.

Myers, K. L. 1999. Cpef: A continuous planning and execution framework. *AI Magazine* 20(4):63–69.

OceanServer Technology, Inc. "Ecomapper AUV". `http://www.ysisystems.com/productsdetail.php?EcoMapper-Autonomous-Underwater-Vehicle-9`. Accessed Feb, 2016.

Pedersen, L.; Smith, T.; Lee, S. Y.; and Cabrol, N. 2015. Planetary lakelander - a robotic sentinel to monitor remote lakes. *J Field Robotics* 32(6):860–879.

Pereira, A. A.; Binney, J.; Hollinger, G. A.; and Sukhatme, G. S. 2013. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *J Field Robotics* 30(5):741–762.

Rao, D., and Williams, S. B. 2009. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conf Robotics and Automation (ACRA), Sydney*.

Robinson, A. R. 1999. Forecasting and simulating coastal ocean processes and variabilities with the Harvard Ocean Prediction System. In Mooers, C. N. K., ed., *Coastal Ocean Prediction*, AGU Coastal and Estuarine Studies Series. Wash., DC: American Geophysical Union. 77–100.

Sanford, T. B.; Dunlap, J. H.; Carlson, J.; Webb, D. C.; Girton, J. B.; et al. 2005. Autonomous velocity and density profiler: Em-apex. In *Proceedings of the IEEE/OES Eighth Working Conference on Current Measurement Technology, 2005*, 152–156. IEEE.

Stofan, E.; Lorenz, R.; Lunine, J.; Aharonson, O.; Bierhaus, E.; Clark, B.; Griffith, C.; Harri, A.-M.; Karkoschka, E.; Kirk, R.; Kantsiper, B.; Mahaffy, P.; Newman, C.; Ravine, M.; Trainer, M.; Waite, H.; and Zarnecki, J. 2009. Titan mare explorer (TiME): first in situ exploration of an extraterrestrial sea.

Thompson, D. R.; Chien, S.; Chao, Y.; Li, P.; Cahill, B.; Levin, J.; Schofield, O.; Balasuriya, A.; Petillo, S.; Arrott, M.; et al. 2010. Spatiotemporal path planning in strong, dynamic, uncertain currents. In *IEEE Intl Conf on Robotics and Automation (ICRA), 2010*, 4778–4783. IEEE.

Troesch, M.; Chien, S.; Chao, Y.; and Farrara, J. 2016. Planning and control of marine floats in the presence of dynamic, uncertain currents. In *Proceedings of ICAPS 2016*. AAAI.

Wang, X.; Chao, Y.; Thompson, D. R.; Chien, S. A.; Farrara, J.; Li, P.; Vu, Q.; Zhang, H.; Levin, J. C.; and Gangopadhyay, A. 2013. Multi-model ensemble forecasting and glider path planning in the mid-atlantic bight. *Continental Shelf Research* 63:S223–S234.

Woods Hole Oceanographic Institution. "Floats & Drifters". `https://www.whoi.edu/main/instruments/floats-drifters`. Accessed Feb, 2016.

YSI Systems. "IVER Autonomous Underwater Vehicle". `http://www.iver-auv.com`. Accessed Feb, 2016.

Risk-averse path planning with observation options

Aino Ropponen, Mikko Lauri and **Risto Ritala**
Department of Automation Science and Engineering
Tampere University of Technology

Abstract

This paper addresses a stochastic shortest path problem. We focus on a problem in which online observation information can be utilized during the traverse. We examine the problem on a directed acyclic graph and at each node, the edge to be observed can be chosen and after that, on the basis of the gathered information, the edge to traverse decided. The information about the edge travel times is uncertain and presented by Gaussian distributions. We have studied the problem using a risk-averse objective function which penalizes uncertainty. Our main interest lies in the optimization of observations i.e. based on the prior-expected value of the observations, which of the edges should be observed? We claim that the scheduling of the observations is an integral part of the dynamic path planning of autonomous agents. We present an algorithm for solving the shortest path problem including optimization of the observations. The result is an offline routing policy for optimal observing and traversing actions.

Introduction

Let us consider a case that an autonomous working machine, e.g. a forest harvester or shuttle carriers at harbors, traverses a rough terrain. The working machine navigates between locations and performs the tasks given to it. Path must be planned with only uncertain information about the travel times between locations. In a junction, the working machine may request external sensing agent – micro aerial vehicle – to fly through the paths. The micro aerial vehicle can observe the current conditions of the paths using e.g. imaging sensors and provide up-to-date information about the travel times. However, the observations are uncertain and slow down the progress of the working machine. Hence, the observation missions should be carried out only when the expected value of the information exceeds the delay caused.

In this paper, we study path planning of an autonomous agent in the presence of a sensing agent capable of observing missions. We represent the terrain where the autonomous working machine traverses as a directed acyclic graph (DAG). The task is to traverse from any given node i to a target node g with minimal travel costs. At each node, the action options are

1. to observe one of the edges
2. to travel along one of the edges.

We examine a case in which the edges are associated with cost representing the travel time along the edge. The travel time depends on the length of the edge as well as on the roughness of the terrain. The time is static but the decision maker (autonomous agent) has only incomplete information about it. Hence, the information about the cost is presented by a probability distribution, more precisely Gaussian distribution. When the autonomous working machine is at a node, the adjacent edges of the node can be observed using an external sensing agent. The sensing agent gathers information about the edge travel time but the information is uncertain and hence presented by a probability distribution. The edge travel time information can be updated by combining the prior information and the sensed observations by using Bayesian filtering. The observation missions are associated with cost representing the time spent in the mission. Therefore, we include the decision of the observation mission to the overall path optimization. The prior-expected value of information for each observation mission is calculated. The optimal mission is the one with the best prior expectation of the posterior optimal performance.

In this paper, the attitude towards uncertainty is risk averse: paths with lower uncertainty are prioritized. We have chosen a linear combination of the mean and the variance of the edge travel time as the objective function. The path planning is in closed-loop (Fu 2001): optimization at a node assumes that the subsequent decisions are done optimally with respect to the observation opportunities. DAG indices a partial order of the nodes. The problem is solved reversing this partial order, starting from the target node. Optimal actions for all the nodes as functions of data resulting from optimal observation missions are solved offline.

This paper is organized as follows. Section 2 reviews the studies related to optimal path planning. In Section 3, the objective function is formulated and the updating method for the edge travel times presented. In Section 4, optimization of the observing and traversing actions is presented. Section 5 presents two numerical case studies illustrating the path planning with optimization of observations. Finally, Section 6 concludes the main points of the research and discusses about the future work.

Background and related work

Algorithms such as Bellman's (Bellman 1958), Dijkstra's (Dijkstra 1959) and A* search (Hart, Nilsson, and Raphael 1968) solve deterministic shortest path problems in which the travel times are known exactly. In stochastic path planning problems, the edge travel times are not deterministic, but presented by a probability distribution. The simplest formulation of the stochastic path planning problem is minimization of the expected travel time, i.e. least expected time (LET) problem. If the edge travel times are time-invariant and uncorrelated, the LET problem reduces to the deterministic path planning in which the travel times are replaced with their expected values. LET problem in a stochastic time-dependent network was first time examined by Hall (1986). In his study, the travel time probabilities depended on the arrival time to the node. The travel time was described as a discrete function of the arrival time and the problem was solved as a minimization of the expected travel time. Other studies exploiting LET formulation include e.g. (Fu and Rilett 1998; Miller-Hooks and Mahmassani 2000; Miller-Hooks 2001; Fu 2001; Gao and Chabini 2006; Gao and Huang 2012).

As the LET formulation does not depend on the uncertainties of the edge travel time, the total travel time can differ highly from the expected value. Therefore, the LET problem represents risk neutral planning and may not be desirable for risk-averse path planning. Formulations suitable for risk-averse path planning include e.g.

(a) maximizing the probability of arriving on time,

(b) minimizing the travel time budget while ensuring predefined on-time arrival probability,

(c) maximizing the expected utility, or

(d) minimizing the sum of the travel time and the weighted variance/standard deviation.

Frank (1969) was the first to formulate the shortest path problem by maximizing the probability of achieving the target in a pre-defined limit (formulation a). Since that, on-time arrival problems have been studied e.g. by (Fan, Kalaba, and Moore 2005; Fan and Nie 2006; Nie and Fan 2006; Nikolova 2010; Lim et al. 2012; Samaranayake, Blandin, and Bayen 2012). A related formulation to the on-time arrival, is minimization of the travel time budget while ensuring a pre-defined on-time arrival probability (formulation b), see e.g. (Chen and Ji 2005; Nie and Wu 2009; Nikolova 2010; Nie et al. 2012; Pan, Sun, and Ge 2013). While the formulation (a) deals with a question "which path most likely takes the agent to the destination in a given time", the formulation (b) respectively deals with a question "how much in advance the agent should start the journey to reach the destination with a given probability". Loui (1983) formulated the stochastic path planning problem as a maximization of the decision maker's expected utility (formulation c). He introduced several utility functions and stated that for general non-linear utility function the Bellman's principle of optimality (Bellman 1965) does not hold. However, for special monotone utility functions, the principle of optimality holds and the problem can be solved using the algorithms for deterministic problems. Since that, expected travel time minimized through some utility function has been studied e.g. by (Murthy and Sarkar 1996; 1998). In studies by (Nikolova 2010; Lim et al. 2012; Chen et al. 2012; Zockaie, Nie, and Mahmassani 2014), the path planning is formulated as a linear combination of the mean and the standard deviation of the edge travel time (formulation d). This is called the mean-risk model (Lim et al. 2012; Nikolova 2010).

Adaptive path planning stands for shortest path problems in which online information attained during the operation is exploited. Interesting studies in that field are e.g. (Dean 2013; Fan, Kalaba, and Moore 2005; Fu 2001). Adaptive path planning problems can in principle be solved (1) in open-loop by re-planning with a deterministic path planning algorithm whenever new information is obtained or (2) in closed-loop by taking into account future availability of information (Fu 2001). Dean (2013) examined adaptive path planning including online information about the travel times. The planning was done in open-loop in the sense that the optimization was repeated when additional information arrived. The observations were uncertain, but optimization of observations was not addressed. Fu (2001) studied adaptive path planning for systems including online information about the travel times. The updated travel times were available during the traverse and hence, were exploitable before the decision in the current node. The a priori travel times were presented as Gaussian distributions, but the updated information indicating the travel times of the outgoing edges from the current node was exact. Fan, Kalaba, and Moore (2005) studied adaptive path planning for stochastic on-time arrival (SOTA) problem. They introduced an algorithm for solving a routing policy that is adaptive to the realized arrival times to the nodes.

The novelty of our study is that we include the optimization of the observations to the overall path optimization of a stochastic network. The information about the network is updated online on the basis of the observed values. Both the a priori information and the observed information about the travel times are uncertain and presented by Gaussian probability densities. There is an additional cost associated with observing, hence computation of the value of observing is reasonable. We present an algorithm for solving an offline routing policy that optimizes both the observing and traversing actions. The optimal traversing action is solved as a function of the data obtained from the observations.

Problem formulation

Let us consider a directed acyclic graph (DAG) with $|N|$ nodes and $|E|$ edges. The edges are associated with a cost that is related to the travel time of the edge. The prior information about the travel time of an edge $e \in E$ is incomplete and represented by a Gaussian distribution with edge-specific means μ_e and variances σ_e^2. As negative travel times are not sensible, the parameters must be chosen such that the probability of negative travel time is insignificantly small. In this paper, the information about the edge travel times is assumed statistically independent.

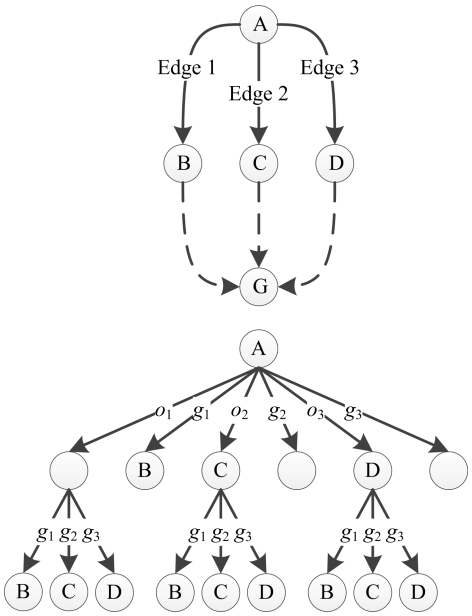


Figure 1: Decision tree of three edge options. The upper figure shows an example of a graph of three edges leaving the node A. Furthermore, there exists paths from nodes B, C, D to the target node G. In the lower figure the corresponding decision tree is presented for a case in which one of the edges leaving the node A can be observed. The branches g_1, g_2, g_3, o_1, o_2, and o_3 represent the action alternatives: traverse (i.e. go) or observe edges 1, 2, or 3, respectively.

The task is to traverse from any given node i to a goal node g with minimal travel costs. For supporting the decision making at a node, the cost of one of the outgoing edges can be observed. However, the observation information is uncertain and induces a cost as observing slows down the operation. Let us denote the observed value of the edge travel time as y_e and the observation uncertainty (variance) as σ^2_{obs}. In this paper, we are concerned with a case in which at each node the action options are either to travel along one of the edges, or first observe one of the edges and then to travel along one of the edges. If there is only one edge leaving the node, observing is useless as the only edge is necessarily traversed. Hence, observing is an option only if there are two or more edges leaving the node. Figure 1 illustrates the decision options for a case with three edge options leaving from a node.

Policy

Let us denote the current information state of the edge travel time as $f(T)$ and the set of possible actions in node i as A_i. Any policy can be written as

$$\pi = \{\pi(i, f(T), g)\}_{i \in N} \tag{1}$$

Here

$$\pi\left(i, f(T), g\right) = \left(a_1\left(f(T)\right), a_2\left(f(T), y_e\right)\right)_{a_1, a_2 \in A_i}. \tag{2}$$

Hence, the policy defines actions $a_1\left(f(T)\right) \in A_i$ and $a_2\left(f(T), y_e\right) \in A_i$ i.e. which edge is observed or traversed first and what is done after the first action. Note that if $a_1\left(f(T)\right)$ is an observation action, then $a_2\left(f(T), y_e\right)$ is a traverse action, whereas if $a_1\left(f(T)\right)$ is a traverse action, then $a_2\left(f(T), y_e\right)$ is null.

By applying the policy π and obtaining data y on the way, the travel time from the node i to the goal node g will have a probability distribution:

$$f\left(T_{ig} | \pi, y, f(T)\right) \tag{3}$$

However, at the planning stage the data is not known. Hence, a policy is evaluated based on the prior expectation of this distribution:

$$f\left(T_{ig} | \pi, f(T)\right) = E_Y\left[f\left(T_{ig} | \pi, y, f(T)\right)\right]. \tag{4}$$

Let μ_{ig} and σ^2_{ig} be the expectation and the variance of this distribution.

Objective function

In this paper, the objective function of the path planning problem is represented as a linear combination of the mean (μ_{ig}) and the variance (σ^2_{ig}) of the path travel time

$$\min_{\pi} \mu_{ig} + \alpha \sigma^2_{ig}, \tag{5}$$

where $\alpha \geq 0$ is a weighting parameter indicating the degree of the risk aversion. For $\alpha = 0$ the problem is risk-neutral and reduces to the LET problem with observation capabilities, whereas for large values of α the problem becomes risk-averse. This objective function is known to be the only form that both penalizes uncertainty and is separable into edges. The separability is a prerequisite for Bellman's principle of optimality (Bellman 1965) that allows solving the optimal path with dynamic programming methods.

Update of the edge variable

After the observation, the edge information is updated by combining the a priori information $N(\mu_e, \sigma^2_e)$ with the observation data $N(y_e, \sigma^2_{obs})$ using Bayesian estimation. As the prior information and the observation uncertainty model are Gaussians, the expected travel time and its uncertainty (variance) can be updated according to the Bayesian theory:

$$\begin{aligned} \widetilde{\mu}_e(y_e) =& \mu_e + \frac{\sigma^2_e(y_e - \mu_e)}{\sigma^2_e + \sigma^2_{obs}} \\ \widetilde{\sigma}^2_e =& \frac{\sigma^2_e \sigma^2_{obs}}{\sigma^2_e + \sigma^2_{obs}}. \end{aligned} \tag{6}$$

For computational reasons, it is practical to introduce a variable $Y(y_e)$ such that

$$Y(y_e) = \frac{\sigma^2_e(y_e - \mu_e)}{\sigma^2_e + \sigma^2_{obs}}. \tag{7}$$

Hence, $\widetilde{\mu}_e(y_e)$ in Eq.(6) takes a form

$$\widetilde{\mu}_e(y_e) = \mu_e + Y(y_e). \tag{8}$$

The prior information about $Y(y_e)$ is distributed as

$$f^{(ap)}(Y(y_e)) = N(Y(y_e); 0, \sigma^2_{Y_e}) \tag{9}$$

where

$$\sigma^2_{Y_e} = \frac{\sigma^4_e}{\sigma^2_e + \sigma^2_{obs}}. \tag{10}$$

By using this notation ($Y(y_e)$ and $\sigma^2_{Y_e}$), the optimal policy can be defined also if the edge is observed only partly.

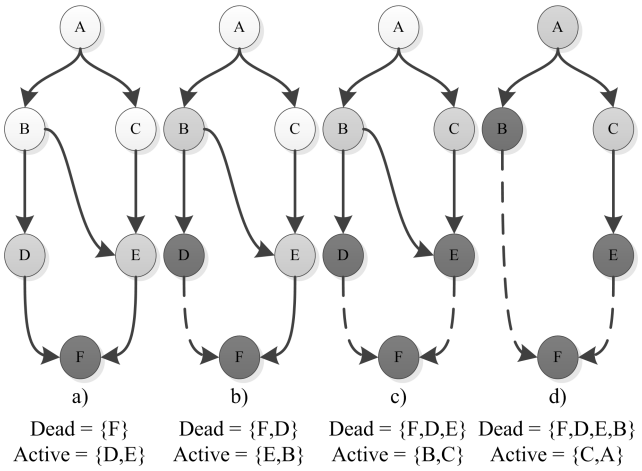


Figure 2: Example of the graph reduction. Light gray nodes are active, dark nodes dead.

Path planning with optimization of observations

This section presents a strategy for planning the path with optimization of observations. As the optimal decisions during the traverse depend on the data obtained, the optimal path cannot be solved on the basis of the prior information only. However, the policy of the optimal actions at each node can be solved offline. The policy is solved by examining the prior expectation of the posterior optimal costs, hence the expectation of the cost before any action is done. The policy defines

1. the optimal first action $a_1\left(f(T)\right)$ i.e. which edge is observed or traversed first,
2. the optimal second action $a_2\left(f(T), y_e\right)$ i.e. what is done after the first action.

The second action is needed only if the first action is observation.

The following subsections present first the order in which the decision policy for nodes is solved and then the method for solving the decision policies for the observing and traversing actions.

Solving the order of nodes

The decision policy is solved for all nodes from which the target node is reachable. The policy is solved reversing the partial order induced by the DAG. Figure 2 shows a simple example of solving the DAG topology. The DAG in this example consists of 6 nodes and 7 edges, node A being the starting point and node F the target. On the first round (Figure 2a), the target node F is selected as dead and nodes D and E are in the queue of active nodes. The order of nodes D and E is arbitrary. During the second round (Figure 2b), node D is first taken as the current node, solved and then put into the queue of dead nodes while node B, having it descendants either dead or active, is appended to the queue of active nodes behind node E. The dashed arrow in the figure denotes that, for example, when solving node B the details of the DAG from node $D{\rightarrow}F$ and node $E{\rightarrow}F$ are irrelevant once the optimal value of objective function at nodes D and e is known. Similarly, when solving node A the details of $B{\rightarrow}F$ and $C{\rightarrow}F$ are irrelevant. The final order of the queue of dead is one of the partial orders induced by the DAG and reversed.

Solving decision policy of a node: general case

After the order of nodes is solved, the decision policy for a node can be solved. Let us denote the set of all paths leaving from the node i and heading to the node g as P_{ig}, and a single path as $p \in P_{ig}$. Without observation capabilities, the value of the objective function is straightforwardly the sum of the edge cost parameters:

$$J_{ig} = \min_{\pi} \sum_{e \in p} \mu_e + \alpha \sum_{e \in p} \sigma_e^2. \tag{11}$$

If observing is possible, the value of the objective function for observation actions is to be solved before the observation data is known. Therefore, the optimal decision after observation must be determined as a function of the observed value. The prior expected value and variance of the cost-to-target when decisions are posterior optimal are:

$$\begin{aligned}
\mu_{ig} =& E\left\{T_{ig}\right\} = \int_{-\infty}^{\infty} T_{ig} F(T_{ig}) \mathrm{d}T_{ig} \\
\sigma_{ig}^2 =& E\left\{T_{ig}^2\right\} - E\left\{T_{ig}\right\}^2 \\
=& \int_{-\infty}^{\infty} T_{ig}^2 f(T_{ig}) \mathrm{d}T_{ig} - \left(\int_{-\infty}^{\infty} T_{ig} F(T_{ig}) \mathrm{d}T_{ig} \right)^2 .
\end{aligned} \tag{12}$$

Let E_i be a set of edges leaving from the node i and $e_k \in E_i$ an edge leaving from the node i and heading to the node j. Let the travel time of the edge e_k be observed resulting y_{e_k} with uncertainty σ_{obs}^2. Then the distribution of accumulated cost-to-target as a function of the observed value is denoted as $F(T_{ig}|y_{e_k})$. The prior expectation of the posterior performance resulting from optimal posterior decisions is distributed as:

$$F(T_{ig}) = \int_{-\infty}^{\infty} F(T_{ig}|y_{e_k}) F^{(ap)}(y_{e_k}) \mathrm{d}y_{e_k}. \tag{13}$$

As the edge costs are independent,

$$\begin{aligned}
F(T_{ig}|y_{e_k}) =& \int_{-\infty}^{\infty} F(T_{e_k}, T_{ig} - T_{e_k}|y_{e_k}) \mathrm{d}T_{e_k} \\
=& \int_{-\infty}^{\infty} F(T_{e_k}|y_{e_k}) F(T_{jg} = T_{ig} - T_{e_k}) \mathrm{d}T_{e_k}.
\end{aligned} \tag{14}$$

Using notation equivalent to Eq.(6), the a priori probability density of the observed value y_{e_k} can be presented as

$$F^{(ap)}(y_{e_k}) = N(y_{e_k}; \mu_{e_k}, \sigma_{e_k}^2 + \sigma_{obs}^2). \tag{15}$$

We denote the posterior mean of T_ig given data y_{e_k} as $\mu_{ig}(y_{e_k})$ and its prior expectation, i.e. the mean of Eq.(13) as $\mu_{ig}(o_{e_k})$. Similarly we denote the variance of Eq.(13) as $\sigma_{ig}(o_{e_k})^2$. Hence, o_{e_k} denotes the observation and y_{e_k} the data obtained. Straightforwardly we get

$$\begin{aligned}\mu_{ig}(o_{e_k}) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} T_{ig}F(T_{ig}|y_{e_k})F^{(ap)}(y_{e_k})\mathrm{d}y_{e_k}\mathrm{d}T_{ig} \\ &= \mu_{e_k}(o_{e_k}) + \mu_{jg} \\ \sigma_{ig}(o_{e_k})^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} T_{ig}^2F(T_{ig}|y_{e_k})F^{(ap)}(y_{e_k})\mathrm{d}y_{e_k}\mathrm{d}T_{ig} \\ &\quad - \mu_{ig}(o_{e_k})^2 \\ &= \sigma_{e_k}(o_{e_k})^2 + \sigma_{jg}^2\end{aligned} \tag{16}$$

By denoting the cost associated with observing edge e as c_e, the value function for all edges leaving the node i can be solved as

$$J_{ig} = \min_{\pi}\begin{cases} \mu_e + \alpha\sigma_e^2 + J_{jg} & \forall e \in E_i \\ \mu_{ig}(o_e) + \alpha\sigma_{ig}(o_e)^2 + c_e & \forall e \in E_i \end{cases} \tag{17}$$

and the decision is done according to:

$$\pi* = \arg\min_{\pi} J_{ig}. \tag{18}$$

In this paper, we focus on solving a case in which at most one observation can be made in a node. However, the above solution can be exploited for cases where sequential observations are possible. When considering follow-up observations, the analysis is as in Eq.(16), but as a function of the data from the first observation. However, this will require use of approximated solutions as the exact solution presented in this paper becomes arduous with even three outgoing edges from a node. In the following subsection the solution is explained in more detail for the case with only one observation from a node.

Solving decision policy of a node: one observation

Let us consider a node A with three edges: e_1, e_2, and e_3 leaving from the node A and heading to the descendant nodes B, C, and D, respectively. Nodes B, C, and D are further having paths to a target node G, as depicted in Figure 1. Let us assume the descendants of the node A have been solved i.e. the optimal policies from descendants to target π_{BG}, π_{CG}, and π_{DG}, as well as the expected cost (e.g. travel time) and variance of traversing from the descendants to target are known. The values of objective functions are

$$\begin{aligned}J_{BG} &= \mu_{BG} + \alpha\sigma_{BG}^2 \\ J_{CG} &= \mu_{CG} + \alpha\sigma_{CG}^2 \\ J_{DG} &= \mu_{DG} + \alpha\sigma_{DG}^2.\end{aligned} \tag{19}$$

The edges e_1, e_2, and e_3 are distinct and their costs are statistically independent, but their descendant nodes can be the same or different. The travel times of the edges are distributed as

$$\begin{aligned}T_{e_1} &\sim N(\mu_{e_1}, \sigma_{e_1}^2) \\ T_{e_2} &\sim N(\mu_{e_2}, \sigma_{e_2}^2) \\ T_{e_3} &\sim N(\mu_{e_3}, \sigma_{e_3}^2).\end{aligned} \tag{20}$$

Hence without observation possibility, the objective function J_{AG} is given as

$$J_{AG} = \min_{\pi}\begin{cases} \mu_{e_1} + \alpha\sigma_{e_1}^2 + J_{BG} \\ \mu_{e_2} + \alpha\sigma_{e_2}^2 + J_{CG} \\ \mu_{e_3} + \alpha\sigma_{e_3}^2 + J_{DG} \end{cases} \tag{21}$$

With observation possibility, there are six action alternatives at a node A: to traverse or observe one of the edges e_1, e_2, and e_3. Let us denote the traversing and observing actions as g_e (i.e. go edge e) and o_e, respectively. Hence, the action $a_1 \in \{g_{e_1}, g_{e_2}, g_{e_3}, o_{e_1}, o_{e_2}, o_{e_3}\}$. The objective function of a node A is

$$J_{AG} = \min_{\pi}\begin{cases} \mu_{e_1} + \alpha\sigma_{e_1}^2 + J_{BG} \\ \mu_{e_2} + \alpha\sigma_{e_2}^2 + J_{CG} \\ \mu_{e_3} + \alpha\sigma_{e_3}^2 + J_{DG} \\ \mu_{ig}(o_{e_1}) + \alpha\sigma_{ig}(o_{e_1})^2 + c_{e_1} \\ \mu_{ig}(o_{e_2}) + \alpha\sigma_{ig}(o_{e_2})^2 + c_{e_2} \\ \mu_{ig}(o_{e_3}) + \alpha\sigma_{ig}(o_{e_3})^2 + c_{e_3}. \end{cases} \tag{22}$$

The computation of the rows 1–3 of Eq.(22) is obvious. Let us compute the fourth row, i.e. the value function for observing first the edge e_1.

As sequential observations were not allowed, after $a_1 = o_{e_1}$, there are three consequential action options: to traverse edge e_1, e_2, or e_3, i.e. $a_2 \in \{g_{e_1}, g_{e_2}, g_{e_3}\}$. The value function after the observation is determined as a minimum of the corresponding objective functions. The formulation is as follows

$$J_{AG}(y_{e_1}) = \min_{\pi}\begin{cases} \widetilde{\mu}_{e_1}(y_{e_1}) + \alpha\widetilde{\sigma}_{e_1}^2 + J_{BG} + c_{e_1} \\ \mu_{e_2} + \alpha\sigma_{e_2}^2 + J_{CG} + c_{e_1} \\ \mu_{e_3} + \alpha\sigma_{e_3}^2 + J_{DG} + c_{e_1}, \end{cases} \tag{23}$$

where $\widetilde{\mu}_{e_1}(y_{e_1})$ and $\widetilde{\sigma}_{e_1}$ are the updated mean and variance after the observation, determined according to Eq.(6). As the observation y_{e_1} does not affect the information about the edges e_2, or e_3, the two lowest rows of Eq.(23) can be solved at once. Let us denote the edge providing minimum of those as $e*$ and the corresponding value function of J_{CG} and J_{DG} as J_{*G}.

As $\widetilde{\mu}_{e_1}(y_{e_1}) = \mu_{e_1} + Y(y_{e_1})$, a critical value Y_{cr} for the observation data, is given as

$$\begin{aligned}Y_{cr} =& \mu_{e*} - \mu_{e_1} \\ &+ \alpha\left(\sigma_{e*}^2 - \widetilde{\sigma}_{e_1}^2\right) + (J_{*G} - J_{BG}).\end{aligned} \tag{24}$$

The critical value Y_{cr} separates which of the two post-observation actions is preferred: if the observed value $Y(y_{e_1})$ is lower than the critical value Y_{cr}, the optimal action after observation is to traverse the observed edge e_1, otherwise the optimal action is to traverse the non-observed

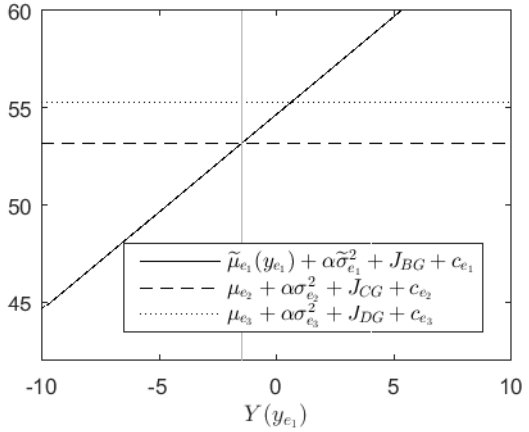


Figure 3: Example of the value functions of Eq.(23). The gray line is the critical value Y_{cr}. In this example $e* = e_2$. Hence, Y_{cr} is the value of the observation $Y(y_{e_1})$ at which the two options $a_2 = g_{e_1}$ and $a_2 = g_{e_2}$ are equally attractive.

edge $e*$ i.e. e_2 or e_3 depending on which one has the lower value of J_{AG}. Hence

$$a_2\left(y_{e_1}\right) = \begin{cases} g_{e_1} & \text{if } Y(y_{e_1}) \leq Y_{cr} \\ g_{e*} & \text{otherwise} \end{cases} \tag{25}$$

Figure 3 gives an example of the critical value and the value functions of Eq.(23).

The optimal edge cost from node A to node G is distributed according to:

$$F(T_{AG}|y_{e_1}) = \begin{cases} \int_{-\infty}^{\infty} N(T_{AG} - t; \widetilde{\mu}_{e_1}(y_{e_1}), \widetilde{\sigma}^2_{e_1}) f_{BG}(t)\mathrm{d}t \\ \int_{-\infty}^{\infty} N(T_{AG} - t; \mu_{e*}, \widetilde{\sigma}^2_{e*}) f_{*G}(t)\mathrm{d}t, \end{cases} \tag{26}$$

where the upper denotes the condition $Y(y_{e_1}) \leq Y_{cr}$ and the lower the condition $Y(y_{e_1}) > Y_{cr}$. The distributions $f_{BG}(t)$ and $f_{*G}(t)$ are expected priors of posterior optimal costs. For further computation, it is sufficient to known only their mean and variance. If the edge e_1 is chosen to observe, the prior expectation of the mean and variance of the posterior cost-to-target solved according to Eq.(16) are as follows

$$\begin{aligned} \mu_{ig}(o_{e_1}) = {} & (\mu_{e*} + \mu_{*G}) \left(1 - F_{01}\left(\frac{Y_{cr}}{\sigma_{Y_{e_1}}}\right)\right) \\ & + (\mu_{e_1} + \mu_{BG}) F_{01}\left(\frac{Y_{cr}}{\sigma_{Y_{e_1}}}\right) \\ & - \sigma_{Y_{e_1}} N_{01}\left(\frac{Y_{cr}}{\sigma_{Y_{e_1}}}\right) \end{aligned} \tag{27}$$

$$\begin{aligned} & \sigma_{ig}(o_{e_1})^2 = \\ & \left(\sigma^2_{e*} + \sigma^2_{*G} + (\mu_{e*} + \mu_{*G})^2\right) \left(1 - F_{01}\left(\frac{Y_{cr}}{\sigma_{Y_{e_1}}}\right)\right) \\ & + \left((\mu_{e_1} + \mu_{BG})^2 + \widetilde{\sigma}^2_{e_1} + \sigma^2_{BG} + \sigma^2_{Y_{e_1}}\right) F_{01}\left(\frac{Y_{cr}}{\sigma_{Y_{e_1}}}\right) \\ & - \sigma_{Y_{e_1}} \left(2(\mu_{e_1} + \mu_{BG}) + Y_{cr}\right) N_{01}\left(\frac{Y_{cr}}{\sigma_{Y_{e_1}}}\right) - \mu_{ig}(o_{e_1})^2. \end{aligned} \tag{28}$$

F_{01} and N_{01} refer to standard normal cumulative distribution and probability density function, respectively. Here it is noted that the variance of prior-expected travel time is not the prior-expected variance.

A similar result is obtained for actions $a_1 = o_{e_2}$ and $a_1 = o_{e_3}$. The value function J_{AG} is solved according to Eq. (22) and the optimal policy according to:

$$\pi* = \arg\min_{\pi} J_{AG}. \tag{29}$$

The optimal policy for a node defines the first action a_1, i.e. which edge is traversed or observed first. If the first action is observation, the optimal policy defines the second action according to Eq. (25). The second action is always traversing. For describing the function $a_2(y_e)$, three values are needed: observed edge e_i, critical value Y_{cr} and alternative edge $e*$.

The mean and variance to be used when computing nodes that node A is a descendant to are

$$\begin{aligned} \mu_{AG} &= \begin{cases} \mu_{e_1} + \mu_{BG} \\ \mu_{e_2} + \mu_{CG} \\ \mu_{e_3} + \mu_{DG} \\ \mu_{AG}(o_{e_1}) + c_{e_1} \\ \mu_{AG}(o_{e_2}) + c_{e_2} \\ \mu_{AG}(o_{e_3}) + c_{e_3} \end{cases} \\ \sigma^2_{AG} &= \begin{cases} \sigma^2_{e_1} + \sigma^2_{BG} \\ \sigma^2_{e_2} + \sigma^2_{CG} \\ \sigma^2_{e_3} + \sigma^2_{DG} \\ \sigma_{AG}(o_{e_1})^2 \\ \sigma_{AG}(o_{e_2})^2 \\ \sigma_{AG}(o_{e_3})^2 \end{cases} \end{aligned} \tag{30}$$

depending on which of the options in Eq.(22) provides the minimum. As the analysis above shows, Eq.(30) is sufficient information about the prior expectation of the posterior optimal distribution for solving the predecessor nodes of the node A. It is independent on the actual functional form of the distribution. As a result, the solution is given analytically as a function of the edge parameters from node A to nodes B, C, and D, and previously solved optimal objective function, i.e. mean and variance of the cost-to-target of the descendant nodes B, C, and D.

Case studies

The following subsections present two simulated case studies illustrating the path planning with optimization of observations. Case 1 is a simple DAG consisting of four nodes and six edges. The purpose is to illustrate how the optimization of observations affects the solution. Case 2 is a larger

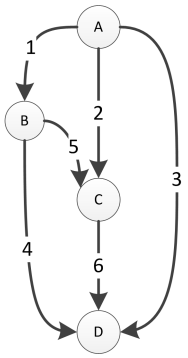


Figure 4: Case study 1: DAG with four nodes and six edges.

Table 1: Edge parameters used in the case study. Start and end refer the nodes where the edge starts and ends, respectively. μ_e is the expected travel time, σ_e^2 is the variance, c_e the observation cost and σ_{obs}^2 the observation uncertainty of the edge.

edge	start	end	μ_e	σ_e^2	c_e	σ_{obs}^2
1	A	B	20	2	0.21	0.2
2	A	C	30	1	0.31	0.3
3	A	D	40	3	0.41	0.4
4	B	D	20	2	0.21	0.2
5	B	C	10	2	0.11	0.1
6	C	D	10	2	0.11	0.1

DAG showing how the method presented in this paper can be applied to more complicated problems.

Case 1: simple case

Let us examine a case study of a DAG presented in Figure 4. The DAG consists of four nodes denoted as A,...,D, and six edges denoted as 1,...,6. Table 1 presents the parameter values associated with each edge. The objective is to travel from the node A to the target node D. Let us first examine the problem without the observing possibility. If the objective is to minimize the expected travel time, i.e. the path planning is a LET problem, all the routes are equally good and the expected travel time is 40. If the objective is to minimize the linear combination of the expected travel time and its weighted variance (see Eq.(5)), paths $A{\rightarrow}C{\rightarrow}D$ and $A{\rightarrow}D$ are equally good with variance 3. The paths $A{\rightarrow}B{\rightarrow}D$ and $A{\rightarrow}B{\rightarrow}C{\rightarrow}D$ are worse due to the higher variance 4 and 6, respectively. The objective function is:

$$J_{ad} = \min \begin{cases} 40 + 4\alpha \\ 40 + 5\alpha \\ 40 + 3\alpha \\ 40 + 3\alpha \end{cases} \tag{31}$$

For $\alpha = 0.1$, the value of the objective function is 40.3 units.

If observing is possible, the expected value of the objective function is 39.787 units. That is 0.513 units lower than the value without observing. Table 2 summarizes the optimal actions at each node. According to Table 2, the optimal action at node A is first to observe edge 1 ($A{\rightarrow}B$). Then if the observed value $Y(y_1)$ (see Eq.(7)) is less or equal to the critical value $Y_{cr} = 0.46$, the optimal action is to traverse edge 1, otherwise the optimal action is to traverse edge 2 ($A{\rightarrow}C$). Respectively, the optimal action at node B is to first observe edge 5 ($B{\rightarrow}C$), and if the observed value $Y(y_5) \leq -0.0095$, to traverse edge 5, otherwise the optimal action is to traverse edge 4 ($B{\rightarrow}D$). It is notable that the edge 3 ($A{\rightarrow}D$) is not optimal in any case.

Table 2: Optimal actions and critical value. The first column indicates the node. The second and third column indicate the optimal first action. If the optimal first action is to observe an edge, columns 4 and 5 present the critical limit of the observed value and the alternative edge. If the observed value $Y(y_e)$ is lower than Y_{cr}, the optimal second action is to travel the observed edge. Otherwise the optimal second action is to travel the alternative edge.

node	action 1	edge 1	Y_{cr}	edge 2
A	observe	1	0.463	2
B	observe	5	-0.0095	4
C	traverse	6		

Table 3: Paths used in a set of 100000 simulations. Left are the paths with their occurrence probabilities and the mean value of the objective function.

path	%	J
$A{\rightarrow}C{\rightarrow}D$	37	40.50
$A{\rightarrow}B{\rightarrow}D$	32	39.79
$A{\rightarrow}B{\rightarrow}C{\rightarrow}D$	31	38.77

For testing the optimization, a numerical simulator was implemented. For each simulation, the real values of the edge cost T_e^{real} are sampled randomly according to the a priori values $N(\mu_e, \sigma_e^2)$. Furthermore, the observed values are sampled randomly such that $y_e \sim N\left(T_e^{real}, \sigma_{obs}^2\right)$. As the real values as well as the observed values vary between simulations, the optimal paths are not same. In a set of 100000 simulations with same initial values, the most common path was $A{\rightarrow}C{\rightarrow}D$, which was used in 37% of the simulations. The paths $A{\rightarrow}B{\rightarrow}D$ and $A{\rightarrow}B{\rightarrow}C{\rightarrow}D$ were used in 32% and 31% of the simulations, respectively. The mean value of the objective function for the whole simulation was $J = 39.786$ units. The paths and their occurrence probabilities together with the mean value of the objective function are given in Table 3. The value of the objective function is highest for path $A{\rightarrow}C{\rightarrow}D$ and lowest for path $A{\rightarrow}B{\rightarrow}C{\rightarrow}D$. That is due to the fact that path $A{\rightarrow}C{\rightarrow}D$ is never observed whereas edges $A{\rightarrow}B$ and $B{\rightarrow}C$ are always observed if path $A{\rightarrow}B{\rightarrow}C{\rightarrow}D$ is chosen. Hence, path $A{\rightarrow}C{\rightarrow}D$ is chosen because the observed value of path $A{\rightarrow}B$ is high, but as path $A{\rightarrow}C$ is not observed, the real value of it can be high.

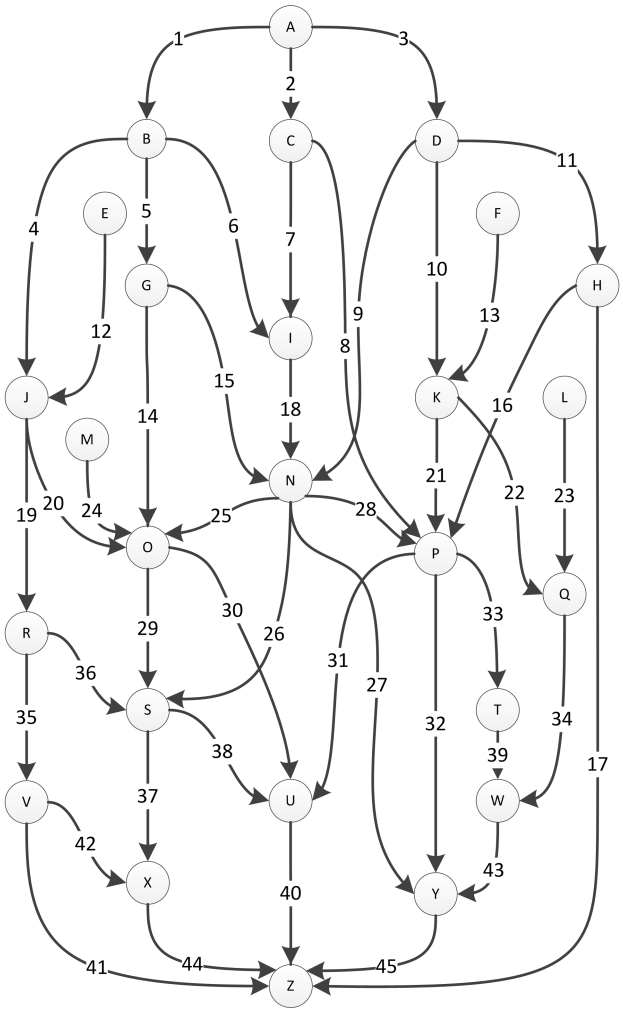


Figure 5: Case study 2: DAG with 26 nodes and 45 edges.

Case 2: larger case

Let us examine a case study of a DAG presented in Figure 5. The DAG consists of 26 nodes denoted as A,...,Z, and 45 edges denoted as $1, ..., 45$. The optimization of the actions needs to be solved only for nodes with at least two edges leaving from the node. In this case there are 15 such nodes. For the remaining 11 nodes with none or only one edge leaving, there is no optimization about the actions to be made.

Table 4 presents the expected travel time (μ_e), variance (σ_e^2), observation cost (c_e), and observation uncertainty (σ_{obs}^2) of each edge. The variance is defined as $\sigma_e^2 = 0.2\mu_e$. The edge 17 is an exception as its variance is $\sigma_{17}^2 = 0.4\mu_{17}$. During the traverse, an external sensing agent may observe the edge travel time, but the observations are uncertain. The observation uncertainty is related to the expected travel time of the edge as $\sigma_{obs}^2 = 0.01\mu_e$. The maximum observation time is 40 i.e. the observation time is $\min(40, \mu_e)$. Hence, the edges with expected travel time longer than 40 are observed only partly. The observation cost consists of a constant part (0.01) and a part depending the observation time ($0.01 \min(40, \mu_e)$), i.e. $c_e = 0.01 + 0.01 \min(40, \mu_e)$.

Table 4: Edge parameters used in the case study. Start and end refer the nodes where the edge starts and ends, respectively. μ_e is the expected travel time, σ_e^2 is the variance, c_e the observation cost and σ_{obs}^2 the observation uncertainty of the edge.

edge	start	end	μ_e	σ_e^2	c_e	σ_{obs}^2
1	A	B	11	2.2	0.12	0.11
2	A	C	10	2	0.11	0.1
3	A	D	15	3	0.16	0.15
4	B	J	19	3.8	0.2	0.19
5	B	G	7	1.4	0.08	0.07
6	B	I	17	3.4	0.18	0.17
7	C	I	19	3.8	0.2	0.19
8	C	P	40	8	0.41	0.4
9	D	N	26	5.2	0.27	0.26
10	D	K	17	3.4	0.18	0.17
11	D	H	24	4.8	0.25	0.24
12	E	J	16	3.2	0.17	0.16
13	F	K	12	2.4	0.13	0.12
14	G	O	36	7.2	0.37	0.36
15	G	N	24	4.8	0.25	0.24
16	H	P	11	2.2	0.12	0.11
17	H	Z	60	24	0.61	0.4
18	I	N	12	2.4	0.13	0.12
19	J	R	32	6.4	0.33	0.32
20	J	O	25	5	0.26	0.25
21	K	P	18	3.6	0.19	0.18
22	K	Q	34	6.8	0.35	0.34
23	L	Q	11	2.2	0.12	0.11
24	M	O	14	2.8	0.15	0.14
25	N	O	14	2.8	0.15	0.14
26	N	S	38	7.6	0.39	0.38
27	N	Y	45	9	0.46	0.4
28	N	P	11	2.2	0.12	0.11
29	O	S	26	5.2	0.27	0.26
30	O	U	38	7.6	0.39	0.38
31	P	U	42	8.4	0.43	0.4
32	P	Y	35	7	0.36	0.35
33	P	T	17	3.4	0.18	0.17
34	Q	W	9	1.8	0.1	0.09
35	R	V	17	3.4	0.18	0.17
36	R	S	17	3.4	0.18	0.17
37	S	X	12	2.4	0.13	0.12
38	S	U	13	2.6	0.14	0.13
39	T	W	9	1.8	0.1	0.09
40	U	Z	8	1.6	0.09	0.08
41	V	Z	23	4.6	0.24	0.23
42	V	X	13	2.6	0.14	0.13
43	W	Y	11	2.2	0.12	0.11
44	X	Z	10	2	0.11	0.1
45	Y	Z	16	3.2	0.17	0.16

Table 5: Optimal actions if observing is not possible. The first column indicates the node. The second the cost from the node to the target node Z. Columns 3 and 4 indicate the optimal first action, i.e. the optimal edge to traverse.

node	J	action	edge
A	102.96	traverse	1
B	91.52	traverse	6
C	93.60	traverse	8
D	88.40	traverse	9
E	89.44	traverse	12
F	83.20	traverse	13
G	85.28	traverse	14
H	63.44	traverse	16
I	73.84	traverse	18
J	72.80	traverse	19
K	70.72	traverse	21
L	48.88	traverse	23
M	62.40	traverse	24
N	61.36	traverse	26
O	47.84	traverse	30
P	52.00	traverse	31
Q	37.44	traverse	34
R	39.52	traverse	36
S	21.84	traverse	38
T	37.44	traverse	39
U	8.32	traverse	40
V	23.92	traverse	41
W	28.08	traverse	43
X	10.40	traverse	44
Y	16.64	traverse	45
Z	0.00		

Optimization Let us choose the node Z as the target. Once the topology is solved, the optimal actions for each node can be solved according to the strategy presented in the previous Section. Tables 5 and 6 present the expected cost-to-target node, the optimal actions and the critical values for the case with observation option and without it.

Let us choose node A as the starting point. Hence, the target is to travel from node A to node Z with minimal costs. Let us first examine the problem without observing possibility. If the objective is to minimize only the expected travel time, paths $A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$ and $A{\rightarrow}D{\rightarrow}H{\rightarrow}Z$ are equally good, with 99 as the expected travel time. If the objective is to minimize the linear combination of the expected travel time and its variance as in Eq. (5), the path $A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$ is optimal. Note, that path $A{\rightarrow}D{\rightarrow}H{\rightarrow}Z$ is no longer optimal due to the higher variance. For $\alpha = 0.2$, the value of the objective function for the optimal path is with 102.96 units, whereas for $A{\rightarrow}D{\rightarrow}H{\rightarrow}Z$ it is 105.36 units.

If observing is an option, the expected value of the objective function is 101.40 units. Hence, the expected travel cost is 1.5% lower using the observation when compared to the case without observation. The difference increases if the uncertainty associated with the edge information increases.

Table 6: Optimal actions and critical value if observing is possible. The first column indicates the node and the second the cost from the node to the target node Z. Columns 3 and 4 indicate the optimal first action. If the optimal first action is to observe an edge, columns 5 and 6 present the critical limit of the observed value and the alternative edge. If the observed value $Y(y_e)$ is lower than Y_{cr}, the optimal second action is to travel the observed edge. Otherwise the optimal second action is to travel the alternative edge.

node	J	action 1	edge 1	Y_{cr}	edge 2
A	101.40	observe	3	0.20	1
B	90.39	observe	6	1.37	5
C	91.94	observe	8	1.24	7
D	86.60	observe	9	0.77	11
E	88.61	traverse	12	0.00	0
F	82.67	traverse	13	0.00	0
G	84.16	observe	14	2.17	15
H	62.42	observe	17	1.25	16
I	73.04	traverse	18	0.00	0
J	71.97	observe	19	2.01	20
K	70.19	observe	22	-1.30	21
L	48.88	traverse	23	0.00	0
M	61.84	traverse	24	0.00	0
N	60.56	observe	26	2.11	25
O	47.28	observe	30	2.31	29
P	51.49	observe	31	2.56	32
Q	37.44	traverse	34	0.00	0
R	39.21	observe	35	-0.83	36
S	21.66	observe	38	1.54	37
T	37.44	traverse	39	0.00	0
U	8.32	traverse	40	0.00	0
V	23.14	observe	41	0.88	42
W	28.08	traverse	43	0.00	0
X	10.40	traverse	44	0.00	0
Y	16.64	traverse	45	0.00	0
Z	0.00				

Table 7: Simulation example 1.

node	action 1	edge	$Y(y_e)$	action 2	edge
A	observe	3	1.45	traverse	1
B	observe	6	-2.09	traverse	6
I	traverse	18			
N	observe	26	0.1	traverse	26
S	observe	38	0.46	traverse	38
U	traverse	40			

Table 8: Simulation example 2.

node	action 1	edge	$Y(y_e)$	action 2	edge
A	observe	3	-1.34	traverse	3
D	observe	9	2.08	traverse	11
H	observe	17	8.57	traverse	16
P	observe	31	-4.48	traverse	31
U	observe	40			

Also the observation uncertainty and the observation cost affect significantly the benefit of the observation.

Simulation For testing the optimization, a numerical simulator was implemented. For each simulation, the real values of the edge travel times T_e^{real} are sampled randomly according to the a priori values $N(\mu_e, \sigma_e^2)$. Furthermore, the observed values are sampled randomly such that $y_e \sim N\left(T_e^{real}, \sigma_{obs}^2\right)$. As the real values as well as the observed values vary between simulations, the optimal paths are not same. Tables 7 and 8 show results of two independent simulations. In the first simulation, the optimal path is through nodes $A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$, in the second simulation through nodes $A{\rightarrow}D{\rightarrow}H{\rightarrow}P{\rightarrow}U$. Both are based on the same optimal policy.

According to Table 6, the optimal action in the node A is to observe edge 3 ($A{\rightarrow}D$), and if the observed value is lower than the critical value $Y_{cr} = 0.20$, the optimal next action is to travel edge 3, otherwise the optimal next action is to travel edge 1 ($A{\rightarrow}B$). In the simulation 1 (Table 7), the observed value is $Y(y_3) = 1.45$ which is higher than the critical value $Y_{cr} = 0.20$. Hence, it is not profitable to travel edge 3 and the optimal next action is to traverse edge 1 ($A{\rightarrow}B$). In the node B, the optimal action is to observe edge 6 ($B{\rightarrow}I$) and as the observed value is lower (-2.09) than the critical value (1.37), edge 6 is traveled. In node I, there is only one edge leaving the node, hence edge 18 ($I{\rightarrow}N$) is traveled without observation. In the following steps edges 26 ($N{\rightarrow}S$), and 38 ($S{\rightarrow}U$) are observed and edges 26, 38, and 40 ($U{\rightarrow}Z$) traveled.

In the simulation 2 (Table 8), edge 3 is again observed first. The observed value $Y(y_3) = -1.35$ is lower than the critical value, and the observed edge is traveled. In node D, edge 9 ($D{\rightarrow}N$) is observed. As the observed value $Y(y_9) = 2.08$ is higher than the critical value $Y_{cr} = 0.77$, the optimal action is to travel edge 11 ($D{\rightarrow}H$). In node H, the optimal action is to observe edge 17 ($H{\rightarrow}Z$). This is long edge with 60 segments. As the maximum observing distance was set to 40 segments, the edge can be observed only partly. In this simulation, the observed value is exceptionally large, $Y(y_{17}) = 8.57$ to compared with the critical value $Y_{cr} = 1.25$, hence the optimal action is to travel the non-observed edge 16 ($H{\rightarrow}P$). In the following steps, edge 31 ($P{\rightarrow}U$) is observed and traveled and finally in the node U, edge 40 ($U{\rightarrow}Z$) is traveled.

Table 9: Paths used in a set of 100000 simulations. In left are the paths with their occurrence probabilities and the mean value of the objective function.

path	%	J
$A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	23.0	100.51
$A{\rightarrow}D{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	22.6	99.21
$A{\rightarrow}D{\rightarrow}H{\rightarrow}Z$	12.5	100.44
$A{\rightarrow}B{\rightarrow}G{\rightarrow}O{\rightarrow}U{\rightarrow}Z$	6.4	102.03
$A{\rightarrow}D{\rightarrow}H{\rightarrow}P{\rightarrow}U{\rightarrow}Z$	6.3	102.32
$A{\rightarrow}D{\rightarrow}N{\rightarrow}O{\rightarrow}U{\rightarrow}Z$	6.2	101.30
$A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}O{\rightarrow}U{\rightarrow}Z$	6.1	102.59
$A{\rightarrow}D{\rightarrow}N{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	4.5	100.90
$A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	4.4	102.25
$A{\rightarrow}B{\rightarrow}G{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	1.4	103.80
$A{\rightarrow}D{\rightarrow}H{\rightarrow}P{\rightarrow}Y{\rightarrow}Z$	1.3	104.79
$A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}O{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	1.3	104.44
$A{\rightarrow}B{\rightarrow}G{\rightarrow}O{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	1.3	104.10
$A{\rightarrow}D{\rightarrow}N{\rightarrow}O{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	1.2	103.23
$A{\rightarrow}B{\rightarrow}G{\rightarrow}N{\rightarrow}O{\rightarrow}U{\rightarrow}Z$	0.39	106.11
$A{\rightarrow}B{\rightarrow}G{\rightarrow}N{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	0.26	106.02
$A{\rightarrow}B{\rightarrow}G{\rightarrow}O{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	0.25	106.28
$A{\rightarrow}D{\rightarrow}N{\rightarrow}O{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	0.24	104.67
$A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}O{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	0.23	106.44
$A{\rightarrow}B{\rightarrow}G{\rightarrow}N{\rightarrow}O{\rightarrow}S{\rightarrow}U{\rightarrow}Z$	0.06	108.08
$A{\rightarrow}B{\rightarrow}G{\rightarrow}N{\rightarrow}O{\rightarrow}S{\rightarrow}X{\rightarrow}Z$	0.01	110.33

In a set of 100000 simulations with same initial values, 21 paths were used. The most common path was $A{\rightarrow}B{\rightarrow}I{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$ which was used in 23% of the simulations. This is the same path that was optimal also without observations. Almost as common was the path $A{\rightarrow}D{\rightarrow}N{\rightarrow}S{\rightarrow}U{\rightarrow}Z$ with portion of 22.6%. The path $A{\rightarrow}D{\rightarrow}H{\rightarrow}Z$ that was optimal when only expected travel time was concerned, is third most common with portion of 12.5%. The paths and their occurrence probabilities are given in Table 9. The mean value of the objective function for the whole simulation was $J = 101.38$ units. This is a bit lower than the value obtained in optimization $J = 101.40$ units (see Table 6). In the similar simulation without observation possibility, the mean value of the objective function was $J = 102.94$ units while the value obtained in optimization was $J = 102.96$ units. Hence, the case including observations provided 1.56 units (1.5 %) lower value than the case without observation.

For comparing the results for cases without observation, Table 10 gives more detailed information about the three most common paths. It shows the a priori values of the expected travel time and the expected

Table 10: Paths used in a set of 1000000 simulations. In left are the paths and the mean value of the objective function. The columns 3 and 4 present the expected travel time (a priori) and expected value of the objective function (a priori) of the path

path	J	μ_p	$\mu_p + \alpha\sigma_p^2$
$A \rightarrow B \rightarrow I \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$	100.51	99	102.96
$A \rightarrow D \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$	99.21	100	104
$A \rightarrow D \rightarrow H \rightarrow Z$	100.44	99	105.36

value of the objective function. As pointed out earlier, paths $A \rightarrow B \rightarrow I \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$ and $A \rightarrow D \rightarrow H \rightarrow Z$ are equally good if only expected travel time is considered (99 units), but path $A \rightarrow D \rightarrow H \rightarrow Z$ is no longer optimal if the variance is included to the objective function. The a priori value of the objective function of path $A \rightarrow D \rightarrow H \rightarrow Z$ is 105.36 units whereas the same value for path $A \rightarrow B \rightarrow I \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$ is 102.96 units and for path $A \rightarrow D \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$ 104 units. However, if observing is possible, path $A \rightarrow D \rightarrow H \rightarrow Z$ becomes favorable as the variance can be decreased by observations. Although only 40 of its 60 segments can be observed, still the variance decreases significantly. It is notable that due to the high variance, the realized travel time can be remarkably lower of higher than the a priori travel time.

An interesting feature is that even though the a priori values of the path $A \rightarrow D \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$ were not optimal, the realized value of the objective function is lowest. That is due to the fact that four of its edges is observed. In nodes A, D, N, and S, always the edges belonging to the path $A \rightarrow D \rightarrow N \rightarrow S \rightarrow U \rightarrow Z$ are observed. Hence, the edges are chosen only when the travel time along them is proven to be low.

Discussion and conclusion

We have examined path planning in a stochastic environment with online observation options. We considered an autonomous agent (working machine) which path we seek to optimize, and an external sensing agent which provides additional up-to-date information for the dynamic optimization. In this paper, we provided an exact solution for optimizing both the path of the working machine and the use of the external sensing agent. Our plan is to test the results of this study with an autonomously operating wheel loader that is assisted by an unmanned hexacopter. The hexacopter is equipped with stereo camera system and other sensors providing up-to-date information about the operating environment.

In our study, the stochastic environment is described as a directed acyclic graph. This is a rather strong assumption as it means that the working machine is not allowed to turn back even if the following edges are exceptionally unfavorable to travel. However, in our intended case with working machine on a rough terrain, turning back is very time consuming or even impossible and hence the assumption is justified. Furthermore, in a case turning back is unavoidable, the problem can be re-planned for allowing returning to the previous location.

As the information about the travel times is uncertain, the decision maker's attitude towards risk must be taken into account. We have studied the problem using a risk-averse objective function which penalizes uncertainty. The penalization is done by an objective function which minimizes the linear combination of the expected travel time and its variance. In a sequential paper to this article (Lauri, Ropponen, and Ritala 2016), we present a solution to a stochastic path planning problem in which the objective is to maximize the probability of arriving on time.

In this paper, we have examined observing mode which allows observing at most one of the edges in a junction. This is justified for example if the sensing agent has limited energy resources and it need to be recharged after each observation mission. The solution could be further improved by enabling more edges to be observed. Observation mode allowing options that either none of the edges is observed or all the edges leaving a node are observed is straightforward to solve. The problem becomes more complicated if both the number and order of edges to be observed is optimized. That means that the routing policy defines whether it is optimal to observe other edges after the first observation. The decision depends on the value attained from the first observation and hence the optimal routing policy and the critical limits are functions of the observed values. That is studied in (Lauri, Ropponen, and Ritala 2016) for DAG with at most two edges leaving the nodes. However, the exact solution with more than two outgoing edges becomes arduous, hence approximated solutions are required for solving larger cases. The solution presented in this paper is a starting point for seeking the approximated solutions.

Other interesting generalizations for future studies include e.g. time-dependent edge costs as well as statistically dependent edge costs. The former means that the expected travel time of an edge depends on the arrival time to the node. The latter means that the information about the executed travel time of one edge, affect the prior information of all the other edges. This might be the case for example if the recent weather conditions cause correlations to the edges near to each other. Both the time-dependent and statistically dependent edge costs increase the complexity of the problem, and approximations are needed for solving these.

Acknowledgements

The work was funded by the Academy of Finland, "Optimal operation of observation systems in autonomous mobile machines (O3-SAM)" which is gratefully acknowledged.

References

Bellman, R. 1958. On a routing problem. *Quarterly of Applied Mathematics* 16(1):87–90.

Bellman, R. 1965. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press.

Chen, A., and Ji, Z. 2005. Path finding under uncertainty. *Journal of Advanced Transportation* 39(1):19–37.

Chen, B. Y.; Lam, W. H.; Sumalee, A.; and Li, Z. 2012. Reliable shortest path finding in stochastic networks with spatial correlated link travel times. *International Journal of Geographical Information Science* 26(2):365–386.

Dean, D. J. 2013. Finding optimal travel routes with uncertain cost data. *Transaction in GIS* 17(2).

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Fan, Y., and Nie, Y. 2006. Optimal routing for maximizing the travel time reliability. *Networks and Spatial Economics* 6(3-4):333–344.

Fan, Y.; Kalaba, R.; and Moore, J.E., I. 2005. Arriving on time. *Journal of Optimization Theory and Applications* 127(3):497–513.

Frank, H. 1969. Shortest paths in probabilistic graphs. *Operations Research* 17(4):583–599.

Fu, L., and Rilett, L. 1998. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B: Methodological* 32(7):499 – 516.

Fu, L. 2001. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transportation Research Part B: Methodological* 35:749–765.

Gao, S., and Chabini, I. 2006. Optimal routing policy problems in stochastic time-dependent networks. *Transportation Research Part B: Methodological* 40(2):93 – 122.

Gao, S., and Huang, H. 2012. Real-time traveler information for optimal adaptive routing in stochastic time-dependent networks. *Transportation Research Part C: Emerging Technologies* 21:196–213.

Hall, R. W. 1986. The Fastest Path through a Network with Random Time-Dependent Travel Times. *Transportation Science* 20(3):182–188.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics* SSC-4(2):100–107.

Lauri, M.; Ropponen, A.; and Ritala, R. 2016. Meeting a deadline: shortest paths on stochastic directed acyclic graphs with allocation of observation resources. *Submitted to Annals of Mathematics and Artificial Intelligence*.

Lim, S.; Sommer, C.; Nikolova, E.; and Rus, D. 2012. Practical Route Planning Under Delay Uncertainty: Stochastic Shortest Path Queries. In *Proceedings of Robotics: Science and Systems*.

Loui, R. P. 1983. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM* 26(9):670–676.

Miller-Hooks, E., and Mahmassani, H. S. 2000. Least expected time paths in stochastic, time-varying transportation networks. *Transportation Science* 34(2):198–215.

Miller-Hooks, E. 2001. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks* 37(1):35–52.

Murthy, I., and Sarkar, S. 1996. A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science* 30(3):220–236.

Murthy, I., and Sarkar, S. 1998. Stochastic Shortest Path Problems with Piecewise-Linear Concave Utility Functions. *Management Science* 44(11-part-2):S125–S136.

Nie, Y., and Fan, Y. 2006. Arriving-on-time problem: discrete algorithm that ensures convergence. *Transportation Research Record: Journal of the Transportation Research Board* 1964:193–200.

Nie, Y. M., and Wu, X. 2009. Shortest path problem considering on-time arrival probability. *Transportation Research Part B: Methodological* 43(6):597 – 613.

Nie, Y. M.; Wu, X.; Dillenburg, J. F.; and Nelson, P. C. 2012. Reliable route guidance: A case study from chicago. *Transportation Research Part A: Policy and Practice* 46(2):403 – 419.

Nikolova, E. 2010. Approximation algorithms for reliable stochastic combinatorial optimization. In *Lecture Notes in Computer Science*, volume 6302 LNCS, 338–351.

Pan, Y.; Sun, L.; and Ge, M. 2013. Finding reliable shortest path in stochastic time-dependent network. *Procedia - Social and Behavioral Sciences* 96:451 – 460. Intelligent and Integrated Sustainable Multimodal Transportation Systems Proceedings from the 13th International Conference of Transportation Professionals (CICTP2013).

Samaranayake, S.; Blandin, S.; and Bayen, A. 2012. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C: Emerging Technologies* 20(1):199 – 217.

Zockaie, A.; Nie, Y.; and Mahmassani, H. 2014. Simulation-based method for finding minimum travel time budget paths in stochastic networks with correlated link times. *Transportation Research Record: Journal of the Transportation Research Board* 2467:140–148.

Online Reinforcement Learning for Real-Time Exploration in Continuous State and Action Markov Decision Processes

Hofer Ludovic and **Gimbert Hugo**
Laboratoire Bordelais de Recherche en Informatique
351 Cours de Libration
33405 Talence
France

Abstract

This paper presents a new method to learn online policies in continuous state, continuous action, model-free Markov decision processes, with two properties that are crucial for practical applications. First, the policies are implementable with a very low computational cost: once the policy is computed, the action corresponding to a given state is obtained in logarithmic time with respect to the number of samples used. Second, our method is versatile: it does not rely on any a priori knowledge of the structure of optimal policies. We build upon the Fitted Q-iteration algorithm which represents the Q-value as the average of several regression trees. Our algorithm, the Fitted Policy Forest algorithm (FPF), computes a regression forest representing the Q-value and transforms it into a single tree representing the policy, while keeping control on the size of the policy using resampling and leaf merging. We introduce an adaptation of Multi-Resolution Exploration (MRE) which is particularly suited to FPF. We assess the performance of FPF on three classical benchmarks for reinforcement learning: the "Inverted Pendulum", the "Double Integrator" and "Car on the Hill" and show that FPF equals or outperforms other algorithms, although these algorithms rely on the use of particular representations of the policies, especially chosen in order to fit each of the three problems. Finally, we exhibit that the combination of FPF and MRE allows to find nearly optimal solutions in problems where ϵ-greedy approaches would fail.

1 Introduction

The initial motivation for the research presented in this paper is the optimization of closed-loop control of humanoid robots, autonomously playing soccer at the annual Robocup competition [1]. We specifically target to learn behaviors on the Grosban robot, presented in Figure 1. This requires the computation of policies in Markov decision processes where 1) the state space is continous, 2) the action space is continous, 3) the transition function is not known. Additionally, in order to provide real-time closed-loop control, the policy should allow to retrieve a nearly optimal-action at a low computational-cost. We consider that the transition function is not known, because with small and low-cost humanoid robots, the lack of accuracy on sensors and effectors makes the system behavior difficult to predict.

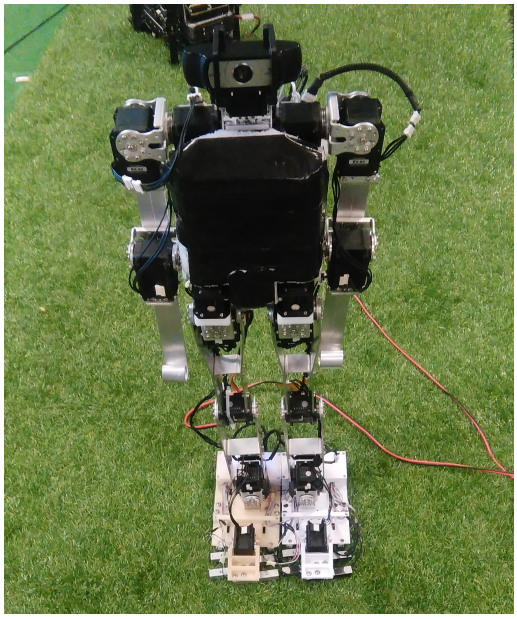


Figure 1: The Grosban robot

More generally, the control of physical systems naturally leads to models with continous-action spaces, since one typically controls the position and acceleration of an object or the torque sent to a joint. While policy gradients methods have been used successfully to learn highly dynamical tasks such as hitting a baseball with an anthropomorphic arm (Peters and Schaal 2008), those algorithms are not suited for learning on low-cost robots, because they need to provide a motor primitive and to be able to estimate a gradient of the reward with respect to the motor primitive parameters. While model-based control is difficult to apply on such robots, hand-tuned open-loop behaviors have proven to be very effective (Behnke 2006). Therefore, model-free learning for CSA-MDP appears as a promising approach to learn such behaviors.

Since the transition and the reward functions are not known *a priori*, sampling is necessary. While an efficient ex-

[1] http://wiki.robocup.org/wiki/Humanoid_League

ploitation of the collected samples is required, it is not sufficient. A smart exploration is necessary, because on some problems, nearly-optimal strategies requires a succession of actions which is very unlikely to occur when using uniformous random actions. On extreme cases, it might even lead to situation where no reward is ever seen, because the probability of reaching a state carrying a reward while following a random policy is almost 0. This problem is known as the combinatory lock problem and appears in discrete case in (Koenig and Simmons 1996) and in continuous problems in (Li, Littman, and Mansley 2009).

For control problems where the action set is discrete and not too large, there are already existing efficient algorithms to tackle the problem of producing an efficient policy from the result of previous experiments. Of course, these algorithms can be used in the continous action space case, by discretization of the action sets. However this naive approach often leads to computational costs that are too high for practical applications, as stated in (Weinstein 2014).

The specificity of continuous action space has also been adressed with specific methods and particularly encouraging empirical results have been obtained thanks for example to the *Binary Action Search* approach (Pazis and Lagoudakis 2009), see also (Busoniu et al. 2010). These methods require to design functional basis used to represent the Q-value function, which we prefer to avoid in order to obtain versatile algorithms.

A recent major-breakthrough in the field of solving CSA-MDP is Symbolic Dynamic Programming which allows to find exact solutions by using eXtended Algebraic Decision Diagrams (Sanner, Delgado, and de Barros 2012), see also (Zamani, Sanner, and Fang 2012). However, those algorithms requires a model of the MDP and rely on several assumptions concerning the shape of the transition function and the reward function. Additionally, those methods are suited for a very close horizon and are therefore not suited for our application.

While local planning allows to achieve outstanding control on high-dimensionnal problems such as humanoid locomotion (Weinstein and Littman 2013), the computational cost of online planning is a burden for real-time application. This is particularly relevant in robotics, where processing units have to be light and small in order to be embedded. Therefore, we aim at global planning, where the policy is computed offline and then loaded on the robot.

Our own learning algorithms are based on the Fitted Q Iteration algorithm (Ernst, Geurts, and Wehenkel 2005) which represents the Q-value as the average of several regression trees. We first present a method allowing to extract approximately optimal continuous action from a Q-value forest. Then we introduce a new algorithm, Fitted Policy Forest (FPF), which learn an approximation of the policy function using regression forests. Such a representation of the policy allows to retrieve a nearly optimal action at a very low computational cost, therefore allowing to use it on embedded systems.

We use an exploration algorithm based on MRE (Nouri and Littman 2009), an optimistic algorithm which represents the knownness of state and action couples using a kd-tree (Preparata and Shamos 1985). Following the idea of extremely randomized trees (Geurts, Ernst, and Wehenkel 2006), we introduce randomness in the split, thus allowing to grow a forest in order to increase the smoothness of the knownness function. Moreover, by changing the update rule for the Q-value, we reduce the attracting power of local maxima.

The viability of FPF is demonstrated by a performance comparison with the results proposed in (Pazis and Lagoudakis 2009) on three classical benchmark in RL: *Inverted Pendulum Stabilization*, *Double Integrator* and *Car on the Hill*. Experimental results show that FPF drastically reduce the computation time while improving performance. We further illustrate the gain obtained by using our version of MRE on the *Inverted Pendulum Stabilization* problem, we finally present the results obtained on the *Inverted Pendulum Swing-Up*, using an underactuated angular joint. This last experiment is run using Gazebo simulator in place of the analytical model.

This paper is organized as follows: Section 2 introduces the notations used for Markov decision processes and regression forests, Section 3 presents the original version of *Fitted Q-Iteration* and other classical methods in batch mode RL with continuous action space, Section 4 proposes algorithms to extract informations from regression forest, Section 5 introduces the core of the FPF algorithm. Section 6 presents the exploration algorithm we used. The efficiency of FPF and MRE is demonstrated through a series of experiments on classical RL benchmarks in section 7, the meaning of the experimental results is discussed in Section 8.

2 Background

2.1 Markov-Decision Process

A *Markov-Decision Process*, or MDP for short, is a 5-tuple $\langle S, A, R, T, \gamma \rangle$, where S is a set of states, A is a set of actions, R is a reward function ($R(s,a)$ denotes the expected reward when taking action a in state s), T is the transition function ($T(s,a,s')$ denotes the probability of reaching s' from s using a) and $\gamma \in [0,1[$ is a discount factor.

A *Deterministic Policy* is a mapping $\pi : S \mapsto A$, where $\pi(s)$ denotes the action choice in state s. Thereafter, by "policy", we implicitely refer to deterministic policy. The Q-value of a couple (s,a) under a policy π with an horizon H is denoted $Q^{\pi}_{H}(s,a)$ and is defined as the expected cumulative and discounted reward by applying a in state s and then choosing actions according to π:

$$Q^{\pi}_{H}(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') Q^{\pi}_{H-1}(s', \pi(s'))$$

We further abreviate Q^{π}_{∞} by Q^{π} for short. The greedy policy with respect to Q is denoted π_Q and always selects the action with the highest Q-value; i.e. $\pi_Q(s) = \underset{a \in A}{\operatorname{argmax}}\, Q(s,a)$. Considering that the action space is bounded to an interval, such a limit exists, although it is not necessarily unique.

It is known that an optimal Q-value function exists (Puterman 1994): $Q^* = \max_{\pi} Q^{\pi}$. The optimal policy π^* is greedy with respect to Q^*: $\pi^* = \pi_{Q^*}$.

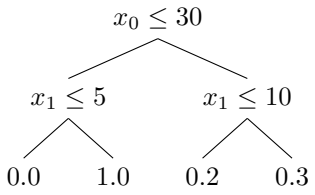


Figure 2: A simple regression tree

Given a complete and finite MDP, standard algorithms exists for finding the optimal policy, including value iteration, policy iteration and linear programming. However, if the transition function or the reward function are unknown, it is necessary to use samples to learn an approximation of the Qvalue denoted $\widehat{Q}$. If the state space or the action space are continuous, it is also necessary to approximate the solution.

When solving offline a MDP while having no direct access to the transition function, it is necessary to use a set of gathered samples. Samples are defined as 4-tuples of the form: $\langle s, a, r, s' \rangle$ where s is the starting state, a the action used, r the reward received and s' the successor state.

2.2 Regression Forests

A *regression tree* is a representation of the approximation of a function $f : X \mapsto Y$ where $X \in \mathbb{R}^k$ and $Y \in \mathbb{R}$. It has a decision tree structure where every non-leaf node is a function mapping X to its children and every leaf is a basic function $\phi : X \mapsto Y$. A simple regression tree with piecewise constant (PWC) approximation is presented in Figure 2. Several algorithms exist to extract regression trees from training set, for a complete introduction, refer to (Loh 2011). Predicting the output y from an entry x requires to find the leaf corresponding to x and then to compute $\phi(x)$, with ϕ the basic function found at the leaf corresponding to x. We will further refer to the value predicted by a tree t for input x by $t(x)$ for short. While some algorithms uses oblique split (Li, Lue, and Chen 2000), the algorithms presented here are only valid for orthogonal splits (splits of the form $x_i \leq v$). We will further note $\mathrm{LC}(n)$ and $\mathrm{UC}(n)$ the lower and upper children of node n, concerning $x_i \leq v$ and $x_i > v$ respectively.

If we define the space X as a hyperrectangle $\mathcal{H}$, each leaf will concern a different part of $\mathcal{H}$. We will further refer to the minimun and maximum value of $\mathcal{H}$ along the dimension i as $\mathcal{H}_{i,m}$ and $\mathcal{H}_{i,M}$ respectively. We define the size of a hyperrectangle $\mathcal{H}$ by $|\mathcal{H}| = \prod_{i=1}^{\dim X} \mathcal{H}_{i,M} - \mathcal{H}_{i,m}$. We use an abusive notation of the norm $\|\mathcal{H}\|$ in place of $\|\mathcal{H}_{i,M} - \mathcal{H}_{i,m}\|$.

A *regression forest* is a set of regression trees: $F = \{t_1, \ldots, t_M\}$. It has been exhibited in (Breiman 1996) that using multipe trees to represent the function leads to a more accurate prediction. The value predicted by a forest F for an input x is $F(x) = \sum_{k=1}^{M} \frac{t_k(x)}{M}$.

2.3 Kd-trees

Kd-trees are a data structure which allows to store points of the same size while providing an $O(\log(n))$ access (Preparata and Shamos 1985). At each leaf of the tree, there is one or several points and at each non-leaf node, there is an orthogonal split. Let X be the space on which the kd-tree τ is defined, then for every $x \in X$, there exist a single path from the root of the kd-tree to the leaf in which x would fit. This leaf is denoted $\mathrm{leaf}(\tau, x)$ is defined on the space X. Each leaf l contains a set of points noted $\mathrm{points}(l)$ and concerns an hyperrectangle $\mathcal{H} = \mathrm{space}(l)$.

3 Previous Work

The use of regression forests to approximate the Q-value of a continuous MDP has been introduced in (Ernst, Geurts, and Wehenkel 2005) under the name of *Fitted Q Iteration*. This algorithm uses an iterative procedure to build $\widehat{Q_H}$, an approximation of the Q-value function at horizon H. It builds a regression forest by using $\widehat{Q_{H-1}}$ and a set of 4-tuples using the rules given at Equations 1 and 2.

$$x = (s, a) \tag{1}$$

$$y = r + \max_{a \in A} \widehat{Q_{H-1}}(s', a) \tag{2}$$

While this procedure yields very satisfying results when the action space is discrete, the computational complexity of the $\max$ part in equation 2 when using regression forest makes it become quickly inefficient. Therefore, in (Ernst, Geurts, and Wehenkel 2005), action spaces are always discretized to compute this equation, thus leading to an inappropriate action set when optimal control requires a fine discretization.

Binary Action Search, introduced in (Pazis and Lagoudakis 2009) proposes a generical approach allowing to avoid the computation of the $\max$ part in equation 2. Results presented in (Pazis and Lagoudakis 2009) show that Binary Action Search strongly outperforms method with a finite number of actions on two problems with rewards including a cost depending on the square of the action used: *Inverted Pendulum Stabilization* and *Double Integrator*. On the other hand, binary action search yields unsatisfying results on *Car on the Hill*, a problem with an optimal strategy known to be "bang-bang" (i.e. optimal strategy is only composed of two actions).

4 Approximation of the Q-value forest

In this part, we propose new methods to extract information from a regression forest while choosing a trade-off between accuracy and computational cost. First, we introduce the algorithm we use to grow regression forest. Then we present an algorithm to project a regression tree on a given subspace. Finally we propose a method allowing to average a whole regression forest by a single regression tree whose number of leaf is bounded.

4.1 Extra-Trees

While several methods exists to build regression forests from a training samples, our implementation is based on Extra-Trees (Geurts, Ernst, and Wehenkel 2006). This algorithm produces satisfying approximation at a moderate computational cost.

The main characteristic of Extra-trees is that k split dimensions are chosen randomly, then for each chosen split dimension the position of the split is picked randomly from an uniformous distribution from the minimal to the maximal value of the dimension along the samples to split. Finally, only the best of the k random splits is used; the criteria used to rank the splits is the variance gain brought by the split. The original training set is splitted until one of the terminal condition is reached. The first terminal condition is that the number of samples remaining is smaller than $n_{\min}$, where $n_{\min}$ is a parameter allowing to control overfitting. There are two other terminal conditions: if the inputs of the samples are all identical or if the output value is constant.

4.2 Improving Extra-trees

We provide two improvements to Extra-trees, in order to remedy two problems. First, due to the terminal conditions, large trees are grown for parts of the space were the Q-value is almost constant because if the Q-value is not strictly constant, the only terminal condition is that the number of samples is lower than $n_{\min}$. We remedy this problem with the help of a new parameter $V_{\min}$ which specifies the minimal variance between prediction and measure necessary to allow splitting. A naive implementation of Extra-Trees leads to a second problem: it may generate nodes with very few samples, which paves the way to overfitting and is bad for linear interpolation. Therefore, we changed the choice of the split values. Instead of choosing it uniformly from the minimum to the maximum of the samples, our algorithm choose it uniformly between the $n_{\min}$-th smallest and highest values, which guarantees that each node of the split tree contains at least $n_{\min}$ samples.

4.3 Projection of a regression tree

Let consider a tree $t : S \times A \mapsto \mathbb{R}$, we can define the projection of the tree t on the state s as another tree $\mathcal{P}(t, s) = t' : A \mapsto \mathbb{R}$. Since s is known, t' does not contain any split depending on s value and therefore contains only splits related to the action space. It is easy to create a hyperrectangle $\mathcal{H}$ corresponding to state s.

$$\mathcal{H}(s) = \begin{pmatrix} s_1 & s_1 \\ \vdots & \vdots \\ s_{D_S} & s_{D_s} \\ \min(A_1) & \max(A_1) \\ \vdots & \vdots \\ \min(A_{D_A}) & \max(A_{D_A}) \end{pmatrix}$$

The pseudo-code for tree projection is shown in Algorithm 1.

Algorithm 1 The tree projection algorithm

```
function PROJECTTREE(t,H)
    return projectNode(root(t), H)
end function
function PROJECTNODE(node,H)
    if isLeaf(node) then
        return node
    end if
    d ← splitDim(node)
    v ← splitVal(node)
    if v > H_{d,M} then
        node ← projectTree(LC(node),H)
    else if v ≤ H_{d,m} then
        node ← projectTree(UC(node),H)
    else
        LC(node) ← projectTree(LC(node),H)
        UC(node) ← projectTree(UC(node),H)
    end if
    return node
end function
```

4.4 Weighted average of regression trees

Let t_1 and t_2 be two regressions trees mapping X to Y, weight respectively by w_1 and w_2, we define the weighted average of the trees as a tree $t' = \mu(t_1, t_2, w_1, w_2)$ such as:

$$\forall x \in X, t'(x) = \frac{t_1(x)w_1 + t_2(x)w_2}{w_1 + w_2}$$

A simple scheme for computing t' would be to root a replicate of t_2 at each leaf of t_1. However this would lead to an overgrown tree containing various unreachable nodes. As example, a split with the predicate $x_1 \leq 3$ could perfectly appear on the lower child of another node whose predicate is $x_1 \leq 2$.

Therefore, we designed an algorithm which merges the two trees by walking simultaneously both trees form the root to the leaves, and performing on-the-fly optimizations. The algorithm pseudo-code is shown in Algorithm 2. An example of input and output of the algorithm is shown in Figure 3. By this way, we also tend to keep an original aspect of the regression tree which is that the top-most nodes carry the most important splits (i.e. splits that strongly reduce the variance of their inner sets of samples).

4.5 Pruning trees

Although our merging procedure helps to reduce the size of the final trees, the combination of M trees might still lead to a tree of size $O(|t|^M)$. Therefore we developed a pruning algorithm which aims at removing the split nodes which bring the smallest change to the prediction function. The only nodes that the algorithms is allowed to remove are nodes that are parent of two leafs. We define the *loss* $\mathcal{L}$ to the prediction function for a node n concerning a hyperrectangle $\mathcal{H}_n$ as:

$$\mathcal{L} = \int_{x \in \mathcal{H}_l} (\phi'(x) - \phi_l)\mathrm{d}x + \int_{x \in \mathcal{H}_u} (\phi'(x) - \phi_u)\mathrm{d}x \quad (3)$$

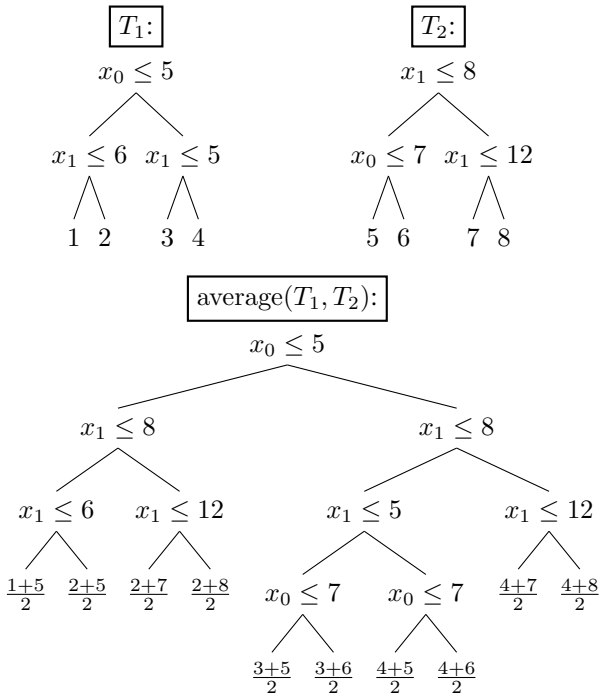


Figure 3: An example of tree merge

Algorithm 2 The averaging tree algorithm

function AVGTREES(t',t_1,t_2,w_1,w_2,$\mathcal{H}$)
  avgNodes(root(t'),root(t_1), root(t_2),w_1,w_2,$\mathcal{H}$)
end function
function AVGNODES(n', n_1, n_2, w_1, w_2,$\mathcal{H}$)
  if isLeaf(n_1) **then**
    if isLeaf(n_2) **then**
      $\phi_{n'} = \frac{w_1\phi_{n_1}+w_2\phi_{n_2}}{w_1+w_2}$
    else
      avgNodes(n', n_2, n_1,w_2,w_1,$\mathcal{H}$)
    end if
  else
    $d \leftarrow \text{splitDim}(\text{n}_1)$
    $v \leftarrow \text{splitVal}(\text{n}_1)$
    $v_m \leftarrow \mathcal{H}_{d,m}$
    $v_M \leftarrow \mathcal{H}_{d,M}$
    if $v_M \leq v$ **then**
      avgNodes(n',n_2, LC(n_1),w_2,w_1,$\mathcal{H}$)
    else if $v_m < v$ **then**
      avgNodes(n',n_2, UC(n_1),w_2,w_1,$\mathcal{H}$)
    else
      split(n') $\leftarrow$ split(n)
      $\mathcal{H}_{d,M} \leftarrow v$
      avgNodes(LC(n'),n_2, LC(n_1),w_2,w_1,$\mathcal{H}$)
      $\mathcal{H}_{d,M} \leftarrow v_M$
      $\mathcal{H}_{d,m} \leftarrow v$
      avgNodes(UC(n'),n_2, UC(n_1),w_2,w_1,$\mathcal{H}$)
      $\mathcal{H}_{d,m} \leftarrow v_m$
    end if
  end if
end function

Where l and u are the lowerchild and upperchild of n respectively, and:

$$\phi' = \frac{|\mathcal{H}_u|\phi_u + |\mathcal{H}_l|\phi_l}{|\mathcal{H}|} \tag{4}$$

The prediction function ϕ' given by equation 4 is a weighted average of the prediction functions of both children weighted by the size of the space concerned by each one. This choice reduces the impact of the prediction on a leaf when merged with a bigger leaf. Our definition of the loss $\mathcal{L}$ in equation 3 also considers the size of the spaces since we compute the integral. The main interest of this method is to reduce the average error on the whole tree by weighting the cost of an error by the size of its space. While most prunning procedures in litterature are centered about reducing the risk of overfitting, our algorithm (Algorithm 3) cares only about reducing the size of the tree, ensuring that the complexity of the representation does not go above a given threshold. Since this procedure is not based on the training set used to grow the forest, it is not necessary to have an access to the training set in order to prune the tree. When merging the M trees of a forest, it is crucial to prune the tree resulting of two merge before applying another merge.

Algorithm 3 The tree pruning algorithm

splits = {} ▷ Map from (node, $\mathcal{L}$) to ϕ, ordered by $\mathcal{L}$
for all $n \in$ preLeafs(t) **do**
  $\mathcal{L} = \text{getLoss}(n)$ ▷ See Eq. 3
  $\phi = \text{getAverageFunction}(n)$ ▷ See Eq. 4
  add $((n, \mathcal{L}), \phi))$ to splits
end for
nbLeafs $\leftarrow$ countLeafs(t)
while nbLeafs $>$ maxLeafs **do**
  $((n, \mathcal{L}), \phi)) \leftarrow \text{popFirst}(\text{splits})$
  $\phi_n \leftarrow \phi$
  removeChild(n)
  if isLastSplit(father(n)) **then**
    $n \leftarrow \text{father}(n)$
    $\mathcal{L} = \text{getLoss}(n)$ ▷ See Eq. 3
    $\phi = \text{getAverageFunction}(n)$ ▷ See Eq. 4
    add $((n, \mathcal{L}), \phi))$ to splits
  end if
  nbLeafs $\leftarrow$ nbLeafs $- 1$
end while

5 Approximation of the optimal policy

In this section, we propose three new methods used to choose optimal action for a given state based on an estimation of the Q-value by a regression forest. While learning of the policy can be computationally demanding since it is performed offline, it is crucial to obtain descriptions of the policies that allow very quick computation of the action, given the current state.

5.1 Learning the continuous policy

In order to compute the best policy given an approximation of the Q-value $\widehat{Q}$ by a regression forest F, we need to solve the following equation:

$$\widehat{\pi^*}(s) = \underset{a \in A}{\operatorname{argmax}}\, F((s,a)) \tag{5}$$

Given s, the most straightforward way to compute $\widehat{\pi^*}(s)$ consists in merging all the trees of F projected on s into a single tree t'. Since the size of t' can grow exponentially with the number of trees, we compute an approximation of t', denoted $\widehat{t'}$ by imposing a limit on the number of leafs using Algorithm 3. Then it is possible to approximate the best actions by simply iterating on all the leafs of $\widehat{t'}$ and computing the maximum of the function ϕ of the leaf in its interval. While this solution does not provide the exact policy which would be induced by F, it provides a roughly good approximation. We refer to this method by *Fitted Q-Iteration*, ***FQI*** for short.

The FQI is computationally too expensive to be used in online situation: the computation of a single action requires exploring a potentially large number of leaves. Therefore, in order to provide a very quick access to the optimal action for a given state, we propose a new scheme. By decomposing the policy function $\pi : S \mapsto A$ into several functions $\pi_j : S \mapsto A_j$ where j is a dimension of the action space, we can easily generate samples and use them to train regression forests which provide estimates of the policy for each dimension. We named this process *Fitted Policy Forest* and abreviate it by FPF. We use two variants, one using a piecewise constant model for the nodes, PWC for short, and another using piecewise linear model for the nodes, PWL for short. We refer to these two methods by ***FPF:PWC*** and ***FPF:PWL*** respectively. Policies resulting of the FPF algorithm provides a quick access. If such a policy is composed of M trees with a maximal number of nodes n, the complexity of getting the action is $O(M\log(n))$. Since the values used for M does not need to be high to provide a good approximation (Ernst, Geurts, and Wehenkel 2005), this complexity makes FPF perfectly suited for real-time applications where online computational ressources are very limited, such as robotics.

6 Exploration

While MRE (Nouri and Littman 2009) provide a strong basis to build exploration algorithm, we found that its performance can be strongly improved by bringing three modifications. First we change the equation used to compute the knownness, second we use bagging technic to improve the estimation of the knownness, and third we modify the rule used for Q-value update.

6.1 Original definition

Multi Resolution Exploration (Nouri and Littman 2009) propose a generic algorithm allowing to balance the exploration and the exploitation of the samples. The main idea is to build a function $\kappa : S \times A \mapsto [0,1]$ which estimate the degree of knowledge of a couple $(s,a) \in S \times A$. During the execution

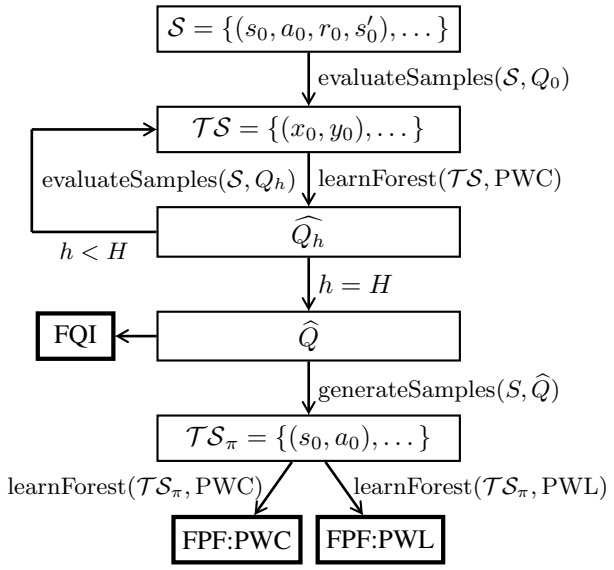


Figure 4: A flowchart of the different methods

of the algorithm, when action a is taken in state s, a point $p = (s_1, \dots, s_{\dim S}, a_1, \dots, a_{\dim A})$ is inserted in a kd-tree, called knownness-tree. Then, the knownness value according to a knownness-tree τ at any point p can be computed by using the following equation:

$$\kappa(p) = \min\left(1, \frac{|P|}{\nu} \frac{1}{\left\lfloor \sqrt[k]{nk/\nu} \right\rfloor \over \|\mathcal{H}\|_\infty}\right) \tag{6}$$

where ν is the maximal number of points per leaf, $k = \dim(S \times A)$, n is the number of points inside the whole tree, $P = \text{points}(\text{leaf}(\tau, p,))$ and $\mathcal{H} = \text{space}(\text{leaf}(\tau, p,))$. A crucial point of this equation is the fact that the knownness value depends on three main aspects: the size of the cell, the number of points inside the cell and the number of points inside the whole tree. Therefore, if the ratio between the number of points contained in a cell and its size does not evolve, its knownness value will decrease.

The insertion of points inside the kd-tree follows this rule: if adding the point to its corresponding leaf l_0 would lead to a number of points greater than ν, then the leaf is splitted into two leafs l_1 and l_2 of the same size, and the dimension is chosen using a round-robin. Then the points stored in l_0 are attributed to l_1 and l_2 depending on their value.

MRE also changes the update rule by using an optimistic rule which replace equation (2) by equation (7):

$$y' = \kappa(s,a)y + (1 - \kappa(s,a))\frac{R_{\max}}{1-\gamma} \tag{7}$$

where $R_{\max}$ is the maximal reward which can be awarded in a single step and y is the result obtained by equation (2). This update can be seen as adding a transition to a fictive state containing only self-loop and leading to a maximal reward at every step. This new transition occurs with probability $1 - \kappa(s,a)$.

6.2 Computation of the knownness value

Initial definition of the knownness is given at Equation (6). Since this definition does only depend on the biggest dimension, we have the following. Consider a leaf l_0 with a knownness τ_0, then adding a point can result in creating two new leafs l_1 and l_2 with respective knowledge of k_1 and k_2 with $k_0 > k_1$ and $k_0 > k_2$. This leads to the unnatural fact that adding a point in the middle of other points can decrease the knowledge of all these points.

We decide to base our knowledge on the ratio between the density of points inside the leaf and the density of points. Thus replacing Equation (6) by Equation (8):

$$\kappa(p) = \min\left(1, \frac{\frac{|\text{points}(\text{leaf}(\tau,p))|}{|\text{leaf}(\tau,p)|}}{\frac{n}{|S\times A|}}\right) \tag{8}$$

where n is the total number of points inside the tree. This definition leads to the fact that at anytime, there is at least one leaf with a knownness equal to 1. It is also easy to see that there is at least one leaf with a knownness strictly lower than 1, except if all the cells have the same density.

6.3 From knownness tree to knownness forest

In order to increase the smoothness of the knownness function, we decided to aggregate several kd-trees to grow a forest, following the core idea of extra-trees (Geurts, Ernst, and Wehenkel 2006). However, in order to grow different kd-trees from the same input, the splitting process needs to be stochastic. Therefore, we implemented another splitting scheme based on extra-trees.

The new splitting process is as follows: for every dimension, we choose at uniformous random a split between the first sample and the last sample. Thus, we ensure that every leaf contains at least one point. Then we use an heuristic to choose the best split.

Once a knownness forest is grown, it is easy to compute the knownness value by averaging the result of all the trees.

6.4 Modification of the Q-value update

The Q-value update rule proposed by MRE improve the search speed, however it has a major drawback. Since it only alters the training set used to grow the regression forest, it can only use the knownness information on state action combination which have been tried. Therefore, even if for a state s and an action a, $\kappa(s,a) \approx 0$, it might have no influence at all.

In order to solve this issue, we decided to avoid the modification of the training set creation, thus using Equation (2). In place of modifying those samples, we simply update the regression forest by applying the following modificator on every leaf of every tree:

$$v' = v\kappa(c) + R_{\max}(1-\kappa(c)) \tag{9}$$

with c the center of the leaf, v the original value and v' the new value.

7 Experimental results

We present experimental results under two different learning setup. First, the results obtained by FPF in a batch reinforcement learning, second, the performances obtained by combining MRE and FPF for online learning.

7.1 Batch reinforcement learning

We used three benchmark problems classical in RL to evaluate the perfomances of the FPF algorithms. While all the methods share the same parameters for computing the Q-value forest, we tuned specifically parameters concerning the approximation of the policy using the Q-value forest. We compared our results with those presented in (Pazis and Lagoudakis 2009), however we do not have access to their numerical data, and rely only on the graphical representation of these datas. Thus, the graphical lines shown for BAS are approximative and drawn thicker than the other to highlight the noise in measurement. We present the result separately for the three benchmarks while discussing results specific to a problem as well as global results. On all the problems, performances of FPF:PWL are better or at least equivalent to those achieved by BAS in (Pazis and Lagoudakis 2009). This is remarkable, because BAS uses a set of basic functions specifically chosen for each problem, while our method is generic for all the problems. The computation cost of retrieving actions once the policy has been calculated appears as negligeable and therefore confirms that our approach is perfectly suited for high-frequency control in embedded systems.

Inverted pendulum stabilization The *inverted pendulum stabilization* problem consists of balancing a pendulum of unknown length and mass by applying a force on the cart it is attached to. We use the description of the problem given in (Pazis and Lagoudakis 2009). The state space is composed of the angular position of the pendulum θ and the angular speed of the pendulum $\dot{\theta}$, the action space is $[-50, 50]$ Newtons, an uniform noise in $[-10, 10]$ Newtons is added. The goal is to keep the pendulum perpendicular to the ground and the reward is formulated as following:

$$R(\theta,\dot{\theta},f) = -\left((2\theta/\pi)^2 + \left(\dot{\theta}\right)^2 + \left(\frac{f}{50}\right)^2\right)$$

except if $|\theta| > \frac{\pi}{2}$, in this case the reward is -1000 and the state is considered as terminal. We set the discount rate γ to 0.95. The transitions of the system follow the nonlinear dynamics of the system described in (Wang, Tanaka, and Griffin 1996):

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml\left(\dot{\theta}\right)^2 \frac{\sin(2\theta)}{2} - \alpha\cos(\theta)u}{\frac{4l}{3} - \alpha ml\cos^2(\theta)}$$

where g is the constant of gravity $9.8[m/s^2]$, $m = 2.0[kg]$ is the mass of the pendulum, $M = 8.0[kg]$ is the mass of the cart, $l = 0.5[m]$ is the length of the pendulum, $\alpha = \frac{1}{m+M}$ and u is the final (noisy) action applied. We used a control step of $100[ms]$ and an integration step of $1[ms]$ (using Euler Method). The reward used in this description of the problem ensure that policies leading to a smoothness of motion

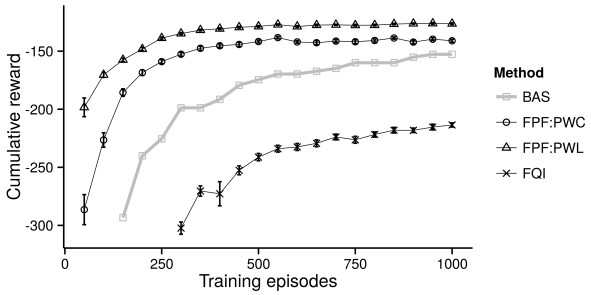


Figure 5: Performance on the Inverted Pendulum Stabilization problem

and using low forces to balance the inverted pendulum are rated higher than others.

The training sets were obtained by simulating episodes using a random policy, and the maximal number of steps for an episode was set to 3000. The performances of the policies were evaluated by testing them on episodes of a maximal length of 3000 and then computing the cumulative reward. In order to provide an accurate estimate of the performance of the algorithms, we computed 50 different policies for each point displayed in Figure 5 and average their cumulative reward (vertical bars denote 95% confidence interval). The parameters used to produce the policies are shown in Table 1.

Learning a policy from the Q-value tree clearly outperform a direct use on this problem and PWL approximations outperform PWC approximations. Results for BAS (Pazis and Lagoudakis 2009) rank systematically lower than both FPF methods. The huge difference of learning speed between FQI and FPF suggests that using regression forest to learn the policy from the Q-value can lead to drastical improvements. On such a problem where the optimal policy requires a fine choice of action, it is not surprising that using linear models to represent the policy provide higher results than constant models.

The best value for $n_{\min}$, the minimal number of samples per leaf, is pretty high (17 for PWC and 125 for PWL). Our understanding of this phenomena is that the Q-value tree tend to slightly overfit the data, additionally, it uses PWC approximation. Therefore, using it directly lead to an important quantization noise. Using a large value for $n_{\min}$ might be seen as applying a smoothing, which is considered as necessary for regression trees sampling a stochastic function according to (Ernst, Geurts, and Wehenkel 2005). The need for a large number of samples is increased for FPF:PWL, because providing an accurate linear interpolation of a noisy application requires a lot of samples.

Double integrator In order to provide a meaningful comparison, we stick to the description of the problem given in (Pazis and Lagoudakis 2009) where the control step has been increased from the original version presented in (Santamaria, Sutton, and Ram 1997). The double integrator is a linear dynamics system where the aim of the controller

Table 1: Parameters used for Inverted Pendulum Stabilization

Parameter	FQI	FPF:PWC	FPF:PWL
Nb Samples	NA	10'000	10'000
Max Leafs	50	50	50
k	NA	2	2
$n_{\min}$	NA	17	125
M	NA	25	25
$V_{\min}$	NA	10^{-4}	10^{-4}

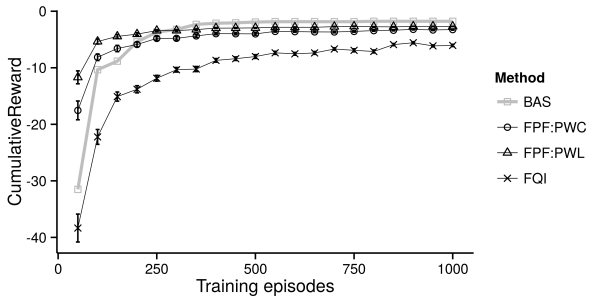


Figure 6: Performance on the Double Integrator problem

is to reduce negative quadratic costs. The continuous state space consist of the position $p \in [-1, 1]$ and the velocity $v \in [-1, 1]$ of a car. The goal is to bring the car to an equilibrium state at $(p, v) = (0, 0)$ by controlling the acceleration $\alpha \in [-1, 1]$ of the car. There are two constraints: $|p| \leq 1$ and $|v| \leq 1$. In case any of the constraint is violated, a penalty of 50 is received and the experiment ends. In all other case, the cost of a state is $p^2 + a^2$. The control step used is $500[\mathrm{ms}]$ and the integration step is $50[\mathrm{ms}]$, the discount factor was set to $\gamma = 0.98$.

The training sets were obtained by simulating episodes using a random policy, and the maximal number of steps for an episode was set to 200. The performances of the policies were evaluated by testing them on episodes of a maximal length of 200 and then computing the cumulative reward. In order to provide an accurate estimate of the performance of the algorithms, we computed 100 different policies for each point displayed in Figure 6 and average their results. The parameters used for learning the policy are shown in Table 2.

On this problem, although none of the proposed methods reach BAS performance when there are more than 300 learning episodes, FPF:PWL learns quicker than BAS with a small number of episodes. It is important to note that while our basic function approximator is constant, a polynome is used for Least-Square Policy Iteration in (Pazis and Lagoudakis 2009), fitting the fact that the optimal policy is known to be a linear-quadratic regulator (Santamaria, Sutton, and Ram 1997).

Car on the hill While there has been several definitions of the *Car on the Hill* problem, we will stick to the version proposed in (Ernst, Geurts, and Wehenkel 2005) which was also used as a benchmark in (Pazis and Lagoudakis 2009).

Table 2: Parameters used for Double Integrator

Parameter	FQI	FPF:PWC	FPF:PWL
Nb Samples	NA	10'000	10'000
Max Leafs	40	40	40
k	NA	2	2
$n_{\min}$	NA	100	1500
M	NA	25	25
$V_{\min}$	NA	10^{-4}	10^{-4}

In this problem an underactuated car must reach the top of a hill. The state space is composed of the position $p \in [-1, 1]$ and the speed $s \in [-3, 3]$ of the car while the action space is the acceleration of the car $u \in [-4, 4]$. If the car violate one of the two constraints: $p \geq -1$ and $|s| \leq 3$, it receives a negative reward of -1, if it reaches a state where $p > 1$ without breaking any constraint, it receive a reward of 1, in all other states, the reward is set to 0. The car need to move away from its target first in order to get momentum.

It is well known that the solution to this problem is a bang-bang strategy, i.e. a nearly optimal strategy exists which uses only the set of actions $\{-4, 4\}$. As stated in (Pazis and Lagoudakis 2009), this problem is one of the worst case for reinforcement learning with continuous action space, since it requires to learn a binary strategy composed of actions which have not been sampled frequently. It has been shown in (Ernst, Geurts, and Wehenkel 2005) that introducing more actions usually reduce the performance of the controller. Therefore, we do not hope to reach a performance comparable to those achieved with a binary choice. This benchmark is more aimed to assess the performance of our algorithms, in one of the worst case.

While the sample of the two previous algorithms are based on episodes generated at a starting point, the samples used for the *Car on the hill* problem are generate by sampling uniformly the state and action spaces. This procedure is the same which has been used in (Ernst, Geurts, and Wehenkel 2005) and (Pazis and Lagoudakis 2009), because it is highly improbable that a random policy could manage to get any positive reward in this problem. Evaluation is performed by observing the repartition of the number of steps required to reach the top of the hill from the initial state $(-0.5, 0)$.

We show the histogram of the number of steps required for each method at Figure 7. For each method, 200 different strategies were computed and tested. There is no significant difference in the number of steps required to reach the top of the hill between the different methods. For each method, at least 95% of the computed policies led to a number of step in the interval $[20, 25]$. Thus we can consider that an FPF or FQI controller take 20 to 25 steps on average while it is mentioned in (Pazis and Lagoudakis 2009) that BAS controller requires 20 to 45 steps on average. Over the six hundred of experiments gathered across three different methods, the maximal number of steps measured was 33. Therefore, we can consider that our results strongly outperforms BAS results.

Car on the Hill is the only problem on which we have

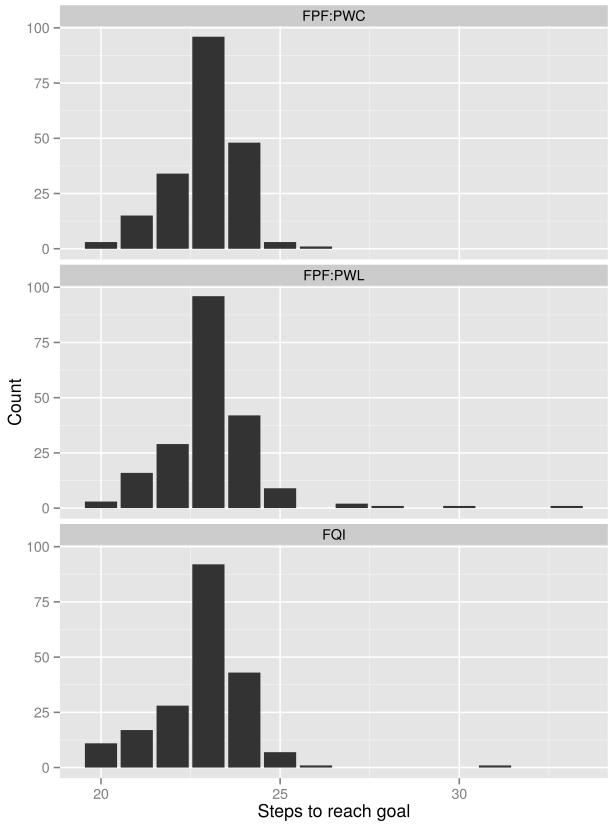


Figure 7: Performance on the Car on the Hill problem

not experienced significant difference between FPF and FQI. Since one of the main advantage of FPF approach is to reduce the quantization noise of the FQI method, this result is logical. Although the number of steps required is not reduced by the FPF approach, the online cost is still reduced by around two orders of magnitude. Therefore, we can affirm that FPF is highly preferable to FQI on this problem.

Computational cost As mentioned previously, a quick access to the optimal action for a given state is crucial for real-time applications. We present the average time spent to retrieve actions for different methods in Figure 8 and the average time spent for learning the policies in 9. Experiments were runned using an AMD Opteron(TM) Processor 6276 running at 2.3 GHz with 16 GB of RAM running on Debian 4.2.6. While the computer running the experiments had 64 processors, each experiment used only a single core.

We can see that using FPF reduces the average time by more than 2 orders of magnitude. Moreover, FPF:PWL presents a lower online cost than FPF:PWC, this is perfectly logical since representing a model using linear approximation instead of constant approximations requires far less nodes. While the results are only displayed for the "Double Integrator" problem due to the lack of space, similar results were observed for the two other problems.

It is important to note that the cost displayed in Figure 8

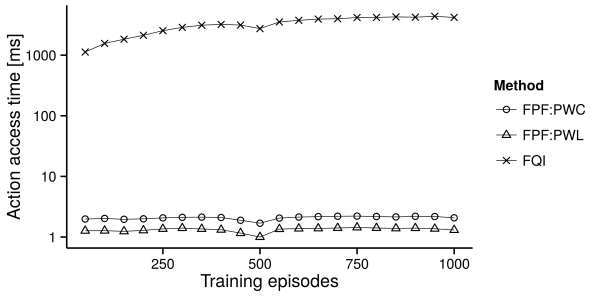


Figure 8: Evaluation time by episod for the Double Integrator

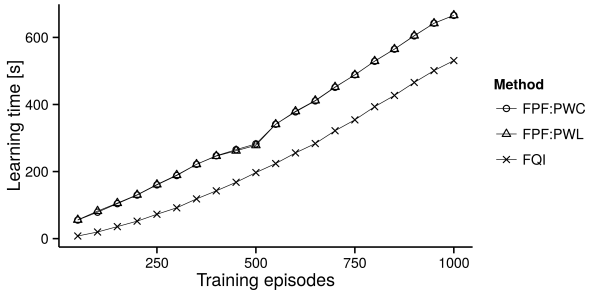


Figure 9: Learning time by episod for the Double Integrator

represents an entire episode simulation, thus it contains 200 action access and simulation steps. Therefore, it is safe to assume that the average time needed to retrieve an action with FPF:PWC or FPF:PWL is inferior to $50\mu s$. Even if the CPU used is two orders of magnitude slower than the one used in the experiment, it is still possible to include an action access at $200Hz$.

The additional offline cost of computing the polices required by FPF is lower than the cost of computing the Q-value using FQI when the number of training episode grows, as presented in Figure 9. Therefore, when it is possible to use FQI, it should also be possible to use FPF without increasing too much the offline cost.

7.2 Online reinforcement learning

We evaluated the performance of the combination of MRE and FPF on two different problems. First, we present the experimental results on the *Inverted Pendulum Stabilization* problem and compare them with the results obtained with random exploration. Second, we exhibit the results on the *Inverted Pendulum Swing-Up* problem. Since online learning on robots can be expensive in time and resources, we did not allow for an early phase of parameter tuning and we used simple rules to set parameters for both problems. In both problems, the policy is updated at the end of each episode, in order to ensure that the system is controlled in real-time. In this section, we denote by *trial* a whole execution of the MRE algorithm on the problem.

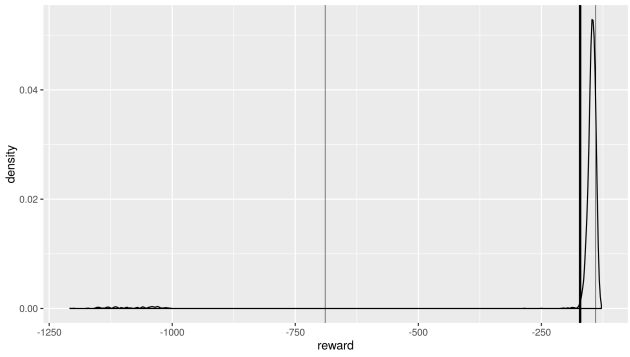


Figure 10: Reward repartition for online learning on Inverted Pendulum Stabilization

Inverted pendulum stabilization This problem is exactly the same as defined in Section 7.1, but it is used in a context of online reinforcement learning. The result presented in this section represent 10 trials of 100 episodes. Each trial was used to generate 10 different policies, every policy was evaluated by 50 episodes of 3000 steps. Thus, the results concerns a total of 5000 evaluations episodes.

The repartion of reward is presented in Figure 10. The reward obtained by the best and worst policy are shown as thin vertical lines, while the average reward is represented by a thick vertical line. Thus, it is easy to see that there is a huge gap between the best and the worst policy. Over this 5000 episodes, the average reward per run was -171, with a minimum of -1207 and a maximal reward of -128. In the batch mode settings, after the same number of episodes, FPF-PWL obtained an average reward of -172, with a minimal reward of -234 and a maximal reward of -139. While the average reward did not significantly improve, the dispersion of reward has largely increased and in some cases, thus leading to better but also worst policy. While this might be perceived as a weakness, generating several policies from the computed Q-value is computationally cheap. Then, a few episodes might be used to select the best policy. From the density of reward presented in Figure 10, it is obvious that by removing the worst 10% of the policies, the average reward would greatly improve.

Another point to keep in mind is the fact that the parameters of FPF have not been optimized for the problem in the MRE setup, while they have been hand-tuned in the Batch setup. Therefore, reaching a comparable performance without any parameter tuning is already an improvement.

Inverted pendulum swing-up For this problem, instead of using a mathematical model, we decided to use the simulator Gazebo[2] and to control it using ROS[3]. Since these two tools are widely accepted in the robotic community, we believe that exhibiting reinforcement learning experiments based on them can contribute to the democratization of RL methods in robotics. We developed a simple model composed of a support and a pendulum which are bounded by

[2]http://gazebosim.org

[3]http://www.ros.org

an angular joint. The angular joint is controled in torque and is underactuated, i.e. the available torque is not sufficient to maintain the pendulum in an horizontal state. The main parameters are the following: the mass of the pendulum is $5[kg]$, the length of the pendulum is $1[m]$, the damping coefficient is $0.1[Nms/\mathrm{rad}]$, the friction coefficient is $0.1[Nm]$, the maximal torque is $\tau_{\max} = 15[Nm]$, the maximal angular speed is $\dot{\theta}_{\max} = 10[\mathrm{rad}/s]$ and the control frequency is $10[Hz]$. The reward function used is the following

$$r = -\left(\left\|\frac{\theta}{\pi}\right\| + \left(\frac{\tau}{\tau_{\max}}\right)^2\right) \tag{10}$$

Where θ is the angular position of the pendulum (0 denote an upward position), and τ represent the torque applied on the axis. If $\left\|\dot{\theta}\right\| > \dot{\theta}_{\max}$, a penalty of 50 is applied and the episode is terminated.

While the system only involves two state dimensions and one action dimension, it presents two main difficulties: first, random exploration is unlikely to produce samples where $\theta \approx 0$ and $\dot{\theta} \approx 0$ which is the target, second, it requires the use of the whole scale of action, large actions in order to inject energy in the system and fine action in order to stabilize the system.

The result presented in this section represent 5 trials of 100 episodes. Each trial was used to generate 10 different policies, every policy was evaluated by 10 episodes of 100 steps. Thus, there is a total of 500 evaluation episodes.

We present the repartition of the reward in Figure 11. The average reward is represented by a thick vertical line and the best and worst policies rewards are shown by thin vertical lines. Again, we can notice a large difference between the best and the worst policy. We exhibit the trajectory of the best and worst evaluation episode in Figure 12. While the worst episode has a cumulated reward of -101, the worst policy has an average reward of -51. According to the repartition of the reward, we can expect that very few policies lead to such unsatisfying results, thus ensuring the reliability of the learning process if multiple policies are generated from the gathered samples and a few episodes are used to discard the worst policy.

8 Discussion

Our results show that using FPF does not only allow to drastically reduce the online computational cost, it also tend to outperforms FQI and BAS, especially when the transition function is stochastic as in the Inverted Pendulum Stabilization problem.

Although using piecewise linear function to represent the Q-value often leads to divergence as mentioned in (Ernst, Geurts, and Wehenkel 2005), the same problem did not appear on any of the three presented problems. In two of the three presented benchmarks, FPF:PWL yields significantly better results than FPF:PWC and on the last problem, results were similar between the two method. The possibility of using PWL approximations for the representation of the policy holds in the fact that the approximation process is performed only once. Another advantage is the fact that on two

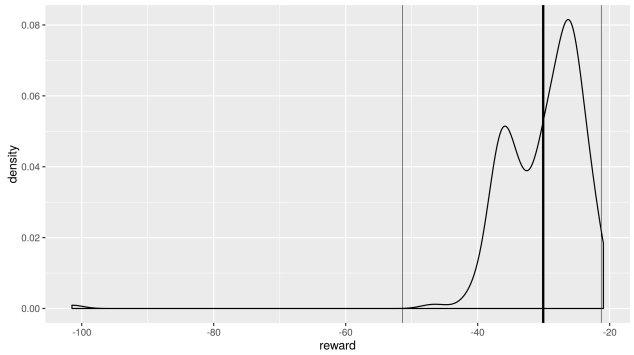


Figure 11: Reward repartition for online learning on Inverted Pendulum Swing-Up

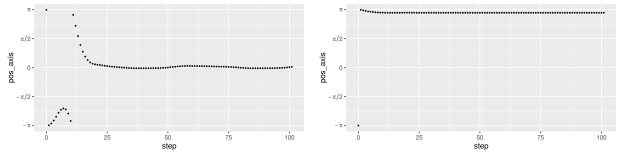


Figure 12: Best and worst episode for Inverted Pendulum Swing-Up

of the problem, the policy function is continuous. However, even when the optimal policy is bang-bang (Car on the hill), using PWL approximation for the policy does not decrease the general performance.

Our experiments on the combination of MRE and FPF showed that we can obtain satisfying results without a parameter-tuning phase. Results also show the strong variability of the generated policies, thus leading to a natural strategy of generating multiple policies and selecting the best in a validation phase.

9 Conclusion

This article introduces Fitted Policy Forest, an algorithm extracting a policy from a regression forest representing the Q-value. FPF presents several advantages: it has an extremely low computational cost to access the optimal action, it does not require expert knowledge about the problem, it is particularly successful at solving problems requiring fine actions in stochastic problems and it can be used with any algorithm producing regression forests. The effectiveness of our algorithm in a batch setup is demonstrated in three different benchmarks. The use of FPF in online reinforcement learning is also discussed and assessed by using MRE as an exploration strategy. Experimental results suggest that exploration can lead to satisfying results without requiring any tuning on the parameters. In the future, we also would like to apply this approach to closed-loop control of Robocup humanoid robots.

References

Behnke, S. 2006. Online trajectory generation for omnidirectional biped walking. *Proceedings - IEEE International Conference on Robotics and Automation* 2006(May):1597–1603.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Busoniu, L.; Babuska, R.; Schutter, B. D.; Ernst, D.; Busoniu, L.; Babuska, R.; Schutter, B. D.; and Ernst, D. 2010. Reinforcement learning and dynamic programming using function approximators. 260.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research* 6(1):503–556.

Geurts, P.; Ernst, D.; and Wehenkel, L. 2006. Extremely randomized trees. *Machine Learning* 63(1):3–42.

Koenig, S., and Simmons, R. G. 1996. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning* 22(1-3):227–250.

Li, L.; Littman, M. L.; and Mansley, C. R. 2009. Online exploration in least-squares policy iteration. *The 8th International Conference on Autonomous Agents and Multiagent Systems* 733–739.

Li, K.-C.; Lue, H.-H.; and Chen, C.-H. 2000. Interactive Tree-Structured Regression via Principal Hessian Directions. *Journal of the American Statistical Association* 95(450):547–560.

Loh, W.-Y. 2011. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1(1):14–23.

Nouri, A., and Littman, M. L. 2009. Multi-resolution Exploration in Continuous Spaces. *Advances in Neural Information Processing Systems* 1209–1216.

Pazis, J., and Lagoudakis, M. G. 2009. Binary action search for learning continuous-action control policies. *Proceedings of the 26th International Conference on Machine Learning (ICML)* 793–800.

Peters, J., and Schaal, S. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4):682–697.

Preparata, F. P., and Shamos, M. I. 1985. *Computational geometry: an introduction*, volume 47.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2012. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *Proceedings of the 26th Conference on Artificial Intelligence*, volume 2.

Santamaria, J. C.; Sutton, R. S.; and Ram, A. 1997. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior* 6(2):163–217.

Wang, H. O.; Tanaka, K.; and Griffin, M. F. 1996. An approach to fuzzy control of nonlinear systems: Stability and design issues. *Ieee Transactions on Fuzzy Systems* 4(1):14–23.

Weinstein, A., and Littman, M. 2013. Open-Loop Planning in Large-Scale Stochastic Domains. In *27th AAAI Conference on Artificial Intelligence*, volume 1, 1436–1442.

Weinstein, A. 2014. *Local Planning for Continuous Markov Decision Processes*. Ph.D. Dissertation, The State University of New Jersey.

Zamani, Z.; Sanner, S.; and Fang, C. 2012. Symbolic Dynamic Programming for Continuous State and Action MDPs.

Planning for Robots with Skills

Matthew Crosby,
Ronald P. A. Petrick
Department of Computer Science
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
{M.Crosby,R.Petrick}@hw.ac.uk

Francesco Rovida,
Mikkel Rath Pedersen, Volker Krüger
Robotics, Vision, and Machine Intelligence Lab
Aalborg University
Copenhagen 2450, Denmark
{francesco,mrp,vok}@m-tech.aau.dk

Abstract

Modern industrial robotics is characterised by a need for flexibility in robot design, in order to minimise programming and development time when a robot's tasks must be changed. To address this problem, a recent approach has proposed that robots be equipped with a set of general, reoccurring operations called 'skills', e.g., picking, placing, or driving. This paper presents a method for automatically generating planning problems from existing skill definitions such that the resulting problems can be solved using off-the-shelf planning software, and the solutions can be used to control robot actions in the world. As a result, a robot can therefore perform new tasks simply by specifying the task's goals via a GUI. The approach is demonstrated on a set of common tasks in a simulated industrial environment and has also been tested successfully on a real-world robotic platform.

Introduction

Robot autonomy is becoming increasingly important in modern industrial robotics, where factory robots often possess low degrees of autonomous operation at the task level, with a relatively large proportion of time spent on robot programming, compared with the time the robots spend performing tasks. This has important consequences for the current trend towards flexible manufacturing which requires frequent changeovers to new products: when a changeover occurs, the robots must be reprogrammed for the new tasks.

Task-level programming provides one way of simplifying the robot control problem. In this paradigm, a human programmer specifies what the robot should do in terms of the high-level actions and objects involved in a task, rather than focusing on the low-level details of the robot or its operating space. Actions are abstracted in a way which hides the complexity of the lower layers from the programmer, allowing users to focus on the task itself. The result is a powerful way to speed up programming, even with complex robots.

One proposal for implementing such a programming framework is based on defining tasks as sequences of *skills*, where skills are identified as the re-occurring actions needed to execute standard operating procedures in a factory (e.g., operations like *pick 'object'* or *place at 'location'*) (Madsen et al. 2015; Pedersen et al. 2016). Embedded within the skill definitions are the sensing and motor operations, or *primitives*, that accomplish the goals of the skill, as well as a set of condition checks that are made before and after execution to ensure robustness. This methodology also provides a process for specifying high-level parameters for skills, while low-level parameters for the primitive operations are mostly inferred through autonomous reasoning by the robot.

Figure 1: A robot operating in a factory environment using the SkiROS system. The robot is executing a six-step plan to place two parts in the white kitting box it is carrying.

While skills have been shown to be a useful tool for human operators to programme robot tasks (Madsen et al. 2015), the goal of increased robot autonomy in the factory environment also relies on the robots *themselves* being able to automatically sequence skills to perform tasks. For instance, when a new skill is introduced to a robot, a skills expert must specify the skill in terms of its input parameters, how it should be executed using low-level primitives, the conditions that must hold of the world state, and the intended changes to the world model. Previous work (Pedersen and Krüger 2015) showed how skill definitions of this form enabled planning problems to be created by hand and used to drive robot actions.

This paper instead introduces techniques for automating the creation of planning domains from the robot's skills and world model, so that the entire process of robot control can itself be automated. In particular, this work focuses on how

a planning problem can be automatically generated from the skill definition itself and, given a world model and a set of goals, how a sequence of parameterised skills can be constructed to achieve these goals. This has important consequences for robot control: using this system, and provided the appropriate skills are implemented, only the goals of the task need to be specified for a robot to complete a new task. This process is demonstrated with a set of skills (e.g., drive, pick, and place) implemented in a skills framework called *SkiROS* (Rovida and Krüger 2015), for a simulated robot system designed for a real factory environment. This approach has also been tested in a factory setting using a real robot and the same set of skills (see Figure 1).

The rest of this paper is organised as follows. First, the related work is considered. Then, the system architecture is introduced including a description of the skills framework. The world model is then outlined, followed by a description of the task planner and the process for converting skills to PDDL. The paper concludes with a set of experiments performed in simulation that demonstrate the system functioning in a mock factory environment resembling the real-world environment for which this system has been implemented.

Related Work

During the last three decades, three main approaches to robot control have dominated the research community: reactive, deliberative, and hybrid control (Kortenkamp and Simmons 2008). Reactive systems rely on a set of concurrently running modules, called behaviours, which directly connect input sensors to particular output actuators (Arkin 1998; Brooks 1986). In contrast, deliberative systems employ a sense-plan-act paradigm, where reasoning plays a key role in an explicit planning process. Hybrid systems attempt to exploit the best of both worlds, through mixed architectures with a deliberative high level, a reactive low level, and a synchronisation mechanism in the middle that mediates between the two (Firby 1989). Most modern autonomous robots use a hybrid approach (Gat 1998; Ferrein and Lakemeyer 2008; Bensalem and Gallien 2009; Magnenat 2010), with researchers focused on finding appropriate interfaces between declarative high-level reasoning and procedural low-level control.

SkiROS (Skills-ROS) (Rovida and Krüger 2015), the skills architecture used in this paper, is a hybrid framework following concepts from model-driven software engineering (Vanthienen, Klotzbücher, and Bruyninckx 2014; Schlegel et al. 2015). SkiROS splits the robot programming process into several layers of abstraction, with two main goals: (i) provide a state-of-the-art architecture for autonomous robot control, and (ii) make high-level robot programming simple and accessible even to non-experts.

Knowledge representation also plays a fundamental role in cognitive robotic systems (Vernon, von Hofsten, and Fadiga 2010), especially with respect to defining world models formalised in an ontology. A prominent example of knowledge processing in robotics is the KnowRob system (Tenorth and Beetz 2012; 2013), which combines knowledge representation and reasoning methods for acquiring and grounding knowledge in physical systems. KnowRob uses a semantic library which facilitates loading and accessing ontologies represented in the Web Ontology Language (OWL). KnowRob uses the ontology to store semantic representations of the world scene in order to reason about object positions in space and time, along with models of the robot hardware and the robot skills. A similar approach is presented in (Björkelund et al. 2012; Stenmark and Malec 2013; Bjørkelund and Edstrom 2011) as part of the Rosetta project, which focuses on how skills should be modelled for industrial assembly tasks. A similar study in (Huckaby 2014) defines a precise taxonomy of skills. However, none of these projects integrate skills into a consistent framework.

Automated planning has also been used for autonomous robot control since the days of Shakey (Nilsson 1984). While early approaches largely separated symbolic planning from other forms of planning like geometric planning, it was recognised that solutions often benefited from a hybrid approach (Cambon, Alami, and Gravot 2009). Recently, robot task planning has become an active research area, with approaches taken from diverse areas such as sampling-based motion planning (Plaku and Hager 2010; Barry 2013), integration of symbolic planning with robot-level processes (Dornhege et al. 2009), and probabilistic back-chaining (Kaelbling and Lozano-Pérez 2013).

A typical approach to robot task planning is to evaluate symbolic actions in a forward manner, sampling geometric choices and backtracking on failure. For instance, (Cambon, Alami, and Gravot 2009) use a symbolic planner that follows several heuristics to guide a geometric search. Symbolic and geometric searches are interleaved, with backtracking in both layers, and probabilistic roadmaps created for all combinations of robot manipulators and objects to represent the search space. Approaches like (Eiter et al. 2006; Dornhege et al. 2009; Erdem et al. 2011; Gaschler et al. 2013) add robot-level functions to a symbolic planning problem through an interface that allows external processes to be invoked during high-level planning. Other approaches like (Srivastava et al. 2014) solve scenarios given symbolic explanations for all failures in the geometric search, which are fed back to the symbolic search. Kaelbling and Lozano-Pérez (2013) perform a hierarchical, back-chaining search, combining geometric abstractions at the robot level with belief space planning.

Other approaches that attempt to bridge the gap between high-level and low-level robotics actions include ROSco (Nguyen et al. 2013) and Smach (Bohren and Cousins 2010) which use Hierarchical Finite State Machines as opposed to the planning approach taken in this paper. Approaches that use planning include ROSPlan (Cashmore et al. 2015) and the work of Vaquero et al. (2015). The former requires manual definition of planning domains, while the latter uses a translation approach specific to their application domain. In contrast, SkiROS is designed so that the user can define and modify skills on the fly, and the planning domains built to use these skills will be automatically generated.

Skills and the SkiROS Architecture

This section introduces the system architecture and skills model used in this work. Developing a robot system is, at

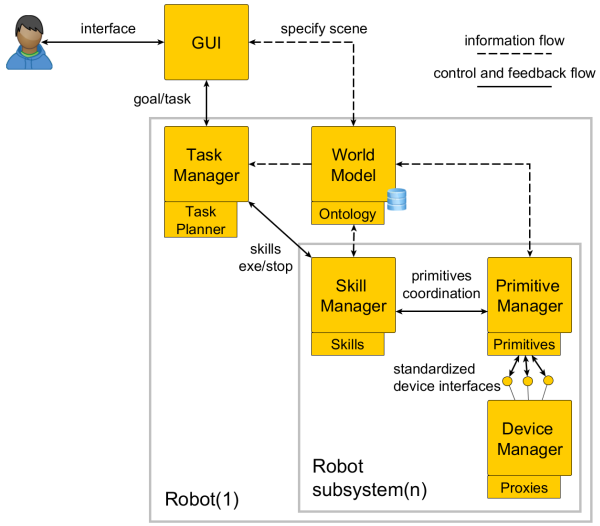


Figure 2: An overview of the SkiROS architecture. The robot presents an external interface from the task manager and the world model, accessed in this case by a GUI. The robot is composed of several subsystems, each one composed of a skill, primitive, and device manager. A skill coordinates the execution of several primitives to realise a world state change. The primitives implement atomic behaviours and interface to the hardware using standardised interfaces.

some level, a software engineering problem. However, robot architectures are distinguished from other software architectures by the special needs of robot systems. The most salient of these requirements, from a system design standpoint, is that robot systems must interact asynchronously and in real time with an uncertain, dynamic environment. At the same time, there is a need to define the tasks the robot can perform in a declarative way, in order to simplify task specification for end users. The skills model attempts to bridge this gap, with skills forming high-level building blocks that can be combined to solve complex tasks, yet containing all the necessary reasoning and control information to be executed by the robot in real time in a dynamic environment.

Skills Model

Robot skills like the ones in (Pedersen et al. 2016) can be thought of as general and robust software constructs that model self-contained, re-occurring operations that a robot might perform. Skills are intended to be designed such that they map easily to simple intuitive tasks. For example, a system might include calibration skills, manipulation skills for operations like picking and placing, as well as driving skills for mobile robots. Skills are implemented by experts to contain the necessary sensing and action operations for self-contained execution on the robot platform.

One benefit of a skills-based system is that non-experts can typically programme a robot task in a straightforward manner by selecting an appropriate skill sequence that results in the desired state changes to the robot's environment. This paper further removes the need for a non-expert user,

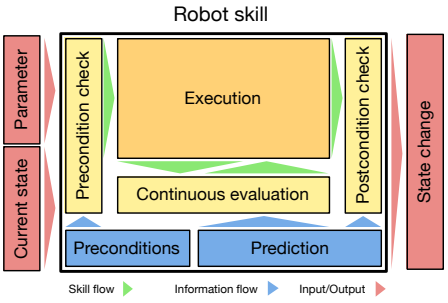


Figure 3: The conceptual model of a SkiROS skill.

and shows how skill sequences can be constructed using planning techniques in a completely automated way.

The skills framework used in this paper, SkiROS (Skills-ROS), is a Robot Operating System (ROS)[1] package implemented as an architecture with several layers of abstraction, as shown in Figure 2. The SkiROS architecture is designed to serve several tasks, including: (i) separating the bottom reactive layers from the top deliberative layers of the robot system, (ii) supporting hardware abstraction, and (iii) modularising robot programming to make it scalable.

The conceptual model of a robot skill is shown in Figure 3. A skill takes as input a set of parameters and a representation of the world state; it outputs a set of state changes. A skill contains both precondition and postcondition checks which monitor the environment, either through sensing or based on the world model. These checks allow the task layer to infer the likely causes of execution failures. For example, a precondition check for a pick skill might be that the item to be picked must be visible to a camera, and a postcondition check might be that the picked item must be in the gripper.

Skills Framework

The SkiROS framework is organised into four layers, each of which is represented by a manager. At the lowest layer is the *device manager*, which loads proxies (drivers which conform to a standard interface) and presents standard interfaces for similar devices (e.g., gripper, arm, camera, etc.). Standardised device interfaces extend the portability of all code, allowing drivers to be changed on the fly, for instance in the case of hardware changes like an updated end-effector. They also greatly simplify the switch between simulation and real-world execution.

The second layer contains the *primitive manager*, which contains motion primitives, software blocks that realise movement controlled with multi-sensor feedback, and services, software blocks that perform a generic computation. The modules are parameterised and loaded in the same way as a skill, but they don't have pre/postconditions and consist only of a *parameters specification* and *execution* part.

The third layer, the *skill manager*, loads skills and provides interfaces to the layer above. It also registers the robot subsystem on the world model, specifying the hardware,

[1] http://www.ros.org/

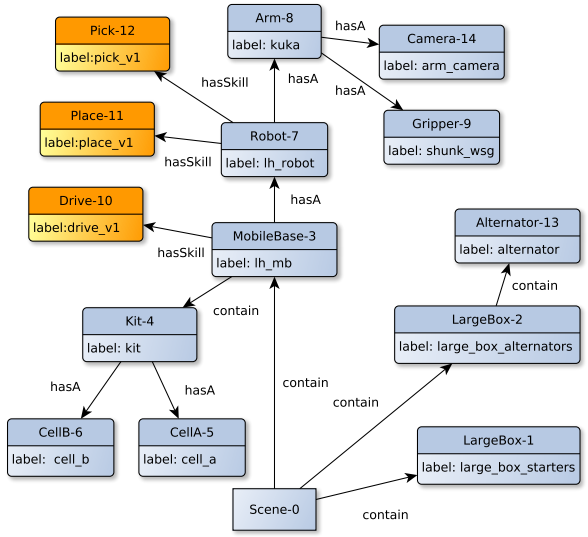


Figure 4: A simplified world model instance with physical (blue) and abstract (orange) objects. All physical objects are connected by a spatial relation in a scene graph structure.

available modules, and available skills. A skill's execution is usually implemented as a finite state machine which coordinates the execution of several parameterised primitives.

Finally, the fourth layer of the architecture is the *task manager* which monitors the presence of subsystems via the world model and acts as a general coordinator. The task manager is the interface for external systems, designed to be connected to a GUI or the manufacturing execution system (MES) of a factory. In this paper, the task manager is extended with an integrated task planner that takes as input a goal and snapshot of the world model and returns a sequence of skills to achieve the current task. The task planner, skills, primitives and proxies are imported as plug-ins using ROS.

World Model

In addition to skills, a key part of SkiROS is the *world model*, which acts as a knowledge integration framework. The world model is a vertical cross-layer component which links all layers together by gathering information from every subsystem at run time, allowing the modules to maintain a shared working memory, and storing the environment and skills information that are used to create the planning domain. In terms of the architecture, the world model can be read and modified from almost every part of the system.

The world state is partially predefined by a human operator in the ontology, partially abstracted from the robot by perception, and completed with the procedural knowledge embedded in the skills and primitives. Each skill manager in the system is responsible for keeping the world model updated with its subsystem information (e.g., hardware, available primitives, skill state, etc.). Similarly, each primitive and skill can extend the scene information with the results of robot operation or sensing. In special cases, the ontology can be extended automatically by the robot, to learn new concepts in a long-term memory (e.g., a new grasping pose).

Knowledge Integration

The core part of the robot's knowledge is organised into an OWL-DL ontology that can be efficiently embedded, edited, and extracted from the system. The SkiROS ontology is comprised of a set of classes C, a set of elements E, a set of relations R, and a set of properties P. Elements are individuals in a particular instance of the world model, for example a box or an alternator. Relations (OWL object properties) are binary relations that link two elements together, while properties (OWL data properties) are binary relations that link an element to a piece of typed data.

Every object in the world is represented in the scene as an *Element* class, which has the properties *type*, *id*, and *label*, along with a flexible list of other potential properties. The id links the Element to the scene, while the type and label categorise it in the ontology. The type is the most important property to this paper as it is used as the object's type in the planning translation. All other data associated with the Element are collected into the properties using a list of variants defined as *parameters*. For example, the Gripper element has the property *is_empty* which is initially set to `true` specifying that the gripper is empty. It is defined as a precondition check in the *Pick* skill to ensure that the gripper does not try to pick up an object while already holding one.

The set R of relations contains a special subset of spatial relations. Apart from the root *scene* element (which has no parent), each element in the world model has a spatial relation to exactly one parent element which ensures that the world model instantiation forms a tree (when only considering edges that represent spatial relations). This is particularly convenient for modelling the objects' spatial transformations. Example spatial relations from the current SkiROS ontology include `RobotAtLocation`, `Holding`, `Contains`, and `Carrying`. We write `RobotAtLocation(robot-1,largebox-1)` and say that `robot-1` is the subject and `largebox-1` is the object. It follows from the properties of this tree structure that an element cannot be both held by one element and contained in another at the same time, or that the robot cannot be in two locations at once.

Figure 4 shows an example instance of a world model. The tree formed by the spatial relations forms the scene graph, a data structure commonly used by modern computer games to arrange the logical and spatial representation of a graphical scene. In this structure, an object's pose is always defined with respect to the parent frame. The skills are connected to robot elements by the non-spatial relation *hasSkill*.

Skills are object-centric models that are parameterised with element types, while their instantiations are expected to link directly to elements of the appropriate type. A condition on a skill must specify either a relation or a property of an element in the world model. If a skill updates the world model by removing a spatial relation property, then it must also state the new subject related to that object as this cannot necessarily be inferred. Type information in skill relations must also be consistent with the world model.

Drive(MobileBase, Container) :

add: RobotAt(Container, MobileBase)

Pick(Gripper, Object, Container) :

pre: empty(Gripper)
pre: robotAt(Container, Robot)
pre: objectAt(Container, Object)
del: empty(Gripper)
add: contains(Gripper, Manipulatable)

Figure 5: Skill definitions in the SkiROS ontology.

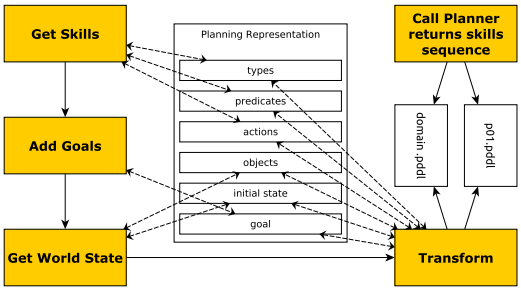


Figure 6: Overview of the task planning process and the creation of its internal planning representation.

Figure 5 shows the parameters, preconditions, and postconditions for the Drive and Pick skills, as defined in SkiROS. The preconditions (similarly, postconditions) are based on the expected and testable requirements of the state of the world prior to execution (similarly, after execution). Relations or properties for which postcondition checks are expected to be false become delete effects, and those expected to be true become add effects. Parameters are formed from the inputs needed for the skill's execution block.

Task Planner

The task planner has three main functions in SkiROS: it creates a PDDL representation of the skills, current state and goals; it calls an external planner to attempt to find a plan for the current goals; and, if a plan is found, it returns a sequence of skills to the task manager. The task planner creates a planning domain (and problem) written in PDDL 1.2 with only the `types` requirement. This means that the output is suitable for use with almost all modern planning systems. In what follows we use the standard definition of STRIPS-like planning actions (Fikes and Nilsson 1971) with *pre*, *add*, and *del* denoting the preconditions, add effects, and delete effects of an action, respectively.

An overview of the task planner is shown in Figure 6. This process is invoked (with the internal planning representation reset) every time the task manager requires a plan for completing the current set of goals. This is either triggered by an operator adding a goal via the SkiROS GUI, or by an external system (e.g., integrated with a factory MES) when SkiROS is deployed as part of a larger system. The central part of Figure 6 shows the task planner's planning library, which contains all the necessary structures to create a PDDL planning problem from the world state, skills, and goals. Specifically, this includes structures for types, predicates (both ground and unground), actions, and (typed) objects. In particular, the main body of a skill as shown in Figure 3 (surrounded by a black box) is not accessible to the task planner. Instead, the task planner uses the information accessible from the world model as shown in Figure 4.

Algorithm 1: Planning Domain Creation

Input : SkiROS World Model (wm), Goals ($goal$)
Output: Initial Planning Representation
`// Parse Skills`
foreach *Skill s : wm* **do**
  types.addAllNewTypes(s)
  predicates.addAllNewPredicates(s)
  actions.addNewAction(s)
`// Add Goal State`
foreach *Goal g : goal* **do**
  goals.add(g);
`// Parse World Model State`
foreach *Predicate p: predicates* **do**
  initState.addAllTrueGroundings(p, wm)
  objects.addAllNewObjects(p,wm)

Initial PDDL Creation

The process of creating the initial planning representation is given in Algorithm 1, which represents the left hand side of Figure 6 and involves three main steps. The first step is to parse the skills that exist in the world model. This involves adding all types and predicates that appear in the skills definitions to the planning library and also creating an action for each skill, which has a direct copy of the preconditions and effects. All relations, properties, and types that do not appear in a skill are therefore not included in the planning library.

The second step instantiates the goal. As goals are specified in the SkiROS GUI using the same predicates and objects in the world model that the skills use, they are simply added to the goal for the planning domain. If the goals contain a predicate or object that has not already been added to the planning library, then the planner returns an error message as no plan can exist given the defined skills.

Finally, the third step of the translation is to obtain the initial state of the planning problem from the current state of the world model. This process iterates over the predicates added in the previous step and queries the world model (through the SkiROS API) to find all ground instances of the predicates that are true. The objects contained in these predicates are added to the planning library as they are found.

The process of iterating through the skills and querying the world model to find true predicates may result in a planning library with less elements and types than in the world model. The omitted data can safely be ignored (and an error given for an incorrect goal) due to the following:

Lemma 1: *Any object with a type that does not appear in a skills definition can never appear in a solution plan.*

Algorithm 2: Planning Domain Refinement

Input : Initial Planning Representation
Output: Final Planning Representation

```
// Add Capabilities
foreach Action a : actions do
    predicates.add(can_a ?robot)
    a.pre.add(can_a ?robot)
    foreach Robot r : hasSkill(a, r) do
        initState.add(can_a r)
// Spatial Relation Constraints
foreach Action a do
    foreach Spatial Relation S(o, s) ∈ a.add do
        if ∄S ∈ a.pre AND ∄S ∈ a.del then
            s.params.add(x, s.type)
            s.pre.add(S(o, x))
            s.del.add(S(o, x))
        else if ∄S ∈ a.pre then
            s.pre.add(S(o, s))
        else if ∄S ∈ a.del then
            s.del.add(S(o, s))
```

Proof Sketch: It is impossible to change the truth value of a predicate that does not appear in an action's effects, and the truth value of a predicate that does not appear in any action's preconditions can never be required for a change in state.

Domain Modification

The right hand side of Figure 6 deals with encoding the properties of the world model in the planning domain. The first part makes sure that skills are only usable by the correct elements, by querying the **hasSkill** relation from the world model. For each action, a new predicate `(can_a ?robot)` is added to the planning representation. This predicate is added as true in the initial state for each robot that can perform a particular skill and is invariant. An additional precondition `(can_a ?robot)` is added to each action to ensure that it can only be instantiated to the correct robots. If the `Robot` parameter is missing from the skill definition then this is added to the action parameters at this time.

The second part of the transformation step adds any preconditions and delete effects that are necessary to maintain the tree structure of the spatial relations in the world model. SkiROS contains methods for internally updating its world model so that it remains consistent, and these methods need to be included in the planning domain as they are not always made explicit in the skill definitions. For instance, referring to the skills in Figure 5, the Drive skill only contains a single predicate which specifies the new location of the robot. This is because the input for the execution block of the drive skill is only the goal location to which the robot has to move. The drive action must then be modified so that `robotAt` is true of only one grounding for the robot performing the drive skill, so the old instantiation must be found (it becomes a precondition) and added as a delete effect of the action.

The algorithm performs the steps in the previous example in a general manner that works for all spatial relations. It iterates over the skills in the planning library and checks each spatial relation in the add effects. If no corresponding spatial relation exists, in either the preconditions or delete effects of the action (i.e., no predicate with matching relation and subject as in the case of the drive skill), then a new predicate of the same spatial relation and the same object, but a new subject variable, is created and added to the preconditions and delete effects of the action. If a related spatial relation exists in just one of the preconditions and delete effects then it is added (with the same subject) to the other.

```
(:action drive
   :param (?R - Agent ?T - Location
     * ?preT - Location)
   :pre (and
     * (can_drive ?R)
     * (RobotAtLocation ?R ?preT))
   :eff (and
     * (not (RobotAtLocation ?R ?preT))
       (RobotAtLocation ?R ?T)))

(:action pick
   :param ( ?A - Arm ?C - Location ?G - Gripper
       ?O - Manipulatable ?R - Agent)
   :pre (and
       (EmptyHanded ?G)
       (RobotAtLocation ?R ?C)
       (ObjectAtLocation ?C ?O)
     * (can_pick ?R))
   :eff (and
       (not (EmptyHanded ?G))
     * (not (ObjectAtLocation ?C ?O))
       (Holding ?G ?O)))
```

Figure 7: The actions from Figure 5 after translation to PDDL. The asterisked lines are added by the translation.

Figure 7 shows the skills from Figure 5 after translation to PDDL. Note that in terms of implementation, the parameter added to the drive skill is removed when returning the parameterised skill to the task manager. The translation adds three new preconditions and two new delete effects over the two actions. The following lemma shows that these additions ensure the world model's tree structure is maintained:

Lemma 2: *Performing an action created by the task planner on a problem whose spatial relations form a tree will result in a state in which the spatial relations still form a tree.*

Proof Sketch: All that needs to be shown is that any deletion of a spatial relation property inserts it elsewhere with the same object (and therefore moves the whole subtree), and that every addition has a corresponding deletion. The former is a constraint on the skill definition. For the latter, every time a new spatial relation appears in the add effects then, by construction of the algorithm, a spatial relation with the same subject must appear in the delete effects. This spatial relation must match the only occurrence of that object in a spatial relation in the current state otherwise the action could not be performed as this must exist (again by construction) as a precondition to the action.

Once the translation is complete, the planning problem is written to domain and problem files in PDDL for use with an external planner. The planner's output (a sequence of instan-

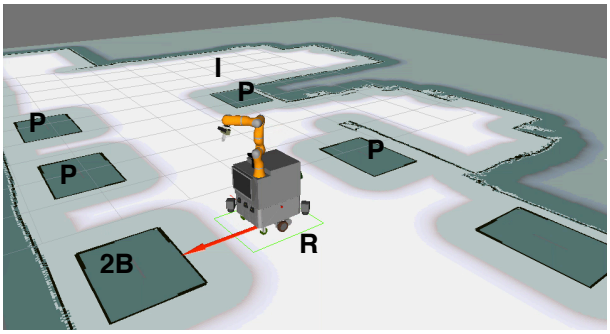


Figure 8: A visualisation of the simulated environment, showing an excerpt of the navigation map containing the robot idle location (I), a number of pallets (P), a pallet with two boxes of parts (2B), and the robot (R).

tiated actions) is parsed and converted back to parameterised skills to be sent to the task manager for robot execution.

Experiments

Figure 8 depicts the simulated environment used for testing.[2] In this setup, kits can contain six different parts: *engine support*, *thermal shield*, *compressor*, *tube*, *alternator*, and *starter*. Four parts (compressor, tube, alternator, and starter) are located in individual pallets; the two remaining parts (engine support and thermal shield) are located in smaller adjacent boxes on a single pallet.

In order to specify this setup in the robot's world model within SkiROS, the 2D poses (including orientation) of each pallet, and the parts contained within, need to be specified manually. The poses are defined in the world model coordinate system, and the transform between this frame and the robot navigation map frame is known. This information can be extracted automatically from the manufacturing execution system (MES) in real-world deployment; for obvious reasons, this is not possible for the simulated environment. The kit that is mounted on the robot is specified as a set of coordinate frames, with one parent frame defining the kit with respect to the robot, and the rest defining the individual compartments in the kit with respect to the kit itself. A part type is associated with each compartment in the kit.

The experiments used a simulated version of a mobile manipulator with an articulated robot arm mounted on a mobile platform. The robot arm was equipped with a 2-finger parallel gripper and an RGB-D camera mounted on the gripper. The execution of skills was simulated using the ROS interfaces employed by the real hardware drivers that were replaced. For example, the MoveIt arm motion planner (that outputs joint trajectories for an arm) and the navigation software (that outputs velocity commands to a mobile base) were not modified. Figure 4 shows a (slightly) simplified version of the spatial relations and components used in the experiment, with picking, placing, and driving skills.

[2]Visualisation was performed using rviz, a 3D tool for ROS (`http://wiki.ros.org/rviz`).

```
drive mobbase-2 loc-1 lbox-10
pick lbox-10 gripper-6 t_shield robot-3
drive mobbase-2 lbox-10 lbox-9
place grip-6 t_shield celld-19 kit-15 robot-3
pick lbox-9 gripper-6 starter robot-3
place grip-6 starter cellb-17 kit-15 robot-3
```

Figure 9: The plan found for the goal of placing two parts (thermal shield and starter) into a kit.

It is not possible, nor necessary, to simulate sensor information in this experiment. However, an inherent part of the skills is that they perform the necessary sensing operations to complete the skill. For this reason, a simple simulated object detection primitive is added, that places an object in the world model that is immediately in front of the robot.

This level of simulation makes it possible to visualise the robot system as it performs the skills, using the same skills that would be running on a real robot system. In terms of Figure 2, only the device layer and a single primitive (i.e., the object detection primitive) is simulated. Therefore the system uses, and is completely integrated in, a complete version of SkiROS, with the same skills as a real robot.

For the experiments, the FastDownward planner (Helmert 2006) was used, with A* search and the landmark-cut heuristic. Since the planning problems created by the translation process did not test the limits of the external planner, and are solved in less than a second (including world model querying, extraction, and translation), there was no benefit in comparing different planners. Instead, any state-of-the-art planner that supports the required features could be used.

Results

The first experiment tested a two skill setup in which only the robotic arm and the pick and place skills were used. The robot was placed in front of a pallet with two smaller boxes containing thermal shields and engine supports. The system was tested with the goal that one of each of the two different types of parts must be placed in the robot's kit. The extraction of the planning domain, and the planning itself, was completed in 0.6s, resulting in a plan with four skills that was successfully executed in 78s.[3] The experiment was then rerun with the goal specifying parts that were not in the vicinity of the robot. In this case, the task planner correctly returned that no plan could be found.

The second experiment introduced a second robot subsystem, the mobile base (which carries the robotic arm) and its associated drive skill. In this case, the task planner automatically included the drive skill in its planning domain. With this addition, a plan could be found for the previously unsolvable problem in which the parts are inaccessible without the ability to drive between locations. When the goal was specified to build a complete kit with six parts, with the robot finishing at an idle location, the PDDL extraction and planning took 0.9s. The resulting plan (with 18 skills) executed

[3]Planning, motion planning, simulation, SkiROS, and visualisation ran on a 2011 laptop with an i7@2.7GHz processor.

correctly in 371s. Figure 9 shows the plan for the goal of filling a kit with two parts (a thermal shield and a starter).

Overall, these experiments demonstrated that the task planning process is able to work as an integrated component in the SkiROS system and that the process is robust enough to find correct plans when different subsets of the currently implemented skills are enabled. The experiments also showed that the planning time is not a significant bottleneck (less than one second in all cases), especially when compared to execution time for these types of tasks.

Conclusions and Future Work

This paper presented a fully implemented software framework for deploying autonomous robot systems in an industrial setting. The system uses a skills model, called SkiROS, to bridge the gap between low-level robot control and high-level planning. Skills are declared explicitly and passed to a task planner which automatically generates the PDDL planning domain. The resulting system was shown to operate successfully in a simulated factory environment and has also been tested in a real-world factory setting. From an end-user perspective, the robot is programmed to perform new tasks by specifying goal conditions; new skills are added by specifying constraints on the world model with no explicit knowledge of planning required.

Work is progressing to test more skill implementations and further explore the relationship between skills and planning. Failure handling will be improved by extending the interaction of the planner with the task manager, to allow for replanning in the case of unsatisfied pre/postconditions and execution failures. To optimise cycle time, the assumption of sequential skill execution will be relaxed, allowing parallel skill execution from temporal plans.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant no. 610917 (STAMINA, `stamina-robot.eu`).

References

Arkin, R. C. 1998. *Behavior-based Robotics*. Cambridge, MA, USA: MIT Press, 1st edition.

Barry, J. L. 2013. *Manipulation with Diverse Actions*. Ph.D. Dissertation, MIT, USA.

Bensalem, S., and Gallien, M. 2009. Toward a more dependable software architecture for autonomous robots. *IEEE Robotics and Automation Magazine* 1–11.

Bjørkelund, A., and Edstrom, L. 2011. On the integration of skilled robot motions for productivity in manufacturing. In *IEEE International Symposium on Assembly in Manufacturing*.

Björkelund, A.; Malec, J.; Nilsson, K.; Nugues, P.; and Bruyninckx, H. 2012. Knowledge for Intelligent Industrial Robots. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*.

Bohren, J., and Cousins, S. 2010. The smach high-level executive [ros news]. *Robotics & Automation Magazine, IEEE* 17(4):18–20.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *Journal of Artificial Intelligence Research* 2(1):14–23.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28(1):104–126.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the robot operating system. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, 1–6.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *The Semantic Web: Research and Applications*, 273–287.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems* 56(11):980–991.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. Dissertation, Yale University, USA.

Gaschler, A.; Petrick, R. P. A.; Giuliani, M.; Rickert, M.; and Knoll, A. 2013. KVP: A knowledge of volumes approach to robot task planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, 202–208.

Gat, E. 1998. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Huckaby, J. 2014. *Knowledge Transfer in Robot Manipulation Tasks*. PhD thesis, Georgia Institute of Technology, USA.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32(9–10):1194–1227.

Kortenkamp, D., and Simmons, R. 2008. Robotic systems architectures and programming. In *Springer Handbook of Robotics*. Springer. 187–206.

Madsen, O.; Bøgh, S.; Schou, C.; Andersen, R. S.; Damgaard, J. S.; Pedersen, M. R.; and Krüger, V. 2015. In-

tegration of mobile manipulators in an industrial production. *Industrial Robot: An International Journal* 42(1):11–18.

Magnenat, S. 2010. *Software integration in mobile robotics, a science to scale up machine intelligence*. PhD thesis, École polytechnique fédérale de Lausanne, Switzerland.

Nguyen, H.; Ciocarlie, M.; Hsiao, K.; and Kemp, C. C. 2013. Ros commander (rosco): Behavior creation for home robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 467–474. IEEE.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International.

Pedersen, M., and Krüger, V. 2015. Automated planning of industrial logistics on a skill-equipped robot. In *Workshop on Task Planning for Intelligent Robots in Service and Manufacturing at IROS 2015*.

Pedersen, M. R.; Nalpantidis, L.; Andersen, R. S.; Schou, C.; Bøgh, S.; Krüger, V.; and Madsen, O. 2016. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing* 37:282–291.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, 5002–5008.

Rovida, F., and Krüger, V. 2015. Design and development of a software architecture for autonomous mobile manipulators in industrial environments. In *IEEE International Conference on Industrial Technology (ICIT)*.

Schlegel, C.; Lotz, A.; Lutz, M.; Stampfer, D.; and Vicente-Chicote, C. 2015. Model-Driven Software Systems Engineering in Robotics: Covering the Complete Life-Cycle of a Robot. *it - Information Technology* 57(2):85–98.

Srivastava, S.; Fang, E.; Lorenzo, R.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Stenmark, M., and Malec, J. 2013. Knowledge-based industrial robotics. *Scandinavian Conference on Artificial Intelligence*.

Tenorth, M., and Beetz, M. a. 2012. Knowledge Processing for Autonomous Robot Control. *Proceedings of the AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*.

Tenorth, M., and Beetz, M. 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research* 32(5):566–590.

Vanthienen, D.; Klotzbücher, M.; and Bruyninckx, H. 2014. The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming. *Journal of Software Engineering for Robotics* 5(1):17–35.

Vaquero, T.; Mohamed, S. C.; Nejat, G.; and Beck, J. C. 2015. The implementation of a planning and scheduling architecture for multiple robots assisting multiple users in a retirement home setting. In *AAAI Workshop on Artificial Intelligence Applied to Assistive Technologies and Smart Environments*.

Vernon, D.; von Hofsten, C.; and Fadiga, L. 2010. *A Roadmap for Cognitive Development in Humanoid Robots*. Springer.

Autonomous Search by a Socially Assistive Robot in a Residential Care Environment for Multiple Elderly Users Using Group Activity Preferences

Sharaf C. Mohamed and Goldie Nejat

Autonomous Systems and Biomechatronics Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON M5S 3G8, Canada

sharaf.mohamed@mail.utoronto.ca; nejat@mie.utoronto.ca

Abstract

In this paper, a novel activity-based search approach is proposed for a socially assistive robot to autonomously search for and find multiple elderly users who are living in a residential care facility within a required time frame. The search maximizes the number of users found within the time frame by utilizing a predictive user model. The model uniquely considers the spatial-temporal preferences of each user, as well as the spatial-temporal preferences of other users sharing the environment and the activities that the users are performing throughout the day in the environment. Since multiple users living in the same environment can have similar activity patterns, these activity patterns can be used to learn mutual preferences (i.e., group influence on individual users). Furthermore, the activities themselves motivate specific regions in the environment where they are performed. As the robot is deployed in a human-centered environment, it autonomously conducts the search for the multiple users while following human etiquette rules. Numerous simulated experiments conducted on a floor of our collaborating residential care facility are presented comparing our proposed activity-based search approach to an approach that only considers the spatial-temporal preferences of the user of interest. Our results show that our proposed search approach is able to find more users across varying time frames and varying user set sizes by considering both the specific user's spatial-temporal activity preferences and mutual preferences among all users. Experiments implemented with a physical robot verify the feasibility of the search being implemented in a real environment.

Introduction

Socially assistive robots are envisioned to co-exist in human centered environments with the elderly in order to provide needed assistance and support. These environments include retirement homes, long-term care facilities, as well as private homes. Our research focuses on the development of autonomous socially assistive robots capable of providing both social and cognitive assistance to the elderly in order to assist with activities of daily living (Louie et al., 2014a; Louie et al., 2014b; Vaquero et al., 2015). The motivation behind the development of such autonomous robots is to effectively integrate robots that can complete a variety of assistive tasks throughout the day in the aforementioned environments, considering the preferences and behavior patterns of the multiple users they interact with.

To-date there have been a handful of user studies that have shown the positive benefits of socially assistive robots providing either social or cognitive stimulation to elderly users living in retirement or long-term care facilities (Oida et al., 2011; Khosla et al., 2012; Hamada et al., 2008; McColl, Louie and Nejat, 2013; Montemerlo et al., 2002). However, the majority of these robots do not move around in their environments, i.e., from one room to another, and only use local motion within a defined activity space to implement activity specific actions, i.e., on top of a table. An exception is the Pearl robot (Montemerlo et al., 2002) which is able to guide users to appointments. Furthermore, none of these robots (including Pearl) actively search in their environment for users who are not collocated with the robot.

The ability to autonomously search for users of interest within their environments increases the number of activities for which socially assistive robots can provide assistance. For example, in a residential care environment (e.g. retirement home or long-term care facility), a robot can find and provide assistance to one elderly user in his/her private room, and then find another group of elderly users in the recreation room to remind them of an upcoming activity.

In this paper, we present a novel user search approach in order to allow a socially assistive robot to autonomously search for and find, within a time frame, a specific set of multiple users living within a residential care facility in order to provide assistance. Our approach models the search as a travelling thief problem, and generates a search plan using dynamic programming and the shortest path in a directed acyclic graph (DAG). The main contribution of our unique

search approach is that in addition to the user's spatial-temporal preferences, it also directly considers the following within the predictive user model: 1) the spatial-temporal preferences of other users sharing the environment, and 2) the activity patterns of all the users throughout the day. The activity patterns associate tasks, start times and end times with the spatial-temporal preferences. Multiple users sharing the same environment can have similar activity patterns and these patterns can be used to learn mutual preferences among the users. Namely, activity patterns of the group can influence individual activity preferences. Our approach models the influence of others on the future activity patterns of one another. In addition, research into human activity recognition has shown that activities can motivate locations (Sheng et al., 2015), hence, herein the robot explicitly considers the activity locations when generating search plans. Furthermore, the robot follows human etiquette rules when searching for users in the intended human-centered environment.

Robot Search of People

A significant amount of research has addressed the problem of using a robot to detect and track people in structured indoor environments while they are collocated in the same region as the robot, e.g. (Montemerlo et al., 2002; Bennewitz, Burgard and Thrun, 2003; Hu, Ma and Dai, 2009; Park and Kuipers, 2013). However, to the authors' knowledge, only a handful of researchers have focused on robotic search of people within indoor environments, when the robot has to actively search the environment to find people that are not collocated with the robot. The literature in this area can be categorized into the search for 1) static or 2) dynamic persons.

Robotic Search for Static Persons

In (Elinas, Hoey and Little, 2003), the HOMER robot used a person finding technique in order to be able to deliver a message to a recipient who was assumed to be at a static location. The robot used a location likelihood function for finding the person at a location in the map of the environment. The robot would visit the closest locations to it first as the best locations and then continue to visit all other possible locations. The search stopped if HOMER found the recipient or all possible locations had been searched. In (Volkhardt and Gross, 2013), a set of navigation points were given to a robot to search for a static person at a particular location in a three-room home environment. The robot iteratively moved to the closest navigation point using a combination of adaptive Monte Carlo planning, E* path planning and motion control via a dynamic window approach. In (Lau, Huang and Dissanayake, 2005), a dynamic programming approach was used for the search of multiple static targets in an environment. The search strategy focused on minimizing the expected average time for detecting each target. The expected proportion of targets in each region was used as information about the targets.

Robotic Search for Dynamic Persons

In (Tipaldi and Arras, 2011), a robot used a spatial affordance map which modelled the spatial-temporal behavior of people. A finite-horizon Markov Decision Process (MDP) was used to generate robot paths which maximize the probability of an encountering a person in an office environment. With respect to finding specific users of interest, in (Bovbel and Nejat, 2014), the Casper robot utilized a predictive search strategy to find a dynamic user in the home environment. The search approach prioritized search locations based on previous patterns of user locations and behaviors. In particular, a Hidden Markov Model was used to determine the relationship between a user's location and behavior at a specific time. While the robot searched the environment, prior location beliefs and the resulting search prioritization were updated based on new observations of the environment. In (Hollinger et al., 2009), a Partially Observable Markov Decision Process (POMDP) model was used to generate search plans using spatial-temporal information of dynamic targets. The objective was to minimize the robot's expected capture time when finding first respondents in an indoor search and rescue scenario. Lastly, our own previous work was the first to focus on the problem of a robot finding a specific set of multiple dynamic people in an indoor environment. The approach used spatial-temporal likelihood functions for the people which considered their daily schedules, the layout of the environment and direct observations by the robot (Schwenk et al., 2014). An MDP was then used to maximize the number of residents found within a deadline time.

Our proposed robot search approach is unique compared to the other existing robotic search approaches as it aims to consider all of the following: 1) finding a specific group of multiple dynamic users, 2) not only considering the location and time preferences for each user separately to define a search plan, but explicitly considering: a) the mutual preferences of all users sharing the environment, and b) the activity patterns for all these users (the influence of group preferences), and 3) taking into account etiquette conventions of space sharing and interaction during robot search. When implementing any of the aforementioned robot search approaches presented in the literature, etiquette rules with respect to robot speed, position and orientation in shared spaces with humans, in particular older adults, were not considered. Guidelines for these have been presented in (Walters et al., 2005; Woods et al., 2006) and have been incorporated into our proposed robotic search approach.

Robot Search Problem

Our search problem consists of a mobile socially assistive robot that needs to find a set of multiple users in an indoor residential environment during time frames throughout the duration of a day in order to interact with them, i.e., provide or obtain information from users. The problem details are presented below.

Activities: An activity, A_m, represents a task a user is undertaking. In the residential care facility A_m can be categorized as the following common activities during the day: sleeping, reading, listening to music, playing games, watching television and eating.

Residential Environment: The search takes place in an environment divided into regions $\mathcal{R}$ (R_1, R_2, …, R_I). Each region is defined by physical boundaries, i.e. walls, and is of a square or rectangular shape. If a region is physically accessible from another region, without passing through a third region, i.e. contains a shared doorway, these regions are said to be neighbors. $t_{R_i,R_{i'}}$ is the time the mobile robot will take to travel from R_i to $R_{i'}$, $\forall \{i \in R, i' \in R\}$, determined using the shortest path between regions, where each subsequent region in a path must be a neighbor of the previous region. Each region R_i is categorized based on the rooms/regions that are present in a residential care facility: a resident's own private bedroom, a hallway, entrance lobby, recreation room, dining room, kitchen, garden, nurses' station and robot charging station. Depending on the region category, different activities can be performed by the residents in a region.

Users: The users U (U_1, U_2, …, U_N) are all residents living in the shared environment. Users perform activities in regions during time intervals, throughout each day.

Search query: The search query S specifies the set of users U' (U'_1, U'_2, …, U'_Z) that the robot needs to find between a certain time frame, t_{start} to t_{end}. This time frame is made-up of discrete time intervals T (T_1, T_2, …, T_Ω) of fixed length $t_{interval}$, where $T_{j,k}$ refers to a subset of T containing all the time intervals, T_ω, between T_j and T_k.

Mobile Robot: The mobile socially assistive robot will execute the search query *S* in the environment. The specific actions that the robot can execute are: *travel between regions*, *search a region*, *interact with user* and *wait*. The navigation plan, P^*, is the sequence of robot actions to implement for the overall search process. The *travel between regions* action allows the robot to move from one region to another. The *search a region action* has the robot execute a local search procedure in the specified region. The *interact with user* action allows the robot to approach the user and perform the necessary interaction. The *wait* action stops the robot from moving. The robot will navigate the environment at a speed of v=1.35 m/s, defined to be the average comfortable walking pace for older adults ages 50-70 years old (Bohannon, 1997). Since the robot is intended for a human-centred environment, the robot implements its actions adhering to social etiquette rules.

Social Etiquette Rules: P^* is subject to safety constraints, interaction constraints, and navigation constraints. Safety constraints are prioritized over interaction constraints, which are prioritized over navigation constraints.

A) Safety Constraints: To ensure the safety of the robot, as well as users, the following etiquette rule is considered:

1) The robot will delay the planning of P^* and execute the *wait* action if any user enters within a distance, d_s, to the robot, where $d_s = 0.5m$.

B) Interacting with a user: while moving to interact with users, the following etiquette rules are considered in order of priority:

1) Direct interactions with users will take place at a distance of d_i between the robot and user, where $1.25m \leq d_i \leq 2.5m$ based on previous studies of comfort levels of people (Woods et al., 2006).

2) When approaching a user of interest, the robot will approach him/her from the frontal direction with an offset of θ to either the right or left, where $30^\circ \leq \theta \leq 70^\circ$ as defined in (Woods et al., 2006).

C) Travelling between regions: while moving between regions, the following etiquette rules are considered in order of priority:

1) When following a user, the robot will be at a distance d_f, where $1.2m \leq d_f \leq 3.6m$. This range represents the acceptable conversation distances between robots and humans (Walters et al., 2005).

2) The robot will navigate on the right side of a hallway.

3) When navigating, the robot will be at a distance of d_n from users in the environment with whom it is not interacting, where $d_n \geq 2.5m$.

When planning P^*, if a path that follows all of the above etiquettes cannot be found, the constraints will be removed, one at a time, from lowest priority to highest priority, until a path is found. If a path is not found and only the safety constraint remains, the robot will execute the *wait* action until it is safe to continue.

Autonomous Activity-Based Search for Multiple Users

The objective of the proposed search approach is to determine the search plan that the robot should execute in order to maximize the expected number of users of interest found within a given time frame.

Environment and User Information Gathering

Since our objective is to develop an autonomous robot capable of searching an environment finding users of interest, we also consider that the robot should be able to autonomously obtain any previous knowledge needed to achieve its goal. Therefore, prior to implementing the proposed search approach, the robot gathers information about the environment and the users. Firstly, the robot autonomously explores the environment in order to generate a 2D map detailing the topology of the environment. The map is generated using the simultaneous localization and mapping (SLAM) algorithm called gmapping (Grisetti, Stachniss and Burgard, 2007) using laser scans provided from an onboard laser range finder and odometry information. This map can then be used by the robot during user search.

The robot also partakes in an observation stage, defined as the user information gathering stage, in which it follows a user around to identity activity patterns. The robot meets the user at his/her private room in the morning, and follows

him/her for the remainder of the day. To follow the user, the robot uses silhouette detection, identifying head and shoulders in the contours which are classified using Support Vector Machines. At the end of the day the robot returns to its charging station. The robot will follow each user in the environment, for a given number of days, recording the user's activity patterns. These activity patterns will be used to generate a database of activity patterns for all users D (D_1, D_2, …, D_Y). D_y represents a single activity pattern which is defined as a tuple containing: a user, a region, an activity, and a time interval.

Proposed Robot Search Approach

Based on the search query provided to the robot, the proposed search approach defines a search plan, α, which includes: 1) a subset of regions to search and the order in which to search these regions, $R = \{R_1 \dots, R_H\}$, 2) the length of time to search each region, $T^{search} = \{t_1^{search}, \dots, t_H^{search}\}$, and 3) the navigation plan P^* which consists of the explicit actions of the robot. The search plan α is achieved using a reward-based technique. Namely, the robot receives a reward for searching a particular region for a specified amount of time, given a time interval for the search, $W\left(R_h, t_h^{search} | \, T_{j,k}\right)$. The goal is to choose R and T^{search} in order to maximize the total reward acquired over the total search for the specified time frame:

$$maximize \sum_{h \in R} \left(W\left(R_h, t_h^{search} | \, T_{j,k}(t_j(h), t_k(h))\right)\right), \quad (1a)$$

where,

$$t_j(h) = \sum_{g=1}^{h-1} (t_{R_{g-1}}^{R_g} + t_g^{search}) + t_{R_{h-1}}^{R_h}, \text{ and} \quad (1b)$$

$$t_k(h) = t_j(h) + t_h^{search},$$

and

$$\sum_{h \in R} \left(t_{R_{h-1}}^{R_h} + t_h^{search_h}\right) \leq t_{end} - t_{start}. \quad (1c)$$

R_g, where $g = 0$, is defined as the previous location of the robot before a search query is executed. $t_j(h)$ is defined as the search start time for R_h and is determined by summing over the prior travel and search times. $t_k(h)$ is the search end time for R_h. $T_{j,k}(t_j(h), t_k(h))$ is defined as the shortest time interval that contains both $t_j(h)$ and $t_k(h)$.

Predictive User Model

To model the probability of user, U_n, occupying region, R_i, during time interval, $T_{j,k}$, the following probability function is defined to consider all ongoing activities:

$$P[R_i | \, T_{j,k}, U_n] = \sum_{m \in A} P[R_i, A_m | T_{j,k}, U_n], \quad (2)$$

where,

$$P[R_i, A_m | T_{j,k}, U_n] =$$

$$P[A_m | \, T_{j,k}, U_n] \, P[R_i | \, T_{j,k}, U_n, A_m].$$

For each user, U_n, the following two probabilities are assigned: 1) $\mathrm{P}[A_m | T_{j,k}, U_n]$, which represents the probability that U_n performs activity A_m during $T_{j,k}$, and 2) $\mathrm{P}[R_i | \, T_{j,k}, U_n, A_m]$, which represents the probability that U_n occupies R_i during $T_{j,k}$, given that U_n performs activity A_m during $T_{j,k}$. Both these probabilities are inferred using a weighted sum between the four likelihood functions $L_{A_m}[T_{j,k}, U_n]$, $L_{A_m}[T_{j,k}]$ $L_{R_i}[A_m, T_{j,k}, U_n]$, and $L_{R_i}[A_m, T_{j,k}]$ discussed below.

User Preferences: $L_{A_m}[T_{j,k}, U_n]$ represents the likelihood of observing U_n undertaking A_m during $T_{j,k}$within the activity patterns D. $L_{R_i}[A_m, T_{j,k}, U_n]$ represents the likelihood of observing U_n undertaking A_m during $T_{j,k}$ in region R_i within the activity patterns D.

Group Preferences: $L_{A_m}[T_{j,k}]$ represents the likelihood of observing any user undertaking A_m during $T_{j,k}$within the activity patterns D. $L_{R_i}[A_m, T_{j,k}]$ represents the likelihood of observing any user undertaking A_m during $T_{j,k}$ in region R_i within the activity patterns D.

Combining the Likelihoods: We combine both the user preference and the group preference for A_m in order to determine $\mathrm{P}[A_m | T_{j,k}, U_n]$:

$$\mathrm{P}[A_m | T_{j,k}, U_n] = \frac{L_{A_m}[T_{j,k}, U_n] + L_{A_m}[U_n] \, L_{A_m}[T_{j,k}]}{2}, \quad (3)$$

where the weighting $L_{A_m}[U_n]$ applied to the group preference for A_m represents the likelihood of observing U_n undertaking A_m within the activity patterns D. We use this weighting since a user who frequently performs A_m at other time intervals, is likely to perform A_m at the time interval $T_{j,k}$ if other users are also performing A_m (i.e., group influence on an individual). For example, a user who normally plays cards in the afternoon, is now playing cards in the morning since a number of other users in the group play cards then.

We combine both the user preference and the group preference for R_i in order to determine $P[R_i | \, T_{j,k}, U_n, A_m]$:

$$P[R_i | \, T_{j,k}, U_n, A_m] = \frac{L_{R_i}[A_m, T_{j,k}, U_n] + L_{R_i}[A_m, U_n] L_{R_i}[A_m, T_{j,k}]}{2}, \quad (4)$$

where the weighting $L_{R_i}[A_m, U_n]$ applied to the group preference for R_i represents the likelihood of observing U_n undertaking A_m in region R_i within the activity patterns D. We use this weighting as a user who frequently occupies R_i when performing A_m in other time intervals, is likely to occupy R_i to perform A_m during $T_{j,k}$ if others in the group are doing the same. For example, a user who usually listens to music in the garden, now joins other users in the group in the recreational room to listen to music with them.

Rewards

We use a combination of region rewards and search coverage to determine the total reward acquired for searching region, R_i, during time interval $T_{j,k}$ for a duration of t_i^{search}:

$$W\left(R_i | \, T_{j,k}, t_i^{search}\right) = W\left(R_i | \, T_{j,k}\right) \mathrm{Sr}(R_i, t_i^{search}). \quad (5)$$

Region Rewards: Each region, R_i, is assigned a reward with respect to the probability of finding users of interest in that region during time interval, $T_{j,k}$:

$$W(R_i|\, T_{j,k}) = \sum_{z \in U'} \mathrm{P}[R_i|\, T_{j,k}, U'_z]. \tag{6}$$

Search Coverage: Given an allotted time, t_i^{search}, the percentage of reward that can be acquired from a region, R_i is based on the total area of the region R_i^{area}:

$$\mathrm{Sr}(R_i, t_i^{search}) = f(t_i^{search}, R_i^{area}), \tag{7}$$

where $\mathrm{Sr}(R_i, t_i^{search})$ is a function of the local search technique used. For example, when using a grid-based minimal cost tour (coverage) technique the function can be linear. Our approach can use a variety of local search techniques to search within a particular region, one of which we have implement in the simulated experiments and robot experiments sections below for completeness.

Obtaining the Search Plan

To determine the search plan α, R and T^{search} are selected in order to maximize Eq. (1a), without violating Eq. (1c). This maximization problem is called the travelling thief problem (Bonyadi, Michalewicz and Barone, 2013) which consists of two interdependent, *np*-hard, sub-problems: 1) the knapsack problem, and 2) the travelling salesman problem. An approximate solution to the travelling thief problem can be found by solving each sub problem independently in an iterative manner.

Knapsack Problem: The knapsack problem is defined as determining the optimal search plan that maximizes the total reward found over the search, without accounting for the travel time between regions:

$$maximize \sum_{h \in R} \left(W\left(R_h, t_h^{search} |\, T_{j,k}(t_j(h), t_k(h))\right)\right), \tag{8a}$$

where,

$$t_j(h) = \sum_{g=1}^{h-1} (t_g^{search}), \text{ and} \tag{8b}$$

$$t_k(h) = t_j(h) + t_h^{search},$$

and

$$\sum_{h \in R} (t_h^{search_h}) \leq t_{end} - t_{start}. \tag{8c}$$

In order to solve the knapsack problem we select a subset of regions of $\mathcal{R}$ defined to be $r_\omega = (r_{\omega,1}, r_{\omega,2}, \dots, r_{\omega,B})$ in order to search at each time interval, T_ω, and a set of search times $T_\omega^{search} = (t_{\omega,1}^{search}, \dots, t_{\omega,B}^{search})$ for each r_ω. r_ω and T_ω^{search} are assigned such that the total reward acquired over the overall time frame of the search is maximized, without exceeding the allotted search time $t_{\mathrm{MAX\_elapsed}}$ during each time interval T_ω:

$$maximize \sum_{\omega \in T} \sum_{b \in r_\omega} \left(W\left(r_{\omega,b}, t_{\omega,b}^{search} | T_\omega\right)\right), \tag{9a}$$

where,

$$\sum_{b \in r_\omega} \left(t_{\omega,b}^{search_h}\right) \leq t_{\mathrm{MAX\_elapsed}}, \ \forall \omega \in T. \tag{9b}$$

Assigning increments in minutes for $t_{\omega,b}^{search}$ allows for the enumeration of all possible combinations of search times. Therefore, this allows us to solve the special case of the knapsack problem using dynamic programming and shortest path algorithm for a directed acyclic graph (i.e., shortest DAG). Figure 1 shows the DAG for our problem. The nodes $N^i_{t_1^{elapsed},\dots,t_\Omega^{elapsed}}$ represent each possible pairing of region and elapsed times, and the edges which represent the time spent searching R_i in each time interval, $E^i_{t_i^{search,1},\dots,t_i^{search,\Omega}}$, are used to connect two nodes, i.e. $N^i_{t_1^{elapsed},\dots,t_\Omega^{elapsed}}$ to $N^{i+1}_{t_1^{elapsed}+t_i^{search,1},\dots,t_\Omega^{elapsed}+t_i^{search,\Omega}}$. R_{I+1} is the terminal state needed to solve for the shortest DAG. A reward is assigned to each edge based on the search time of each interval:

$$E^i_{t_i^{search,1},\dots,t_i^{search,\Omega}} = \sum_{\omega \in T} W\left(\mathrm{R}_i, t_i^{search,\omega} \middle| T_\omega\right). \tag{10}$$

To determine the optimal solution to the knapsack problem we find the path with the maximum reward from $N^1_{0,\dots,0}$ to $N^{I+1}_{t_{MAX\_elapsed},\dots,t_{MAX\_elapsed}}$:

$$maximize \sum_{i \in \mathcal{R}} E^i_{t_i^{search,1},\dots,t_i^{search,\Omega}}. \tag{11}$$

The resulting plan is a set of unordered subsets r_ω. Each r_ω contains R_i if E^i has $t_i^{search,\omega} \neq 0$ and has a corresponding search time $t_i^{search,\omega}$.

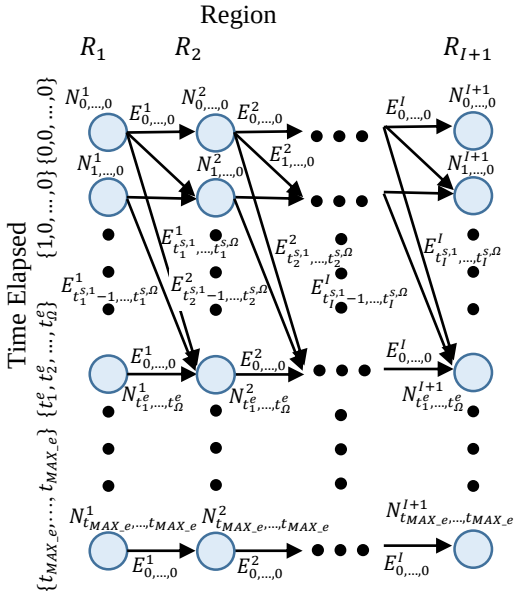


Figure 1: directed acyclic graph for the knapsack problem, where t_ω^e represents $t_\omega^{elapsed}$, $t_i^{s,\omega}$ represents $t_i^{search,\omega}$, and t_{MAX_e} represents $t_{MAX\_elapsed}$.

Travelling Salesman Problem: The travelling salesman problem is used to determine the order in which to search the subsets r_ω. Every permutation of r_ω is considered and the permutation that results in the shortest travel time is chosen:

$$minimize \sum_{\omega \in T} \sum_{b \in r_\omega} \left(t_{r_{\omega,b-1}}^{r_{\omega,b}} \right). \tag{12}$$

$r_{\omega,b}$, where $\omega = 0$ and $b = 0$, is defined as the current region that the robot is in when a search query is executed. $r_{\omega,b}$, where $\omega \neq 0$ and $b = 0$, is defined as the last region of the previous time interval, $r_{\omega-1,B}$.

If the search time and travel time are longer than $t_{interval}$, we implement an iterative knapsack problem with a reduced value for $t_{\text{MAX\_elapsed}}$ until the below condition is satisfied:

$$\sum_{b \in r_\omega} \left(t^{r_{\omega,b}}_{r_{\omega,b-1}} + t^{search}_{\omega,b} \right) \leq t_{interval}, \ \forall \omega \in T. \quad (13)$$

Simulated Experiments

We developed a simulator in order to test the performance of our proposed activity-based multi-user search approach. The simulation environment consists of the layout of a floor of our collaborating residential care facility where twenty-six users are sharing the environment, Fig. 2. We define a single day to start at 7 am and end at 9pm. Both the common rooms and private rooms are used throughout the day by the elderly residents to perform different activities. There are defined time periods for the activities. Namely, eating a meal occurs three times a day for a duration of an hour. Recreational activities (reading, listening to music, playing games and watching TV) can occur in either the morning or afternoon. Sleeping can occur after meals as well as early morning and night. Both recreational activities and sleeping can have varying lengths of time with a minimum time of 15 minutes. An example of the robot finding a user of interest in the recreational room in the simulated environment is shown in Fig. 3.

User Simulated Activity Patterns

To simulate the activity patterns of the users for the simulations, each activity has: 1) a lower bound on its minimum duration and an upper bound on its maximum duration, 2) a set of regions in which the activity can occur, and 3) the time periods over which it can occur. Each user is given a random preference for each activity, a random minimum and maximum duration within the aforementioned bounds for which the activity will last, and random preferences of all the possible regions the activity can be performed in. For each user, a set of activities is generated for the duration of the day based on the random preferences defined above. After completing an activity, the user's preference for the activity is reduced by half for the remainder of the day. The duration of and the region where the activity takes place are chosen randomly from the bounds and preferences given for each user.

User Information Gathering

The robot follows each user in the environment mapped out by the robot for 1 day and stores all their activity patterns in *D*.

Local Search within a Region

For the local search of each region, R_i is subdivided into grid cells, $C_i = \{C_{i,1}, C_{i,2}, \ldots, C_{i,Q}\}$, of size 3m x 3m. The size of each cell corresponds to the robot's sensory range for detecting and identifying users. The time it takes for the robot to move between two adjacent cells C_i and C_j is a function of the robot's speed v=1.35 m/s, and defined to be 2.22s. The time it takes to search a cell to identify if U'_z is occupying the cell is defined to be 10s. To search a region, the robot implements a minimum cost tour of the grid cells using a heuristic approach. Depending on the allotted search time t_i^{search}, the robot searches as many cells along the planned tour as possible. Utilizing this local search technique, the search coverage defined in Eq. (7) becomes:

$$\text{Sr}\left(R_i, t_i^{search}\right) = \frac{1}{12.22} \frac{t_i^{search}}{R_i^{area}}, \quad (14)$$

where R_i^{area} is in m^2, and t_i^{search} is in seconds.

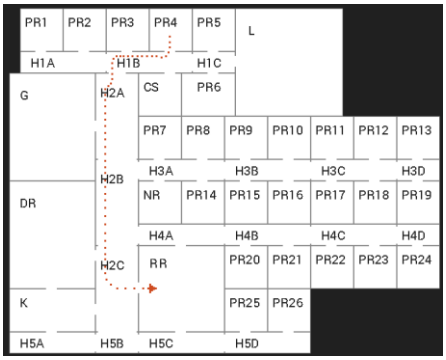


Figure 2: The map of the floor layout of the simulated residential care facility with the following room categories and sizes: Lobby (L) is 15m x 15m; Private Room (PR), Nurses' Station (NR) and Robot Charging Station (CS) are 6m x 6m; Hallway (H)- each section of H is 3m x 12m; Garden (G) and Dining Room (DR) are 15m x 12m; Recreational Room (RR) is 12m x 12m; and Kitchen (K) is 6m x 12m. Red dotted line represents an example robot path from region to region.

Figure 3: Robot (orange) finding a user of interest in the recreational room. The user is sitting on a red chair reading.

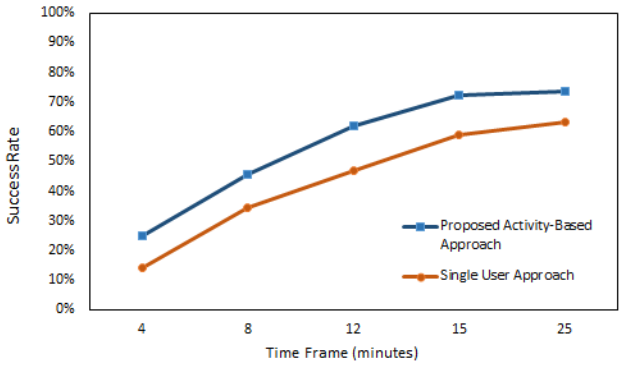
Implementation

We compared the performance of our activity-based search approach which considers both single user and group prefer-

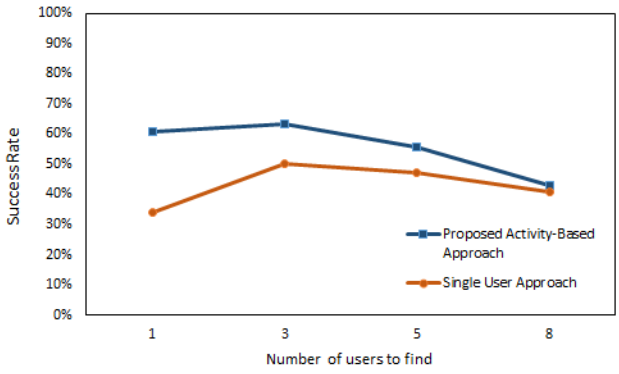
ences to an approach that considers only single user preferences to determine a search plan. Both search approaches were tested with 25 different search queries S with U'=1,3,5,8 and with $t_{end} - t_{start}$= 4,8,12,15,25 minutes. The start times were selected to be both in the morning and afternoon (i.e., 10 am, 11:30 am, 1pm, 2:30pm and 4:30 pm) and the time frame is divided into three discrete time intervals (T_1,T_2,T_3). As the focus of this work is on the global search between regions, both approaches used the heuristic local search approach to obtain the minimum cost tour of the grid cells when searching within a region. The performance metrics used for the comparison were: 1) the success rate for the different time frames, and 2) the success rate for the set of users to find.

Results

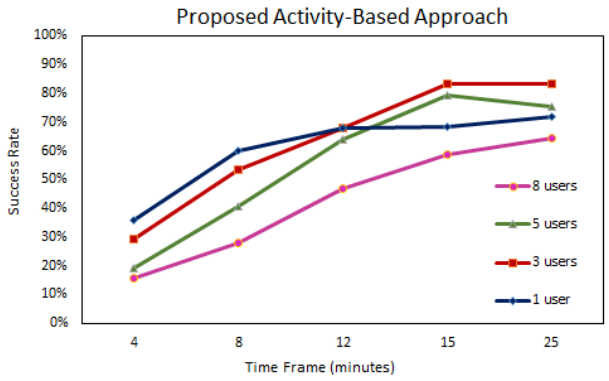
The results for both approaches are presented in Figure 4. Fig. 4(a) shows that, overall, our proposed approach has higher success rates for finding the users of interest across all search time frames, and Fig. 4(b) shows that, overall, our proposed approach is able to find more users across varying user set sizes. The detailed results for each user set and time frame for our activity-based proposed approach are also presented in Fig. 4(c).



(a)



(b)



(c)

Figure 4: (a) success rate with respect to search time frame averaged across number of users, (b) success rate with respect to number of users to find averaged across time frames, (c) success rate for each user set with respect to the time frame for our proposed activity-based approach.

Robot Experiments

Experiments with our socially assistive robot were conducted to verify the use of our search approach in a real environment finding real people. The robot has a differential drive mobile base with a human-like upper torso. The robot obtains information about its environment and users within the environment using a variety of sensors including a laser range finder, encoders, two 2D cameras and a 3D camera. The robot is also capable of autonomously navigating in an indoor environment using the ROS navfn planner (ROS, 2014). User detection is achieved through the detection of head and shoulder contours using depth and 2D images from the 3D sensor (within a range of 0.5m-4.5m). User identification is achieved using the OKAO™ Vision software library (Omron, 2007) by finding facial features within 2D images provided from one of the robot's 2D cameras. More details on the robot can be found in (Louie et al., 2014b).

The experimental environment was a scaled down version of the floor layout above consisting of 8 regions defined to be a combination of private and common rooms connected by hallways. The size of each region ranged from 50 to 144 m^2. Five experiments were conducted in order to have the robot find five users of interest within 15 minutes and provide them with a reminder of an upcoming activity. We investigated the ability of the robot to successfully execute the overall search plan α in order to find these users.

In each experiment, the robot was able to follow the generated search plan by navigating from region to region and searching the specified locations within each region. Examples of the robot's actions and etiquette behaviors are presented in Fig. 5. The overall success rate for finding users was 88%. Table 1 represents the success rate for each of the five different scenarios with respect to the regions the users were located in. The lower success rate for experiment 1 was

a result of the fact that since the physical robot used a conservative obstacle avoidance technique which took it longer to navigate through the regions that were more cluttered, it did not have enough time to search all regions in its plan. Namely, there needs to be a balance between how conservative the physical robot's movements are in the real environment with respect to the search time frame. In general, the success rate was higher for the experiments than the simulations due to the smaller environment and room sizes.

Table 1: Experimental Results

Experiment #	User Distribution within the 8 Regions	Success Rate of Finding Users
1	[1,1,1,1,0,1,0,0]	60%
2	[1,2,0,0,1,1,0,0]	80%
3	[0,2,2,0,1,0,0,0]	100%
4	[3,0,0,2,0,0,0,0]	100%
5	[0,0,5,0,0,0,0,0]	100%

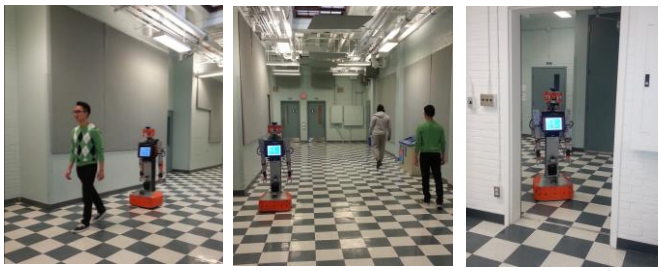


Figure 5: Example Robot Actions and Etiquette Behavior (from top left to bottom right): robot following a person to gain information; robot travelling between regions; robot entering a region; robot searching a region; and robot interacting with user.

Conclusion

In this paper, we address the problem of an autonomous socially assistive robot searching for a set of multiple elderly users who are living in a residential care facility within a required time frame, in order for the robot to provide them with assistance. We present the development of a novel activity-based search approach which maximizes the expected number of dynamic users of interest that need to be found within a given time frame during the day. We model the problem as a travelling thief problem. Our approach is unique in that it considers the spatial-temporal preferences of each user, the spatial-temporal preferences of other users sharing the environment (group preferences) and the activities that the users are performing throughout the day in the environment in order to determine a global search plan. Furthermore, robot etiquette behaviors for human-centered environments are considered when executing a plan. We have compared our activity-based search approach to a common approach that only considers the spatial-temporal preferences of the users of interest in simulated experiments conducted on a floor of a residential care facility. The results show that our proposed search approach has higher success rates in finding users of interest across varying time frames and varying user set sizes. The experiments with a physical robot are promising and show the feasibility of the search approach being implemented in a real environment.

Acknowledgements

The authors would like to thank NSERC (Natural Sciences & Engineering Research Council of Canada), the Canada Research Chairs Program, and Dr. Robot Inc. The authors would also like to thank Seung (Rio) Hong for his assistance with this work.

References

Bennewitz, M.; Burgard, W; and Thrun, S. 2003. Adapting Navigation Strategies Using Motion Patterns of People. IEEE International Conference on Robotics and Automation (ICRA), 2(1): 2000-2005.

Bohannon, R. W. 1997. Comfortable and maximum walking speed of adults aged 20-79 years: reference values and determinants. Journal of Age and Aging, 26(1): 15-19.

Bonyadi, M. R.; Michalewicz, Z.; and Barone, L. 2013. The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. IEEE Congress on Evolutionary Computation (CEC), 1037-1044.

Bovbel, P.; and Nejat, G. 2014. Casper: An Assistive Kitchen Robot to Promote Aging in Place. Journal of Medical Devices-Transactions of the ASME, 8(3): 1-2.

Elinas, P.; Hoey, J.; and Little, J. J. 2003. HOMER: Human Oriented Messenger Robot. AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments, 45-51.

Grisetti, G.; Stachniss, C.; and Burgard, W. 2007. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. IEEE Transactions on Robotics, 23(1): 34-46.

Hamada, T.; Okubo, H.; Inoue, K.; Maruyama, J.; Onari, H.; Kagawa, Y.; and Hashimoto, T. 2008. Robot therapy as for recreation for elderly people with dementia-Game recreation using a pet-type robot. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 174-179.

Hollinger, G.; Singh, S.; Djugash, J.; and Kehagias, A. 2009. Efficient multi-robot search for a moving target. The International Journal of Robotics Research, 28(2): 201-219.

Hu, C.; Ma, X.; and Dai, X. 2009. Reliable Person Following Approach for Mobile Robot in Indoor Environment. International Conference on Machine Learning and Cybernetics (ICMLC), 3(1): 1815-1821.

Khosla, R.; Chu, M. T.; Kachouie, R.; Yamada, K.; and Yamaguchi, T. 2012. Embodying care in Matilda: an affective communication robot for the elderly in Australia. ACM SIGHIT International Health Informatics Symposium, 295-304.

Lau, H.; Huang, S.; and Dissanayake, G. 2005. Optimal search for multiple targets in a built environment. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 3740-3745.

Louie, G.; Li, J.; Vaquero, T.; Nejat, G. 2014a. A focus group study on the design considerations and impressions of a socially assistive robot for long-term care. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 237-242.

Louie, G.; Vaquero, T.; Nejat, G.; and Beck, J. C. 2014b. An Autonomous Assistive Robot for Planning, Scheduling and Facilitating Multi-User Activities. IEEE International Conference on Robotics and Automation (ICRA), 5292-5298.

McColl, D.; Louie, G.; and Nejat, G. 2013. Brian 2.1: A Socially Assistive Robot for the Elderly and Cognitively Impaired. IEEE Robotics & Automation Magazine, 20(1): 74-83.

Montemerlo, M.; Pineau, J.; Roy, N.; Thrun, S.; and Verma, V. 2002. Experiences with a Mobile Robotic Guide for the Elderly. AAAI National Conference on Artificial Intelligence, 587-592.

Oida, Y.; Kanoh, M.; Inagaki, M.; Konagaya, Y.; and Kimura, K. 2011. Development of a robot-assisted activity program for elderly people incorporating reading aloud and arithmetic calculation. Asian Perspectives and Evidence on Health Promotion and Education, 67-77.

Omron. 2007. OKAO Vision. Available: http://www.omron.com/r_d/coretech/vision/okao.html

Park, J. J.; and Kuipers, B. 2013. Autonomous Person Pacing and Following with Model Predictive Equilibrium Point Control. IEEE Conference on Robotics and Automation (ICRA), 1060-1067.

ROS "Navfn" 2014. Available: http://wiki.ros.org/navfn.

Schwenk, M.; Vaquero, T.; Nejat, G.; and Arras, K.O. 2014. Schedule-Based Robotic Search for Multiple Residents in a Retirement Home Environment. AAAI Conference on Artificial Intelligence, 2571-2577.

Sheng, W.; Du, J.; Cheng, Q.; Li, G.; Zhu, C.; Liu, M.; and Xu, G. 2015. Robot semantic mapping through human activity recognition: A wearable sensing and computing approach. Journal of Robotics and Automation, 68(1): 47-58.

Tipaldi, G. D.; and Arras, K. O. 2011. I want my coffee hot! Learning to find people under spatio-temporal constraints. IEEE International Conference on Robotics and Automation (ICRA), 1217-1222.

Vaquero, T.; Mohamed, S.; Nejat, G.; and Beck, J. C. 2015. The Implementation of a Planning and Scheduling Architecture for Multiple Robots Assisting Multiple Users in a Retirement Home Setting. AAAI Workshop on Artificial Intelligence Applied to Assistive Technologies and Smart Environments, 1-6.

Volkhardt, M.; and Gross, H. M. 2013. Finding People in Home Environments with a Mobile Robot. European Conference on Mobile Robots (ECMR), 282-287.

Walters, M. L.; Dautenhahn, K.; Koay, K. L.; Kaouri, C.; Boekhorst, R.; Nehaniv, C.; Werry, I.; and Lee, D. 2005. Close encounters: Spatial distances between people and a robot of mechanistic appearance. IEEE-RAS International Conference on Humanoid Robots (Humanoids), 450-455.

Woods, S. N.; Walters M. L.; Koay, K. L.; and Dautenhahn, K. 2006. Methodological Issues in HRI: A Comparison of Live and Video-Based Methods in Robot to Human Approach Direction Trials. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 51-58.

Instinct: A Biologically Inspired Reactive Planner for Embedded Environments

Robert H. Wortham, Swen E. Gaudl, Joanna J. Bryson
Dept of Computer Science, University of Bath
Claverton Down, Bath, BA2 7AY, UK

Abstract

The Instinct Planner is a new biologically inspired reactive planner, based on an established behaviour based robotics methodology and its reactive planner component — the POSH planner implementation. It includes several significant enhancements that facilitate plan design and runtime debugging. It has been specifically designed for low power processors and has a tiny memory footprint. Written in C++, it runs efficiently on both ARDUINO (ATMEL AVR) and MICROSOFT VC++ environments and has been deployed within a low cost maker robot to study AI Transparency. Plans may be authored using a variety of tools including a promising visual design language, currently implemented using the DIA drawing package.

INTRODUCTION

From the 1950's through to the 1980's the study of embodied AI assumed a cognitive symbolic planning model for robotic systems — SMPA (Sense Model Plan Act) — the most well known example of this being the Shakey robot project (Nilsson, 1984). In this model the world is first sensed and a model of the world is constructed within the AI. Based on this model and the objectives of the AI, a plan is constructed to achieve the goals of the robot. Only then does the robot act. Although this idea seemed logical and initially attractive, it was found to be quite inadequate for complex, real world environments. Generally the world cannot be fully modelled until the robot plan is underway, since sensing the world requires moving through it. Also, where environments change faster than the rate at which the robot can complete its SMPA cycle, the planning simply cannot keep up. Brooks (1995) provides a more comprehensive history, which are not repeated here.

In the 1990's Rodney Brooks and others (Breazeal and Scassellati, 2002) introduced the then radical idea that it was possible to have intelligence without representation (Brooks, 1991). Brooks developed his subsumption architecture as a pattern for the design of intelligent embodied systems that have no internal representation of their environment, and minimal internal state. These autonomous agents could traverse difficult terrain on insect-like legs, appear to interact socially with humans through shared attention and gaze tracking, and in many ways appeared to posses behaviours similar to that observed in animals. However, the systems produced by Brooks and his colleagues could only respond immediately to stimuli from the world. They had no means of focusing attention on a specific goal or of executing complex sequences of actions to achieve more complex behaviours. The original restrictions imposed by Brooks' subsumption architecture were subsequently relaxed with later augmentations such as timers, effectively beginning the transition to systems that used internal state in addition to sensory input in order to determine behaviour.

Following in-depth studies of animals such as gulls in their natural environment, ideas of how animals perform action selection were originally formulated by Nico Tinbergen and other early ethologists (Tinbergen, 1951; Tinbergen and Falkus, 1970). Reactions are based on pre-determined drives and competences, but depend also on the internal state of the organism (Bryson, 2000). Bryson (2001) harnessed these ideas to achieve a major step forwards with the POSH (Parallel Ordered Slipstack Hierarchy) reactive planner and the BOD (Behaviour Oriented Design) methodology, both of which are strongly biologically inspired. It is important to understand the rationale behind biologically inspired reactive planning. It is based on the idea that biological organisms constantly sense the world, and generally react quickly to sensory input, based on a hierarchical set of behaviours structured as Drives, Competences and Action Patterns. Their reactive plan uses a combination of sensory inputs and internal priorities to determine which plan elements to execute, ultimately resulting in the execution of leaf nodes in the plan, which in turn execute real world actions. For further reading see Gurney, Prescott, and Redgrave (1998), Prescott, Bryson, and Seth (2007) and Seth (2007).

At run-time, the reactive plan itself is essentially fixed. Various slower reacting systems may also be used to modify priorities or other parameters within the plan. These slower reacting systems might be compared with emotional or endocrinal states in nature that similarly affect reactive priorities (Gaudl and Bryson, 2014). Similarly the perception of senses can be affected by the internal state of the plan, an example being the latching (or hysteresis) associated with sensing (Rohlfshagen and Bryson, 2010).

In nature, the reactive plan is subject to possible learning that may change the plan parameters or even modify the

structure of the plan itself as new skills and behaviours are learned. This learning may take place ontogenetically, i.e. within the lifetime of an individual, or phylogenetically, by the process of natural selection, across the lifetimes of many individuals. Bryson's BOD approach suggests that humans provide most of the necessary learning in order to improve the plan over time, in place of natural selection. However, Gaudl (manuscript in preparation) successfully uses genetic algorithms to automate part of this learning process, albeit within a computer game simulation.

A reactive plan is re-evaluated on every plan cycle, usually many times every second, and this requires that the inquiries from the planner to the senses and the invocation of actions should respond quickly. This enables the reactive plan to respond quickly to changes in the external environment, whilst the plan hierarchy allows for complex sequences of behaviours to be executed. Applying these ideas to robots we can see that for senses, this might imply some caching of sense data. For actions, it also implies that long running tasks (relative to the rate of plan execution), need to not only return success or failure, but also another status to indicate that the action is still in progress and the plan must wait at its current execution step before moving on to its next step. The action may be executing on another thread, or may just be being sampled when the call to the action is made. This is implementation specific and does not affect the functioning of the planner itself. If re-invoked before it completes, the action immediately returns an In-Progress response. In this way, longer running action invocations do not block the planner from responding to other stimuli that may still change the focus of attention by, for example, releasing another higher priority Drive.

Each call to the planner within the overall scheduling loop of the robot starts a new plan cycle. In this context an action may be a simple primitive, or may be part of a more complex pre-defined behaviour module, such as a mapping or trajectory calculation subsystem. It is important to note that the BOD methodology does not predicate that all intelligence is concentrated within the planner. Whilst the planner drives action selection, considerable complexity can still exist in sensory, actuation and other probabilistic or state based subsystems within the overall agent (Bryson, 2001).

The computer games industry has advanced the use of AI for the simulation of non player characters (Lim, Baumgarten, and Colton, 2010). Behaviour trees are similarly hierarchical to POSH plans, but have additional elements that more easily allow logical operations such as AND, OR, XOR and NOT to be included within the plan. For example it is possible for a goal to be reached by successfully executing only one of a number of behaviours, trying each in turn until one is successful. Bryson's original design of POSH does not easily allow for this kind of plan structure.

Behaviour trees are in turn simplifications of Hierarchical Task Network (or HTN) planners (Ghallab, Nau, and Traverso, 2004). Like POSH, HTN planners are able to create and run plans that contain recursive loops, meaning that they can represent any computable algorithm. An interesting parallel can be drawn here with Complexity theory. Holland (2014) argues that a Complex Agent System (CAS) is often characterized by the fact that it can be decomposed into a set of hierarchical layers, with each layer being Turing complete. For a biological entity Holland identifies these layers as existing at the levels of DNA, organelle, cell, organ, organism and social levels. For an artificial agent we can identify these layers as computer hardware, operating system, application programming language, reactive planner, plan, agent and social levels. Thus we can argue that to create an artificial agent truly capable of emergent implicit behaviour, we should strive to ensure that the Planner on which its behaviour depends should be Turing complete, particularly allowing looping and recursion.

THE INSTINCT PLANNER

The Instinct Planner is a reactive planner based on Bryson's POSH (Bryson, 2008, 2001). It includes several enhancements taken from more recent papers extending POSH (Rohlfshagen and Bryson, 2010; Gaudl and Bryson, 2014), together with some ideas from other planning approaches, notably Behaviour Trees (BT — Lim, Baumgarten, and Colton, 2010). A POSH plan consists of a *Drive Collection (DC)* containing one or more *Drives*. Each *Drive (D)* has a priority and a releaser. When the *Drive* is released as a result of sensory input, a hierarchical plan of *Competences*, *Action Patterns* and *Actions* follows.

- *Action Pattern (AP)*: Action patterns are used to reduce the computational complexity of search within the plan space and to allow a coordinated fixed sequential execution of a set of elements. An action pattern—$AP = [\alpha_0, \ldots, \alpha_k]$—is an ordered set of Actions that does not use internal precondition or additional perceptual information. It provides the simplest plan structure in POSH and allows for the optimised execution of behaviours. An example would be an agent that always shouts and moves its hand upwards when touching an hot object. In this case, there is no need for an additional check between the two Action primitives if the agent should always behave in that manner. APs execute all child elements before completing.

- *Competence (C)*: Competences form the core part of POSH plans. A competence $C = [c_0, \ldots, c_j]$ is a self-contained basic reactive plan (BRP) where $c_b = [\pi, \rho, \alpha, \eta], b \in [0, \ldots, j]$ are tuples containing π, ρ, α and η: the priority, precondition, child node of C and maximum number of retries. The priority determines which of the child elements to execute, selecting the one with the highest priority first. The precondition is a concatenated set of senses that either release or inhibit the child node α. The child node itself can be another Competence or an Action or Action Pattern. To allow for noisy environments a child node can fail a number of times, specified using η, before the Competence ignores the child node for remaining time within the current cycle. A Competence sequentially executes its hierarchically organised child-nodes where the highest priority node is the competence goal. A Competence fails if no child can execute or if an executed child fails.

- *Drive (D)*: A Drive—$D = [\pi, \rho, \alpha, A, v]$—allows for the design and pursuit of a specific behaviour as it maintains its execution state. The Drive Collection determines which Drive receives attention based on each Drive's π, the associated priority of a Drive. ρ is the *releaser*, a set of preconditions using senses to determine if the drive should be pursued. α is either an Action, Action Pattern or a Competence and A is the root link to the Drive Collection. The last parameter v specifies the execution frequency, allowing POSH to limit the rate at which the Drive can be executed. This allows for coarse grain concurrency of Drive execution (see below).
- *Drive Collection (DC)*: The Drive Collection—DC—is the root node of the plan—$DC = [g, D_0, \ldots, D_i]$. It contains a set of Drives $D_a, a \in [0 \ldots i]$ and is responsible for giving attention to the highest priority Drive. To allow the agent to shift and focus attention, only one Drive can be active in any given cycle. Due to the parallel hierarchical structure, Drives and their sub-trees can be in different states of execution. This allows for cooperative multitasking and a quasi-parallel pursuit of multiple behaviours at the Drive Collection level.

For a full description of the POSH reactive planner see Bryson (2001).

Enhancements and Innovations

The Instinct Planner includes a full implementation of what we term *Drive Execution Optimisation (DEO)*. DEO avoids a full search of the plan tree at every plan cycle which would be expensive. It also maintains focus on the task at hand. This corresponds loosely to the function of consciousness attention seen in nature (Bryson, 2011). A form of this was in Bryson's original POSH, but has not been fully implemented in subsequent versions. The Drive, Competence and Action Pattern elements each contain a *Runtime Element ID*. These variables are fundamental to the plan operation. Initially they do not point to any plan element. However, when a Drive is released the plan is traversed to the point where either an Action is executed, or the plan fails at some point in the hierarchy. If the plan element is not yet completed it returns a status of In Progress and the IDs of the last successful steps in the plan are stored in Runtime Element ID in the Drive, Competence and Action Pattern elements. If an action or other sub element of the plan returns success, then the next step in the plan is stored. On the next cycle of the drive, the plan hierarchy is traversed again but continues from where it got to last plan cycle, guided by the Runtime Element IDs. A check is made that the releasers are still activated (meaning that the plan steps are still valid for execution), and then the plan steps are executed. If a real world action fails, or the releaser check fails, then the Runtime Element ID is once again cleared. During execution of an Action Pattern (a relatively quick sequence of actions), sensory input is temporarily ignored immediately above the level of the Action Pattern. This more closely corresponds to the reflex behaviour seen in nature. Once the system has started to act, then it continues until the Action Pattern completes, or an element in the Action Pattern explicitly fails. Action Patterns are therefore not designed to include Actions with long running primitive behaviours.

In addition to these smaller changes there are three major innovations in the Instinct Planner that increase the range of plan design options available to developers:

- Firstly, the idea of runtime alteration of drive priority. This implementation closely follows the RAMP model of Gaudl and Bryson (2014) which in turn is biologically inspired, based on spreading activation in neural networks. Within the Instinct Planner we term this *Dynamic Drive Reprioritisation (DDR)*. DDR is useful to modify the priority of drives based on more slowly changing stimuli, either external or internal. For example, a recharge battery drive might be used to direct a robot back to its charging station when the battery level becomes low. Normally this drive might have a medium priority, such that if only low priority drives are active then it will return when its battery becomes discharged to say 50%. However, if there are constantly high priority drives active, then the battery level might reach a critical level of say 10%. At that point the recharge battery drive must take highest priority. A comparison can be drawn here with the need for an animal to consume food. Once it is starving the drive to eat assumes a much higher priority than when the animal experiences normal levels of hunger. For example, it will take more risks to eat, rather than flee from predators.
- Secondly, the idea of flexible latching provides for a more dynamic form of sense hysteresis, based not only on plan configuration, but also the runtime focus of the plan. This implementation follows the work of Rohlfshagen and Bryson (2010). Within the Instinct Planner we term it *Flexible Sense Hysteresis (FSH)*. This hysteresis primarily allows for noise from sensors and from the world, but Rohlfshagens paper also has some basis in biology to avoid dithering by prolonging behaviours once they have begun. If the Drive is interrupted by one of a higher priority, then when the sense is again checked, it will be the Sense Flex Latch Hysteresis that will be applied, rather than the Sense Hysteresis.
- Thirdly, we enhance the Competences within the plan, such that it is possible to group a number of competence steps by giving them the same priority. We refer to this as a *priority group*. Items within a group have no defined order. Within a priority group, the Competence itself can specify whether the items must all be successfully executed for the Competence to be successful (the AND behaviour), or whether only one item need be successful (the OR behaviour). In the case of the OR behaviour, several items within the group may be attempted and may fail, before one succeeds. At this point the Competence will then move on to higher priority items during subsequent plan cycles. A Competence can have any number of priority groups within it, but all are constrained to be either AND or OR, based on the configuration of the Competence itself. This single enhancement, whilst sounding straightforward, increases the complexity of the planner code significantly, but allows for much more compact plans, with a richer level of functionality achievable within a single

Competence than was provided with the earlier POSH implementations.

Multi Platform

The Instinct planner itself is able to run both within MICROSOFT VISUAL C++ and the ARDUINO development environments (Arduino, 2016) as a C++ library. The ARDUINO uses the ATMEL AVR C++ COMPILER (Atmel Corporation, 2016a) with the AVR LIBC library (Atmel Corporation, 2016b) — a standards based implementation of `gcc` and `libc`. This arrangement harnesses the power of the VISUAL C++ Integrated Development Environment (IDE) and debugger, hugely increasing productivity when developing for the ARDUINO platform, which has no debugger and only a rudimentary IDE. We have a complete implementation of the Instinct Planner on an ARDUINO based robot named R5. The robot runs using various test plans, see figure 1. Due to the very compact memory architecture of Instinct, the planner is able to store plans with up to 255 elements within the very limited 8KB memory (RAM) available on the ARDUINO MEGA (ATMEL AVR ATMEGA2560 MICROCONTROLLER). The 255 element limitation arises from the use of a single byte to store plan element IDs within the ARDUINO environment.

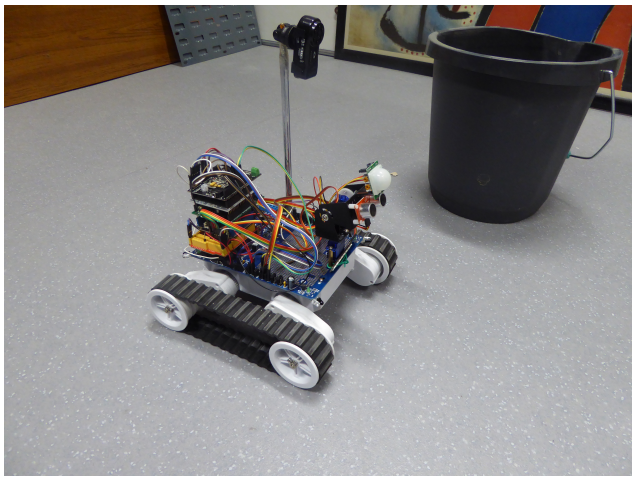


Figure 1: The R5 ARDUINO based Maker Robot in a laboratory test environment. The camera mounted on the robot is used to record robot activity, but is not used by the robot itself.

The robot itself has active infrared and ultrasonic distance sensors, a head capable of scanning its environment, a passive infrared (PIR) sensor to assist in the detection of humans interacting with it, and proprioceptive sensing of odometry (distance travelled) and drive motor current. It has simple and more complex underlying behaviours that can be invoked by the planner, such as the ability to turn in the direction of the most clear pathway ahead, or to use its head to scan for the presence of a human. It also has a multicoloured headlight that may be used for signalling to humans around it. Finally, it has an electronically erasable programmable read only memory (EEPROM) that permanently stores both the robot's configuration parameters and the Instinct plan. This leverages the planner's ability to serialise plans as a byte stream, and then reconstitute the plan from that stream at startup.

Memory Management

In order to produce a planner that operates effectively in an environment with severely limited working memory resources (RAM), considerable design effort has been applied to the memory management architecture within the planner. There are 6 separate memory buffers, each holding fixed record length elements for each element type in the plan — Drives, Competences, Competence Elements, Action Patterns, Action Pattern Elements and Actions. An instance of Instinct has a single Drive Collection — the root of the plan.

Within each plan element, individual bytes are divided into bit fields for boolean values, and the data is normalised across elements to avoid variable length records. This means, for example, that Competence Elements hold the ID of their parent Competence, but the Competence itself does not hold the IDs of each of its child Competence Elements. At runtime a search must be carried out to identify which Competence Elements belong to a given Competence. Thus, the planner sacrifices some search time in return for a considerably more compact memory representation. Fortunately this search is very fast, since the Competence Elements are stored within a single memory buffer with fixed length records. Testing shows the time taken by this searching was negligible in comparison with the plan cycle rate of the robot.

Plan elements, senses and actions are referenced by unique numeric IDs, rather than by name. The memory storage of these IDs is defined within the code using the C++ `#typedef` preprocessor command, so that the width of these IDs can be configured at compile time, depending on the maximum ID value to be stored. This again saves memory in an environment where every byte counts. Consideration of stack usage is also important, and temporary buffers and similar structures are kept to a minimum to avoid stack overflow.

Fixed strings (for example error messages) and other data defined within programs are usually also stored within working memory. Within a microcontroller environment such as ARDUINO this is wasteful of the limited memory resource. This problem has been eliminated in the Instinct Planner implementation by use of AVR LIBC functions (Atmel Corporation, 2016b) that enable fixed data to be stored in the much larger program (flash) memory. For code compatibility these functions have been replicated in a pass-through library so that the code compiles unaltered on non-microcontroller platforms.

Instinct Testing Environment

As a means to test the functionality of the Instinct Planner within a sophisticated debugging environment, we have an implementation of the planner within MICROSOFT VISUAL C++, and have tested a very simple simulation of a robot within a grid based world. The world allows multiple robots to roam, encountering one another, walls and so on. This could be extended in future, with a graphical user interface

to better show both the world and the real time monitoring available from within the plan. However our current research focusses on the real time debugging of actual robots (Wortham, Theodorou, and Bryson, 2016). Building transparency into robot action selection can help users build a more accurate understanding of the robot, see below.

The Instinct Planner code is not fundamentally limited to 255 plan elements, and will support much larger plans on platforms with more memory. In MICROSOFT VISUAL C++ for example, plans with up to 65,535 nodes are supported, simply by redefining the `instinctID` type from `unsigned char` to `unsigned int`.

Instinct Transparency Enhancements

The planner has the ability to report its activity as it runs, by means of callback functions to to a monitor C++ class. There are six separate callbacks monitoring the Execution, Success, Failure, Error and In-Progress status events, and the Sense activity of each plan element. In the VISUAL C++ implementation, these callbacks write log information to files on disk, one per robot instance. This facilitates the testing and debugging of the planner. In the ARDUINO robot, the callbacks write textual data to a TCP/IP stream over a wireless (wifi) link. A JAVA based Instinct Server receives this information, enriches it by replacing element IDs with element names, and logs the data to disk. This communication channel also allows for commands to be sent to the robot while it is running.

With all nodes reporting all monitor events over wifi, a plan cycle rate of 20Hz is sustainable. By reducing the level of monitoring, we reduce the volume of data sent over wifi and plan cycle rates of up to 40Hz are achievable. In practice a slower rate is likely to be adequate to control a robot, and will reduce the volume of data requiring subsequent processing. In our experiments a plan cycle rate of 8Hz was generally used.

Figure 2 shows how the planner sits within the robot software environment and communicates with the Instinct Server.

Instinct Command Set

The robot command set primarily communicates with the planner which in turn has a wide range of commands, allowing the plan to be uploaded and altered in real time, and also controlling the level of activity reporting from each node in the plan. When the robot first connects to the Instinct Server, the plan and monitoring control commands are automatically sent to the robot, and this process can be repeated at any time while the robot is running. This allows plans to be quickly modified without requiring any re-programming or physical interference with the robot.

Creating Reactive Plans with iVDL

POSH plans are written in a LISP like notation, either using a text editor, or the ABODE editor (Brom et al., 2006; Bryson, 2013). However, Instinct plans are written very differently, because they must use a much more compact notation and they use IDs rather than names for plan elements,

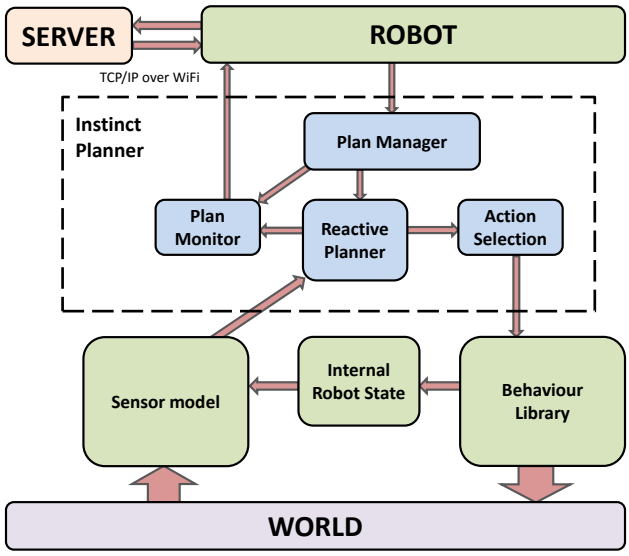


Figure 2: Software Architecture of the R5 Robot showing interfaces with the World and the Instinct Server. The Instinct Planner provides the action selection subsystem of the robot.

senses and actions. We have developed the *Instinct Visual Design Language (iVDL)* based on the ubiquitous Unified Modelling Language (UML) notation. UML is supported by many drawing packages and we have developed a simple PYTHON export script to allow plans to be created graphically within the DIA drawing tool (Macke, 2014). The export script takes care of creating unique IDs and allows the plans to use named elements, thus increasing readability. The names are exported alongside the plan, and whilst they are ignored by the planner itself, the Instinct-Server uses this export to convert IDs back into names within the log files and interactive display.

Figure 3 shows the Instinct plan template within Dia. We use the UML class notation to define classes for the six types of element within the Instinct plan, and also to map the external numerical identifiers (IDs) for senses and robot actions to names. We use the UML aggregation connector to identify the connections between the plan elements. This can be read, for example, as "A Drive can invoke an Action, a Competence or an Action Pattern".

Figure 4 shows a plan for the R5 robot. At this level of magnification the element details are not legible, but this screen shot gives an impression of how plans can be laid out.

This particular plan searches the robot's environment, avoiding objects and adjusting its speed according to the space around it. As it moves around it attempts to detect humans within the environment. The robot also temporarily shuts down in the event of motor overload, and it will periodically hibernate when not in open space to conserve battery power. Such a plan might be used to patrol hazardous areas such as industrial food freezers, or nuclear facilities.

The plan was designed and debugged within the space of a week. During the debugging, the availability of the trans-

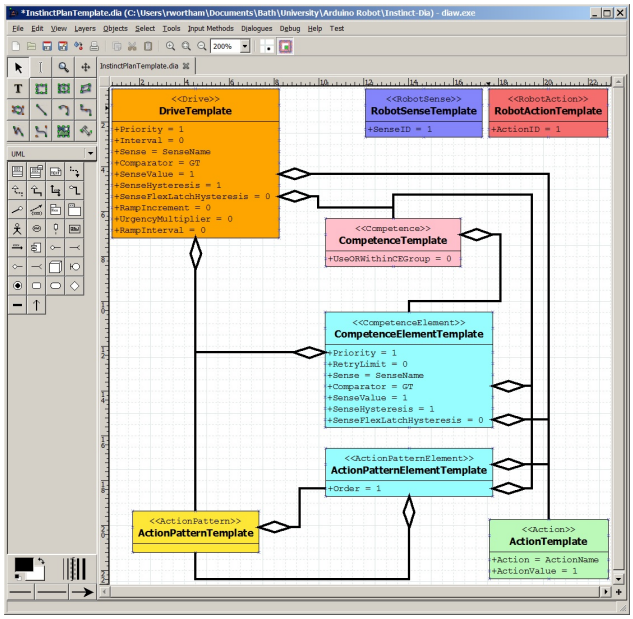


Figure 3: Instinct Plan element types and their relationship, shown within the DIA drawing tool.

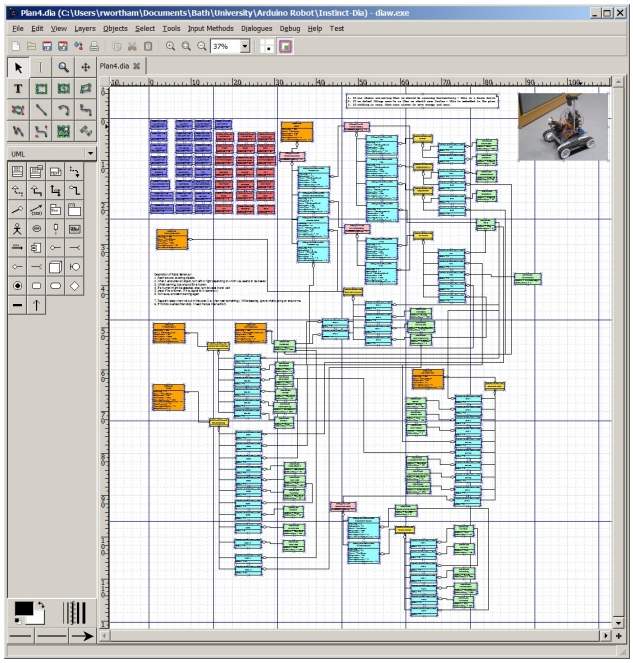


Figure 4: The plan used by the R5 robot to enable it to explore an environment, avoid obstacles, and search for humans. The plan also includes emergency behaviours to detect and avoid excessive motor load, and to conserve battery by sleeping periodically.

parency data logged by the Instinct Server was extremely useful, because mere observation of the robot's emergent behaviour is frequently insufficient to determine the cause of plan malfunction.

The actual positioning of plan elements within the drawing is entirely up to the plan designer. Since Dia is a general purpose graphical editor, other symbols, text and images can be freely added to the file. This is useful at design time and during the debugging of the robot. It also provides an additional vehicle for the creation of longer term project documentation. We suggest that an in-house standard is developed for the layout of plans within a development group, such that developers can easily read one another's plans.

Plan Debugging and Transparency

Currently, work is underway within the Artificial Models of Natural Intelligence (AmonI) research group at the University of Bath[1] to create a new version of the ABODE plan editor (Theodorou, Wortham, and Bryson, 2016). This version directly writes Instinct plans, and also reads the real-time transparency data emanating from the Instinct Planner, in order to provide a real-time graphical display of plan execution. In this way we are beginning to explore both runtime debugging and wider issues of AI Transparency.

CONCLUSIONS AND FURTHER WORK

The Instinct planner is the first major re-engineering of Bryson's original work for several years, and allows deployment in practical real time physical environments such as our ARDUINO based maker robot.

By using a very lean coding style and efficient memory management, we maximise the size of plan that can be dynamically loaded and the performance in terms of execution rate.

The transparency capabilities, novel to this implementation of POSH, provides the necessary infrastructure to deliver real time plan debugging. Work is currently underway to leverage this architecture with a real time visual debugging tool, initially to assist the work of reactive plan designers, but also as a research tool for the investigation of wider AI Transparency issues.

The Visual Design Language (iVDL) is a novel representation of reactive plans, and we demonstrate that such plans can be designed using a standard drawing package and exported with a straightforward plug-in script. We envisage the development of similar plug-ins for other drawing tools such as MICROSOFT VISIO.

Although primarily developed for physical robot implementations, the Instinct Planner has obvious applications in teaching, simulation and game AI environments. We envisage extending the current Instinct Testing Environment to provide a richer, GUI based test platform for Instinct, and for use as a teaching tool to teach the concepts of reactive planning in general and the Instinct Planner in particular.

Finally, we would like to see the implementation of Instinct on other embedded and low cost Linux computing environments such as the RASPBERRY PI (Raspberry Pi Foundation, 2016). With more powerful platforms such as the PI, much larger plans can be developed and we can test both the runtime performance of very large plans, and the design

[1] AmonI — http://www.cs.bath.ac.uk/ai/AmonI.html

efficiency of iVDL with multi-user teams.

References

Arduino. 2016. *Arduino Website*. https://www.arduino.cc/.

Atmel Corporation. 2016a. *Atmel Studio Website*. http://www.atmel.com/Microsite/atmel-studio/.

Atmel Corporation. 2016b. *AVR Libc Reference Manual*. http://www.atmel.com/webdoc/avrlibcreferencemanual/.

Breazeal, C., and Scassellati, B. 2002. Robots that imitate humans. *Trends in Cognitive Sciences* 6(11):481–487.

Brom, C.; Gemrot, J.; Bida, M.; Burkert, O.; Partington, S. J.; and Bryson, J. J. 2006. POSH Tools for Game Agent Development by Students and Non-Programmers. In *The Nineth International Computer Games Conference: AI, Mobile, Educational and Serious Games.*, 1–8. Bath, UK: The University of Bath.

Brooks, R. a. 1991. Intelligence Without Representation. *Artificial Intelligence* 47(1):139–159.

Brooks, R. 1995. Intelligence Without Reason. In Steels, L., and Brooks, R. R. A., eds., *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents*. Mahwah, New Jersey, USA: L. Erlbaum Associates. 25–81.

Bryson, J. J. 2000. The study of sequential and hierarchical organisation of behaviour via artificial mechanisms of action selection. M.Phil. Thesis, University of Edinburgh.

Bryson, J. J. 2001. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. Ph.D. Dissertation, MIT, Department of EECS, Cambridge, MA. AI Technical Report 2001-003.

Bryson, J. 2008. *Parallel-rooted, Ordered Slip-stack Hierarchy*. http://www.cs.bath.ac.uk/˜jjb/web/posh.html.

Bryson, J. J. 2011. A Role for Consciousness in Action Selection. In Chrisley, R.; Clowes, R.; and Torrance, S., eds., *Proceedings of the AISB 2011 Symposium: Machine Consciousness*, 15—-20. York: SSAISB.

Bryson, J. J. 2013. *Advanced Behavior Oriented Design Environment (ABODE)*. http://www.cs.bath.ac.uk/˜jjb/web/BOD/abode.html.

Gaudl, S., and Bryson, J. J. 2014. The Extended Ramp Goal Module: Low-Cost Behaviour Arbitration for Real-Time Controllers based on Biological Models of Dopamine Cells. *Computational Intelligence in Games 2014*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Series in Artificial Intelligence. Elsevier/Morgan Kaufmann.

Gurney, K. N.; Prescott, T. J.; and Redgrave, P. 1998. The Basal Ganglia viewed as an Action Selection Device. In *Eighth International Conference on Artificial Neural Networks*, 1033–1038. London, UK: Springer.

Holland, J. H. 2014. *Complexity: A Very Short Introduction*. Oxford University Press.

Lim, C. U.; Baumgarten, R.; and Colton, S. 2010. Evolving behaviour trees for the commercial game DEFCON. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6024 LNCS(PART 1):100–110.

Macke, S. 2014. *Dia Diagram Editor*. http://dia-installer.de/.

Nilsson, N. J. 1984. Shakey the Robot. Technical report, SRI International, Technical Note 323.

Prescott, T. J.; Bryson, J. J.; and Seth, A. K. 2007. Introduction. Modelling natural action selection. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 362(1485):1521–9.

Raspberry Pi Foundation. 2016. *Raspberry Pi Website*. https://www.raspberrypi.org/.

Rohlfshagen, P., and Bryson, J. J. 2010. Flexible Latching: A Biologically-Inspired Mechanism for Improving the Management of Homeostatic Goals. *Cognitive Computation* 2(3):230–241.

Seth, A. K. 2007. The ecology of action selection: insights from artificial life. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 362(1485):1545–1558.

Theodorou, A.; Wortham, R. H.; and Bryson, J. J. 2016. Why is my Robot Behaving Like That? Designing Transparency for Real Time Inspection of Autonomous Robots. In Prescott, T. J., ed., *EPSRC Principles of Robotics Workshop, Proceedings of the AISB 2016 Annual Conference*.

Tinbergen, N., and Falkus, H. 1970. *Signals for Survival*. Oxford: Clarendon Press.

Tinbergen, N. 1951. *The Study of Instinct*. Oxford, UK: Oxford University Press.

Wortham, R. H.; Theodorou, A.; and Bryson, J. J. 2016. What Does the Robot Think? Transparency as a Fundamental Design Requirement for Intelligent Systems. In Prescott, T. J., ed., *IJCAI-2016 Ethics for Artificial Intelligence Workshop*.

Strategic Plannning for Autonomous Systems over Long Horizons

Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, and Bram Ridder
Kings College London, London, WC2R 2LS
firstname.lastname@kcl.ac.uk, bram.ridder@gmail.com

Abstract

Planning plays a role in achieving long-term behaviour (persistent autonomy) without human intervention. Such behaviour engenders plans which are expected to last over many hours, or even days. Such a problem is too large for current planners to solve as a single planning problem, but is well-suited to decomposition and abstraction planning techniques. We present a novel approach to bottom-up decomposition into a two-layer hierarchical structure, which dynamically constructs planning problems at the abstract layer of the hierarchy using solution plans from the lower layer.

We evaluate this approach in the context of persistent autonomy in autonomous underwater vehicles, showing that compared to strictly top-down approaches the bottom-up approach leads to more robust solution plans of higher quality.

1 Introduction

This paper introduces a novel technique for planning in the context of persistent autonomous systems within tight deadlines and energy constraints. Persistent autonomy entails planning long-term behaviour for one or more autonomous vehicles achieving purposeful and directed activity over hours, days, or even weeks without human intervention. This includes many challenges, including robust execution, detection of errors and recovery (Faria et al. 2014; Cashmore et al. 2014). However, there is a challenge that precedes execution: generating plans for missions that extend over hundreds or thousands of actions, within hours or days of activity.

We show in Section 5 that such a problem is too large for current planners to solve as a single planning problem, but is well-suited to decomposition and abstraction planning techniques. Decomposition into a hierarchical structure is exploited by HTN planners (Erol, Hendler, and Nau 1994; Nau, Ghallab, and Traverso 2015), which rely on a top-down approach, exploiting pre-constructed plans to tackle separate component elements of the hierarchy. In contrast we propose dynamically decomposing the problem in two layers using a bottom-up approach. The tactical layer at which task plans are constructed from the original actions in the domain, and the strategic layer, which encapsulated actions that represent the completion of a task. The original problem is decomposed into disjunct tasks using a clustering algorithm, each task is planned for independently at the tactical layer and forms a pre-constructed plan. Then a problem is composed at the strategic layer, to find an execution order of the pre-constructed plans that satisfies the original planning problem.

We explore this approach in the context of the FP7 project PANDORA, managing a fleet of Autonomous Underwater Vehicles (AUVs). These vehicles are tasked with maintaining a seabed facility unsupervised. The structures on the seabed must be inspected on a regular basis. The AUVs must interact with control panels within set time windows to manage the site within time and resource constraints, refueling autonomously.

Decomposing a task, in general, is not trivial. Planners like SGPlan (Chen, Wah, and wei Hsu 2006) and REALPlan (Srivastava 2000) have explored this in the past with mixed success. The latter decomposes a plan based on the number of resources available, e.g. it crease a different plan for each available robot. In contrast, the decomposition approach used in the PANDORA project is based on "locality". The observation we exploit is that many of these long-term autonomy missions involve located executives interacting with their environment to achieve goals. These goals can be clustered, geographically and temporally, into a set of discrete tasks. A task can be associated with the area in which operations will be performed to accomplish its goals.

The key difference in our approach is in the construction of the strategic model. Similar to an HTN model, the strategic level contains macro actions that encapsulate plans at the tactical level. However, these plans are not constructed top-down, but automatically generated by a planner, bottom-up. A planner is used to construct plans for each task. The expected time and resource requirements to complete the task are taken automatically from the plan. These values are used as costs for the corresponding strategic action encapsulating this task. The strategic problem can then be constructed using these abstracted task-actions.

The paper is organized as follows. In Section 2 we give an overview of the relevant background of persistent autonomy and planning for long horizons. In Section 3 we describe our decomposition and abstraction technique in more detail. Then, in Section 4 we describe how a strategic mission plan can be executed, and some efficiency that can be gained. Finally, in Section 5 we describe the evaluation method for testing our framework and include the results of our evalua-

tion.

2 Related Work

Integrating planning systems with robotic systems for onboard planning in long-term missions is not a new concept, for example NASA's EUROPA framework was used for the EO-1 mission (Sherwood et al. 2006). Related to AUVs specifically a planning system was developed in cooperation with MBari to track algae blooms (Fox, Long, and Magazzeni 2012), within T-REX (Teleo-Reactive EXecutive) for reasoning onboard AUVs (McGann et al. 2008b), and using ROSPlan to integrate a planner with the COLA2 control architecture for AUVs in subsea intervention tasks (Cashmore et al. 2015; Palomeras et al. 2012).

We take the ideas introduced in these works and approach the challenge of *persistent autonomy*: missions that require robust planning and execution, with horizons of days, or even weeks. Alternative strategies for long term autonomy typically focus on execution monitoring or onboard replanning (eg (Smith, Rajan, and Muscettola 1997; McGann et al. 2008b; 2008a; Cashmore et al. 2015)). Nevertheless, these approaches are founded on the same ambition for long term autonomous behaviour and recognise the role of planning in achieving it. The contribution of this paper is in the formulation of a decomposed planning problem, through a bottom-up decomposition approach.

3 Bottom-Up Top-Down Strategic Missions

In this section we formalise the planning problem and the decomposition of it. We use PDDL2.1 (Fox and Long 2003) to describe our example domain and problems. In general the approach is not tied to choice of description language.

Definition 1. *PDDL2.1 Planning Problem. A PDDL2.1 planning problem is the tuple* $\Pi := \{P, V, A, T_p, T_v, I, G\}$*, where* P *is a set of propositions;* V *is a vector of real variables, called fluents; both are manipulated by* A*, a set of durative and instantaneous actions.* $I(P, V)$ *is a fuction over* $P \cup V$ *which describes the initial state of the problem. Similarly* $G(P, V)$ *is a function that describes the goal condition.* T_p *is the set of time initial literals (TILs). A TIL* (t, p) *describes that proposition* p *becomes true at time* t*.* T_v *is the set of time initial fluents (TIFs). A TIF* (t, v, x) *describes that fluent* v *is assigned the value* $x \in \mathbb{R}$ *at time* t*.*

A durative action a is described as a tuple: $a := \{pre_a, eff_a, dur_a\}$ where pre_a represents the action's preconditions – conditions that must hold for the action to be applied – eff_a represents the action's effects, and dur_a is a duration constraint, a conjunction of numeric constraints corresponding to the duration of the action a.

A single condition is either a single proposition $p \in P$, or a numeric constraint over V. A precondition is a conjunction of zero or more conditions. Each durative action A has three subsets of preconditions: $pre_{\vdash a}, pre_{\leftrightarrow a}, pre_{\dashv a} \in pre_a$. These represent the conditions that must hold at its start, throughout its execution, and at the end of the action, respectively.

Action effects are described by $eff^+_{\vdash a}$, $eff^-_{\vdash a}$, effects which add and remove propositions at the start of the action, respectively. Similarly $eff^+_{\dashv a}$, $eff^-_{\dashv a}$ add and remove propositions at the end of the action. Numeric effects $eff^{num}_{\vdash a}$, $eff^{num}_{\dashv a}$ assign values to fluents at the start and end of the action. Finally, continuous numeric effects $eff_{\leftrightarrow a}$ describe continuous change throughout the action's duration. (Fox and Long 2003).

A solution to a planning problem is a sequence of actions and timestamps $\langle (a_0, t_0), (a_1, t_1) \ldots, (a_n, t_n) \rangle$ applicable in I for which the resultant state S' satisfies the goal: $S' := G$.

A planning problem can contain fluents which are *resources* (Coles et al. 2014), we define resources as follows:

Definition 2. *Resource. Given a planning problem* Π *a resource is defined as a numerical fluent* $v \in V$ *whose value can be altered by the effect of an action* $a \in A$ *and also appears in the precondition of an action or is contained by a goal. A special resource is time as it can be constrained by TILs in the domain.*

An example of a resource in a logistical domain might be fuel, which can altered by consuming it or produced by refuelling.

Decomposition into Tasks

Given a planning problem Π we search for a decomposition that separates the goal G into a set of *tasks* T where each task $task \in T$ is $task \subseteq G$. For each $task$ we construct new planning problem $\Pi' = \Pi$, where $\Pi'_G = task$. At the tactical planning stage we do not know how many resources are available for each task, so we remove any constraints on these. These constraints are imposed on the planner at the strategic level.

We define a task's initial state to be a subset $I' \subseteq P$ describing the requirements for beginning the task. These requirements for beginning each task must be carefully chosen, we need to make sure that these requirements are reachable after the execution of any other task. In our work we rely on a domain expert to assign an initial state to each task. For example, in the PANDORA domain we identify, for each task, a location that is close to the area within which that task will be performed, $L(task)$. We call this the *jump off* point for $task$. The initial state I' of each task is that the AUV is located at the jump off point. These jump off points are located above the seabed structures, which makes them easily reachable. The task's initial state is used in the generation of the strategic problem, discussed in the next section.

Each problem Π' is passed to a planner, and a plan found. The solution plan $plan(\Pi')$ and its resource requirements are saved.

Whilst in general many decompositions are possible, it is sensible to decompose a problem such that related goals are combined in a single task. Decomposition of goals can be computed using a clustering algorithm or can be hand coded based on expert information of the domain. In PANDORA we exploit the fact that the domain is already separated into multiple seabed structures. While part of the same seabed facility, the structures are separated spatially, and are a simple way to split the goals into sets based on location. Moreover, control panels can be interacted with only within certain

time windows. If we create a task containing a goal with an associated time window with any other goal, then we might impose artificial constraints on the strategic level, leading to an unsolvable strategic problem. For example, consider the goals g_1, g_2, and g_3 that have non-overlapping time windows, such that g_1 needs to be achieved before g_2 and g_2 needs to be achieved before g_3. If we create a task that combines g_1 and g_3 and another task that contains g_2 then we render the planning problem unsolvable. For this reason we put those goals that are dependent on deadlines in separate tasks.

Generating a Strategic Problem

In order to generate the strategic problem, first an estimate of the resource use for each task must be computed. These estimates are computed from the tactical plans $plan(\Pi')$. The strategic domain and problem describe the execution of a task in an abstracted way, similar to HTN planning, by encapsulating the task plan as a single action. Therefore, the $plan(\Pi')$ is encapsulated in an abstract action a_{task} for which:

- pre_a models the bounds on the resources used by $plan(\Pi')$, and the initial state I'.
- eff_a^{num} describes the change in resource over $plan(\Pi')$.
- $eff_a^+ \cup eff_a^-$ describes the change in distinct variables over $plan(\Pi')$.
- dur_a is the duration of $plan(\Pi')$.

The domain and an example problem file from PANDORA are shown in figures 1 and 2 respectively. The *complete_mission* actions correspond to the tactical plans for a single task. The duration of these actions is determined by the function *mission_duration*, which is defined in the initial state. These mission durations (along with any other possible resource requirements) are set in the initial state to be equal to the duration of the corresponding task plan. For example, in figure 2 the duration of each mission is assigned in the initial state, eg:

```
(= (mission_duration Mission10) 117.739)
```

As a precondition of the *complete_mission* action, the executive vehicle must be at the mission jump-off location $L(T)$ for the task t, and there must still be enough resource and time available to achieve the task completely.

4 Efficient Execution of Strategic Missions

There are some improvements we can add in the way that the strategic plan is executed. It is possible to simply replace the strategic *complete_mission* actions with the tactical plans they represent. However, a more efficient plan can be found by replanning the tactical problem during execution of the strategic plan. Example strategic and tactical plans are shown in figures 3 and 4.

There are several reasons to replan the tactical problem before its encapsulating *complete_mission* action is executed in the strategic plan.

1. depending on the execution of previous tactical plans, the current amount of available resource might differ from what was expected to be available;

```
(define (domain strategic)
(:requirements ...)
(:types waypoint mission vehicle)

(:predicates
  (connected ?wp1 ?wp2 - waypoint)
  (at ?v - vehicle ?wp - waypoint)
  (vehicle_free ?v - vehicle)
  (in ?m - mission ?wp - waypoint)
  (completed ?m - mission)
  (active ?m - mission)
  ...
)

(:functions
  (distance ?wp1 ?wp2 - waypoint)
  (mission_duration ?m - mission)
  (charge ?v - vehicle)
  (mission_total)
)

(:durative-action complete_mission
  :parameters (?v - vehicle ?m - mission ?wp - waypoint)
  :duration ( = ?duration (mission_duration ?m))
  :condition (and
    (over all (vehicle_free ?v))
    (over all (active ?m))
    (at start (in ?m ?wp))
    (at start (at ?v ?wp))
    (at start (>= (charge ?v) (mission_duration ?m)))
  )
  :effect (and
    (at start (not (at ?v ?wp)))
    (at end (increase (mission_total) 1))
    (at end (decrease (charge ?v) (mission_duration ?m)))
    (at end (completed ?m))
    (at end (at ?v ?wp))
  )
)

(:durative-action do_hover ...
(:durative-action dock_auv ...
(:durative-action recharge ...
(:durative-action undock_auv ...
))
```

Figure 1: A fragment of a strategic domain. The body of some domain-specific operators is ommitted for space. The *complete_mission* operator corresponds to the tactical plan of a task.

2. similarly, the *complete mission* (strategic) action might be dispatched earlier or later than was anticipated, which might have knock-on effects on deadlines in the tactical task;
3. depending on the direction of approach from the previous action of the (strategic) plan, the tactical plan might be planned differently to exploit better routes between elements of the task.

The execution of a tactical plan, including any tactical replanning, or rescheduling, is handled by an onboard executive, in our evaluation we use ROSPlan (Cashmore et al.

```
(define (problem strategic_mission)
(:domain strategic)
(:objects
  v - vehicle
  mission_site_start_point_0 wp_auv0
  ... - waypoint
  Mission0 Mission1 Mission10 Mission12
  Mission13 Mission14 Mission15
  ... - mission
)
(:init
  (vehicle_free v)
  (at auv wp_v0)
  (= (charge v) 1200)
  (= (mission_total) 0)

  (recharge_at mission_site_start_point_0)

  (active Mission0)
  (active Mission1)
  (active Mission10)
  (active Mission12)
  (active Mission13)
  (active Mission14)
  (active Mission15)
  ...
  (at 4100 (not (active Mission0)))
  (at 7100 (not (active Mission1)))
  (at 86400 (not (active Mission10)))
  (at 86400 (not (active Mission12)))
  (at 86400 (not (active Mission13)))
  (at 86400 (not (active Mission14)))
  (at 86400 (not (active Mission15)))
  ...
  (in Mission0 mission_site_start_point_1)
  (in Mission1 mission_site_start_point_1)
  (in Mission10 mission_site_start_point_1)
  (in Mission12 mission_site_start_point_1)
  (in Mission13 mission_site_start_point_1)
  (in Mission14 mission_site_start_point_1)
  (in Mission15 mission_site_start_point_1)
  ...
  (= (mission_duration Mission0) 261.868)
  (= (mission_duration Mission1) 242.065)
  (= (mission_duration Mission10) 117.739)
  (= (mission_duration Mission12) 154.668)
  (= (mission_duration Mission13) 157.892)
  (= (mission_duration Mission14) 151.502)
  (= (mission_duration Mission15) 135.29)
  ...
  (connected mission_site_start_point_0 wp_v0)
  (= (distance mission_site_start_point_0
      wp_v0) 56.7891)
  ...
)
(:metric maximize (mission_total))
(:goal (> (mission_total) 0))
)
```

Figure 2: A fragment of an example strategic problem.

2015). In our tactical domain, we use a conservative model of action duration and cost, so we expect that most tasks will

```
0.000: (do_hover auv wp_auv0 wp0)  [291.548]
291.548: (complete_mission auv mission9 wp0)  [194.639]
486.188: (complete_mission auv mission8 wp0)  [236.909]
723.098: (do_hover auv wp0 wp1)  [270.416]
993.516: (dock_auv auv wp1)  [20.000]
1013.516: (recharge auv wp1)  [1800.000]
2813.517: (undock_auv auv wp1)  [10.000]
2823.517: (do_hover auv wp1 wp0)  [270.416]
3093.934: (complete_mission auv mission11 wp0)  [284.545]
3378.481: (complete_mission auv mission10 wp0)  [293.488]
3671.970: (do_hover auv wp0 wp1)  [270.416]
3942.387: (dock_auv auv wp1)  [20.000]
3962.387: (recharge auv wp1)  [1800.000]
5762.388: (undock_auv auv wp1)  [10.000]
5772.388: (do_hover auv wp1 wp0)  [270.417]
6042.806: (complete_mission auv mission1 wp0)  [342.707]
6385.514: (do_hover auv wp0 wp1)  [270.417]
6655.931: (dock_auv auv wp1)  [20.000]
6675.931: (recharge auv wp1)  [1800.000]
8475.932: (undock_auv auv wp1)  [10.000]
8485.932: (do_hover auv wp1 wp0)  [270.416]
8756.350: (complete_mission auv mission0 wp0)  [381.766]
9138.117: (do_hover auv wp0 wp1)  [270.416]
9408.534: (dock_auv auv wp1)  [20.000]
9428.534: (recharge auv wp1)  [1800.000]
11228.535: (undock_auv auv wp1)  [10.000]
```

Figure 3: A strategic plan for the abstract level. The *complete_mission* action corresponds to the tactical plan of a task.

```
% 0.000: (do_hover auv wp0 wp1) [34.547]
% 34.548: (check_panel auv wp1 ip0) [10.000]
% 44.549: (correct_position auv wp1) [10.000]
% 55.208: (valve_state auv wp1 p0) [10.000]
% 70.000: (do_hover auv wp0 wp2) [9.921]
% 79.922: (turn_valve auv wp2 p0 v0) [30.000]
% 109.923: (correct_position auv wp2) [10.000]
% 149.924: (turn_valve auv wp2 p0 v1) [30.000]
% 179.925: (correct_position auv wp2) [10.000]
```

Figure 4: A tactical plan for a single task. This plan is generated to provide resource estimates in the construction of an abstract strategic problem, and encapsulated at that level in a *complete_mission* action.

be completed within the estimated time. However, when executing plans onboard a physical platform there is always the chance that actions might fail, take longer than expected, or be accomplished more quickly.

To take advantage of extra or diminished resource (1), or alteration in deadline (2), we replan a tactical task before it is executed. This process takes 10 seconds, and can be performed in parallel with other strategic actions.

When planning tactical tasks in the construction of the strategic problem, the task jump-off point $L(t)$ is used as the intial position of the executive. After the strategic plan has been generated, the initial position of the executive can be improved by considering the previous actions in the strategic plan. Thus, the executive no longer needs to visit the jump off point, but can move directly to the most convenient first

location in the execution of the task.

To ensure efficient linkage to the next task, the latest destination in the strategic plan can be used as the initial location for the executive in the task to be replanned. $L(t)$ will be ignored and replaced with the necessary connecting navigation actions to get between the latest destination in the strategic plan and the best entry point in the next task.

Finally, it is necessary to revalidate the strategic plan after the replanning or completion of any tactical task. It is possible that a tactical plan takes longer to complete than expected, and from the current time the strategic plan no longer respects the task deadlines. Similarly, a tactical plan might take more resource than accounted for by our conservative action model, and there is no longer enough resource to complete subsequent actions in the strategic plan.

To account for these points, we use the ROSPlan executive for dispatch of the strategic, as well as tactical plan. More generally, execution monitoring techniques developed for tactical plans can be used for the execution of plans at both levels in the hierarchy.

5 Results

We tested this approach in the context of the FP7 project PANDORA, managing a fleet of Autonomous Underwater Vehicles (AUVs). In order to do this, we have built an underwater environment simulation. The simulator possesses an in-built editor used to model missions with very long horizons, allowing us to experiment with multi-hour and multi-day missions, combining multiple tasks, such as inspection of a complex site (figure 5) and valve turning, under deadlines and resource constraints. The simulation provides uncertainty about action durations, creates unpredictable features such as marine life, simulates currents, and allows the AUV to discover new features in the environment.

We compare against a selection of top-down decompositions to show that the tactical information gathered by the bottom-up top-down (BUTD) approach leads to heigher quality, and more robust solutions. We compare against a purely tactical approach to demonstrate that the scenario is too large to solve as a single planning problem, and some heirarchical decomposition is a possible solution.

We used the BUTD decomposition approach to separate a set of missions into tasks, with results between 10 and 30 tasks. We then found strategic solutions to these problems. To compare, we used the same set of tasks in a top-down decomposition, in which the tactical missions had not been planned, and their estimated resource usage computed. As the top-down approach has not yet planned on the tactical level, another estimate of each task's resource requirements is required. We expect that some prior knowledge of resource use would be known, and so use the estimated resource usage computed from the BUTD decomposition as this a priori knowledge, and used it in four different estimates of task resource usage:

- *mean*: a naive strategy that takes the mean resource use over all tasks, and assigns this to all tasks;
- *conservative* a conservative strategy that assigns to all tasks the 80th percentile of resource use over all tasks;
- bucket-mean: the tasks were divided into sets of similar type – inspection, valve-turning, etc. – and size. Then, the mean resource use from each set was assigned as the estimated resource use for each task in that set;
- bucket-conservative: the tasks were divided in the same was as bucket-mean, but the 80th percentile from each set was used instead of the mean.

We used the planner POPF (Coles et al. 2010). POPF is a temporal and metric planner, which allows us to model the synchronisation aspects of our problems (including deadlines for interaction with valves), the constraints on energy over long missions, and optimise plans based on a metric function. The metric to be optimised was the number of tasks completed. POPF was given 1800 seconds and 8GB of RAM to find the best possible solution. In the BUTD approach POPF was given 10 seconds per tasks to perform tactical planning for each task. This reduced the amount of time given to solving the BUTD strategic problem.

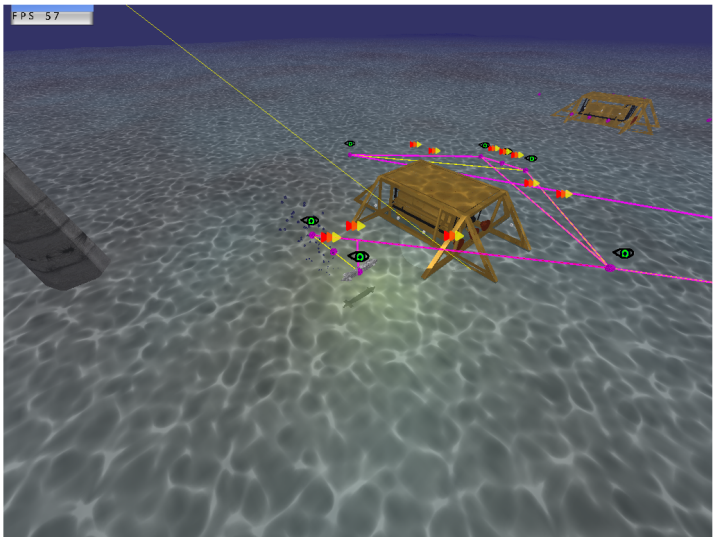
Due to the extra information from the tactical level, we expected that the strategic solutions generated using the BUTD approach would plan more efficient solutions in terms of number of tasks completed. Moreover, as the tasks had already been tactically planned, the resource usages of fewer tasks would have been underestimated, and the solutions are expected to be more robust.

Table 1 compares the number of tasks solved by BUTD and the various top-down strategies. As can be seen from the table, BUTD solves more tasks than any of the top-down strategies. Table 2 shows the number of tasks for which the resource usage is underestimated. In the strategic problems generated by the top-down strategies, these tasks have assigned resource estimates lower than that derived from planning a tactical solution. As a result, these tasks are very likely to run out of time and resource before successfully completing.

In the top-down approaches, the conservative approach performs the poorest, which is to be expected – the large uniform estimates for the resource usage of every task leads to a solution with many unnecessary refueling actions and missed deadlines. Both mean and bucket-mean strategies lead to higher quality solutions, but are not robust, underestimating the resource use of 32% and 27% of tasks respectively. The bucket-conservative approach performs best of the top-down appraches. The approach is the most robust, underestimating the resource requirements of only 14% of tasks, and generating the highest quality strategic solutions amongst all top-down approaches.

BUTD outperforms all top-down approaches, confriming our expectations. The variance in the resource usage between tasks of similar type and size is due to constraints that are only visible at the tactical level of the original domain, and as such are not available prior to planning the task itself. With this information, the BUTD strategic solutions are the most robust – none of the tasks are guaranteed to fail – and of the highest quality.

Table 3 shows the results for attempting to plan missions without decomposing into a tactical/strategic heirarchy. The number of tactical goals achieved by the best plan. It is clear



0.000: (correct_position auv0 wp_auv0) [3.000]
3.001: (do_hover_fast auv0 wp_auv0 strategic_location_7) [11.403]
14.405: (correct_position auv0 strategic_location_7) [3.000]
17.406: (observe_inspection_point auv0 strategic_location_7 inspection_point_22) [10.000]
27.407: (correct_position auv0 strategic_location_7) [3.000]
30.408: (do_hover_fast auv0 strategic_location_7 strategic_location_4) [11.673]
42.082: (correct_position auv0 strategic_location_4) [3.000]
45.083: (do_hover_controlled auv0 strategic_location_4 strategic_location_5) [4.000]
49.084: (observe_inspection_point auv0 strategic_location_5 inspection_point_20) [10.000]
59.085: (correct_position auv0 strategic_location_5) [3.000]
62.086: (do_hover_controlled auv0 strategic_location_5 strategic_location_6) [4.000]
66.087: (observe_inspection_point auv0 strategic_location_6 inspection_point_21) [10.000]
76.088: (correct_position auv0 strategic_location_6) [3.000]
79.089: (do_hover_fast auv0 strategic_location_6 strategic_location_4) [4.000]
83.090: (correct_position auv0 strategic_location_4) [3.000]
86.091: (observe_inspection_point auv0 strategic_location_4 inspection_point_19) [10.000]
96.092: (correct_position auv0 strategic_location_4) [3.000]
99.093: (do_hover_controlled auv0 strategic_location_4 strategic_location_5) [4.000]
103.094: (do_hover_fast auv0 strategic_location_5 strategic_location_8) [13.124]
116.219: (correct_position auv0 strategic_location_8) [3.000]
119.220: (observe_inspection_point auv0 strategic_location_8 inspection_point_23) [10.000]
129.221: (correct_position auv0 strategic_location_8) [3.000]
132.222: (do_hover_fast auv0 strategic_location_8 strategic_location_5) [13.124]
145.348: (correct_position auv0 strategic_location_5) [3.000]
148.349: (do_hover_controlled auv0 strategic_location_5 strategic_location_4) [4.000]
152.350: (do_hover_fast auv0 strategic_location_4 strategic_location_7) [11.673]
164.023: (correct_position auv0 strategic_location_7) [3.000]
167.024: (do_hover_fast auv0 strategic_location_7 wp_auv0) [11.403]
178.429: (correct_position auv0 wp_auv0) [3.000]
181.430: (do_hover_fast auv0 wp_auv0 mission_site_start_point_0) [40.307]

Figure 5: The simulation (left) and current plan (right) of a tactical mission for an inspection task.

that a purely tactical approach is insuffiecient for these sizes of problem.

6 Conclusion

We have presented an novel approach to the heirarchical decomposition a planning problem in the context of persistent autonomy. Our approach constructs an abstracted "strategic" layer of the heirarchy from solutions to the tactical tasks planned at the level of the original domain. The resulting problems can be solved and dispatched using a breadth of execution frameworks already available for executing plans onboard a robotic platform.

We briefly described the decomposition of a mission into a set of tasks, based on geographical and temporal clustering. This simple solution enables us to generate the problem heirarchy, but is not necessarily the best approach. In fact, any decomposition sacrifices the possibility or more optimal solutions that are formed from the interweaving of action in multiple tasks. Moreover, our spatial and temporal decomposition is specific to our domain. We propose to investigate the use of more general decompositions in future work.

In generating an strategic layer there is a trade-off when estimating the resource constraints of abstracted actions that encapsulate collections of lower-level behaviour. A more conservative approach is more robust, but over-estimation of resource use leads to less efficient plans that do not utilise all of the time and resource actually available. We show that it is possible to precompute much of this information in the automatic generation of the strategic layer, creating tighter bounds on resource use that remain robust.

We demonstrate these benefits, simulating long-term missions by autonomous underwater vehicles in a dynamic environment.

Tasks Completed					
number of tasks	BUTD	Top-down conservative	bucket-mean	mean	bucket-conservative
10	**10**	6	**10**	**10**	**10**
10	**10**	6	**10**	**10**	**10**
10	**10**	6	**10**	**10**	**10**
15	**15**	4	**15**	**15**	**15**
15	**15**	5	**15**	**15**	**15**
15	**15**	5	**15**	**15**	**15**
20	**20**	4	**20**	**20**	14
20	**20**	4	**20**	**20**	14
20	**20**	4	**20**	**20**	**20**
25	**24**	4	16	13	18
25	**23**	4	16	13	18
25	**25**	4	14	10	24
30	**25**	3	11	10	22
30	15	3	11	10	**22**
30	**25**	3	11	10	23

Table 1: Comparing BUTD and Top-down performance over long missions. The number of tasks completed in strategic missions of varying size.

Tasks Underestimated				
number of tasks	conser -vative	bucket mean	mean	bucket con -servative
10	1	5	4	2
10	0	3	2	1
10	2	3	2	3
15	1	7	6	2
15	0	3	3	1
15	4	5	4	5
20	1	9	8	2
20	0	5	4	1
20	4	5	4	5
25	1	11	10	2
25	0	6	5	1
25	6	7	6	7
30	1	13	12	2
30	0	7	6	1
30	6	7	6	7

Table 2: The quality of the different top-down representations, in terms of number of tasks given a duration that is shorter than the actual estimate of time required to complete the task. These tasks are likely to run out of time during execution.

Goals achieved		
goals	BUTD	Separate tasks
106	106	9
106	106	9
106	106	9
150	150	9
150	150	9
150	150	9
212	212	14
212	212	9
212	212	9
265	258	11
265	250	9
265	265	9
310	270	8
310	190	9
310	270	10

Table 3: Comparing the BUTD decomposition against planning for each goal separately. The number of goals achieved by the best plan.

References

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. Auv mission control via temporal planning. In *IEEE International Conference on Robotics and Automation (ICRA'14)*.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*.

Chen, Y.; Wah, B. W.; and wei Hsu, C. 2006. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research* 26:369.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20rd International Conference on Automated Planning and Scheduling (ICAPS'10)*, 42–49.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2014. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *CoRR*.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Htn planning: Complexity and expressivity. In *AAAI*, volume 94, 1123–1128.

Faria, M.; Pinto, J.; Py, F.; Fortuna, J.; Dias, H.; Martins, R.; Leira, F.; Johansen, T. A.; de Sousa, J. B.; and Rajan, K. 2014. Coordinating uavs and auvs for oceanographic field experiments: Challenges and lessons learned. In *2014 IEEE International Conference on Robotics and Automation (ICRA'14)*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Res. (JAIR)* 20:61–124.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based policy-learning for autonomous feature tracking. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

McGann, C.; Py, F.; Rajan, K.; Ryan, J. P.; and Henthorn, R. 2008a. Adaptive control for autonomous underwater vehicles. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence, (AAAI'08)*, 1319–1324.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. S. 2008b. A deliberative architecture for AUV control. In *2008 IEEE International Conference on Robotics and Automation (ICRA'08)*, 1049–1054.

Nau, D. S.; Ghallab, M.; and Traverso, P. 2015. Blended planning and acting: Preliminary approach, research challenges. In *Proceedings 29th AAAI Conference on Artificial Intelligence (AAAI'15)*.

Palomeras, N.; El-Fakdi, A.; Carreras, M.; and Ridao, P. 2012. Cola2: A control architecture for auvs. *IEEE Journal of Oceanic Engineering* 37(4):695–716.

Sherwood, R.; Chien, S.; Tran, D.; Davies, A.; Castaño, R.; Rabideau, G.; Mandl, D.; Frye, S.; Shulman, S.; and Szwaczkowski, J. 2006. *Enhancing science and automating operations using onboard autonomy*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration.

Smith, B. D.; Rajan, K.; and Muscettola, N. 1997. Knowledge acquisition for the onboard planner of an autonomous spacecraft. In *10th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW'97)*.

Srivastava, B. 2000. Realplan: Decoupling causal and resource reasoning in planning. In *AAAI/IAAI*, 812–818.

Opportunistic Planning for Increased Plan Utility

Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, and Bram Ridder*

Abstract

This paper explores the execution of planned missions in situations in which opportunities to achieve additional utility can arise during execution. The missions are represented as temporal planning problems, with hard goals and time constraints. Opportunities are soft goals with high utility. The probability distributions for the occurrences of these opportunities are not known, but it is known that they are unlikely so it is not worth trying to anticipate their occurrence prior to plan execution. However, as they are high utility, it is worth trying to address them dynamically when they are encountered, as long as this can be done without sacrificing the achievement of the hard goals of the problem. We formally characterise the opportunistic planning problem, introduce a novel approach to opportunistic planning and compare it to an on-board replanning approach in an example domain involving autonomous underwater vehicles.

1 Introduction

There are many examples of long-horizon control problems in which the goal is to complete specific tasks under time and resource constraints. To do so requires goal-achieving activities to be *planned*. An executive system then executes the resulting plan in the physical world to bring about the desired goals. This picture is complicated by the fact that most physical environments are dynamic, leading to uncertainty about the effects of actions (Paulos et al. 2015). One way to handle this uncertainty is to build a plan as a *policy* (a mapping from states to actions), allowing reactive control during execution, but the current Reinforcement Learning-based approaches to policy construction (Kaelbling, Littman, and Cassandra 1995a; Pineau, Gordon, and Thrun 2006a; Sanner and Boutilier 2009; Ong et al. 2009) do not scale to handle long-horizon tasks. It is computationally most efficient to plan without taking uncertainty into account. When large parts of a plan can be expected to execute without incident, it is more efficient to exploit the strategy of *replanning on failure*, rather than to try to plan ahead for contingencies. During the execution of a plan, execution activity will divert from the original plan when failures occur and actions to achieve the original goals are replanned from the resulting unexpected state.

Another motivation for diverting from an original plan arises when unforeseen *opportunities* to achieve additional utility present themselves. Replanning on failure is a widely recognised technique, but responding to opportunities demands a different behaviour. In contexts in which these opportunities are unlikely, and might arise without warning during execution of a plan, the construction of a policy or a contingent plan that can exploit them is generally impractical. An example of a domain in which opportunities can arise is in the pursuit of planetary space science, where an unexpected high-value science phenomenon might occur during the execution of a long traverse. Instead of missing the phenomenon and having to be directed back by human experts (as was the case when the Mars rover, Opportunity, missed Block Island in 2009), the intelligent vehicle should autonomously detect the phenomenon and determine, without recourse to human advice, whether there are resources available to devote to it.

The domain we consider in this paper is the autonomous inspection and maintenance of underwater installations. We begin, in Section 2, by briefly introducing the field of Automated Planning, the technology we have exploited to address opportunistic planning. In Section 3, we then describe the operational context, introducing the relevant concepts and explaining the planning problem. In Section 4 we explain what we mean by *opportunistic planning*, and in Section 5 we formalise this problem. In Section 6 we discuss probabilistic approaches to similar problems. In Section 7 we explain how we have addressed opportunistic planning within a deterministic planning framework. We then present results for a number of experiments and discuss future work directions.

2 Automated Planning

Planning is the process of considering and organising actions to achieve goals, before starting to execute them. In planning, the actions that must be performed are not predetermined by the goals, but are selected, from amongst a typically large number of alternative actions. The choice is guided by an effort to achieve the goals whilst optimising various metrics. Ordering choices and resource allocations are made, and evaluated, as part of the selection process.

*M. Cashmore, M. Fox, D. Long, D. Magazzeni and B. Ridder are with the Dept. of Informatics, King's College London, Strand, WC2R 2LS, UK.

The selection of a particular action affects choices that can be made subsequently, so has an important impact on the quality of the eventual plan. The consequence of this approach is that neither the number of actions in a plan, nor the makespan or resource allocation of the plan, are predetermined. This distinguishes planning from scheduling, where the actions to be performed are predetermined but the timing of actions, and the allocation of resources to them, are not.

Planning relies on the use of a model of the available actions to support both prediction of their effects on a state and the identification of states from which the actions are applicable. A standard modelling language used to represent actions for this purpose is the *Planning Domain Description Language* (PDDL), originally developed in 1998 by a committee led by Drew McDermott (McDermott et al. 1998), but later extended through several variants, including PDDL2.1 (Fox and Long 2003), PDDL2.2 (Edelkamp and Hoffmann 2004), PDDL3 (Gerevini et al. 2009) and PDDL+ (Fox and Long 2006). The extensions of most relevance to us here are PDDL2.1 and PDDL2.2, which introduced actions with duration and the opportunity for concurrency and management of deadlines. In this language, a planning problem is formally described by providing two files: the *domain* and the *problem*. The problem file consists of two parts: the *initial state* and the *goals*.

Definition 1 *A* **state** *is a set of known true facts consisting of boolean and numeric variables. A Boolean variable is expressed as a proposition consisting of a predicate and a vector of typed arguments, which is assigned the value True or False. A numeric variable is expressed as a function applied to a vector of typed variables, which is assigned to a numeric value.*

Definition 2 *An* **action** *is a tuple $\langle P, A, D\rangle$, representing a function from state to state, described in terms of its preconditions, P, and effects, $A \cup D$. The Boolean effects in A are the facts that are added by the action, while the Boolean effects in D are the facts that are removed by the action. The numeric effects in $A \cup D$ are to* increase *or* decrease *a numeric variable by some numeric quantity, or to* assign *a value to a numeric variable. An action may be applied in any state in which the preconditions are true, and it produces a state in which the effects are true.*

Definition 3 *A* **planning problem** *is a tuple, $\langle D, I, G\rangle$, where D is the domain file specifying the types, functions and predicates required to describe the problem, and containing the set of action schemas available to the planner. I is the initial state, consisting of all the facts that are known to be true when planning begins. G is the goal state, consisting of the hard goal conditions that must be achieved by the planner. The problem instance description varies, depending on the problem to be solved, while the domain is a fixed description of what the planner can do to change the state of the world.*

A temporal planning problem is an extension of a planning problem in which actions have *duration*. An action, A, is specified as having a $start$ and an end, and the temporal constraint, that the start precedes the end ($A_{start} < A_{end}$), is always enforced. The duration of an action might be flexible, so that the planner can choose it dynamically. Durative actions can specify invariant conditions that must hold over their entire interval. When durative actions are present, the planner must maintain a *simple temporal network* (Dechter, Meiri, and Pearl 1991) to enable the enforcement of temporal consistency during planning.

The actions used to model a domain usually encapsulate a behaviour that is managed, in execution, by one or more controllers, handling sensing and actuation to achieve a specific effect. The planner is concerned not with the execution of actions, but their organisation into larger collections in order to efficiently achieve a collection of goals. Thus, an action to navigate between waypoints will be implemented by controllers that attempt to use motors and localise via sensing, while the planner is concerned with deciding which locations to visit, for what purpose and in what order.

3 The Operational Context: Underwater Maintenance and Inspection Tasks

In this paper, we focus on long-term maintenance and inspection of underwater installations, using an Autonomous Underwater Vehicle (AUV). This work was carried out in the EU FP7 project, PANDORA[1]. The PANDORA project explored the achievement of *persistent autonomy*, through planning, task learning, plan execution within resource limits and adaptive response to unanticipated events.

The PANDORA project considers an underwater oil installation, consisting of manifolds, pipelines, valves and welds, requiring regular inspection and maintenance. The installation must be maintained over long periods, such as days or weeks, without human intervention. Because of energy and time constraints, mission plans must ensure that the best use is made of limited resources such as on-board energy. The situation is complicated by the fact that environmental conditions (such as currents and marine life) might affect how long tasks take to complete, and when they are available for completion. There is also uncertainty, both in the layout of the installation and the condition of its components (both of which might have changed since the construction of the installation).

The overall objective of the PANDORA project is for a suitably equipped AUV to: (i) construct long-term mission plans to ensure an effective monitoring of the site over time, and (ii) to execute the operations in these mission plans whilst managing uncertainty and responding to unexpected events. The AUV is equipped with a retractable gripper for turning valves, and a water jet for cleaning.

The daily operations to be performed by the AUV include: inspecting pillars, manifolds, welds and pipelines, reading valve-sensors, turning valves, cleaning components exposed to bio-fouling, and updating the mapped layout of the site. This latter task involves investigating objects that appear in unexpected locations, such as collapsed pillars, buried chains and pipeline segments, and other phenomena that could affect the welfare of the installation.

[1] http://persistentautonomy.com/

4 Opportunistic Planning

During the execution of a plan by an AUV, unexpected events might occur that provide *opportunities* for the vehicle to increase the overall utility of its operations. An example is that a part-submerged section of an anchor chain, or other structure, might be spotted during the execution of a mission. This event provides an opportunity to perform an unplanned inspection, or chain-following activity, provided that resources permit the execution of the necessary extra actions. Opportunities are not modelled or anticipated by the planner, and they can be managed without requiring the planner to reason with probabilities. They can be treated as dynamically occurring soft goals. These are distinguished from the goals specified in the problem instance description, which are treated as hard goals that must be satisfied.

To manage unexpected opportunities within a deterministic planning framework, we use a *conservative* planning approach. Conservative planning is a method that seeks to exploit the classical planning framework, while simultaneously recognising the underlying, but unknown, stochastic behaviour of the execution environment. This means that well-researched methods in temporal-metric planning can be exploited. In this approach, rather than seeking to produce a plan with optimal utility, we seek to produce a robust plan, in which we can have very high confidence that the goals will be achieved within the time and energy budget of the vehicle. We then use opportunities to increase the utility of the plan during its execution.

To construct the mission plans, we use the POPF planner (Coles et al. 2010), which takes planning domain models written in the temporal planning language PDDL2.2 (Edelkamp and Hoffmann 2004). A temporal planner is required because the valve-turning tasks impose temporal constraints. They are constrained to be turned within specified time windows. For example, in a given mission it might be necessary to reset a valve within a one-hour window timed to occur six hours into the future from the start of the plan. These constraints necessitate reasoning with deadlines and synchronisation of activities. Thus, although we consider only a single AUV executing actions in sequence, and therefore no concurrent activity, these deadlines raise synchronisation issues which make online methods such as the online receding horizon approach (Burns et al. 2012; Ross et al. 2008) impractical.

We assume that opportunities are rare, but offer high utility gain when they are spotted and exploited. Thus, opportunities in this framework are somewhat similar to *high impact, low probability* events (HILPs) (Lee, Preston, and Green 2011), although in this setting we are considering rare events with a positive value, while HILPs are typically treated as risks that threaten execution. We further assume that the probability density function governing the distribution of these opportunities in the physical space is unknown, so we cannot plan to anticipate them or determine their expected utility.

Our conservative planning strategy is based on the assumption that, when executed, the actions in a plan will have durations that are normally distributed around their means, and that actions will in fact take much longer than their mean durations. To build a robust plan we therefore use estimated durations for the actions that are longer than the means. For example, to have 95% confidence, we use 1.65 standard deviations from their means as the estimated durations of the actions. 1.65 standard deviations from the mean is the 95th percentile of the Gaussian distribution.

As a plan containing multiple actions is executed, the use of the 95th percentile as an estimate for the nominal execution time of each action leads to an accumulating expected error. So, if k actions all with independently distributed mean execution times m and standard deviations s are executed in sequence, the sum of the estimated durations will yield a total time for execution of $k(m + 1.65s)$. The time actually required to achieve the 95th percentile for the combined sequence of actions is only $km+1.65s\sqrt{k}$, showing that the estimate based on individual 95th percentiles yields a $1.65s(k - \sqrt{k})$ over-estimate of the time required for 95% confidence in execution of the entire sequence. Our proposed opportunistic planning method is designed to exploit this over-estimate for other tasks that arise opportunistically.

As a practical example, suppose that navigating the traverse between two waypoints on the installation has a mean time of 11,507 seconds (3.2 hours), and a standard deviation of 925 seconds (about 15 minutes). If 5 successive traverses between waypoints are to be executed, the use of the nominal time estimates will yield an estimated duration of 65,166 seconds (about 18 hours). The 95th percentile for the estimated duration of the combined sequence is 60,948 seconds, so using the 95th percentile for nominalisation will lead to an expected overestimate for the execution time of 4,218 seconds: just over an hour, which is about 7% of the 95th percentile time for the execution of the complete sequence.

Opportunities can only be spotted and exploited during the execution of *preemptible* actions, or at points between the execution of actions. In our application, the only preemptible actions are the navigation actions (of different types, corresponding to different modes of movement). We consider opportunities that are physically located in space, so it is generally the case that they will arise during movement between locations, when large areas are scanned as part of the navigation action.

With these points in mind, the opportunistic planning problem is as follows:

- The problem is a temporal planning problem, with deterministic actions and a collection of hard goals specified in the problem instance description.
- The problem exists in a 3-dimensional space, with tasks requiring the executive to perform actions at particular locations and actions allowing the executive to move between locations (possibly in more than one way).
- The initial state is uncertain in a limited way: there is a possibility that, at random locations, instances of objects exist that offer high reward if certain actions are performed at their locations, but their existence and locations are not known to the planner. There is also uncertainty

about whether these objects will be observed, even if the executive passes close to them.

- Although the probability distribution of opportunities is unknown, it is assumed that they are rare and it is therefore entirely likely that the plan for the original goals will be completely executed without an opportunity ever being encountered.
- The executive is required to satisfy the hard goals of the original problem, and to collect as much reward as possible from opportunities, given that the hard goals are achieved.

Since the durations of actions can be longer than expected, the goals might not be achieved when executing a plan that is expected to satisfy them, due to failure to meet deadlines. In fact, during execution actions can fail for various reasons and the goals might become unachievable as a consequence. In this paper we do not focus on what happens when actions fail. Instead, we are interested in the possibility that a conservative assumption about the time required to execute actions used in the original plan might lead to slack time that can be used to pursue opportunities.

In this paper we formalise the opportunistic planning problem, we propose a way to obtain good quality solutions to it and we compare the proposed approach with the simple alternative to replan whenever the observed state diverges from the predicted state during execution.

5 The Opportunistic Planning Model

We present a formal description of the opportunistic planning problem. We assume that P is a temporal planning problem, consisting of a *domain* and a *problem instance*, expressible in PDDL2.2 (Edelkamp and Hoffmann 2004). The domain provides a finite, enumerated type representing locations, W, in a 3-dimensional space. In the PDDL family of languages, the members of this type are all explicitly named in the definition of the planning problem instance. We suppose that P represents a problem in which goals are associated with locations (for example, pillars are located at waypoints), so that the executive must visit those locations in order to complete the achievement of the goals. We further suppose that the domain file of P contains at least one action schema that allows an executive to move between locations (possibly subject to accessibility constraints, restricting which pairs of locations are directly connected).

Definition 4 *An* **opportunity** *is a tuple, $\langle T, Og, U\rangle$, where T is the name of a* PDDL *enumerated type in P, (x, y, z); Og is a goal, with one free variable, v of type T; and U is a utility value in $\mathbb{R}$. The goal $Og[v]$ is called an opportunistic goal.*

An opportunity is a soft goal schema that is associated with objects of a particular type, T, appearing in the domain of P. The idea is that instances of T can be discovered and added to the world during plan execution, each leading to the creation of a new soft goal by instantiation of free variable v in the opportunistic goal, Og. For example, Og might be an inspection goal, and v might be instantiated by the object "pillarA" of type Pillar, resulting in a new soft goal to have inspected pillarA. In this work, we assume that soft goals always correspond to performing operations on single objects. An opportunistic planning domain consists of the original domain, P, and a collection of opportunities.

In a real world situation, opportunities are distributed in some way around the physical area being explored. In a simulation, they can be placed randomly around in the space. In both cases, they exist to be discovered, but are not modelled by the planner. They arise when new objects are identified at locations that may have been previously unmapped and inaccessible. If an opportunity is present, it can only be discovered if the executive passes within sensing distance of its location (a distance dependent on the type and effectiveness of sensors available) and with some associated probability, which is unknown.

The modelling language PDDL2.2 provides a feature called Timed initial literals (TILs) which record, in the problem instance description, time windows during which goals are achievable.

Definition 5 *An* **opportunistic planning problem** *is a tuple $\langle P, I, G, A, Opps, R\rangle$, where P is a temporal planning problem (as described above), I is the initial state (including timed initial literals that determine deadlines for goals), G is a set of hard goals (they must all be achieved in any goal state), A is a distinguished subset of actions in P that are preemptible, $Opps$ is a set of opportunities, and R is a function giving the mean and standard deviation of the duration of any grounded instance of an action in P. The durations of action instances are specified at the 95th percentile of the distributions whose parameters R reports.*

If an opportunity is discovered, replanning is initiated and an extended initial state is constructed. If the opportunity is discovered at time t, the TILs in the extended initial state must be displaced by t (to allow for the time that has passed since the start of execution). Any TIL with time earlier than t are discarded and those later than t have their times reduced by t. We call these *time-corrected* TILs. For example, consider a TIL with an original time of t, when replanning is initiated 30 minutes into the execution of the plan. The TIL t occurs at $(t - 30)$ minutes from the extended initial state. Hence, the time of the *time-corrected* TIL is $(t - 30)$.

Definition 6 *A* **monotonic extension** *of an initial state description, I, extends I with a new collection of objects and waypoints, $O = \{o : T\} \cup \{w : W\}$ and facts F, such that each $f \in F$ includes at least one object in O, and new soft goals Og, formed by grounding the opportunities associated with type T using objects in O. Connectivity is added, linking the new waypoints so that the newly added opportunity can be reached. The extended initial state, I', adds O to the objects in I and records the opportunity utility as a reward for actions achieving goals in Og. I' contains all the facts and time-corrected TILs in I as well as facts F.*

The extended initial state will locate discovered opportunities at new locations and new paths will be available by which they can be accessed. According to the topography of the space, some paths might require additional intermediate locations to have been added to the state.

Definition 7 *An* **opportunistic plan fragment** *in state S is a plan constructed to achieve a grounded opportunistic goal from state S.*

Our approach integrates opportunistic plan fragments with the original plan, in order to exploit an opportunity within the context of achieving the hard goal set. Opportunistic plan fragments, once integrated with a plan, can be visualised as sub-plans (which might be long chains of actions) that branch off from the main plan trajectory, finally returning to the main plan at a point enabling its continued execution to result in the achievement of the hard goal set. The means by which this integration is achieved is discussed in Section 7.

6 Relationship to other Probabilistic Models

Earlier work (Fox and Long 2002; Gough, Fox, and Long 2004) explores a different model of opportunities in which the opportunities and their locations are known in advance of starting the execution of the plan. Opportunistic plan fragments are computed offline, and executed online if their resource requirements are met. Woods et al. (Woods et al. 2009) assume that the *types* of opportunities that can arise are known, and that all opportunities of the same type can be exploited by the same opportunistic plan fragment. Plan fragments are precomputed and stored in a plan library. The relevant plan fragment is then inserted into the plan whenever an opportunity of its type is identified, and resources allow.

The opportunistic planning problem can be seen as a special case of a Partially-Observable Markov Decision Problem (POMDP), with an infinite state space (due to the continuous 3-dimensional distribution of locations of possible opportunities). A general solution to such a problem is a policy, mapping each possible state to an action. If the probability distribution over the opportunity space were known, the problem could be modelled as an explicit POMDP (Kaelbling, Littman, and Cassandra 1995b). Whether or not to pursue an opportunity in a certain belief state amounts to whether the expected utility of pursuing the opportunity, in addition to achieving the hard goal set, all within the resource envelope available, exceeds the expected utility of completing the current plan under execution with lower resource pressure. The problem can be modelled but, even with recent work on improving efficiency (Ong et al. 2009; Pineau, Gordon, and Thrun 2006b; Ross et al. 2008), the decision-theoretic approach will not scale to the sizes of problems that arise in practical applications. Moreover, the offline decision-theoretic reasoning cannot be done at all in the absence of knowledge about the probability distribution over the opportunity space. A further problem in creating a POMDP model is that states must record histories in order to capture the fact that repeated observations of a part of the physical space do not have independent probabilities of leading to discovery of an opportunity: if nothing is seen on one observation, then the probability that there is anything there to be seen is much lower. Finally, the continuous space presents a very significant challenge in representing the state space, since we cannot know in advance which locations are of interest, or, therefore, which areas of the space might be observed or even become accessible.

Although a POMDP model appears very difficult to realise and a full policy impossible to achieve with current approaches, a partial policy might be more tractable. One possible partial policy structure is a *contingent plan* in which alternative branches are built explicitly into the plan structure (Pryor and Collins 1996; Drummond, Bresina, and Swanson 1994). Contingent planning is very expensive, so various methods have attempted to limit the number of contingent branches constructed. In particular, Coles (Coles 2012) considers over-subscription planning with resource uncertainty. In her approach, the configuration of the world and all goals, including opportunities and their locations, are known in the initial state, which makes it unsuitable for tackling the problem we have characterised.

Burns *et al* (Burns et al. 2012) use an online receding horizon approach to consider anticipatory on-line planning in which plans take into account goals that are likely to arise, in order to be better prepared for achieving them efficiently. This is a relevant idea, but the difficulty in applying it to the problem we present is that, in our model, opportunities are assumed to be rare, making it unlikely that investment of resource in searching for an opportunity, rather than in simply completing the main mission goals, will pay dividends.

All of these approaches rely on some knowledge of the PDF over opportunities. By contrast, *replanning* does not require any knowledge about probability distributions, either over the opportunity space or over the use of resources by actions. In a reactive online method, a replanning strategy responds to an opportunity by throwing away the plan under execution, and building a new, conservative, plan for the union of the hard goal set and the opportunity. It then executes this plan instead, whenever its available resources are sufficient to achieve the new goal set.

Replanning is therefore a plausible approach to our problem. However, we hypothesise that replanning will be unnecessarily expensive, because it will replan parts of the problem for which there is already a detailed, and resource-valid, plan structure in place.

7 The Proposed Approach

We propose an approach to solving the opportunistic planning problem (Definition 5) as described in Section 5. Our approach tackles opportunities (Definition 4) by inserting opportunistic plan fragments (Definition 7) into an existing robust plan, generated using conservative planning.

In our implementation of opportunistic planning, we consider the distributions of the action durations and plan at the 95th percentiles of these distributions. This provides a stable baseline for robust confidence in the completion of the plan. Conservative planning seems an inefficient way of allocating time to tasks, but this apparent inefficiency is offset by the fact that plan utility is likely to be improved upon at execution time. The executive may decide to use any resource gained during execution to carry out extra tasks, such as pursuing opportunities, on top of the basic plan.

We manage execution of an opportunistic plan via the use of an execution stack. When a decision is made to pursue

an opportunity, the tail of the executing plan is pushed onto the stack. The initial state is monotonically extended (Definition 6) and replanning is initiated so that an opportunistic plan fragment is constructed. As long as this successfully completes within the planning time bound, and the resulting plan fragment can be executed within allocated resources, execution of the opportunistic plan fragment begins. When the opportunistic plan fragment has finished executing, the remainder of the main plan is popped off the stack, and its execution is then resumed. With this execution method, it is possible for an opportunistic plan fragment under execution to be suspended and stacked, if a new opportunity is detected during its execution. This is illustrated in figure 2.

When an opportunistic plan fragment is incorporated, some steps from the main plan might become redundant. In this case, some reasoning is needed to return to the latest possible state on the main plan trajectory (obviating as many redundant steps as possible). There is much work on the task of *plan merging*, e.g. (Alami et al. 1998; Alami, Ingrand, and Qutub 1998). In our approach we simply prune redundant steps and insert the plan fragment. In particular, when an opportunity is planned, the main plan suffix is pruned by removing all of the navigation actions at the front of the suffix. Figure 1, part (a), shows an opportunistic plan fragment that has been inserted into the plan, while part (b) shows the structure of a contingent branching plan. It can be seen that, in principle, opportunistic planning explores many fewer states.

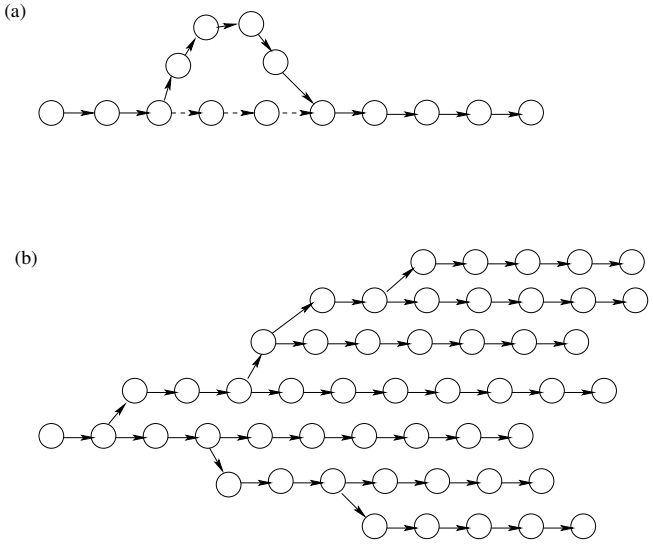


Figure 1: (a) The main plan with an opportunistic plan fragment attached. The fragment rejoins the plan suffix at the first necessary point for completion of the hard goal set. (b) The structure of a contingent plan. Each branch leads to a different goal set, depending on resource availability at the branch nodes.

7.1 The Implementation

In our implementation, the types of opportunities that can be identified are *inspections* and *investigations*. In particular, we restrict our attention to pillar inspections and a particular kind of investigation called chain-following. New objects of types Chain and Pillar are detected during AUV operations. When a new object is spotted, a new opportunity is created, by instantiating the corresponding opportunistic goal, as described in definition 4. The consequent construction processes, by which the extended initial state and the new soft goal are set up, are described in Definition 6. Our implementation of the Opportunistic Planning method behaves as follows, and is detailed in Algorithm 1:

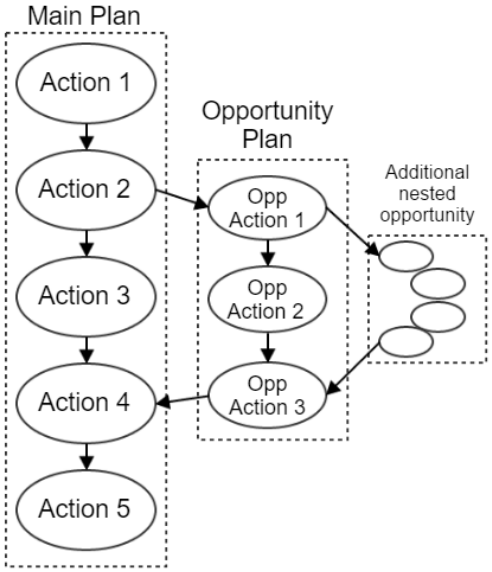


Figure 2: Plan execution with opportunity insertion. Actions 2, 3, and 4 of the main plan are navigation actions (or more generally "support actions") which are subsumed by the opportunistic plan fragment, and may be skipped. The opportunistic plan achieves the "weakest preconditions" of the tail end of the main plan, while adhering to the deadline constraints.

- construct a sequential strategic plan to achieve the goal set (top level missions) within a conservative resource bound;
- start executing that plan under operational control, keeping track of unspent resources;
- branch off the plan to handle an opportunity within the unspent resource bound, storing the plan suffix (this is recursive);
- return to the plan suffix as soon as possible.

The execution of this algorithm, showing the management of the plan stack, is shown in Figure 3.

A limitation of our approach so far is that we treat navigation actions as different from any other actions. They are only needed to move the AUV to places where tasks can be done, and are never in the plan to achieve top-level goals. They can therefore always be safely removed from the suffix as long as we can reach the next interesting waypoint after completion of an opportunistic plan fragment. Integrat-

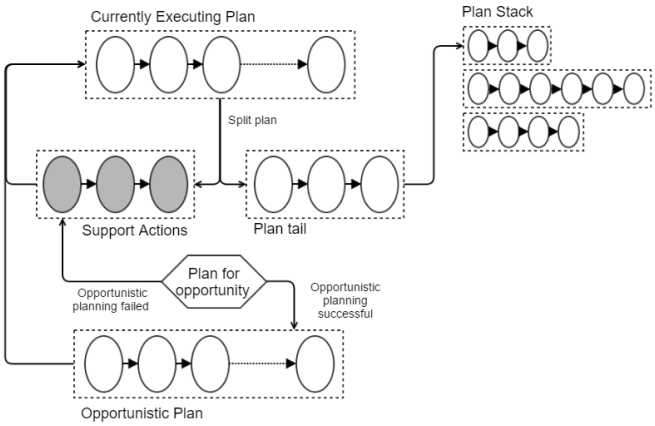


Figure 3: The execution of the algorithm, showing how plan suffixes are stacked. We start at the currently executing plan. When an opportunity is detected, the support actions are pruned and the plan suffix is stacked. Then we plan for the opportunity. If the opportunistic planning is successful, then the opportunistic plan becomes the new currently executing plan, otherwise the support actions are executed.

ing opportunistic plan fragments becomes more complex if other actions are preemptible, and we will consider such extensions in our further work.

In Algorithm 1:

- input: time limit in seconds, missionID identifies the mission goals;
- line 1: now() is the current time. (at AUV MissionEndPoint) is included as a goal;
- line 3: replanRequested can be set true by external processes;
- line 12: We set by hand which actions can dispatch early (all except for turn_valve);
- line 16: AUVs are busy while still executing an action;
- line 18: An action's default timeout can be chosen per operator, either as duration*T for some T, or duration+T for some other T;
- line 22: opportunistic_plan_requested is a communication variable that is declared and initialised externally, and then set true by an external process;
- lines 26-29: These lines find the finishing location for the opportunistic plan, and remove the "goto" actions from the parent plan;
- line 32: If the opportunistic mission was not possible, then the "goto" actions are reinserted at the start.

A final point about the implementation is that we do not currently use the utility component of an opportunity. This is because we have restricted the system to detecting and considering only one opportunity at a time, and an opportunity will always be pursued if time and resources allow. However, in general there might be several opportunities available, in which case a means is required for distinguishing them. Utility provides a way in which opportunities can be ranked for consideration. One approach would be to rank the opportunities by utility, then execute the first one in the ranking that fits into the available time and resources. Again, this is a topic for future work.

8 Experiments

Our hypothesis is that the opportunistic planning approach, just described, is more efficient than replanning the entire hard and soft goal set every time a new soft goal is identified. This might seem to be a "straw man" comparison, because it might seem obvious that replanning is apparently facing a much harder challenge than that oof planning to achieve a local opportunity within a well-defined context. However, this is not always the case. When replanning, the planner throws away all of the constraints of the defunct plan and has complete freedom about the timing of activities, as long as they fit within their respective time windows. This allows the planner to optimise activity around deadlines. By contrast, opportunistic planning has to fit all activity into the local time and resource envelopes of the global plan, which necessitates a *myopic* approach to opportunities (now or never), and might be over-constraining.

We therefore contrast our approach with a replanning method, to identify whether we gain any significant advantage, in terms of overall plan utility and resources spent planning, from the opportunistic approach.

In the case where no opportunity is observed during execution, the replanning strategy and the opportunistic planning strategy will both simply execute the main plan to achieve the hard goals, with no deviation (except in response to plan failure, which we ignore here). Therefore, the differences lie only at the point where an opportunity is discovered. In the replanning case, we construct a new initial state and replan for the entire goal set. In the opportunistic planning case, we plan only for the opportunity, together with a goal to return to the start of the plan suffix. Both approaches begin by constructing the monotonically extended initial and goal states. The opportunistic approach benefits from what is usually a simpler planning problem in exchange for los-

Algorithm 1: opportunisticPlanningMethod

```
input  : timelimit : Int, missionID : Int, missionEndPoint :
         Waypoint
output : boolean
problem ← generateProblemFile(now(), missionID,
  missionEndPoint);
plan ← makePlan(problem);
replanRequested ← false;
freeTime ← 0;
if plan.length() > timelimit then
    return false;
end
else
    while plan.length() > 0 do
        currentAction ← plan.pop();
        dispatchTime ← currentAction.dispatchTime;
        if !canDispatchEarly(currentAction) and
          now() < dispatchTime and !replanRequested then
            wait();
        end
        currentAUV ← currentAction.AUV;
        while AUV.isBusy() and !replanRequested do
            wait();
            if now() > currentAction.timeout then
                dispatch(cancelAction);
                replanRequested ← true;
            end
            if opportunistic_plan_requested then
                opportunistic_plan_requested ← false;
                currentEndPoint ←
                  currentLocation();
                prunedActions ← {};
                while plan.first() == "goto" do
                    currentEndPoint ←
                      plan.first().destination;
                    prunedActions.push_back(plan.pop());
                end
                plans.push_back(plan);
                if
                  !opportunisticPlanningMethod(freeTime,
                  opportunisticMissionID, currentEndPoint)
                  then
                    plans.insert(prunedActions, 0);
                end
                plans.pop(plan);
            end
        end
        if !replanRequested then
            dispatch(currentAction);
            freeTime ← now() - dispatchTime;
        end
        else
            replanRequested ← false;
            problem ←
              generateProblemFile(now());
            plan ← makePlan(problem)
        end
    end
end
return true
```

ing the possibility of finding a better plan by exploiting the remaining resources to achieve the opportunity and original goals together. Our experiments consider the situation at a point at which an opportunity has been discovered.

We perform the comparison by setting up a *main mission*, with hard goals, and an opportunity. The main mission is taken to be a valve-turning mission, possibly involving many valves, and the opportunity mission is an inspection (we do not consider investigations in this experiment). In the valve-turning mission, the AUV is required to approach and set two valves within a deadline. The effect of setting a deadline is to bound the resource available for exploiting opportunities. The inspection mission is not time-limited. When it arises as an opportunity, a plan to exploit it must fit within the available resource envelope. Inspection missions are of several sizes, ranging between 2 and 32 inspection points.

In our simulation, the main mission elements and the opportunities are located within an area 50m by 50m and set at least 5m apart. They are positioned successively, with uniform probability over the available area. The deadlines for valves are set to different values, making the planning problems harder as the deadlines are tightened. The opportunistic planning strategy requires the opportunity to be exploited within the free resource window, *before* the completion of other mission components. The replanning strategy does not require this, but both strategies require the overall plan to be completed by the mission deadline.

In Table 1 we report our results for a collection of randomly generated problem instances. The opportunistic planner is given 10 seconds to solve the problem. In general, the window of opportunity is short, partly because it is most often the case that we will discover an opportunity while navigating, in which case we do not want to stop the vehicle unless we decide to pursue the opportunity, and partly because the energy and computational resources on board the AUV is limited. It is also important that the time taken evaluating an opportunity should not be significant compared with execution time of actions, otherwise we endanger the main mission itself by wasting resources on multiple opportunity evaluations. This latter problem arises if the signal processing that leads to recognition of an opportunity is unable to determine that multiple sightings of the same object are actually not distinct opportunities.

The replanning strategy was allowed 30 minutes of CPU time to generate a best possible plan. We report the best plan found in that time, with the time it took to find that plan (POPF2 uses an anytime strategy of plan improvement, reporting plans as they are found).

The bolded results are the cases in which the combined mission is solvable with a higher quality solution within the 30 minute bound. In four of these cases, the replanning strategy would outperform the opportunistic strategy, but in the bold and italicised case, the plan takes so long to find that the combined planning and execution time exceeds the time available for the complete plan. Indeed, in almost all cases, the complete plan is so much longer than the opportunistic plan that it would not be possible to complete within the duration of the intended mission time for the whole problem.

Part of the difficulty for the replanning strategy arises

from the forward search paradigm of POPF2. The existence of deadlines leads to the planner pushing activity later along the time line than is appropriate and it fails to search the parts of the search space in which the short plans exist. In future work we will explore alternative temporal planning strategies in order to better understand the impact of this planning artefact on the qualit of the plans.

In one of our test cases the opportunistic planner failed to find a plan within 10 seconds, so the plan reverts to the main mission plan. In this case, the replanning strategy takes 3 minutes to find a plan that is far too long to be used in place of the main mission plan, so this represents a waste of the time spent in this attempt.

These results show very clearly that the cost of a complete replan is much higher than the cost of planning for an opportunity alone. Even though planning for the combined mission should offer, in principle, a chance to find a better quality solution than the one found by simply linking the opportunistic plan fragment to the front of the existing plan, the reality is that it is very hard to achieve this. A more capable planning strategy might be more successful in finding better plans, but the time taken to do so would certainly be far greater than the time required to find the opportunistic plan. Each such plan construction attempt spends the very resource that is required to exploit the opportunity itself, so it is an impractical approach to repeatedly evaluate opportunities by using a full replanning approach.

9 Conclusions and Future Work

In this paper we have defined the concept of *opportunistic planning*, a method for robust planning and plan execution under limited uncertainty. We have presented a fully implemented method for opportunistic planning of missions and the interleaving of mission execution with utility-increasing opportunities. The results of our experiments show that opportunistic planning is a good compromise between scalability and robustness, allowing the practical management of uncertainty. We have demonstrated that, in terms of time to plan and resulting plan utility, opportunistic planning significantly outperforms a replanning method.

In this paper we focus on long-term maintenance and inspection of underwater installations, using an Autonomous Underwater Vehicle (AUV). The locality of goals and the presence of low probability/high-reward opportunities make this domain an ideal target for the opportunistic planning technique. These aspects are also present in many other robotics domains (e.g. ground robots for disaster recovery). One avenue of future work is to investigate other types of scenario, to discover how the opportunistic planning approach could be generalised to other planning domains.

Our current approach to opportunistic planning demonstrates improvements over a replanning strategy, but has some limitations. In particular: we do not evaluate the expected gain, in terms of accumulated resource, of reducing our confidence in achievement of the hard goal set. For example if, at some point p, into the execution of a plan, we are willing to reduce our confidence in successful execution of the plan suffix to the 94th percentile, how much resource could we save for spending on an opportunity spotted at p?

As an alternative to allowing the expected accumulation of resources following the execution of a sequence of actions it would be possible to adjust the sum of the nominal durations to account for the length of the sequence. So, for k actions each with identical mean and standard deviation, the nominal durations can be reduced to $m + \frac{1.65s}{\sqrt{k}}$.

More generally, where several actions are sequenced to achieve a goal it is possible to discount the sum of the nominal durations to allow for the expected accumulated benefits of using the 95th percentile as the nominal durations of the individual components.

In our future work we intend to experiment with trading off confidence against utility, by doing this reasoning *online* at the point at which we have evaluated an opportunity. The actions in the plan suffix are not changed, but the confidence in completing it successfully is traded for the benefits of the opportunity. For a very high value opportunity it might even be worth, in order to free up more resource, requesting the sacrifice of a component mission from the command level planner.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *The International Journal of Robotics Research* 17(4):315–337.

Alami, R.; Ingrand, F. F.; and Qutub, S. 1998. A scheme for coordinating multi-robots planning activities and plans execution. In *ECAI*, 617–621.

Burns, E.; Benton, J.; Ruml, W.; Yoon, S.; and Do, M. B. 2012. Anticipatory Online Planning. In *Proceedings of 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20rd International Conference on Automated Planning and Scheduling (ICAPS'10)*, 42–49.

Coles, A. 2012. Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty. In *Proceedings of European Conference on AI (ECAI'2012)*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artif. Intell.* 49(1-3):61–95.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-Case Scheduling. In *Proceedings of 12th National Conference on Artificial Intelligence (AAAI)*, 1098–1104.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical report, Technical Report 195. Albert Ludwigs Universitat, Institut fur Informatik, Freiburg. Germany.

Fox, M., and Long, D. 2002. Single-trajectory opportunistic planning under uncertainty. In *Proceedings of the UK Planning Special Interest Group (PLANSIG'02)*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Res. (JAIR)* 20:61–124.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)* 27:235–297.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.* 173(5-6):619–668.

Gough, J.; Fox, M.; and Long, D. 2004. Plan execution under resource consumption uncertainty. In *Proceedings of the ICAPS Workshop on Connecting Planning and Execution*, 24–29.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1995a. Planning and acting in partially observable stochastic domains. *Articial Intelligence* 101:99–134.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1995b. Planning and acting in partially observable stochastic domains. *Articial Intelligence* 101:99–134.

Lee, B.; Preston, F.; and Green, G. 2011. *Preparing for High-impact, Low-probability Events Lessons from Eyjafjallajkull*. Chatam House.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – the planning domain definition language. Technical report, Yale Center for Computational Vision and Control.

Ong, S.; Png, S.; Hsu, D.; and Lee, W. 2009. POMDPs for robotic tasks with mixed observability. *Robotics: Science and Systems*.

Paulos, J.; Eckenstein, N.; Tosun, T.; Seo, J.; Davey, J.; Greco, J.; Kumar, V.; and Yim, M. 2015. Automated self-assembly of large maritime structures by a team of robotic boats. *IEEE T. Automation Science and Engineering* 12(3):958–968.

Pineau, J.; Gordon, G.; and Thrun, S. 2006a. Anytime point-based approximations for large POMDPs. *Journal of Articial Intelligence Research* 27:335–380.

Pineau, J.; Gordon, G.; and Thrun, S. 2006b. Anytime point-based approximations for large POMDPs. *Journal of Articial Intelligence Research* 27:335–380.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research* 32:663–704.

Sanner, S., and Boutilier, C. 2009. Practical Solution Techniques for First-Order MDPs. *Artificial Intelligence* 173(5-6):748–788.

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an ExoMars Rover-like mission. *J. Field Robotics* 26(4):358–390.

Mission		Opp plan	Full replan	Plan duration		
Main	Opp	time	time	Opp Mission	Complete Opp Plan	Replanned plan
V2_400	I_16	0.36	38.18	851.384	1265.032	2437.496
V2_500	I_16	5.54	7.46	1541.168	2076.155	2596.156
V2_600	I_16	5.34	7.28	1541.168	2117.136	2269.701
V2_700	I_16	5.32	9.56	1541.168	2117.136	2283.134
V2_800	I_16	5.38	6.24	1541.168	2117.136	**2048.833**
V2_900	I_16	5.4	9.16	1541.168	2117.136	**1900.069**
V2_1000	I_16	0.38	21.42	851.384	1265.032	2615.245
V2_1100	I_16	0.34	7.28	888.554	1302.202	2048.833
V2_1200	I_16	2.4	11.9	1440.568	1854.216	2511.960
V2_1300	I_16	0.36	6.34	851.384	1265.032	2772.985
V2_1400	I_16	0.42	6.28	851.384	1265.032	2772.985
V2_1500	I_16	0.34	7.82	851.384	1265.032	2946.391
V2_1600	I_16	0.38	14.54	851.384	1265.032	2175.901
V2_1700	I_16	0.4	15.6	851.384	1265.032	2897.665
V2_1800	I_16	0.42	6.24	851.384	1265.032	2772.985
V2_1900	I_16	0.38	6.44	851.384	1265.032	2772.985
V2_2000	I_16	0.36	2.62	851.384	1265.032	2490.490
V2_400	I_32	5.08	148.17	2233.961	2564.254	3531.784
V2_500	I_32	2.2	165.62	1768.98	2129.213	5332.514
V2_600	I_32	3.7	78.19	1777.177	2137.41	3623.974
V2_700	I_32	4.08	272.84	1815.849	2176.082	4877.45
V2_1000	I_32	4.66	104.04	2686.638	3093.992	4263.605
V2_2000	I_32	4.32	100.16	2457.922	2865.276	3778.601
V2_2500	I_32	4.67	68.78	2457.922	2865.276	4212.37
V2_3000	I_32	4.36	132.32	2469.124	2876.478	3948.493
V2_3500	I_32	4.21	119.14	1861.244	2191.537	4925.07
V2_5000	I_32	5.16	81.04	1997.34	2327.633	5460.531
V2_1000	I_2	0.02	3.36	141.138	678.167	**580.58**
V2_1000	I_6	0.06	182.02	374.415	911.444	***774.337***
V2_1000	I_8	0.06	4.82	504.346	863.481	977.001
V2_1000	I_10	0.1	135.62	582.795	1017.846	1383.791
V2_2000	I_10	0.1	7.44	700.198	1294.007	1585.303
V2_2000	I_12	0.14	3.38	772.926	1545.852	**1414.551**
V2_2000	I_14	0.14	3.70	675.458	1141.406	2040.448
V2_2000	I_16	0.14	2.60	676.458	1142.406	2490.490
V2_2000	I_18	0.18	44.68	878.395	1470.264	3116.591
V2_2000	I_20	0.44	15.18	1231.22	1767.021	3358.226
V2_2000	I_22	1.08	17.42	1752.10	2152.423	4114.774
V2_2000	I_24	0.98	34.48	1643.92	2017.172	2914.554
V2_2000	I_26	2.6	289.40	2141.87	2485.068	6029.480
V2_2000	I_28	3.38	265.72	3088.31	3667.984	5987.822
V2_2000	I_30	-	179.76	-	398.45	4187.185
V2_2000	I_32	4.14	218.74	2689.84	3057.283	4475.005
V2_2000	I_34	5.24	89.32	3125.49	3621.695	4615.062

Table 1: Table of experimental results. Planning time and plan durations are measured in seconds.

Goal Reasoning with Informative Expectations

Benjamin Johnson and **Mark Roberts**
NRC Posdoctoral Fellow
Naval Research Laboratory
Washington, DC
firstname.lastname.ctr@nrl.navy.mil

Thomas Apker and **David W. Aha**
Navy Center for Applied Research in AI
Naval Research Laboratory (Code 5514)
Washington, DC
firstname.lastname@nrl.navy.mil

Abstract

In complex and dynamic scenarios, autonomous vehicles often need to intelligently adapt their behavior to unexpected changes in the their environment. Goal Reasoning provides a methodology for autonomous agents to deliberate and adapt their goals to more intelligently react to changing conditions. This paper implements a Goal Reasoning system based on the Goal Lifecycle, and grounds the implementation in the information measures and expectations used by the vehicles to asses their performance. The implemented system, termed Goal Reasoning with Information Measures (GRIM), is demonstrated using a disaster relief scenario in which a small team of vehicles is tasked with surveying a pre-defined set of geographical regions. This demonstration shows how area search goals can be progressively refined, and how they can be adapted to resolve problems encountered by the vehicles during execution.

1 Introduction

Complex applications of robotics often require one or more autonomous vehicles to react intelligently to changes in the operational scenario. A change in the observed state of the environment (e.g., the robot senses an unexpected event) or the internal state of the vehicle (e.g., the vehicle is consuming fuel faster than expected) may necessitate a change in the robot's behavior. This change can manifest as a change to the vehicle's plans, tasks, or even the underlying goals that it is trying to achieve. Intelligent adaptation to unexpected changes is vital to the design of autonomous systems for complex applications.

Goal Reasoning (GR) research aims to develop autonomous agents that can deliberate on, and change, their own goals (Vattam et al. 2013). Such agents would be able to adapt to new and unexpected observations about their environment by creating new goals to pursue. Similarly, they would be able to modify their existing goals to account for unplanned changes by reprioritizing and reordering their goals. Such capabilities become even more valuable in applications where multiple robots must act collaboratively to accomplish their goals; GR would allow the robots to adjust their goals to align with those of the other agents, or to leverage the assistance of other robots.

An example application that would benefit from GR is the control of autonomous vehicles in the Foreign Disaster Relief (FDR) domain (U.S. Department of Defense 2011). The FDR domain focuses on providing humanitarian aid in the wake of natural disasters, and is an area that could greatly benefit from the deployment of autonomous vehicles. In such situations, autonomous vehicles could be used to provide rapid surveys of the disaster area, identifying important locations and traversable routes for the responders. Additionally, the vehicles could be used to enhance the reliability and range of communications, by serving as mobile communication relay points.

To apply GR techniques to applications like FDR operations, it is important for them to be grounded in the information and capabilities that regulate the behavior of the vehicles. That is, deliberation about the goals of an autonomous agent should be performed based on the metrics that define and govern those goals. This paper investigates that by grounding the systems described in Roberts et al. (2015a)[1] and Apker, Johnson, and Humphrey (2016). This grounding frames the goal refinement process in terms of information gathered during the execution of the goal, and is implemented in a system called Goal Reasoning with Information Measures (GRIM). The GRIM system also includes a set of strategies to resolve problems that arise during execution.

The work here presents early steps in creating a full GR system for a team of robots assisting with FDR operations with multiple, possibly conflicting, goals for the system. The paper demonstrates a multi-vehicle system performing GR with respect to a set of area-survey goals. An extended version of this paper, including a preliminary evaluation of the effectiveness of the RESOLVE strategies, can be found in Johnson et al. (2016).

[1] ActorSim, an implementation of the Goal Lifecycle, is available online at `http://makro.ink/actorsim/`

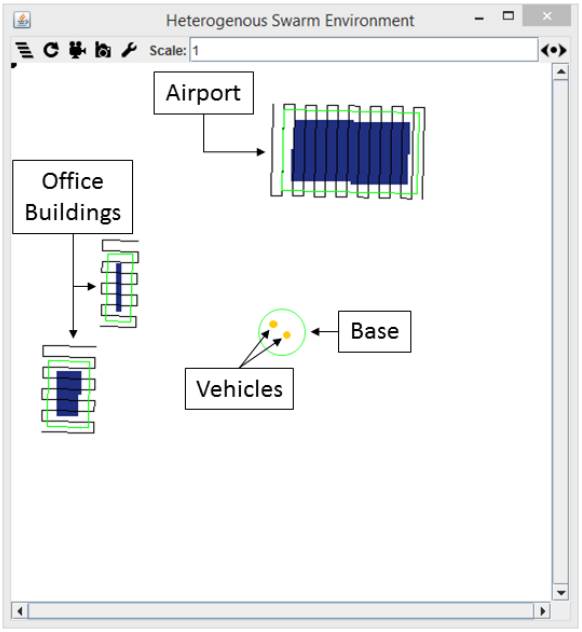


Figure 1: Map of the example scenario. Two vehicles begin in a base region, and are tasked with surveying three different regions of interest: the Airport and the two Office Buildings.

The paper is structured as follows. Section 2 describes a motivating example. Section 3 provides a more in-depth description of GR and the instantiation that is extended in this work. Section 4 demonstrates the GRIM system with respect to the motivating example. Section 5 discusses this work in the scope of other, related work, as well as the future expansion of the system. Finally, Section 6 concludes the paper.

2 Motivating Example

Consider, as a motivating example, an unmanned FDR mission where a team of Unmanned Air Vehicles (UAVs) must survey several pre-defined regions to identify and locate an important official. Figure 1 shows a map of such a scenario, in which a team of UAVs must survey three different regions, each with different characteristics. The Airport is the largest of the regions, composed primarily of flat, open space. Each of the two Office Buildings, on the other hand, are significantly smaller and have considerably more complicated terrain. There is also a Base region, where the UAVs begin the scenario.

The system's first task is to search the regions to locate the official. Once the location of the official is known, the system must then establish and maintain a communications relay for that official. The establishing of a relay is made more difficult in the Office Buildings (in comparison to the Airport) due to the more complicated and cluttered terrain.

These goals are further complicated by other restrictions and factors involved in the mission, such as:

- The need for the UAVs to refuel at the nearby base station.
- Changes to the set of resources (e.g., vehicles) that are available to the system.
- The presence of uncontrolled or adversarial environmental factors (e.g., wind or road blockages).
- Additional goals, with varying or dynamic priorities/importance (e.g., the discovery of a medical emergency that must be immediately addressed).

The system controls a team of two UAVs and can assign them to any of the search areas. Furthermore, due to uncontrolled factors, it is assumed that the vehicles will under-perform their expectations during execution, resulting in slower-than-expected searches of the areas.

3 Goal Reasoning and the Goal Lifecycle

GR focuses on developing agents that can deliberate on and modify their goals during execution within a dynamic environment. The work presented here leverages and adapts the *Goal Lifecycle* of Roberts et al. (2014), Roberts et al. (2015a), and Roberts et al. (2015b), an adaptation of which is shown in Figure 2. This provides a framework for the refinement of goals and the resolution of problems that arise during execution.

The set of goals ("goal nodes" in (Roberts et al. 2015b)) G are stored in a data structure called the Goal Memory. GR is performed by transitioning each goal $g \in G$ through the *modes* (represented by boxes) of the Goal Lifecycle via *strategies* (the arcs). Progression through the modes in the Goal Lifecycle represents increasing refinement in the goal detail. The step-like structure of the goal modes enforces the concept that each mode builds on the previous mode: each transition strategy can only occur from specific modes in the Lifecycle.

The remainder of this section briefly summarizes the key strategies of the Goal Lifecycle; specific details are given in Section 4, as the strategies relate to the goals for the motivating scenario described in Section 2. For the remainder of the paper, transition *strategies* are denoted with small caps (e.g., FORMULATE) and the resulting goal *modes* are denoted by monospace small caps (e.g., `FORMULATED`).

The FORMULATE strategy determines when a new goal g is created and enters the Goal Lifecycle from an external source (e.g., user input, or a triggering event). This strategy takes an abstract goal as an input, and transitions it to a `FORMULATED` mode, by defining the initial constraints, the measures defining success or failure, and its prerequisites. The result of FORMULATE is that a new

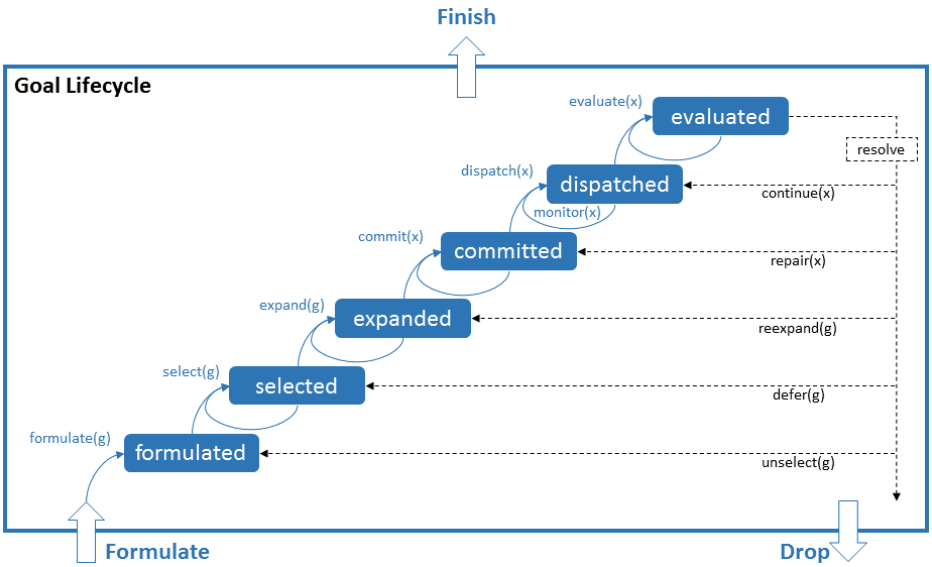


Figure 2: An adapted version of the Goal Lifecycle, from Roberts et al. (2015a). Goals transition through the modes (boxes) via the strategies (arcs), where higher-level modes represent a higher level of goal refinement.

goal is entered into the Goal Memory with the information (i.e., constraints , measures, and prerequisites) required for further refinement.

The SELECT strategy takes a FORMULATED goal g, and determines whether the system activates it. A goal transitions to SELECTED (meaning that it will be actively pursued by the system) only if its prerequisites are satisfied and the system has the available resources to pursue, both of which are defined by the FORMULATE strategy. As such, some goals may not be SELECTED and will instead remain in the FORMULATED mode until their prerequisites are met and the required resources become available.

The EXPAND strategy takes a SELECTED goal g and generates one or more expansions (i.e., plans) $x \in X$ to achieve the goal. An EXPANDED goal defines how the system can satisfy the constraints that were created during the FORMULATE strategy, and generates expectations for how each expansion will perform when executed. If one or more feasible plans are created, the goal transitions to an EXPANDED mode. Otherwise, the EXPAND strategy fails and the goal remains in the SELECTED mode.

Once a goal g has been successfully EXPANDED the COMMIT strategy chooses one of the feasible expansions x for execution. Such a choice involves assessing the costs of each of the expansions, as well as the likelihood that they will successfully execute the goal (per the FORMULATED constraints). A COMMITTED expansion defines how the system will satisfy g, and provides the set of expectations for the performance of the plan's execution.

The DISPATCH strategy sends the COMMITTED expansion x to the executive to run. This process amounts to allocating resources and generating metrics for plan execution. A successfully DISPATCHED expansion defines the criteria by which a goal is evaluated to ensure that it detects and reacts to discrepancies in the expected performance of the expansion.

During execution, two strategies manage updates that impact the mode of the goal g. First, EVALUATE is a passive strategy that is called whenever new information impacts the goal. It can be called by an external process or by the goal itself. In contrast, the MONITOR strategy, when enabled, proactively tracks the execution of the DISPATCHED expansion x to ensure that its expected performance will still result in successful completion of the goal. Additionally, MONITOR ensures that the prerequisite conditions for the goal remain met and that the allocated resources remain available. If MONITOR detects a problem, it triggers EVALUATE directly, and x progresses to an EVALUATED mode, indicating that there is some discrepancy in the performance of the expansion that should be addressed.

When a goal transitions to EVALUATED, the RESOLVE strategy assesses any discrepancies that were detected, and determines how the system should resolve the discrepancy (i.e., which mode the goal should transition to). If the EVALUATED goal g is determined to have met all of the constraints and success-conditions that were generated during the goal formulation, it is resolved with FINISH and marked as completed. If g violates the formulated constraints, DROP marks it as unsuccessful (it can then be reformulated with new constraints). Both

FINISH and DROP results in the removal of g from Goal Memory.

Otherwise, if g still meets its formulated constraints but does not meet its success conditions, the RESOLVE strategy transitions the goal to one of the earlier modes in the Goal Lifecycle. If EVALUATE determines that the DISPATCHED plan x is still feasible, the goal is resolved back to the DISPATCHED mode (referred to as CONTINUE). If the COMMITTED plan can be fixed without major changes by reallocating system resources, it resolves back to the COMMITTED mode (REPAIR). If the committed plan is infeasible, but another feasible plan $\bar{x} \in X$ exists, the goal g is resolved back to the EXPANDED mode and commits to a different, feasible plan $\bar{x}$ (REEXPAND). If no feasible expansion exists given the currently available resources, g is resolved back to the SELECTED mode, where it can be expanded once the necessary resources become available (DEFER). Finally, if the goal no longer meets its prerequisites for selection, but it still satisfies its constraints, it is resolved back to a FORMULATED mode until the prerequisites for selection are met once again (UNSELECT).

4 Goal Refinement with Information Metrics

Goal Refinement for unmanned FDR missions can be framed in terms of refining a set of constraints and expectations for a set of measurable information measures. This section describes a GR system, called Goal Reasoning with Information Measures (GRIM), and demonstrates this system via simulation. The GRIM system instantiates the Goal Lifecycle (discussed in Section 3), and provides centralized control for a small team of 2 UAVs. Returning to the motivating example, described in Section 2, the goal refinement strategies of the Goal Lifecycle are defined here for the area search goals, while the relay goal (which will be the focus of future research) is only included as an abstract goal (i.e., it is not specified below). The information measures for an area search goal, defined in this section, describe the degree to which the defined region has been "searched".

The metric used to evaluate the uncertainty in an area search will differ based on the sensors and algorithms used to conduct the search. For simplicity, the vehicles conduct searches in this example by following a lawnmower waypoint pattern, and the information measure used is the length of that search pattern that has yet to be traversed, though this metric could easily be adjusted to a more accurate measure of uncertainty. This measure was chosen as a simple approximation for the information gathered during the survey task, and future work will explore more accurate measures of the uncertainty in an area survey.

Formulate

The FORMULATE strategy, in the case of the area survey, defines three parameters that describe the constraints under which each can be considered as successful or failed. These parameters are:

1. *maximum uncertainty*: the upper bound on the uncertainty in the search area (i.e., the uncertainty of the area before any information has been gathered),
2. *acceptable uncertainty*: the level of uncertainty at which the goal is considered complete, and
3. *deadline*: the time by which the search must be complete.

The maximum uncertainty specifies amount of uncertainty in the search area prior to any search, and represents the total information that can be gathered about an area during a survey. The acceptable uncertainty parameter defines the level of uncertainty at which the area can be reliably deemed to be empty of an official (i.e., a finishing criteria for the search). Finally, the deadline is set by estimating the time required to arrange follow-on interactions with a located official, as a function of the area's type and terrain. It defines the point in time at which the search must be finished, in order for it to be considered successful. For both (1) and (2), the constraints are defined by the metric used for uncertainty: the length of the search path (in meters) that has yet to be traversed by the vehicles. For (3), the constraint is defined in terms of mission time (seconds).

Figure 3 displays the constraints on each of the formulated goals as a function of the area uncertainty (in meters of untraversed search path) and the execution time. Each of the dashed lines in this figure represents the allowable area of uncertainty for a survey area (i.e., the Airport and the Office Buildings) at a given time. Because of the more complicated interactions of the Office Buildings, the deadline (i.e., the time at which the area uncertainty must be within the defined acceptable level of uncertainty) for each of these is earlier than the deadline for the Airport. At all times up until the deadline, no constraint is placed on the allowable area of uncertainty, so it is set as the full length of the search path for each of the search areas (28,267 meters for the Airport, and 10,303 and 7,537 meters for the smaller Office Buildings). At the deadline, the area of uncertainty is required to not exceed the defined acceptable level of uncertainty. This acceptable level of uncertainty was defined as 500 meters for each of the search areas, but is omitted from Figure 3 for clarity.

The result is a FORMULATED goal $g \in G$, which defines the constraints on the execution of the goal (maximum uncertainty and deadline), as well as the criteria for successful completion of the goal (acceptable uncertainty).

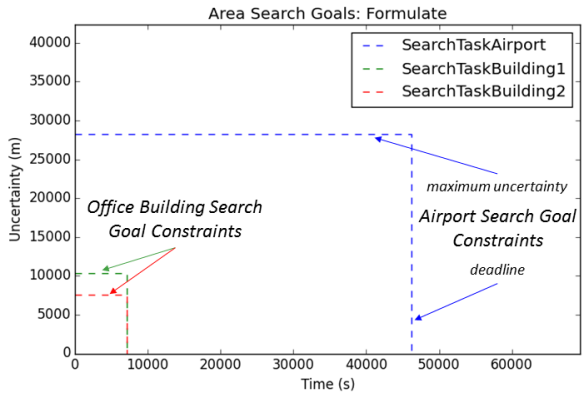


Figure 3: Maximum uncertainty and deadline constraints for the Airport and Office Building search goals. The performance of each goal must remain within the pictured constraints during execution, by reaching the acceptable uncertainty level before the deadline.

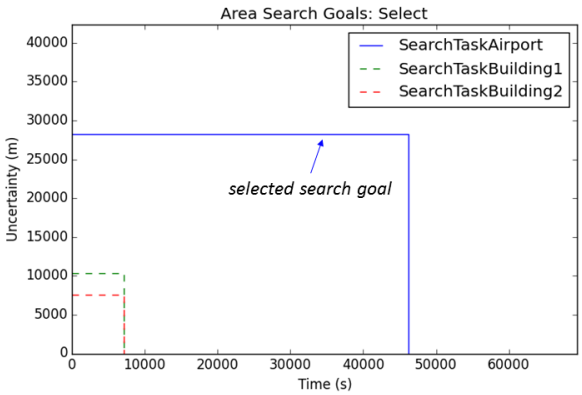


Figure 4: GRIM selects the Airport search goal g, while the Office Building search goals remain unselected. The selected search goal progresses progresses to the EXPAND strategy.

Select

The next strategy in the Goal Lifecycle is for GRIM to SELECT one or more goals to pursue. The SELECT strategy requires comparing high-level estimates of expected performance and value (cost/reward) for each goal, and assessing the available resources for the system. This strategy determines which goals are operational (i.e., those that are selected). For the motivating example presented here, only a single goal is allowed to be SELECTED at a given time.

Figure 4 shows the results of this process, where GRIM elects to pursue the Airport search goal because it is deemed most likely to be successfully searched within the formulated constraints. For this example, the SELECT strategy choses the goal with the smallest ratio of $\frac{max\ uncertainty}{deadline}$. Due to the significantly longer deadline, the ratio for the Airport is (despite the larger uncertainty) smaller than the ratio for the Office Buildings, and GRIM selects the Airport goal.

In short, a SELECTED goal g is one that is being pursued by the GRIM system via later strategies in the Goal Lifecycle, while a goal that is *not* SELECTED remains paused in the FORMULATED mode.

Expand

After selecting a particular goal g, EXPAND generates a set of plans[2] X to accomplish it. For each plan $x \in X$ that is generated, a set of expectations is also generated that describe its expected performance with respect to the metrics used in the formulation strategy. A successful EXPAND strategy generates at least one feasible plan (i.e., it is expected to satisfy the formulated constraints). In the example used here, a feasible plan is one where the expected value of the uncertainty at the deadline is no greater than the defined acceptable uncertainty.

Figure 5 displays the expectations of four feasible plans that are generated during the EXPAND strategy, for the selected goal g. The expectations are shown as the expected change in the uncertainty of g (i.e., the length of the search pattern that has been traversed) over time, and each expanded plan uses the same set of waypoints. The resulting plans are for a single vehicle moving at normal speed ("1vehicleNorm"), a single vehicle moving at a faster speed ("1vehicleFast"), two vehicles moving at normal speed ("2vehicleNorm"), and two vehicles moving at a faster speed ("2vehicleFast"). It is assumed that a faster vehicle speed improves the search rate at the cost of higher fuel consumption and that, for each plan, the bulk of the UAVs effort will be expended determining where the official is not located (i.e., a nearly complete search of the area will be required).

The result is an EXPANDED goal g, which has one or more feasible plans $x \in X$, each with a set of performance expectations that satisfy the completion criteria generated by the FORMULATE strategy.

Commit

Once the goal g has been EXPANDED into a set of feasible plans, GRIM must COMMIT to a single plan. To do so, it assesses the costs (i.e., the expended resources, including time) of each feasible plan, and commits to the least costly plan.

Figure 6 highlights the COMMITTED plan ("1vehicleNorm"), which was chosen because the expectations lie well within the formulated constraints while conserving the most resources. By using only a single vehicle,

[2]The original Goal Lifecycle (Roberts et al. 2015a) used the term *expansion* to refer to the possible plans that could be applied to a goal; the remainder of this paper uses the term *plan* interchangeably with the term *expansion*.

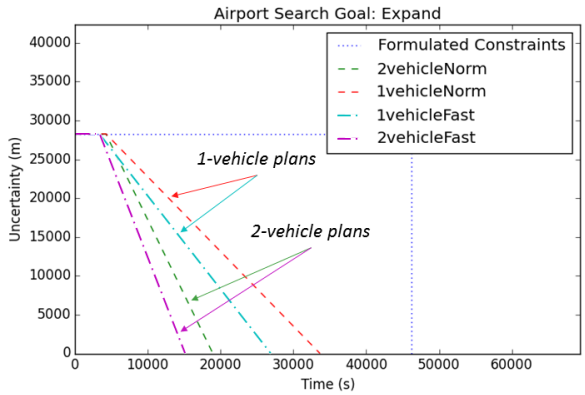


Figure 5: Plots of expected survey performance for each expansion x of the Airport search goal. The "fast" plans increase the vehicle speed to result in a quicker expected completion time, at a higher fuel cost.

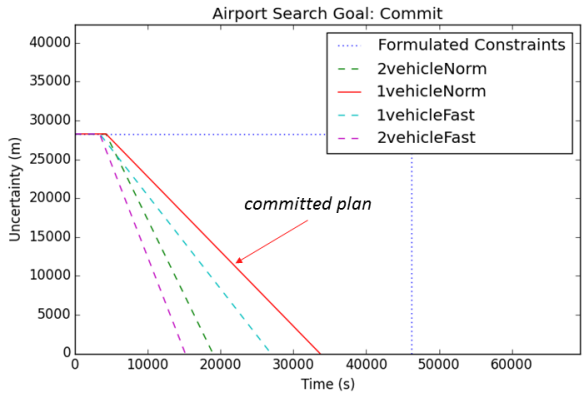


Figure 6: GRIM commits to the 1-vehicle, normal-speed expansion. This plan was selected to conserve resources while still resulting in successful completion.

GRIM leaves the second vehicle available in reserve, and by committing to the plan that moves the vehicle at the normal speed the system preserves fuel for other tasks, such as searching another area or providing a relay for a discovered official.

In short, the COMMITTED expansion $x \in X$ is the plan that GRIM chooses to enact in order to pursue the goal g.

Dispatch

Once GRIM has a COMMITTED plan, it must then DISPATCH that plan to the appropriate vehicles. To do this, it must also determine the expected performance bounds for successful plan execution. These bounds represent the worst-case scenario from which the plan can still be expected to satisfy the formulated constraints. To generate these bounds, GRIM uses the expected performance

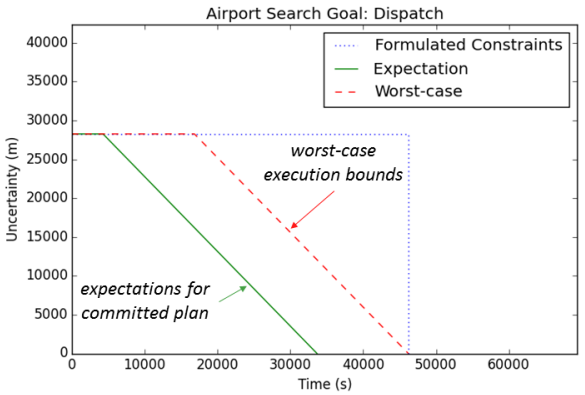


Figure 7: GRIM dispatches the committed expansion x (with expectations) to the applicable vehicle. As part of the plan dispatch, the system generates worst-case bounds on the successful performance of the plan.

of the plan adjusted such that it is expected to just barely satisfy the constraints; that is, the worst-case bounds represent the execution for which the expectations reach the acceptable level of uncertainty at the deadline. If the performance exceeds these bounds during execution, GRIM will need to adapt the goal g, by applying the RESOLVE strategies described in Section 3. Figure 7 displays the expectations and worst-case execution bounds of the dispatched plan.

In GRIM, a plan is dispatched by scheduling predefined commands for execution by vehicles. When the vehicles receive these commands, they are passed to a synthesized Finite State Automaton (FSA) that is running on the vehicle, and executed according to the rules that were used to synthesize that FSA. A more thorough description can be found in Apker, Johnson, and Humphrey (2016). In the case of the dispatched "1vehicleNorm" plan, the command to search the Airport region is sent to a single vehicle, and the vehicle's speed is left at its default, more fuel-efficient value. The other vehicle is allowed to determine its own behavior (it proceeds to search a different area).

The DISPATCHED plan x is the one that is being enacted by GRIM, and defines both the expected performance of the plan and bounds on when that expected performance will fail to satisfy the constraints and completion criteria of the formulated goal g.

Monitor

During execution, GRIM will actively MONITOR the progress of the DISPATCHED expansion to ensure that it will satisfy the constraints of the selected goal. Figure 8 shows the system's estimate of the search area uncertainty over time, as well as the constraints, expectations, and worst-case bound. In Figure 8a the execution is proceeding slower than expected, but still remains

within the worst-case bound; if the execution were to proceed from this point forward at the *expected rate*, it would satisfy the formulated constraints on the goal. Figure 8b shows the execution at a later point in time, where it first exceeds the worst-case bound; if it were to continue to execute at the expected rate, it would not complete the search within the formulated constraints. As such, MONITOR triggers the EVALUATE strategy.

In summary, the MONITOR strategy actively tracks the execution performance of the dispatched plan x, and triggers the EVALUATE strategy when the performance violates any of the goal's constraints or the plan's worst-case bounds.

Evaluate and Resolve

Once the monitor detects that an execution has violated some constraint or bound, it passes the goal to the EVALUATE strategy. If the execution had satisfied the acceptable level of uncertainty constraint that was generated during formulation, the goal would be passed to the FINISH strategy, where it would be marked as successfully completed. If it violated the other formulated constraints (i.e., it passed the deadline without reaching the acceptable level of uncertainty), the DROP strategy would mark it as failed. In this example, the execution violates the worst-case bound generated during DISPATCH and the goal g is passed to the RESOLVE strategy, where GRIM attempts to change the execution such that it may still satisfy the constraints on g.

The RESOLVE strategy attempts to fix g by working through previous modes in the Goal Lifecycle. In the system described here, the RESOLVE strategy set is limited to the REPAIR, REEXPAND, and UNSELECT strategies.

First, GRIM attempts to REPAIR the expansion by adjusting the expansion chosen in the COMMIT strategy. This involves changing the vehicle speed by committing to a different instance of the 1-vehicle plan from the original expansion: "1vehicleFast". The new instance of the expansion is `COMMITTED`, and it is dispatched and monitored as before. Figure 9 shows the expectation and worst-case bound for the newly repaired plan, which now requires the vehicle to move more quickly and expend more fuel. However, as the execution continues, the repaired plan also fails to meet expectations, and at time 39,975 the system execution crosses the new bound and triggers the EVALUATE strategy, again activating the RESOLVE strategies.

This time, when GRIM attempts to RESOLVE the failing goal, it finds that neither instance of the originally expanded plan can be expected to satisfy the formulated constraints. Thus, it cannot REPAIR the expansion, and it instead attempts to REEXPAND goal g. Doing so allows GRIM to attempt to generate new plans that might be feasible by incorporating resources that were not used by the current expansion. In the example shown here, the second vehicle (which was not used in the originally committed expansion) is available for the newly expanded plans. As a result, the re-expansion strategy finds two new feasible plans (the single-vehicle plans are deemed infeasible): using both vehicles at their default ("2vehicleNorm") and fast ("2vehicleFast") speeds. With these new plans, the goal g returns to the `EXPANDED` mode, and progresses through the Goal Lifecycle again, committing and dispatching the "2vehicleNorm" plan.

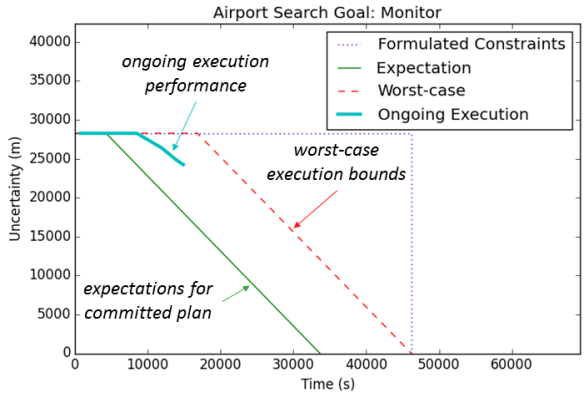
Figure 10 shows the expectation and worst-case bounds for the newly re-expanded and `DISPATCHED` plan p, which now assigns both vehicles to search the area. Because the 2nd UAV must first traverse to the search region, it does not arrive in time to assist the search before the deadline, and the new plan also fails to meet expectations. At time 43,670 the execution violates the bounds of the new expansion, and GRIM uses the REPAIR strategy to increase the speed of both vehicles. At time 44,380 MONITOR again triggers the EVALUATE and RESOLVE strategies, but both the REPAIR and REEXPAND strategies fail. GRIM then proceeds to UNSELECT the Airport search goal and consider other goals to pursue. In this case, which was designed specifically to fail (in order to demonstrate the RESOLVE strategies), both of the other search goals have already passed their deadlines, and are dropped as they are deemed to have failed. Once the execution time passes the deadline for the Airport search goal, it will also be dropped.

The EVALUATE strategy assesses the performance of the goal and determines which RESOLVE strategy to activate; the RESOLVE strategy will FINISH a completed goal, DROP a failed goal, or REPAIR, REEXPAND, or UNSELECT a goal with an infeasible expansion.

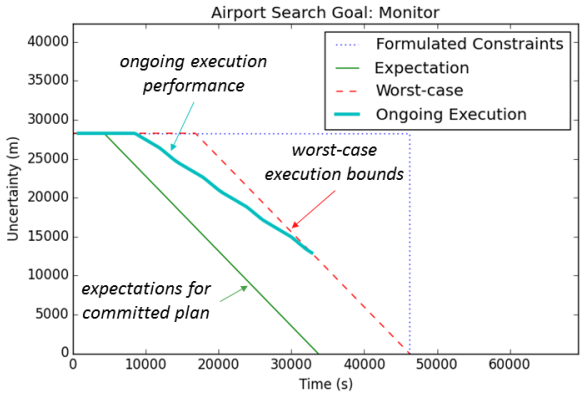
5 Discussion

This paper describes initial efforts towards grounding a GR system, termed GRIM, in the information measures used by the controlled vehicles during execution. In particular, this work adapts the Goal Lifecycle introduced in Roberts et al. (2014; 2015a) and instantiates it in the GRIM system: a centralized GR system that provides commands to independent vehicles. Each vehicle interprets the commands via a play-calling architecture that leverages a formally synthesized FSA and executes the required behaviors via an application of artificial physics, termed *physicomimetics*; more details can be found in Apker, Johnson, and Humphrey (2016) for the play-calling architecture, Kress-Gazit, Fainekos, and Pappas (2009) for the controller synthesis process, and Apker and Martinson (2014) for physicomimetic vehicle control.

Other implementations of GR systems have been developed. Vattam et al. (2013) describes a GR agent as an autonomous agent that is "aware of its own goals and [can] deliberate upon them," and provides a useful survey of related GR research. Autonomous agents that deliberate on their goals are not an isolated concept, and sig-



(a) Execution monitor at time 15,000.



(b) Execution monitor at time 32,824.

Figure 8: GRIM monitors the performance of the dispatched plan during execution. Violation of the goal constraints or plan performance bounds will trigger the EVALUATE strategy, causing the system to react to the change in the goal.

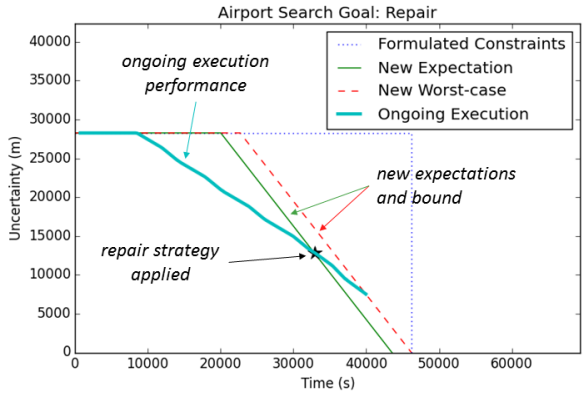


Figure 9: After a violation of the original plan's bounds, the plan is repaired to increase the UAV speed), and new expectations and bounds are generated for the repaired plan. GRIM continues to monitor the execution of the goal until time 39,975.

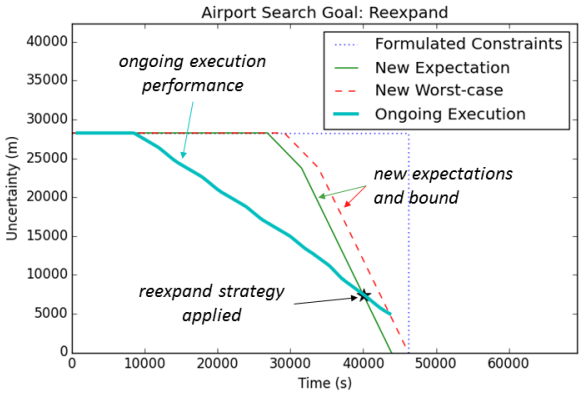


Figure 10: After a violation of the repaired plan's bounds, the goal is reexpanded and the 2-vehicle plan is selected. New expectations and bounds are generated for the reexpanded plan, and GRIM continues to monitor the execution of the goal until time 43,670.

nificant research has been conducted towards those ends (Norman and Long 1996; Altmann and Trafton 2002; Cox 2007; Molineaux, Klenk, and Aha 2010; Thangarajah et al. 2010; Harland et al. 2014).

The individual strategies used in the Goal Lifecycle are, themselves, important research topics, and each can be accomplished in a variety of ways. Goal formulation, for example, may occur externally to the system (i.e., a user may provide a goal), or may be conducted autonomously during execution. For example, Klenk, Molineaux, and Aha (2013) present a GR system in which the autonomous agent automatically detects and explains discrepancies during execution, which then facilitates the generation of new goals for the agent. Alternatively, new goals can also be learned or guided by human input through methods such as case-based reasoning (Weber, Mateas, and Jhala 2012; Jaidee, Muoz-Avila, and Aha 2013). As with goal formulation, the specific method for goal selection can vary widely, from domain-specific rule-based selection (Shapiro et al. 2012; Thangarajah et al. 2010) to the evaluation of domain-independent heuristics (Wilson, Molineaux, and Aha 2013), or goal priorities (Young and Hawes 2012).

Similarly, the plan generation strategy can vary among applications or systems, and may involve trajectory generation (Yilmaz et al. 2008; LaValle and Kuffner 2001) or occur at a more abstract level (Blythe 1999; Kress-Gazit, Fainekos, and Pappas 2009). In many cases, plan generation will also generate expectations for the plan's execution performance, though in some cases it may be necessary to generate expectations separately, as in Auslander et al. (2015).

The Goal Lifecycle provides a formal structure for these strategies, such that the resulting system can deliberate on and adapt its goals to dynamic and unpredictable events. This paper extends the Goal Lifecycle within the FDR domain by grounding its implementation using the vehicle's information measures, and by implementing and demonstrating the RESOLVE strategies. This work focused specifically on area survey goals within a disaster relief scenario, though other related goals exist that must also be characterized in a similar fashion. For example, once an official is located by the vehicles conducting the area search it may be necessary to provide a continuous communications relay for that official, which involves formulating a goal of a new type (relay) in GRIM. Likewise, other potential goals (e.g., medical evacuation or logistics supply delivery) may arise during execution of the FDR scenario. Each goal should be defined in the Goal Lifecycle and grounded in the metrics used to evaluate its performance. These are topics for future extensions of the GRIM system.

Future extensions will also investigate the use of more complex algorithms and metrics in the implementation of the Goal Lifecycle. A more accurate measurement of the uncertainty remaining in an area survey will allow GRIM to improve its performance estimates and react accordingly. Additionally, more complex strategies would improve the system's capabilities. For example, a planner or scheduler could be used to SELECT goals while accounting for the likelihood of discovering an official in each region, thus enabling GRIM to more intelligently choose which goals to pursue. Likewise, adapting the plan expectations (i.e., recognizing that the vehicles are not completing the survey at the expected rate, and changing the expectations accordingly) would enable GRIM to more quickly identify and evaluate problems, and thus improve the likelihood that it could RESOLVE any discrepancies.

6 Conclusion

This paper demonstrated, via simulation, how a GR system can FORMULATE, SELECT, EXPAND, COMMIT to, and DISPATCH area search goals to a team of autonomous vehicles using the team's information measures and expectations. The system, GRIM, will then MONITOR the performance of these vehicles with respect to their goals, and trigger the EVALUATE strategy when a problem is detected in the execution. When the execution performance violates the pre-determined bounds for the plan, GRIM automatically attempts to RESOLVE the problems by repairing the plan or re-expanding the goal into a new set of plans.

This demonstration showed how a GR system can be useful for autonomous systems operating in dynamic environments, and how to ground it to the information measures used by the system to evaluate its performance. Future work on this subject will extend the system to process additional goal types, and use additional and more complex strategies. This will enable a more thorough evaluation of the benefits of such a system via an experiment with randomly generated scenarios.

Acknowledgments

This work was performed at the Naval Research Laboratory. The authors were funded by the Office of Strategic Defense.

References

Altmann, E. M., and Trafton, J. G. 2002. Memory for Goals: An Acctivation-Based Model. *Cognitive Science* 26(1):39–83.

Apker, T., and Martinson, E. 2014. *Distributed Autonomous Robotic Systems: The 11th International Symposium*. Berlin, Heidelberg: Springer Berlin Heidelberg. chapter A Physics Inspired Finite State Machine Controller for Mobile Acoustic Arrays, 47–58.

Apker, T. B.; Johnson, B.; and Humphrey, L. 2016. LTL Templates for Play-Calling Supervisory Control. In *Proceedings of AIAA Science and Technology Exposition*.

Auslander, B.; Floyd, M. W.; Apker, T.; Johnson, B.; Roberts, M.; and Aha, D. W. 2015. Learning to Estimate : A Case-Based Approach to Task Execution Prediction. In *Proceedings of the 23rd International Conference on Case-Based Reasoning*, 15–29.

Blythe, J. 1999. Desicion-Theoretic Planning. *AI Magazine* 20(2):37–54.

Cox, M. T. 2007. Perpetual Self-Aware Cognitive Agents. *AI Magazine* 28(1):32–46.

Harland, J.; Morley, D. N.; Thangarajah, J.; and Yorke-Smith, N. 2014. An Operational Semantics for the Goal Life-Cycle in BDI Agents. *Autonomous Agents and Multi-Agent Systems* 28(4):682–719.

Jaidee, U.; Muoz-Avila, H.; and Aha, D. W. 2013. Case-Based Goal-Driven Coordination of Multiple Learning Agents. In *Proceedings of the 21st International Conference on Case-Based Reasoning*, 164–178.

Johnson, B.; Roberts, M.; Apker, T.; and Aha, D. W. 2016. Goal Reasoning with Information Measures. In *Proceedings of the 4th Annual Conference on Advances in Cognitive Systems*.

Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Goal-Driven Autonomy for Responding to Unexpected Events in Strategy Simulations. *Computational Intelligence* 29(2):187–206.

Kress-Gazit, H.; Fainekos, G. E.; and Pappas, G. J. 2009. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics* 25(6):1370–1381.

LaValle, S. M., and Kuffner, J. J. 2001. Rapidly-Exploring Random Trees: Progress and Prospects. In Donald, B.; Lynch, K.; and Rus, D., eds., *Algorithmic and Computational Robotics: New Directions*, 293–308.

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-Driven Autonomy in a Navy Strategy Simulation. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.

Norman, T., and Long, D. 1996. Alarms: An implementation of motivated agency. *Intelligent Agents II Agent Theories, Architectures, and Languages* 219–234.

Roberts, M.; Vattam, S.; Alford, R.; Auslander, B.; Karneeb, J.; Molineaux, M.; Apker, T.; Wilson, M.; McMahon, J.; and Aha, D. W. 2014. Iterative Goal Refinement for Robotics. In *Working Notes of the Planning and Robotics Workshop at ICAPS*. Portsmouth, NH: AAAI.

Roberts, M.; Apker, T.; Johnson, B.; Auslander, B.; Wellman, B.; and Aha, D. W. 2015a. Coordinating Robot Teams for Disaster Relief. In *Proceedings of the 28th International FLAIRS Conference*.

Roberts, M.; Vattam, S.; Alford, R.; Auslander, B.; Apker, T.; Johnson, B.; and Aha, D. W. 2015b. Goal Reasoning to Coordinate Robotic Teams for Disaster Relief. In *Planning and Robotics: Papers from the ICAPS Workshop*. Jerusalem, Israel: AAAI.

Shapiro, S.; Sardina, S.; Thangarajah, J.; Cavedon, L.; and Padgham, L. 2012. Revising Conflicting Intention Sets in BDI Agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 1081–1088.

Thangarajah, J.; Harland, J.; Morley, D.; and Yorke-Smith, N. 2010. Operational Behaviour for Executing, Suspending, and Aborting Goals in BDI Agent Systems. In Omicini, A.; Sardina, S.; and Vasconcelos, W., eds., *Declarative Agent Languages and Technologies VIII: Papers from the AAMAS Workshop*, 1–21.

U.S. Department of Defense. 2011. Department of Defense Support to Foreign Disaster Relief (Handbook for JTF Commanders and Below). *GTA 90-01-030*.

Vattam, S.; Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Breadth of Approaches to Goal Reasoning : A Research Survey. In Aha, D. W.; Cox, M. T.; and Muñoz-Avila, H., eds., *Goal Reasoning: Papers from the ACS Workshop (Techinical Report CS-TR-5029)*, 111. College Park, MD: University of Maryland, Department of Comnputer Science.

Weber, B. G.; Mateas, M.; and Jhala, A. 2012. Learning from Demonstration for Goal-Driven Autonomy. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 1176–1182.

Wilson, M.; Molineaux, M.; and Aha, D. W. 2013. Domain-Independent Heuristics for Goal Formulation. In *Proceedings of the 26th International FLAIRS Conference*, 160–165.

Yilmaz, N. K.; Evangelinos, C.; Lermusiaux, P. F.; and Patrikalakis, N. M. 2008. Path Planning of Autonomous Underwater Vehicles for Adaptive Sampling Using Mixed integer Linear Programming. *IEEE Journal of Oceanic Engineering* 33(4):522–537.

Young, J., and Hawes, N. 2012. Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game. In *Proceedings of the 8th Annual International Conference on Artificial Intelligence and Interactive Digital Entertainment*, 87–92.

Planning and Monitoring with Performance Level Profiles

Maor Ashkenzi and **Michael Bar-Sinai** and **Ronen I. Brafman**
Computer Science Department
Ben-Gurion University
maorash,barsinam,brafman@cs.bgu.ac.il

Abstract

Despite the existence of powerful formal languages for writing robot controllers, most existing functional modules are written using standard programming languages. The existence of such a code base raises critical challenges: 1. How to enable automated analysis, monitoring, and reuse of existing code? 2. How to convey to customers the expected level of performance of an autonomous robot? 3. Perhaps most crucial: how to quickly identify abnormal behavior of autonomous robots? To address these issues we suggested the use of *performance-level profiles (PLPs)* (Brafman, Shani, and Shimony, 2014), a formal, yet intuitive, language for specifying the expected properties of functional modules, designed with the above aims in mind. PLPs add novel elements to action specification languages that are important for robotic applications, such as update frequency, run-time statistics, progress measures, and trigger conditions, taking into account the different roles modules can play. In recent years, we further developed their language and designed tools that utilize their power. Besides automated monitoring-code generation, we provide a PLP2PDDL compiler that generates planning domains automatically and a PLP based action dispatcher that supports true separation between a replanning-based controller and the underlying modules, providing plug&play functionality. We describe the updated PLP form, the tools we provide for automated code-schema generation for monitoring, statistics gathering, planning domain generation, action dispatching and our experience in using them in two robotics projects.

INTRODUCTION

The robotics research community has made much progress designing software engineering tools that make it easier to develop robotic modules and controllers (e.g., Ingrand et al. (1996); Simmons and Apfelbaum (1998); Ingham, Ragno, and Williams (2001) to name a few). Many of these tools (e.g., BIP Basu, Bozga, and Sifakis (2006); Basu et al. (2008)) are based on formal concepts and formal languages that provide added benefits in the form of guarantees about the properties of the resulting code, and constructs for composing different modules, and allow proving and inferring properties of the actual system. An ideal situation would be for practitioners to adopt such technology. Yet, in practice, functional modules are often built using standard programming languages, even if on top of infrastructure such as ROS.

Unfortunately, as noted by Abdellatif et al. (2012): "Systems built by assembling together independently developed and delivered components often exhibit pathological behavior. Part of the problem is that developers of these systems do not have **a precise way of expressing the behavior of components** at their interfaces, where inconsistencies may occur."

Addressing this issue is crucial to our ability to deploy autonomous robots in open environments. If we cannot define normal behavior precisely, how will we identify abnormal behavior? Certainly, we would not want to wait until the last moment where some evidently undesirable outcome occurs, nor is the public likely to tolerate such performance. Moreover, even before deployment, customers would want to understand what they are getting before spending large amounts of money on an autonomous robot. Non-paying customers, such as users of open-source code, would greatly benefit from having such information, too.

Finally, another pragmatic consideration is that the lack of *machine readable* behavior specification prevents the development and use of tools that utilize such specifications to *automatically* support monitoring, validation, and planning.

We encountered some of these issues in an ongoing project led by Israel Aerospace Industries (IAI) seeking to build an autonomous compact track loader (CTL) for discovery and evacuation of land-mines. For IAI, replacing its own code base and methodology was not an option, yet it constantly grappled with the question of how to provide clear performance guarantees to the CTL's potential customers. With the text based SSS (System/Subsystem requirements Specification) specification format their engineers use, it was difficult to foresee how each module would behave, and various performance problems encountered at run-time were identified at a late stage, although with appropriate monitoring earlier warnings could have been issued.

Motivated by these considerations, we defined a formal language for specifying what we call *performance-level profiles (PLPs)* (Brafman, Shani, and Shimony, 2014). PLPs describe a number of key aspects of the performance of functional modules.[1] They combine ideas from planning language, such as PDDL 2.1 (Fox and Long, 2003), probabilis-

[1] We use the term "module" loosely referring to some piece of software, or even hardware, that provides some functionality. Starting with a sensor that provides raw data, a software module that

tic PDDL (Younes and Littman, 2004), and RDDL (Sanner, 2010), diverse goal notions, such as achievement and maintenance goals (Ingrand et al., 1996; Kaminka et al., 2007), and a new *repeat* construct aimed at making explicit the frequency by which input parameters are read and output parameters are published. Unlike action languages that are limited expressively because they are closely tied to the state-of-the-art in planning technology, PLPs seek to provide expressivity that can be used for other tasks, such as performance monitoring.

PLPs have a number of potential roles. At the design stage, PLPs can replace less formal text-based specifications such as the SSS (systems/subsystem requirements specification). SSS are often used by system engineers to describe requirements from a software module. They have rigid structure, but field content is textual. Second, as noted above, PLPs can be used as a basis for describing the commitments made by module designers to module users, whether customers or other developers. Perhaps more importantly, thanks to the more rigid, machine readable syntax of PLPs, they can be manipulated automatically for the purpose of online monitoring, validation, and planning.

In the past few years, based on our experience in the CTL project, we added new features to this language and developed tools for automated code-schema generation for monitoring, statistics gathering, and planning. These tools play an important role in our long-term agenda of providing plug&play functionality for autonomous robots. Given a PLP, we can generate planning operators, currently in PDDL, that can be used by a planner. The code-generation software queries the user for appropriate information that "glues" the PLPs parameters to the underlying ROS elements. Using the information in the PLP, the generated action dispatcher can receive an execution engine's requests and dispatch the relevant modules code with appropriate parameters. This dispatcher code uses the information in the PLP's associated "glue" file to determine how to trigger the underlying model, i.e what should be sent to which ROS topics.

We describe the current form of PLPs and the tools we developed, and explain how PLPs together with ROSPLAN help provide an abstract planning layer. We also explain how we integrate techniques from the area of plan compilation to allow the use of a classical planner in a domain with uncertainty and partial observability (Palacios and Geffner, 2009; Albore, Palacios, and Geffner, 2009; Shani and Brafman, 2011).

PLPs

The primary objective of a PLP is to clarify the role and expected behavior of a module. As a simple example, imagine a module designed to grasp an object. The expected outcome is that the object is held by the arm. However, in most realistic settings, this effect is not guaranteed. There is some probability of failure, and failure can come with some side effects, such as the object falling, or being broken. And while the running time is most likely not deterministic, we can try to describe properties of its distribution. We can also describe its rate of progress. For instance, the grasping module is usually not static until the object is captured, and so we expect its position to change at some minimal rate. Moreover, the probability of success and failure may depend on various properties, such as the shape and size of the object. Furthermore, in certain settings it may be unrealistic to specify the success probability, as too many external things could impact it. For example, if another arm is attempting to catch the object at the same time, it may difficult to predict which one will succeed.

PLPs are described in XML-based format. Each XML document must conform to the schema defined for the corresponding PLP type (*achieve, maintain, observe, detect*). These schema are described using XML Schema Definition (XSD) files. XML and XSD were chosen for their simplicity and wide-spread use and support. Any tool for editing XML and verifying its structure based on an XSD file can be used as a PLP editor. The precise syntax of the four schema is laborious, and can be found in https://github.com/PLPbgu/PLP-repo together with an example of a PLP of each type. Below we provide an informal description of the information contained in the respective XML documents.

Overview

PLPs have two abstract components. The first circumscribes the conditions under which the profile is provided – conditions that must hold for it to be valid. Such conditions include various properties of the world before and during execution as well as available resources. The second specifies the effect of the action – what success means, what are the possible failure modes, what is the probability of each, what is the distribution over running times, and what is its rate of progress. More generally, it can specify a statistical profile of various aspects of normal module behavior at run-time.

There are four types of PLP, corresponding to four types of functional modules. *Achieve* modules attempt to achieve a new state of the world or generate a new object. For example, changing the orientation of the robot to some goal orientation. *Maintain* modules attempt to maintain some property. For example, a module that maintains some orientation; or, a module that ensures that the robot remains within some confined area. *Observe* modules attempt to recognize some property of the current state of the world. For example, the robot's location, or whether there is a cup on the table. Finally, *Detect* modules monitor the state of the world until some condition holds.

Many robotic modules operate by repeatedly updating or modifying some data-structure or signal based on information that is constantly being updated. To model such constructs we introduce the *repeat* wrapper for the *achieve* and *observe* modules. For example, a path-planning module may update the path as it obtains updated maps. It can be viewed as performing an *achieve* task repeatedly.

generates some higher-level data from it, simple code that generates basic motion, or more complex code that provide complex motion that is informed by some sensor input, and higher level functionality that performs a clear task.

Variables and Resources

The formal definition of PLPs rests on the specification of properties of states of the world. These are defined by specifying properties of various state variables. It is desirable to have a coherent specification of such variables, so that the relationship between modules is clear.

In addition, each module may need access to certain resources. These resources could be energy or memory, or they could be some actuator, or some region of space. These must be specified, much like state variables, and coherent and consistent use of these names is required. In fact, resources can be viewed as a special class of state variables, whose state indicates the status of the resource (e.g., available, > 100 gallons, etc.). However, because we believe that they carry special significance to programmers and operators, we distinguish them from other variables.

Common Elements

All modules specify the following elements:

Parameters: Variables supplied to the module as input or provided by the module as its output. Some input parameters can also be output parameters (i.e., after undergoing some processing). There are some special classes of parameters:

- Error parameters. That is, parameters that specify the accuracy of other parameters. For example, if localization is performed using a Kalman filter, an error estimate can be obtained from its co-variance matrix.
- Execution parameters. These parameters are considered the trigger for the underlying modules execution. For example, the target location for a navigation action.
- Unobservable parameters. Some parameters' value is unknown and cannot be sensed before the action is executed. For example, a navigation module that cannot avoid obstacles not in its initial map. Its PLP will specify this, e.g., using a Boolean parameter "clear path", whose value is not observable prior to the execution of the navigation command. Although the robot cannot act on this parameter, the user must be aware of this requirement.

Set of variables: Local variables and their range.

Application Context: Conditions specifying the contexts in which the PLP is valid. It contains the following elements:

- **Required resources:** List of resources required. If the resource is quantifiable, a required quantity is mentioned. If the resource is needed for operation and then freed (e.g., memory, some tool, some actuator), the requirement status must be mentioned. Possible values are "exclusive" or some frequency of use.
- **(Optional) Maximal rate of change:** Maximal change in resource level per time unit.
- **Preconditions:** conditions on the world at the start of execution time under which the PLP is defined. These conditions can refer only to parameters.
- **Concurrency conditions:** conditions on the world at execution time under which the PLP is defined. They can also refer to the rate of change of parameters. For example, stating that successful navigation requires respecting some maximal speed limit.
- **Concurrent modules:** conditions on modules that must or must-not be executed concurrently.
- **(optional) Parameter frequency:** The frequency by which each parameter must be read or written. The frequency by which a parameter is updated together with its accuracy (which is available if an error parameter exists) can affect the accuracy of output. For example, a navigation module that aims to reach a specified position may need to obtain position information with certain rate and certain accuracy to ensure success.

Side-effects: Each module has an intended effect, or role. However, it may also have side-effects that are a result of executing this module, but are not a measure of its success or failure. For example, if the module consumes some resource, a natural side effect is that the level of this resource is reduced. Side effects are described by a conditional assignment to a parameter, which could depend on a local variable (such as running time, or distance traveled). Intuitively, side-effects are changes caused by the module that could potentially impact other modules.

Expected Progress Rate: Some modules perform continuous work to achieve or maintain their goals. For example, when navigating, the position will change at some rate as long as the robot is not at its destination. Or, when grasping an object, the hand will move closer to the object. In this field, one specifies the minimal rate of change per time unit, as well as the time unit itself (e.g., $\Delta(x) \geq 1$ meter every 1 minute). This field was recently added both because it is consistent with our goal of providing as much information about module behavior, and because it helps recognize problematic behavior earlier on. Suppose, for example, that we predict that the robot will arrive at the next way-point in 10 minutes, yet it gets stuck somewhere in between for some reason. Without progress measures, we would typically raise an alert after more than 10 minutes (e.g., two standard deviations beyond the expected running time). However, with progress measures, we can recognize the problem much earlier.

PLP Types

There are four PLP types: *achieve, maintain, observe, detect*. We briefly describe each type. See the link provided for more information.

Achieve modules attempt to reach a state of the world in which some desirable property holds. For example, fuel tank is full, robot is standing, plane has landed, etc. Achieve also covers cases where the goal is to generate some virtual object, such as a map or a path. Beyond the common elements, their PLP contains the following: the achievement goal – a Boolean condition defined over suitable parameters, failure modes – which are various ways the module could fail to achieve the goal, the probabilities associated with success and each failure mode, and the running time distribution given success and given failure.

Maintain modules attempt to maintain the value of a variable or the truth value of some more complex condition, e.g., maintain heading, maintain speed, maintain perimeter clean, etc. The condition need not be true initially, and so the mod-

ule may need to initially attain the condition, as in a closed-loop controller that always attempts to decrease some distance to the desired goal condition. For this reason we do not want to force users to split such cases into *achieve* and then *maintain.* Our goal is to describe code, not to force a certain architecture.

In addition to common elements, the PLP of *maintain* modules contains: the condition to be maintained, whether it is initially true,termination conditions, one for successful termination and one for failure, failure modes, the probability of successful termination and different failure modes, and the runtime distribution given success and failure. A success termination condition is not necessary, and the run-time distribution will often be memoryless (i.e., exponential).

Sometimes, one needs to maintain some condition in order to achieve a goal. For example, one may reach a target position by maintaining a pre-computed path to the goal, either by iteratively reaching way-points, or by ensuring that the heading is always in the direction of the path. One can model this behaviour using an achieve module whose goal is to reach the target position, or as a maintain module that maintains heading along the path direction, with the success termination condition being "at-the-goal". The latter definition is less abstract, and clarifies what the module actually needs to do. Given this definition, we can detect problems by alerting the operator or system whenever the actual heading is not in the direction of the path, rather than only when the goal is not reached. Of course, ultimately, the user decides which model suits her best.

Observe modules attempt to identify the value of some variable(s) or a Boolean condition in the current world state. For example: observe distance to wall, observe whether robot is standing, observe whether object is held. Beyond the common elements, *observe* PLPs contain an observation goal – a Boolean condition to be verified or a parameter whose value is to be observed. We also describe the probability of failure to observe, the probability the observation is correct (if Boolean) or some form of error specification, such as confidence interval and confidence level, the running-time distribution given success and given failure.

Detect modules attempt to identify some condition that is either not true now, or that is not immediately observable. For example, detect intruder, detect temperature change, detect motion, detect obstacle, etc. In addition to the common elements, detect modules contain the detection goal, i.e., the condition being detected, and the probability the condition will be detected given that it holds (*true* positive) and given that it does not hold (*false* positive).

REPEAT

In many robotic applications, various modules run continuously, updating some data-structure, such as a map, or a path, or monitoring the environment for some trigger, such as detection of an intruder. Such modules are essentially loops that execute some underlying routine many times, e.g., map and path update, or repeatedly analyze some input until a condition holds. While, in code terms, they offer nothing special, in terms of their spec, they raise a number of issues – for example, the rate by which updates occurs, the rate in which input is expected to be received, and the termination condition. For this purpose we provide a special Boolean *repeat* field for *achieve* and *observe* PLP. When its value is true, additional information must be supplied including:

Execution Frequency – How many times per second is the underlying module executed. We assume that if the module has an output parameter, then the frequency by which it is updated is the same as the execution frequency.

Input Frequency – For each input parameter that can trigger a repetition, it is possible (optional) to specify its minimal expected update frequency. One can view this as a special type of concurrency condition.

Termination Condition – Once this condition is true, the module stops executing the loop.

Repeat can be used to implement *detect* and *maintain* using repeated *observe* and *achieve* with a suitable termination condition. For example, *detect* can be implemented using an observe module that observes the detect condition, repeatedly, until it is true. Since our philosophy is to provide constructs that map naturally to functional modules, rather than to educate users in using some minimal vocabulary, we allow both options.

An interesting issue regarding repeat modules is the automated derivation of the parameters of the entire module given the properties of the repeat construct (frequencies, termination condition) and the properties of the module that is repeatedly executed (i.e., success probability, running time). For example, suppose that one has computed a map of the environment with some accuracy, and has now generated a path based on this map. At some positions along this path, the distance to the nearest obstacle is 50cm. This implies that the localization error must be smaller than 50cm. This, in turn, likely requires more accurate localization, that can potentially be obtained by increasing the rate by which images, or scanner readings are obtained and analyzed. We believe that automatic inference of such constraints could be valuable in many applications, and we pose this problem as an interesting question for future work.

ROS-BASED PLP MONITORING TOOLS

We developed two main tools so far: PLP schema files written in XSD, and automated code generation for run-time monitoring and statistics gathering. We briefly describe each. The code-generation software as well as code samples can be found at `https://github.com/PLPbgu/PLP-repo`. The code generator is written in Java and outputs a ROS package containing Python scripts for a ROS node, a PLP object, and additional auxiliary classes. The ROS node, called *PLP ROS harness*, is responsible for communicating with the ROS environment, and detecting triggers. The PLP object is responsible for the calculations and has no direct connection with the ROS environment. This decoupling allows programmers to use the PLP object offline and makes it easy to modify the code to work in other environments.

Editor Friendly PLP Schema

We developed an XML-based language for describing PLPs, and provide a formal definition of it in the form of an XML

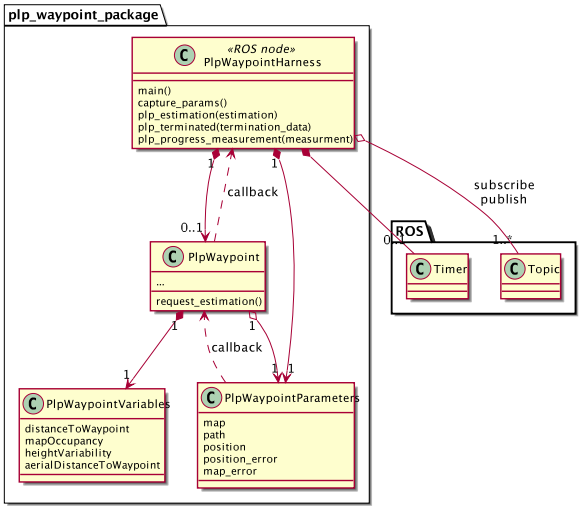


Figure 1: Class diagram of the PLP package, and how it relates to the ROS environment. The class structure resembles the structure of the PLP file. Object ownership is directional, from the ROS node to the PLP classes. Calls from the PLP class back to the ROS node are done via a callback mechanism, decoupling the PLP logic from the ROS environment.

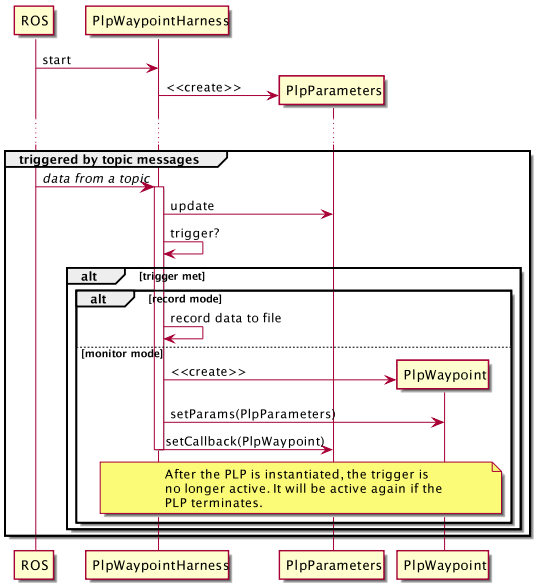


Figure 2: The triggering sequence of a PLP. Note that a trigger may result in initiating a PLP, recording the PLP parameters at trigger time to file, or both.

schema document (XSD file). This allows developers to use their tool of choice for writing and validating PLPs. We believe this approach serves PLP users better than creating a bespoke editor, since developers value productivity, which normally suffers when a new tool is introduced. As many text editors and IDEs, such as Emacs, NetBeans, and VisualStudio, offer XSD-based validation and code completion, an XSD language definition is directly usable for most programmers.

Code Generation for Monitoring

Given a PLP file, our tools can generate a ROS package,[2] comprised of a single ROS node, a PLP object, and auxiliary classes. When executed, the node monitors the performance of the module described by the PLP. Figure 1 describes the generated classes and how they relate to each other.

The generated code structure is designed to allow both online (interacting with a ROS environment) and offline (outside of ROS) use of the PLP logic. Additional goals are making the code reusable and easy to comprehend. To this end, the generated classes reflect the structure of the PLP file: Specific classes for the PLP parameters and variables, a single class for the PLP logic, and a class for a specialized ROS node, called the *harness node*. To allow offline usage, the PLP node is not directly aware of the harness node (which is a part of a ROS environment). Rather, it communicates with it using a callback mechanism. During offline usage, client code can provide a PLP object with a different callback, or use the PLP logic directly.

For generating the harness node, we use a *glue file* that contains the mapping between PLP parameters to their location in ROS (e.g., the topic). The glue file also contains the required imports in order to work with the needed messages and classes.

This generated ROS package performs the following functionality: The PLP Harness saves the most recent value of each parameter as referred to by the given glue file. Each time a value is updated, the PLP Harness checks whether the PLP trigger holds (start condition). When it does, the harness node creates a PLP object (see Figure 2). The newly created PLP object is updated whenever one of its parameters are updated, and uses the harness to publish alerts and predictions regarding the task it monitors. In particular, it can send alerts when preconditions, concurrency conditions, progress measures or other required conditions are violated, and can generate predictions regarding the success probability and expected execution time. For PLPs that require periodical update, the harness node generates a timer requesting periodical PLP evaluations. Figures 3 and 4 show the call sequences resulting from a message arriving at the harness node and a timer event, respectively.

There are two main parts that are left as "templates" for the programmer to implement:

1. Variable updaters – when a parameter is updated, the methods for recalculating the variables will be called. These functions are left to be filled in by the user.
2. Condition validation functions – each function checks if some basic condition specified somewhere in the PLP holds. The user is asked to fill-in code that validates only the bare essential conditions.

Each of these functions is generated with comments describing what needs to be implemented in a clear and simple manner.

[2] We do not support code generation for *repeat* constructs, yet.

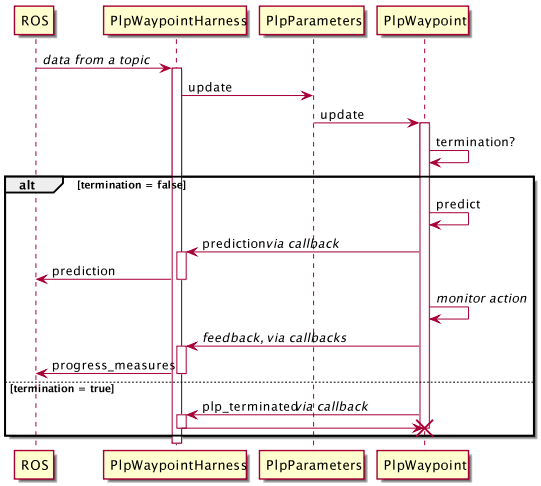


Figure 3: When a message arrives at the PLP ROS harness, it triggers a sequence of calls and callbacks. The result may be the PLP publishing predictions and progress measures, or detecting a termination condition.

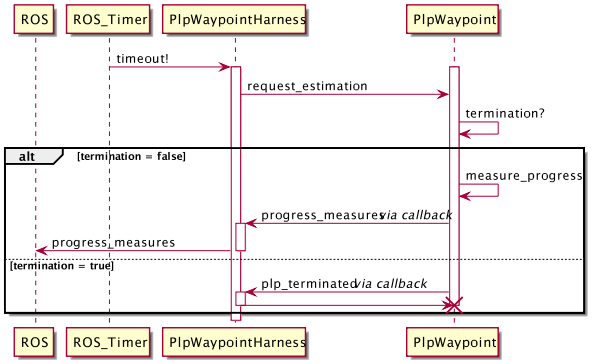


Figure 4: A PLP whose evaluation requires periodical measurements can use a ROS timer.

Code Generation for Data Gathering

An important component of a PLP is a statistical profile of some of its properties, such as success rate and run-time distribution. Our statistics collecting code is generated automatically from a PLP and is, in fact, an operating mode of the aforementioned ROS harness node. When started in "data gathering mode," the ROS harness node records the parameters when the PLP's trigger condition becomes true (see Figure 2). This allows for the following technique: run multiple simulations of a given task, with the ROS harness in data gathering mode. For each iteration, in addition to the file generated by the ROS harness node, record the final result (success and failure probabilities, run-time distributions, etc.) and use this data set for predictive analysis using PLP parameters.

Use Methodology

Our suggested methodology of use for PLPs is described in Figure 5. A PLP can be used instead, or in addition to traditional specification languages by the system architect. By using a PLP, he/she is required to provide more precise, quantifiable requirements, possibly providing more guidance to the programmer. For example, if the expected success rate of the grasping module is high, the programmer may want to update the trajectory of the arm during its motion based on input from the gripper camera. Next, the code is written, and a PLP describing its expected properties is defined (or modified, if defined earlier). Then, using automated code generation, data-gathering code is generated and integrated into the ROS package. Whenever the module executes, it triggers the measurement of various run-time and success statistics which are used to update the predictive analysis carried out by the PLP object. At any point that a PLP is available, one can use the automated code generator to generate monitoring code. If the code changes, one can edit the PLP to reflect such changes, and regenerate the code. The generated code works online, monitoring the module's behavior and/or gathering statistics.

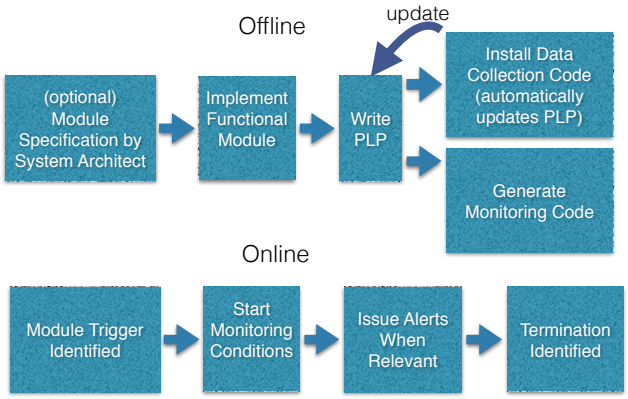


Figure 5: Suggested Use Methodology

Using PLPs for Planning

PLPs can be viewed as a rich action representation language. As such, they contain the information required to generate planning operators of various types. We exploit this in order to facilitate automated planning, providing developers with yet another advantage for using PLPs as a specification/description language. In this context, PLPs provide an abstract layer on top of the actual system that makes its integration with a planner simpler, and allows for plug&play capabilities, where new capabilities, whether software of hardware based, can easily be integrated and orchestrated through the use of a more abstract planning layer. While we are not there, yet, ROSPLAN Cashmore et al. (2015) took an important step in this direction, and PLPs with their support tools take us closer to realizing this vision.

The process we envision is as follows: First, system engineers provide PLPs for their modules. The PLP2PDDL compiler generates planning domain descriptions from these PLPs. The domain files are now provided to the planner/executor, ROSPLAN in our case, which is able to plan, and modify plans. Using the information in the glue files, we

generate a PLP dispatcher that connects the execution engine's request, i.e ROSPlan, to the PLPs underlying modules. If new capabilities are added, all that is required is to add the new PLPs, and rerun the process. At present, our compilation is a batch process, but we hope to make the process modular, so that only the new PLPs will require compilation.

In what follows we describe our PLP2PDDL transformations that address the inherent discrepancy between the naturally partially observable domain of robotics with the classical model specified in PDDL. Then, we explain the more technical aspect of our integration with ROSPLAN.

Generating Planning Domains from PLPs

PLPs capture information about uncertainty and partial observability. For example, *observe* and *detect* modules have no immediate counterparts in PDDL, which assumes a classical, and thus deterministic, fully observable domain. To address this, we provide two compilation modes. In the first, we assume that the domain is almost fully observable. Sensing is used only to supply the value of parameters for actions. For example, the location of a cup to be grasped. The second is a more sophisticated compilation that mimics some of the ideas used to compile conformant and contingent planning problems into classical problems (Palacios and Geffner, 2009; Albore, Palacios, and Geffner, 2009; Shani and Brafman, 2011)

Our PLP2PDDL compiler takes as input PLP specifications and outputs a PDDL domain and a template problem file. In the near-fully-observable mode we add one new kind of proposition. For every parameter par, which value can be sensed by an *observe* PLP, we add: $KVpar$ which intuitively stands for $pars$ *value is known*. The initial state is given by the user, and is transformed into the initial state of for the problem under the closed-world assumption. Thus, initially, $\neg KVpar$ holds for every parameter.

The PDDL action generated from a PLP contains all preconditions and concurrency conditions that appear in the PLP. For conditions whose value cannot be altered by any other PLP, a message will be prompted to the user, asking if this condition is needed. We do this because some of these conditions might be unnecessary for planning, adding unwanted complexity. For PLPs which require an input parameter, par, for which a sensing action exists, a precondition $KVpar$ is added to the PDDL actions preconditions to ensure that the sensing action will be executed before applying that action. PLP side effects, both conditional and unconditional are added as effects to the PDDL action. Other effects are determined by the type of module:

1. *Achieve* PLPs: the goal condition is added as an effect.
2. *Maintain* PLPs: if the maintained condition is initially false, it is added as an effect. If it is initially true, we ignore it because it has no significance in non-temporal planning.
3. *Detect* PLPs: if the action finished successfully, the detected condition holds. In the non-temporal case, we can view it as an action that achieves a condition. Thus, the detection goal is added as an effect.
4. *Observe* PLPs: if the observation goal is a parameter par, $KVpar$ is added as an effect. If the observation goal is a condition, it is of no significance in this compilation mode because we assume it is known.

In the partially observable mode we follow the basic compilation rules described in the near-fully-observable mode, with some new additions and changes. We generate a domain with a richer set of variables: for every ground predicate p, we add two new predicates: Kp and $K\neg p$, for p *is known* and $\neg p$ *is known*.

Initial State Using the user's input initial state, we generate a new initial state as follows: For every literal l that holds in the user provided initial state, we include l and Kl in the generated initial state. In addition, we optimistically close the initial state for every primitive proposition that is unknown as follows:

1. If there is a sensing action for p or an action that achieves p, we optimistically assume that p holds and add p to the generated initial state, but not Kp. Instead, we include $\neg Kp \wedge \neg K\neg p$.
2. If p cannot be sensed or produced, we optimistically add p and Kp.

Actions Every action that requires p as a precondition, will instead require $p \wedge Kp$. Regarding the effects of the actions, there are three main cases:

1. Sensing parameter values: We handle this case as described in the near-fully-observable mode.
2. Sensing conditions: in *observe* PLPs that sense whether a primitive proposition p holds or not have the following conditional effects: $(p, Kp \wedge \neg K\neg p), (\neg p, K\neg p \wedge \neg Kp)$.
3. Every deterministic effect p is augmented by $Kp \wedge \neg K\neg p$.
4. Every conditional effect (p, q) is augmented by (Kp, Kq) and $(\neg K\neg p, \neg K\neg q)$.

The result is a classical planning problem in which the planner acts based on assumptions on the value of all propositions. However, because of the modified form of the operators, it must verify its assumptions when possible (i.e., via a sensing action or using a deterministic effect). This forces the planner to add sensing actions, which are useless in standard classical planning. Of course, the assumptions may be wrong, at which point the execution engine (i.e., the dispatcher) will stop plan execution and will replan with the new information. Very roughly, one can view this compilation scheme as a simplified version of the compilation used by the SDR planner (Shani and Brafman, 2011). A more sophisticated compilation would require more information about the initial belief state.

ROSPlan Integration

Another tool we built is a plan dispatcher to be used for the domain generated by the PLP2PDDL compiler. In ROSPlan, part of the *Planning System* responsibilities is to generate a plan and dispatch the actions according to the plan. The action dispatch is achieved using a shared ROS topic, which every action node subscribes to. The message published on

this topic specifies the name of the action that needs to be executed. Every node that is subscribed to this topic, receives the message and checks whether or not it is responsible for the given action. In addition, the actions themselves are responsible for updating the Planning System of success/-failure and updating the *Knowledge Management System*, which holds the current state of the world. This design is somewhat problematic for PLP module dispatch. PLPs advocate no change to the code written by the user, they are wrappers that one can attach to one's robotic module without changing the code. This means that we do not want to force the user to write code that listens to a specific topic for execution or updates the KMS. Given some PLPs' descriptions, a PLP dispatcher will be generated in order to dispatch a plan containing the given PLPs. Our PLP dispatcher is composed of a single node for each PLP and a launch file to run them. Each node provides a layer of abstraction between the Planning System and KMS, and the relevant PLPs underlying module. Each PLPs dispatcher node will be responsible for the following:

1. Listening to the required ROSPlan node that requests action execution. Once a message is received, the dispatcher will activate the required PLP module's trigger (in order to execute it).
2. Constantly monitoring the termination conditions of the PLP's underlying module. Once a termination condition holds, the dispatcher will update the planning system whether the action failed or succeeded.
3. Receiving the PLP module's output parameters and, if needed, saving them in the ROS MongoDB. This is crucial because we might need to pass them to another PLP as part of a trigger.
4. Updating the KMS with the current state of the world according to the results of the PLP modules execution.
5. When a domain generated in the partially observable mode is used, if a plan failed because of a wrong assumption regarding a predicate's value, the dispatcher will update the value of the predicate in the KMS and inform the planning system that the action failed, thus triggering a replan (in implementation stages).

This is achieved using a *glue* file which maps PLP parameters to PDDL parameters, so that the execution parameters can be sent to the PLPs' modules. This abstraction layer allows us to successfully integrate with ROSPlan while still following the PLPs guidelines of not requiring any alteration to the user's code. There is currently a working demo of the PLP dispatcher. The code generator that automatically generates the PLP dispatcher given a domain is still in implementation stages.

USE CASES

Our main use case involves the use of PLPs for monitoring within the autonomous CTL, led by Israel Aerospace Industries. To date, most work done is within the Gazebo simulation environment, for which a CTL model and working environments were constructed. A PLP ROS package is

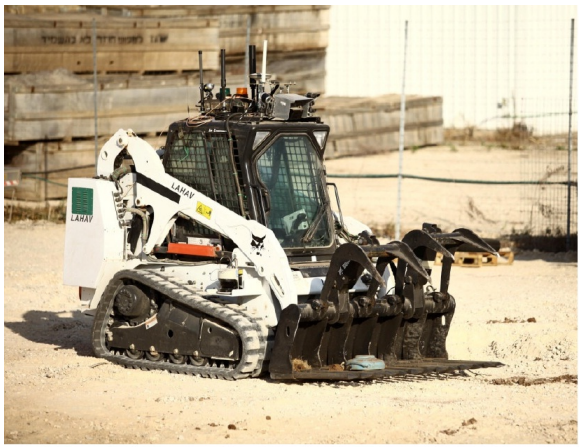


Figure 6: The Autonomous CTL

installed on the real CTL, but we do not have enough data from this installation at this point.

PLPs were used to generate monitoring code for two modules: the way-point driver and the module that processes the IBEO laser scanner. The way-point driver has an *achieve* PLP whose goal is to arrive at the next way-point (provided by the path-planner). Among some of its properties are the following two progress conditions: the length of the planned path should decrease during an appropriate interval, and the aerial distance to the next way-point should decrease every (different) interval. During testing, the ROS node based on the way-point driver PLP helped the team find an error in the path planning module. Even though the CTL was reaching its navigation goal, the PLP node was complaining that the planned path length did not decrease. Looking into this issue, the team found that once the path planning module decided on a given path, it did not update it while the CTL was moving. Rather, it was re-publishing the same, initial path over and over again, until the CTL has reached the goal.

Another ongoing project employing PLPs is an autonomous service robot that uses AI planning technology to respond to user requests. The robot can perform actions like pressing an elevator button, moving between rooms, etc. For each action, a PLP is defined. Besides the benefit of automated monitoring code generation, noted earlier, here planning plays a key role in supporting autonomy, and we use the near-fully-observable mode. The correspondence between plan actions, PLPs, and concrete functional modules via the PDDL compiler and the glue file, allowed us to build a working PLP dispatcher that provides an abstraction layer between the ROSPlan dispatcher and the PLPs underlying modules. Our PLP dispatcher receives the PDDL actions to execute and dispatches the plan online, as described earlier.

RELATED WORK

PLPs for achievement goals are based on existing action languages, mixing features from a number of sources. Preconditions and effects go back to STRIPS (Fikes and Nilsson, 1971). Concurrency conditions were integrated into PDDL when it was extended to handle temporal actions (Fox and

Long, 2003), which obviously included a specification of (deterministic) running time of operators and the ability to specify resources and their consumption over time. PLPs emphasize a slightly different version of concurrency condition, which attempts to address the issue of action (here, module) interaction introduced by Boutilier and Brafman (2001). Requiring exclusive use of a resource is a technique for preventing harmful interactions, or simply interactions whose effect cannot be predicted. Thus, a module can require exclusive use of an arm, preventing other modules from interfering with its use. This idea goes back to the classical use of mutex in concurrent systems (Dijkstra, 1965), and is implemented in some languages for concurrent programming. A related idea appears in Structured Reactive Controllers (Beetz, 1999) where the notion of embedability is defined. That notion asserts that a module may be executed concurrently with a set of other modules without their execution interfering with its ability to reach its goal.

In addition, PLPs address uncertainty by borrowing ideas from PPDDL (Younes and Littman, 2004) and RDDL (Sanner, 2010). For each of these aspects (time, concurrency, uncertainty, resources) there are languages that provide more powerful constructs, whereas PLPs attempt to address all essential aspects of the performance of a module, while providing a good trade off between expressiveness and intuitiveness. Thus, while PDDL2.1 can describe temporal actions, it does not describe actions with stochastic durations, and while RDDL describes probabilistic effects, it does not specify temporally extended actions, etc.

Maintenance goals are also a well known concept (e.g., Ingrand et al. (1996)). In principle, a maintenance goal could be specified using a version of temporally extended actions (Fox and Long, 2003) in which some effects of an action take place immediately at the beginning of its execution and are true throughout its execution. Such specification is a bit counter-intuitive, though, as maintain modules usually maintain a condition that is already true. *detect* and *observe* modules, on the other hand, are motivated by the ideas of observation in POMDPs and contingent planning, whereas the ideas behind the *repeat* module, and in particular, the introduction of input/output frequencies appears new, to the best of our knowledge.

The formalism that appears most similar to PLP in terms of constructs offered is PRS (Ingrand et al., 1996). It has achieve, maintain, and observation goals, context conditions, and properties, which can mention resource consumption. But herein lies a fundamental difference. PRS is part of a large set of languages for *building* robotic modules and controllers that have been developed and improved over the years. Other well-known languages are TCA/TDL (Simmons and Apfelbaum, 1998), RPML (Ingham, Ragno, and Williams, 2001), BIP (Basu, Bozga, and Sifakis, 2006; Basu et al., 2008), or even synthesis based tools for automated controller generation, such as Kress-Gazit and Pappas (2010). PLPs on the other hand are descriptive rather than prescriptive. This why they contain important information about expected performance, such as success probability, expected running time, and expected progress that is missing from the above languages. Such descriptions can be used as a more structured, machine readable, specification language, describing the performance that the designer seeks, or as the input to automated tools that utilize information about expected behaviour for monitoring, validation, or planning. However, they are *not* a tool for programming robots. However, unlike formal languages for constructing robotic software, PLPs are not provably correct. That is, we rely on the fact that the PLP is a faithful specification of the written code, but we cannot guarantee it. Statistics gathering tools can, to some extent, verify the correctness of this specification, but only probabilistically.

Finally, the automated code generated from PLPs implements a middle-ware layer between the planner and the underlying modules where the PLP provides the information required to build the interface between these layers. This allows for a plug&play-like behavior where the system can make use of new capabilities, whether software or hardware, by processing their PLPs. From the planner's/controller's perspective, the system is composed of PLPs, and it must combine them appropriately to achieve its task. Currently, this combination is in the form of a classical sequential plan+replanning, but in the future, we hope to incorporate more complex planners that support temporal and probabilistic reasoning. Similar motivation underlies the ROAR architecture Degroote and Lacroix (2011) where the control layer views the systems as composed of a set of resources, or agents with capabilities, and desired behavior requires these agents to satisfy various constraints. This architecture uses constraint reasoning combined with pre-written recipes and scripts. Constraint reasoning is used to make the interaction between resources as transparent as possible to the script writer. Our middle layer is composed of code generated automatically that enables using a planner, hopefully in a fully autonomous manner. Another system, Dyknow Heintz, Kvarnström, and Doherty (2010), provides tools for generating middle-ware for knowledge processing – that is, the generation of meaningful symbolic descriptions from sensor information. This middle-ware, however, is very different in flavor from ROAR and PLPs, but is rather loosely related to our abstract notions of *observe* and *detect*. Dyknow provides a rich set of tools, aimed at generating more abstract, symbolic "sensors", whereas *observe* and *detect* make no attempt to provide such abstractions. That is, an *observe* module may observe low level variables, and the generation of more symbolic variables could be done by another module that processes this data. In this respect, the two ideas are orthogonal.

SUMMARY AND FUTURE WORK

We described *performance level profiles*, a language for specifying the expected performance of functional modules. PLPs are motivated by the need to provide more precise, quantifiable, and machine-readable descriptions of module behavior. With such information, one can detect abnormal behavior more easily, generate monitoring code automatically, as well as provide clear guarantees to users.

PLPs are motivated by standard planning languages, but attempts to make the specification more intuitive and appropriate by resorting to the well known notions of achievement

and maintenance goals, adding also analogous notions for the sensing-side of robot activity. In addition, using the *repeat* wrapper, we are able to naturally give a clear role to the notion of frequency and latency, commonly used by designers of robotic hardware and software.

We developed a first set of tools for building and using PLPs. We began by building tools for performance monitoring – being able to provide operators with useful information about unexpected or problematic conditions, such as modules operating without the required preconditions and concurrent conditions, information about running times, and effects of modules used online. Afterwards, we built a compiler that receives PLP specifications and outputs a PDDL domain and template problem file. Finally, we built a PLP dispatcher that provides a layer of abstraction between ROSPlan and the underlying modules, thus allowing planning and plan dispatch without any change to these modules.

In defining PLPs we preferred intuitiveness and redundancy to economy and succinctness. We envision that future use will lead to additional fields describing properties that are useful to know. Our point is that users can always decide not to specify certain aspects of their module, and automated reasoning tools can always use only a subset of the information. For example, for generating PDDL planning domains, we use only part of information in the PLP. If a stronger planner is available, a compiler to a more informative action can be used, instead.

Acknowledgments: We thank the reviewers for their useful comments and our collaborators in the ROBIL project. This work was supported by ISF Grant 933/13, by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev, by MAFAT through its ROBIL project, and by the Lynn and William Frankel Center for Computer Science.

References

Abdellatif, T.; Bensalem, S.; Combaz, J.; de Silva, L.; and Ingrand, F. 2012. Rigorous design of robot software. *Robotics and Autonomous Systems* 60(12):1563–1578.

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Basu, A.; Gallien, M.; Lesire, C.; Nguyen, T.-H.; Bensalem, S.; Ingrand, F.; and Sifakis, J. 2008. Incremental component-based construction and verification of a robotic system. In *European Conference on AI*.

Basu, A.; Bozga, M.; and Sifakis, J. 2006. Modeling heterogeneous real-time components in bip. In *International Conference on Software Engineering and Formal Methods (SEFM-06)*, 3–12.

Beetz, M. 1999. Structured reactive controllers: Controlling robots that perform everyday activity. In *Agents*, 228–235.

Boutilier, C., and Brafman, R. I. 2001. Planing with concurrent interacting actions. *Journal of AI Research* 14:105–136.

Brafman, R. I.; Shani, G.; and Shimony, S. E. 2014. Performance level profiles. In *ICAPS'14 Workshop on Planning and Robotics (PlanRob)*. 99–105.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, 333–341.

Degroote, A., and Lacroix, S. 2011. Roar: Resource oriented agent architecture for the autonomy of robots. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 6090 –6095.

Dijkstra, E. W. 1965. Solution of a problem in concurrent programming control. *Communication of the ACM* 8(9):569.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Aritificial Intelligence* 2:189–208.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Heintz, F.; Kvarnström, J.; and Doherty, P. 2010. Bridging the sense-reasoning gap: DyKnow-Stream-based middleware for knowledge processing. *Advanced Engineering Informatics* 24(1):14–26.

Ingham, M. D.; Ragno, R. J.; and Williams, B. C. 2001. A reactive model-based programming language for robotic space explorers. In *Int. Symp. oon AI, Robotics, and Automation for Space*.

Ingrand, F.; Catilla, R.; Alami, R.; and Robert, F. 1996. A high level supervision and control language for autonomous mobile robots. In 43-49., ed., *IEEE ICRA*.

Kaminka, G. A.; Yakir, A.; Erusalimchik, D.; and Cohen-Nov, N. 2007. Towards collaborative task and team maintenance. In *Autonomous Agents and Multi-Agent Systems*.

Kress-Gazit, H., and Pappas, G. J. 2010. Automatic synthesis of robot controllers for tasks with locative prepositions. In *ICRA*, 3215–3220.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res. (JAIR)* 35:623–675.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description.

Shani, G., and Brafman, R. I. 2011. Replanning in domains with partial information and sensing actions. In *IJCAI*, 2021–2026.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *IEEE IROS*.

Younes, H., and Littman, M. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Appendix: Compilation Example

The following is an example showing the outcome of compilation in the near-fully-observable mode. There are two PLPs: *observe_gateway* which senses the location of a required gateway and *walk_through_gateway* which moves the robot through the required gateway. Their description appears in Figures 8 and 9. The generated PDDL domain file and template problem file appear in Figures 7 and 10.

```
(define (domain PLP_DOMAIN)
(:requirements [:strips] [:adl])
(:predicates (at ?par) (connected ?par1 ?par2 ?par3)
)
(:action walk_through_gateway
:parameters (?areaA ?areaB ?gateway)
:precondition (and (at areaA)
(connected areaA areaB gateway) (K_GATEWAY_LOCATION gateway))
:effect (and (at areaB)
(forall (gw) (not (K_GATEWAY_LOCATION gw))))
)
(:action observe_gateway
:parameters (?areaA ?areaB ?gateway)
:precondition (and (at areaA) (connected areaA areaB gateway))
:effect (K_GATEWAY_LOCATION gateway)
))
```

Figure 7: Generated Domain File

```
<achieve_plp name="walk_through_gateway">
  <parameters>
    <execution_parameters>
      <param name="areaA" />
      <param name="areaB" />
      <param name="gateway" />
    </execution_parameters>
    <input_parameters>
      <param name="gateway_location" >
        <field value="gateway" />
      </param>
      <param name="laser_scan" read_frequency="5" />
      <param name="odometry" read_frequency="5" />
      <param name="arm_controller" read_frequency="1" />
      <param name="current_Aspeed" />
      <param name="current_Lspeed" />
    </input_parameters>
...
  </parameters>
  ...
  <preconditions>
    <predicate_condition name="at">
      <field value="areaA" />
    </predicate_condition>
    <predicate_condition name="connected">
      <field value="areaA" />
      <field value="areaB" />
      <field value="gateway" />
    </predicate_condition>
    <formula_condition key_description="no_angular_speed">
      <expression value="begin_Aspeed" />
      <operator type="="/>
      <expression value="0" />
    </formula_condition>
    <formula_condition key_description="no_linear_speed">
      <expression value="begin_Lspeed" />
      <operator type="="/>
      <expression value="0" />
    </formula_condition>
  </preconditions>
  ...
  <side_effects>
  <forall_effect>
<param name="gw" />
<assignment_effect
key_description="losing_observed_locations">
  <param name="gateway_location" >
<field value="gw" />
  </param>
  <expression value="NULL" />
</assignment_effect>
  </forall_effect>
  </side_effects>
  ...
  <achievement_goal>
    <predicate_condition name="at">
      <field value="areaB" />
    </predicate_condition>
  </achievement_goal>
  ...
</achieve_plp>
```

Figure 8: Walk-Through-Gateway PLP

```
<observe_plp name="observe_gateway">
  <parameters>
    <execution_parameters>
      <param name="areaA" />
      <param name="areaB" />
      <param name="gateway" />
    </execution_parameters>
    <input_parameters>
      <param name="camera_controller" />
      <param name="odometry" />
      <param name="arm_controller" />
      <param name="begin_Aspeed" />
      <param name="begin_Lspeed" />
    </input_parameters>
    <output_parameters>
      <param name="gateway_location">
        <field value="gateway" />
      </param>
    </output_parameters>
    ...
  </parameters>
  ...
  <preconditions>
    <predicate_condition name="at">
      <field value="areaA" />
    </predicate_condition>
    <predicate_condition name="connected">
      <field value="areaA" />
      <field value="areaB" />
      <field value="gateway" />
    </predicate_condition>
    <formula_condition key_description="no_angular_speed">
      <expression value="begin_Aspeed" />
      <operator type="="/>
      <expression value="0" />
    </formula_condition>
    <formula_condition key_description="no_linear_speed">
      <expression value="begin_Lspeed" />
      <operator type="="/>
      <expression value="0" />
    </formula_condition>
  </preconditions>
  <observation_goal_parameter>
    <param name="gateway_location">
      <field value="gateway" />
    </param>
  </observation_goal_parameter>
</observe_plp>
```

Figure 9: Observe-Gateway PLP

```
(define (problem PLP_PROBLEM)
(:domain PLP_DOMAIN)
(:objects OBJ1 OBJ2 ... OBJ_N)
(:init ATOM1 ATOM2 ... ATOM_N)
(:goal CONDITION_FORMULA)
)

#COMMENTS:
1. Add objects for the following predicates:
- (at ?par)
- (connected ?par1 ?par2 ?par3)
2. As confirmed with the user - assumes the following
    is true in the initial state (no need to mention them):
- [begin_Aspeed = 0] @ walk_through_gateway
- [begin_Lspeed = 0] @ walk_through_gateway
- [arm_moving = FALSE] @ walk_through_gateway
- [begin_Aspeed = 0] @ observe_gateway
- [begin_Lspeed = 0] @ observe_gateway
- [arm_moving = FALSE] @ observe_gateway
```

Figure 10: Generated Problem File Template

Productivity Challenges for Mars Rover Operations

Daniel Gaines, Robert Anderson, Gary Doran, William Huffman, Heather Justice, Ryan Mackey, Gregg Rabideau, Ashwin Vasavada, Vandana Verma, Tara Estlin, Lorraine Fesq, Michel Ingham, Mark Maimone, and **Issa Nesnas**

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109
{*firstname.lastname*}@jpl.nasa.gov

Abstract

Achieving consistently high levels of productivity for surface exploration missions has been a challenge for Mars missions. While the rovers have made major discoveries and accomplished a large number of objectives, they often require a great deal of effort from the operations teams and achieving objectives can take longer than anticipated. This paper describes the early stages of a multi-year project to investigate solutions for enhancing surface mission productivity. A primary focus of this early stage is to conduct in-depth studies of Mars Science Laboratory science campaigns to gain a deeper understanding of the factors that impact productivity, and to use this understanding to identify potential changes to flight software and ground operations practices to increase productivity. We present the science campaigns we have selected along with a conceptual model of how surface missions achieve objectives that is used to guide the study. We also provide some early thoughts on the technologies, and their interactions, which we believe will play an important role in addressing surface mission productivity challenges.

Introduction

The Curiosity rover has been exploring Gale Crater and Mount Sharp since its landing in August 2012. During this time, the Mars Science Laboratory (MSL) mission has accomplished many significant objectives. It has achieved the success criteria for the prime mission, collected evidence that indicates Mars was once habitable, collected over a dozen samples and driven more than 12 kilometers (Grotzinger et al. 2015; Vasavada et al. 2014). Curiosity is currently in its extended mission and continues to make new discoveries as it explores Mount Sharp.

While the Mars rovers, including Spirit, Opportunity and Curiosity, have demonstrated an incredible ability to survive far beyond their designed lifetimes, they still represent limited opportunities to explore the planet. As such, there is great interest in getting the most out of these landed assets over the course of the missions.

Maintaining high levels of productivity for the Curiosity rover is challenging. While the operations team has made significant accomplishments with the rover, doing so often requires a large amount of human effort in planning, coordinating, sequencing and validating the development of command products for the rover. Further, limitations in communication opportunities and anomalies on the vehicle can sometimes cause delays in accomplishing the team's objectives. These productivity challenges can result in the underutilization of the vehicle's resources.

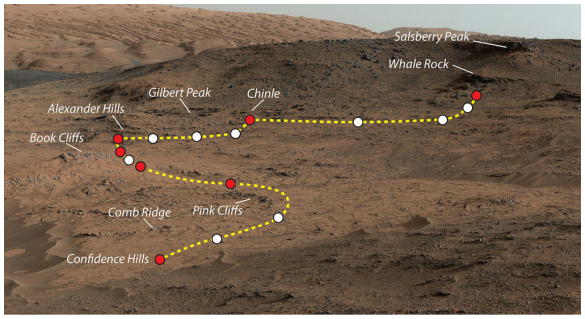
We are conducting a multi-year project to address these productivity challenges. Beginning with in-depth case studies of key MSL campaigns, we will develop a better understanding of the factors that promote and hinder surface mission productivity. Based on the findings from the study, we are developing designs for flight software and ground operations practices to address these challenges. The designs will be prototyped on research rovers and evaluated in realistic operations scenarios.

We are currently in the first year of the project, conducting the MSL case studies. The remainder of this paper describes the case studies we have selected for our investigation with motivations for why we think they will yield interesting results. Next, we provide an overview of the mission operations process to provide context for the discussion of mission productivity. We then provide a conceptual model of how mission objectives are accomplished. The model provides guidance in the collection and analysis of data in the case studies and highlights some of the factors that may influence productivity. Finally, we provide our early thoughts on the types of changes to flight software and ground operations that we anticipate will be important in attaining high levels of surface mission productivity.

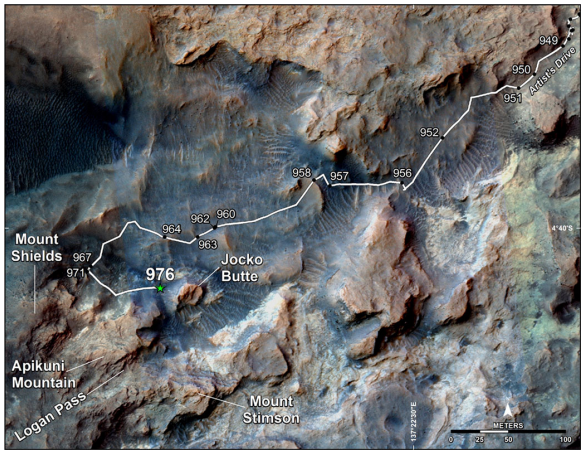
Illustrative Campaigns

We are conducting an in-depth study of some of Curiosity's science campaigns to help increase our understanding of the factors that contribute to and detract from surface mission productivity. For each campaign, we are examining how the operations team decides what to accomplish each day, how well these objectives are achieved and how results from one day feed into and inform objectives for the next.

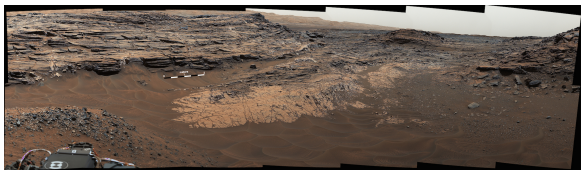
Figure 1 shows the campaigns that we have selected for study. Curiosity began exploring Pahrump Hills (Figure 1 (a) in the fall of 2014 (Stack et al. 2015). Pahrump Hills is an interesting case study for multiple reasons. The light-toned outcrop of Pahrump Hills was the first exposure



(a) Pahrump Hills (Curiosity Mastcam)



(b) Artist's Drive (Mars Reconnaissance Orbiter HiRISE)



(c) Marias Pass (Curiosity Mastcam)

Figure 1: Examples of Curiosity's science campaigns.

of bedrock making up the base of Mount Sharp that was encountered during the mission. The campaign was also significant in the way in which the exploration of this formation was conducted. The science team decided to conduct a "walkabout", a practice used by field geologists when studying unexplored geological areas on Earth. The team made multiple passes of the area with each pass informing a subsequent, more detailed study. We chose to focus on the first walkabout which explored the region with primarily remote sensing instruments (mast-mounted imagery and spectroscopy) in order to identify locations to return to for more detailed follow-up study with arm-mounted instruments. Another interesting factor was the geography of Pahrump Hills was conducive to developing a strategic plan for the initial walkabout. The sloping hills made it possible to see nearly the complete formation from a single panoramic image allowing scientists and engineers to plan a route to explore the area.

After completing investigations at Pahrump Hills, Curiosity departed the area in the spring of 2015 with the objective to reach higher levels of Mount Sharp for continued exploration. Curiosity followed a route referred to as Artist's Drive, shown in Figure 1 (b) (Jet Propulsion Laboratory Press Release 2015). Along the way, the science team conducted a science campaign to capture images of the surrounding topography in order to build a record of the stratigraphy (i.e., layering and structure) of the sedimentary rock layers exposed in the valley walls. The orbital imagery provided by the Mars Reconnaissance Orbiter's HIRISE instrument enabled the team to identify locations where gaps in the surrounding terrain provided the opportunity for imaging the far terrain.

While the orbital imagery enabled the team to develop a strategic route for driving through Artist's drive, the geography of the region made it more challenging for day to day (also referred to as tactical) driving. The orbital data provides good information about the general terrain the rover will encounter, but it is not sufficient resolution for the actual drive path planning. Instead, images acquired from the end of the previous drive are used, in conjunction with orbital data, to plan the next drive. Ridges and valleys in the surrounding terrain often prevented the rover from getting a good view of the terrain in which it would be driving the next day, and wheel wear concerns limited the desirability of using onboard autonomous hazard detection to extend drives into unseen terrain. This often made it difficult for the engineers to plan the next drive path.

Along the route toward higher levels of Mount Sharp, Curiosity took the opportunity to explore an area where the Murray Formation (the type of rock from Pahrump Hills) came into contact with an overlying geological unit called the Stimson Unit. The contact was explored in an area named Marias Pass, shown in Figure 1 (c) (Milliken et al. 2016). This campaign has interesting similarities and contrasts with the earlier Pahrump Hills campaign. Both campaigns sought to explore and characterize a geological area. However, the more challenging terrain in the area and discoveries made during exploration resulted in a more dynamic campaign than the Pahrump Hills walkabout cam-

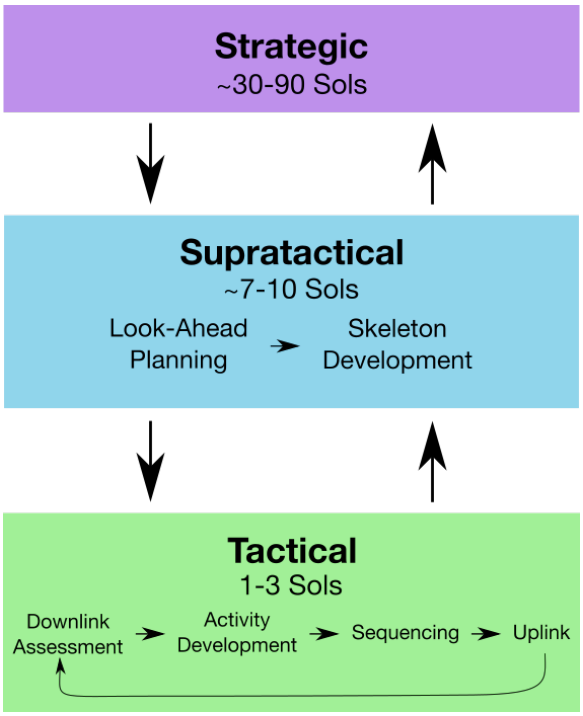


Figure 2: Overview of MSL operations.

paign.

Overview of MSL Mission Operations

One of the challenges a surface mission has compared to an orbital mission is that a surface mission is impacted more significantly by a prior unknown and changing environmental conditions. While orbital imagery provides valuable information to guide activity, it does not capture all the conditions that affect the rover. For example, while orbital imagery may indicate that exploring a particular region is promising to achieve a science objective, the specific science targets are not known until additional data is collected from the rover itself, such as images from its mast-mounted cameras. Further, orbital data is insufficient for fully predicting specific terrain conditions that will impact the rover's traversability and its able to perform close-contact operations on targets of interest.

As such, surface operations must be reactive and respond to the results of activity carried out during the previous sol (Martian day). This daily planning activity is referred to as "tactical" operations and is patterned after the tactical operations developed for the Mars Exploration Rovers (Mishkin et al. 2006).

MSL operations are organized into three major phases: strategic, supratactical and tactical (Chattopadhyay et al. 2014). Figure 2 illustrates the relationship among these phases. These processes are structured to enable the team to achieve long-term science objectives, managing the limited rovers resources, while still responding to the dynamic nature of surface exploration.

Strategic planning focuses on developing long-term plans, typically spanning weeks or months, to achieve high-level objectives. For example, in the Pahrump Hills campaign from Figure 1 (a), the strategic plan spanned several months and specified a multi-pass approach to exploring the region in order to achieve the high-level objectives of performing a comprehensive study of the formation. Strategic planning for the Artist's Drive campaign (Figure 1 (b)) included the development of a Strategic Traverse Route (STR) to provide guidance for selecting paths for the rover on its longer-term objective to reach higher levels of Mount Sharp. These strategic plans provide vital guidance in achieving mission objectives, but they are not directly executable. They must be adapted to take into account the current conditions and adjusted to respond to unanticipated conditions as the rover explores the environment. For example, while the STR provides guidance on the direction the rover should travel, the actual tactical routes may deviate from the route in order to respond to local terrain conditions. And in some cases, significant alterations to the STR were required when a particular path was discovered to be non-traversable or in cases where unexpected science objectives were identified.

The supratactical stage provides a bridge between the long-term strategic plan and the day-to-day, highly reactive tactical process. The process is designed to coordinate the complex science instruments and manage the constraints and resources required to conduct campaigns. The supratactical process produces "look-ahead plans" which span several sols, typically a week, of activity. These plans help maximize the use of vehicle resources. For example, if an energy-intensive, multi-sol sampling experiment is coming up, the look-ahead plan provides guidelines on how much energy can be used each sol of operations. The process also helps with coordination among the large science team spread across the globe.

The process feeds into the tactical process by delivering a "skeleton" plan for each sol of tactical planning. The skeleton provides the tactical team with the major objectives for the plan, e.g. drive toward a particular location, or perform close-contact operations. It includes a rough structure of the activities, including coordination of science activities around communications windows and other engineering activities and guidelines on how much resources, such as energy, time and available data volume, can be expended during the execution of the plan.

The tactical planning process forms the highly reactive phase of surface operations. It includes an assessment of the state of the vehicle and the performance of the previous plan's activities. During activity development stage, specific science and engineering objectives are identified based on the high-level objectives of the current campaign and guidelines provided in the skeleton plan. The developed activity plan is translated into sequencing command products to be executed on the vehicle. These command products are verified, reviewed and delivered for uplink to the rover.

Unlike the Mars Exploration Rovers (MER) mission, which included automated planning ((Bresina, Ari K.J̃nsson, and Rajan 2005)), the MSL mission does not currently include automated planning to assist in activity plan devel-

opment. Instead, MSL operators employ a combination of helper scripts and user-interface operations to support plan development, both as part of supratactical and tactical operations. Helper sequences are used to build the for a variety of purposes including, laying out communication windows according to the overflight database, placing activities to representing turning on and off the CPU, and generating heating activity per the appropriate thermal table. User-interface support includes support for laying out activities back-to-back, snapping an activity to the start or end of another and flagging errors when the activity plan violates rules as defined in the activity dictionary. As Bresina et al. observed, this type of mundane planning support is often more valuable for operations plan development than complex goal achievement.

Factors Impacting Surface Mission Productivity

In general, when we speak of mission productivity, we are referring to how effectively the operations team is able to achieve their objectives. This can include how much effort is required by the team to accomplish a given objective as well as how long it takes, e.g., number of sols, to achieve objectives.

Given the key role of objectives on productivity, we began our case study design by developing a conceptual model, shown in Figure 3, of how the team achieves objectives in a surface mission. Several of the authors have worked surface operations on the MER and MSL missions and this model is based on the authors' experience. During the case study, we will be seeing how well the model explains the data observed in the case study as well as collecting feedback in interviews with other operations personnel.

The general flow of the diagram begins with the team identifying candidate activities that can be used to accomplish their intent. These activities are developed and refined during operations planning until a set of command products is ready to be uplinked to the vehicle. The vehicle executes these activities and produces results which are conveyed back to Earth through telemetry and data products. This information, in turn, is used to support the development of subsequent activities and, potentially, new intent. The crossed out activities illustrate typical stages in the conceptual model in which activity is limited in some way. During operations planning, this can include restricting the scope of an activity, deferring an activity to a later planning day or even descoping an activity entirely. During execution, it can include partial or complete failure of an activity. The following subsections describe each stage and the factors that can limit productivity in more detail.

Both the Supratactical and Tactical team, from Figure 2, perform steps (A) through (C) of Figure 3. The Supratactical team makes these assessments when deciding what activity to include in the high-level look-ahead plan, and feeds this information to Tactical through the skeleton plan. The Tactical team makes similar decisions with the detailed tactical plan, taking into account the latest data from the vehicle.

Step (A): Activity Development

The diagram begins with Activity Development, in Step (A), where the team considers activities that could be performed that would contribute toward achieving their objectives. Objectives may be science objectives, such as characterizing a geological formation, or engineering, such as performing a vehicle maintenance operation of subsystem inspection.

In terms of the operations timeline discussed in the previous section, Activity Development can occur as part of the Supratactical timeline, delivered to Tactical in the Skeleton plan, or during the early stages of the Tactical timeline.

Throughout the planning process, the team makes use of their knowledge of the vehicle's capabilities to help develop command products that the rover will be able to achieve. In the Activity Development stage, this knowledge is used to help determine if an activity is feasible given the abilities and limitations of the science instruments and other actuators. This includes, for example, understanding the detection sensitivity of science instruments and knowing the range of slopes the mobility system can safely traverse. At times, the team will use their vehicle model to come up with creative new ways of using the vehicle's capabilities in ways not previously considered. For example, after landing the team developed methods for driving and performing arm operations with sample cached in the sampling system. In other examples, the team developed a technique for using the MAHLI instrument as a goniometer (Johnson et al. 2015), as well as using the rover's inertial measuring unit to perform a gravimetry survey (Lewis, Peters, and Gonter 2016).

Depending on the type of activity, varying levels of knowledge of the current state of the vehicle may be required. For example, in order to select specific targets for the mast-mounted instruments, knowledge of the position the rover will be in along with navigation images of the surrounding terrain is required. Similarly, activities related to using the arm in close contact with the surface typically require up to date knowledge of the rover and the terrain. In contrast, many activities such systematic survey imagery and atmospheric measurements do not require as extensive knowledge of the current state of the vehicle.

The level of rover state knowledge required to accomplish an activity may prohibit certain activities from being accomplished on a given sol. The amount of knowledge about the state of the vehicle may depend on the downlinked data from the previous plan as well as activity in the current plan. For example, the downlink which contains the latest information about the state of the rover from the prior plan may be delayed, or the communications window between the rover and relay orbiter may have not transferred sufficient data to support all the desired activity. Or, the current plan may include an event that changes the state of the vehicle, e.g. by driving, such that insufficient state knowledge will be available for performing certain activities, e.g. ground-targeted imaging, after the event.

The rover typically performs activities that result in significant changes to its state during the daytime. There are usually one or two communication windows with relay orbiters during the latter part of the day which allow the rover to relay its latest state and other collected data to Earth. Un-

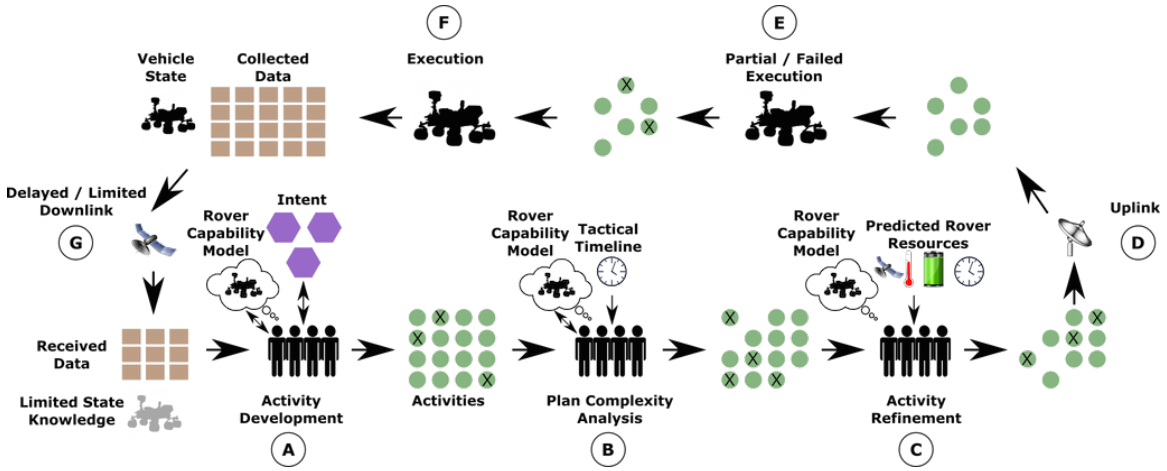


Figure 3: Factors impacting surface mission productivity.

der what is referred to as "nominal" operations, this data will be received by the operations team in the morning on Earth. The team on Earth will then have all day, during the rover's night on Mars, to develop command products that will be sent to the rover during the next morning on Mars. Mishkin has referred to this as working the Martian night shift since the operations team works during the Martian night (Mishkin et al. 2006).

A significant factor in the availability of vehicle state knowledge is the relative duration of a day on Earth and a day (aka sol) on Mars. A Martian sol is approximately 40 minutes longer than an Earth day. As such, if the operations team wishes to continue to work during the Martian night, they must continually shift the times in which they work on Earth. For example, if the team starts their shift at 8:00am one day, they would start their shift at 8:40am the next. Subsequent shift start times would be 9:20am, 10:00am, 10:40am, etc. Over the course of about a month, the team will have transitioned their shift start times around the clock. This mode of operations is referred as "working Mars time" and is highly taxing to the team. Due to the stress this mode of operations places on the team, the MER and MSL missions limited Mars time operations to the first 3 months of the mission.

The vast majority of the surface mission is conducted with the team restricting operations to the daytime on Earth. The consequence is that the operations team is often out of sync with the activity of the rover on Mars. Figure 4 illustrates the impact this can have on the data available to the team during planning. In the diagram, the end-of-day relay from the rover arrives on Earth during the night. If the team had still been working Mars time operations, they would arrive to work at this point and begin the tactical process. Instead, the team begins later in the day. Meanwhile the rover is waking up for its next Mars day. The team on Earth will not have sufficient time to develop a new set of command products by this time. Instead, by the time the team has completed the tactical process, they must wait for the next Mars morning to uplink the products to the vehicle.

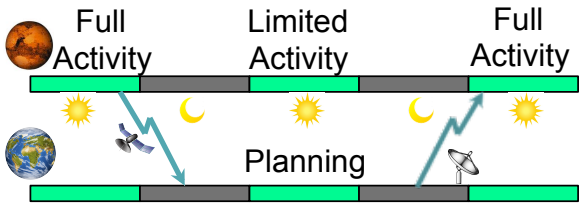


Figure 4: Mars activity vs. Earth planning.

This often limits what the team can command the vehicle to do during the middle sol of Figure 4. If the vehicle were allowed to make significant changes to its state, in particular driving to a new location, this would significantly limit the types of activities the team could command on the subsequent sol. These limited activity sols are referred to as "restricted sols" because the latency of data often restricts the type of activity the team can perform.

A similar situation arises when the team takes days off for weekends and holidays. In these cases, the team will create plans that span multiple sols (aka multi-sol plans). Again, activities that result in significant changes to vehicle state are limited since they will impact the activity that can be done in later sols of the plan.

Step (B): Plan Complexity Analysis

Step (A) discussed how the activities considered depend on the general capability of the rover and the team's knowledge of its state. In Step (B), the team considers the complexity involved in implementing the activities under consideration. As with Step (A), this step may begin with Supratactical planning and continue into the early stages of Tactical operations. The purpose of this stage is to help ensure the team does not take on more activities than can be completely planned and validated during the scheduled shift duration. If the team attempts to perform too much activity it may result in the team not completing the tactical timeline and thus risk missing the next uplink deadline. Or it could mean overload-

ing the team which could lead to mistakes.

As such, the team carefully evaluates the activities it chooses to work on such that they can be completed during the tactical timeline. It is extremely challenging to pick an appropriate set of activities that allows the team to maximize what can be accomplished during the tactical shift without exceeding the capacity of the timeline. It requires a lot of experience and good judgment to make these decisions. Further challenging the decision making is the fact that what can fit in the tactical timeline is continually changing. The first time a new type activity is performed will require more focus and effort from the team. But after that type of activity has been performed several times, it may consume much less of the timeline. New ground tool developments can also result in increasing the capacity of the tactical timeline.

Step (C): Activity Refinement

During Activity Refinement, Step (C), the team takes into account the vehicle resources that are required to perform the proposed activities. The resources the team considers include energy, data volume and time available to perform activities. This stage uses models of the rover and activities to make predictions about the amount of resources that activities will consume and the amount of resources available on the vehicle.

Some of the resource constraints are more or less fixed. For example, the team avoids depleting the battery below a certain level and filling up the data product file system. Other constraints are more transient. For example, the Supratactical team often provides guidelines on the battery state of charge to maintain at the end of the plan. This end-of-plan battery constraint will vary from plan to plan, depending on the activity in the Supratactical look-ahead plan. Similarly, the team may self-impose tighter data collection constraints on itself in preparation for data intensive plans that are known to be upcoming in the near future.

The fidelity of the models used in the stage of operations play an important role. Missions tend to be conservative in their estimates to avoid inadvertently exceeding available resources. The model may overestimate the time and energy consumed by an activity, e.g., by allocating an overly generous margin of time around it. As a consequence, this stage may over-prune activities because the model predicts they would exceed resource constraints when in practice there may have been sufficient resources available.

Step (D): Uplink

Given the complexity of communicating with a tiny robot on a distant, spinning planet millions of kilometers away, the missions have a remarkably reliable channel for sending command products to the rover. However, problems can arise that result in loss of activity during uplink. Noise encountered on the millions of kilometers trip can corrupt the signal beyond the means of error correction codes to correct. Equipment failures on Earth stations can occur with insufficient time to repair before the uplink window. In general, the amount of data required to uplink is very small compared to the amount of data downlinked from the rover. However, there are still rare situations in which the capacity of an uplink window is insufficient to transmit all the desired command products. In each of these cases, some or all of the commanded activity can be lost.

Steps (E) and (F): Execution

After receiving command products from Earth, the rover begins executing the new plan and collecting new data. The commands products are mainly in the form of sequences, files containing lists of commands to execute.

In the large majority of cases, execution proceeds as expected and the rover is able to achieve the desired results. At other times, activities may have partial success or completely fail. There are a variety of causes of unsuccessful execution. Sometimes the command product may have included an uncaught command error which results in a problem during execution. Other times, the current state of the vehicle may have been unexpected. Sometimes sequences are written to take into account uncertainties in the state of the vehicle, but such sequences add complexity to develop, consuming the capacity of the timeline, and the expressivity of the sequencing language can limit what can be sequenced. Different activities have increased levels of autonomy to account for unexpected conditions. For example, the rover is capable of autonomous navigation, which enables the rover to drive to locations without a prior knowledge of the terrain through which it will traverse.

Throughout the plan execution, the rover will produce and collect data products which record the results of its activities. It will also generate telemetry, which includes critical information about the state and health of the vehicle. All of this data is stored onboard awaiting transmission to Earth.

Steps (G): Downlink

The vast majority of the data received from the rover is sent via relay from one of the Mars orbiters. The rover must wait for the orbiter to fly overhead before it can transmit data. The amount of data that can be transferred to the orbiters varies with each window depending largely on the elevation of the orbiter in the sky as it passes over the rover. Data is prioritized by the operations team such that information critical to assessing the health of the vehicle and for planning the next sol's activities is sent earlier in the communication window.

Once the data is onboard the orbiter, it must wait for the orbiter to have a communication opportunity with Earth before the data can finally reach Earth and then get transferred to the operations team for analysis. As with uplink, technical problems may occur during downlink which can result in unexpected delays in data reaching the operations team.

The data becomes input to the next round of planning. It may be used to support the development of further activity, e.g. an interesting target for further study may be identified in a downlinked image, and it may result in new high-level objectives being formed, e.g. unexpected signatures in a spectral analysis may result in a new objective to characterize an area.

Case Study Results

Using the conceptual model in Figure 3 as a guide, we developed a data collection schema that includes intent, activities, constraints and data along with relations among these entities. We worked through each sol of the campaigns, sifting through the plans, acquired data and telemetry, and written reports from operations personnel to collect and organize data with respect to this schema. The data gathering process was a combination of manually reading through activity plans and operations reports along with scripts we developed to assist in the collection process. The scripts we developed included utilities to identify links between data products and the activities that used that data and utilities to collect data on predicted and actual vehicle resource allocations. The objective in gathering this data is to identify cases of low and high productivity during each campaign and to help identify the factors that contributed to each.

A full description of the results of the case study are beyond the scope of this paper. Following is a brief summary of the results. Table 1 presents a rough breakdown of the sol-by-sol activity conducted in each campaign in terms of how activity on each sol contributed toward campaign objectives. Sols labeled "Campaign" were those that directly contributed to the campaign objectives with remote sensing and/or drives. "Campaign Multi-Sol" sols are those in which significant activity was performed toward the campaign objectives as part of a multi-sol plan, either due to a weekend or restricted planning. The reason for calling these sols out separately is that the presence of the multi-sol plan limited the team's options for these sols. For example, had there not been a multi-sol plan, the team may have opted for to move up activity that was performed in a subsequent plan (e.g. a drive activity) which would have reduced the overall number of sols required to achieve the campaign objectives. The "Extra Drives" label denotes sols in which unexpected drives were required. The sols labeled "Deferred" were sols in which campaign objectives were unexpectedly deferred due to the need to respond to an issue identified during tactical plan development or in response to an event from received downlink data. For the Pahrump Hills campaign, the deferred sols were due to an unexpected interaction, identified during tactical planning, between Pahrump Hills objectives and high-priority observations of the comet Siding Springs making its closest approach to Mars. For Artist's Drive, the deferred sol was due to the need to repeat an activity from the previous sol, un-related to the Artist's Drive objectives, as received data showed the activity did not have the intended result. Sols labeled "Post-Drive Multi-Sol" were those sols in which the team was not able to achieve substantial campaign objectives due to lack of data following a drive during a multi-sol plan. Finally, Finally, "Runout" are sols of very low activity that used in cases the team had to create multi-sol plans but the tactical timeline capacity did not allow for sufficient time to develop activities for all sols of the plan.

Comparing the campaigns in Figure 1, we note that despite having different high-level objectives, the sol breakdown for Artist's Drive and Marias Pass appear to be the most similar. This is due to these campaigns having a similar number of restricted plans and both being conducted in similar, challenging terrain conditions.

Comparing the sol breakdown for Pahrump Hills with the other two campaigns shows that restricted sols have a major productivity impact for these types of campaigns. Table 2 shows the number of nominal vs. restricted shifts for each campaign. Pahrump Hills had a total of 9 tactical shifts of which 7 were during restricted periods of the mission. In contrast Artist's Drive and Marias Pass had more total shifts and few restricted shifts than Pahrump Hills.

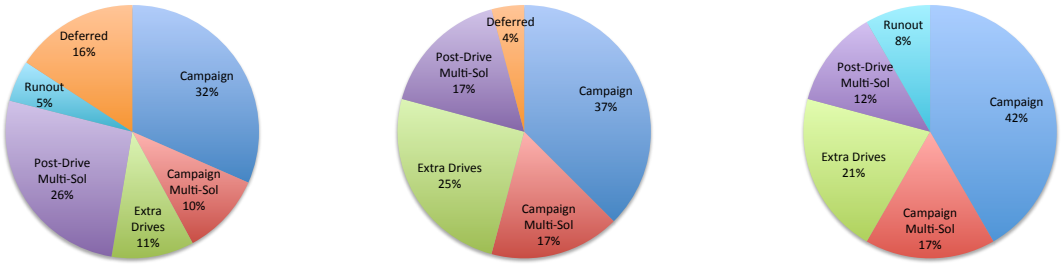
The reason for the differences in number of restricted sols between Pahrump Hills and the other campaigns is largely luck of campaign timing. The Pahrump Hills Walkabout campaign happened to begin just as a restricted period was about to start. On the other hand, the Artist's Drive campaign began just after a restricted period had ended. Marias Pass began toward the end of a nominal period but solar conjunction began before the restricted period began. By the time conjunction was over and the team returned to operations, the restricted period had completed. Thus, it is only toward the end of the Marias Pass campaign that another restricted period impact operations.

The number of restricted sols is anticipated to increase for future missions as the current fleet of sun-synchronous orbiters are replaced with non-sun-synchronous orbiters. There are important science motivations for non-sun-synchronous orbiters, such as studying the Recurring Slope Lineae (RSL). However, such an orbit does not provide the consistent pattern of passes at the end of the rover's day. This will result in many sols in which the operations team does not have sufficient time to develop new plans in response to the latest rover data, thus increasing the number of restricted sols.

Table 2 also highlights the significance of terrain for these types of campaigns. Table 3 summarizes the traverses performed in each campaign. Note that the Artist's Drive traverse marked as "Drive Fault" was also limited by viewshed. Rather than double-count it, we counted it as "Drive Fault" and not "Viewshed Limited". The increased terrain occlusions encountered during the Artist's Drive campaign lead to a larger number of traverses being limited by viewshed than encountered during the Pahrump Hills Campaign. Although the terrain at Marias Pass was more challenging that Pahrump Hills, it had the same number of traverses limited by viewshed. This is likely because the Marias Pass campaign included returning the previously explored areas allowing the team to make use of terrain imagery collected on previous sols. The more challenging terrain of Artist's Drive and Marias Pass resulted in drive faults and the rover stability issues in the associated campaigns.

It is interesting to compare the Pahrump Hills and Marias Pass campaigns as they had similar high-level objectives but were conducted with different exploration strategies. Unlike the Pahrump Hills Walkabout, the Marias Pass campaign did not have an extensive strategic plan to direct activity. This was largely due to the geography of the Marias Pass valley. HiRISE imagery provided a high level overview of the region, with sufficient detail to indicate that the area contact a promising contact between the Stimson formation and

Sol Type	Pahrump Hills	Artist's Drive	Marias Pass
Campaign	6	9	10
Campaign Multi-Sol	2	4	4
Extra Drives	2	6	5
Post-Drive Multi-Sol	5	4	3
Deferred	3	1	0
Runout	1	0	2
Total Sols	19	24	24



Pahrump Hills **Artist's Drive** **Marias Pass**

Table 1: Breakdown of sols for all campaigns.

Sol Type	Pahrump Hills	Artist's Drive	Marias Pass
Nominal Shifts	2	16	12
Restricted Shifts	9	4	4
Total Shifts	11	20	16

Table 2: Summary of shift types for all campaigns.

Sol Type	Pahrump Hills	Artist's Drive	Marias Pass
Nominal	5	6	3
Viewshed Limited	2	5	2
Drive Fault	0	1	2
Insufficient Stability	0	0	1
Total Traverses	7	12	8

Table 3: Summary of traverses for all campaigns.

Murray formation, but contained insufficient detail to form a strategic plan for exploring the location. Because the valley was elevated above the Artist's drive route the rover had been following, it was not possible to obtain the same type of Mastcam panorama that was available for planning the Pahrump Hills Walkabout.

Despite the absence of a detailed strategic plan for the Marias Pass campaign, the team was able to make quick tactical decisions and respond to new data as it arrived such as identifying drive routes and selecting key science targets. This can explain the why Table 1 shows a comparable number of campaign-oriented sols as Artist's Drive. In other words, it seems that the number of restricted shifts and terrain challenges was a bigger factor that the availability of a guiding strategic plan, given the teams ability to react.

It is also interesting to compare the walkabout approach employed at Pahrump Hills vs. the linear approach used at Marias Pass. Although the team intended to use a linear strategy at Marias Pass, then ended up backtracking to explore data collected near the Sol 992 location. There was additional backtracking in the sols that followed the end of our case study sol range. It was suggested by one of the scientists in our interviews that perhaps the Marias Pass campaign would have been overall more efficient had it employed a walkabout approach.

A full assessment of the benefits of these two exploration strategies is beyond the scope of a single case study. The interested reader is referred to Yingst et al. for additional discussion on this topic (Yingst et al. 2015). Their conclusion is that a walkabout approach can take more time to execute, but has the potential for achieving higher quality results. One of the objectives we have with this case study is to leverage what we have learned from these productivity challenges to identify flight and ground approaches that can reduce the overhead of employing a walkabout approach.

We performed a series of analyses on how the team allocated vehicle resources throughout each campaign. This included tracking of predicted and actual allocations of flight computer duration, energy and data volume. Figure 5 shows an example using the flight computer duration allocation during the Pahrump Hills campaign. Multi-sol plans are indicated with vertical black lines.

The analyses of resource allocations followed a similar pattern for each of the campaigns. The impact of multi-sol planning due to weekends and restricted sols had the largest impact in how effectively the team was able to allocate resources toward campaign activity. The analysis showed a general decrease in overall activity across multi-sol plans. This is likely due to limitations in how much activity can be developed during the tactical timeline. In addition, the team is limited in the types of activity that can be performed after a drive without ground-in-the-loop. The analysis also showed a significant decrease in the allocation of resources to campaign objectives following drives during multi-sol plans. This is because ground-in-the-loop is required to per-

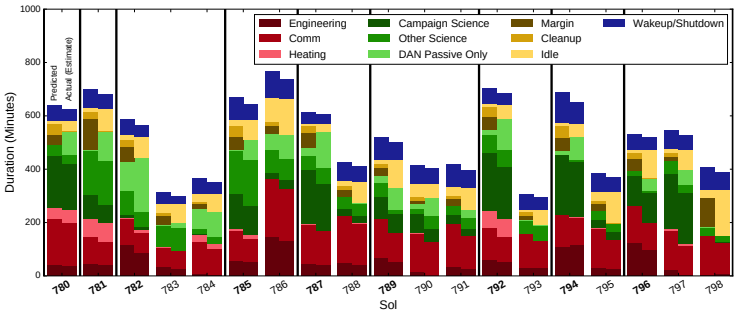


Figure 5: Allocation of flight computer duration for Pahrump Hills Walkabout campaign.

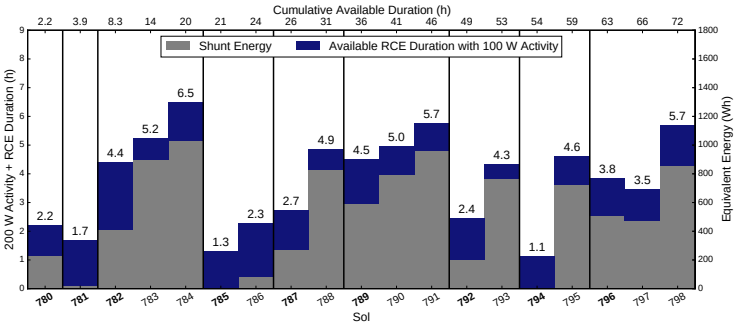


Figure 6: Estimate of extra duration availability for Pahrump Hills Walkabout campaign.

form the majority of the activities needed to accomplish the campaign objectives.

The resource analysis indicated that the team was not constrained by energy during this campaign. There was sufficient unused energy and sufficient non-productive vehicle awake time to support an estimated additional 72 hours of campaign-related activity over the span of the 19 sols for the Pahrump Hills campaign, as shown in Figure 6. Similar analysis estimated an additional 62 hours and 69 hours of campaign activity could have been performed during the Artist's Drive and Marias Pass campaigns, respectively.

As part of the case study, we were also interested in understanding the types of decisions that were made with ground-in-the-loop cycles. Following is a summary of common types of ground-in-the-loop decisions across the campaigns:

- Selecting targets for ChemCam, Mastcam and contact science: While distant imagery of terrain provided sufficient information to indicate the value of traversing to an area, the scientists required the higher quality imaging of the area, obtained when the rover arrives at the site, to select specific targets.
- Drive planning: Post-drive imagery is also used to provide the data necessary to plan the next traverse, including allowing the scientists to refine their selection of end-of-drive location and the engineering team to design a route for the rover to follow.
- Stability assessment for contact science: Prior to deploying the arm and performing contact science, the team must use data from the rover's current position to assess the vehicle's stability.
- Responding to problems in activity executions: It is a complicated mission and plan execution does not always go as expected. Unexpected terrain conditions can cause a drive to end early, resulting in the engineering team assessing the reasons for the problem and re-planning the drive. There are also cases where remote sensing observations do not work as expected. During the campaigns there was a case when imagery was re-acquired due to lighting issues with the first attempt, and cases where ChemCam observations of extremely small features needed to be re-acquired when previous attempts missed.

Addressing Surface Mission Productivity Challenges

The ultimate goal of this case study is to help identify changes to flight software and ground operations that will increase productivity for future missions. Based on the authors' operations experience and preliminary analysis of the results from the case study, we have developed a broad concept for the technologies we anticipate playing important roles in addressing these productivity challenges. The general theme of the changes we are considering is to move more knowledge of intent and more authority for decision making to the rover. Figure 7 provides an overview of the concept which we refer to as a Self-Reliant Rover.

Following is a summary of the key technologies in the concept and motivation for how we anticipate they will support increased productivity.

Goal Elaboration

We believe that an important step in increasing productivity of surface operations is changing the interface the operations team uses to interact with the rover. For the most part, the operations team interfaces with the vehicle with sequences which essentially provide detailed instructions on how the vehicle is to perform the team's desired activities. This approach is inefficient in a variety of ways. It is a time-consuming process to develop and validate the command sequences, especially in situations where the team must try to take into account uncertainty of the state that the rover will be in when the sequences are executed. Because the team does not know the actual state of available resources at the time of plan execution and because conservative resource models are often used, this approach tends to result in under-subscribing rover resources. In other words, the vehicle frequently has more time and energy to perform activities than predicted. Finally, this approach provides very limited ability to respond to unexpected events during execution.

We would like to move to an interface with the vehicle in which we tell the vehicle what tasks we want it to accomplish, rather than telling it how to accomplish tasks. This is important because how a task is accomplished may depend on the current state of the vehicle which is unknown to the operations team at plan development time.

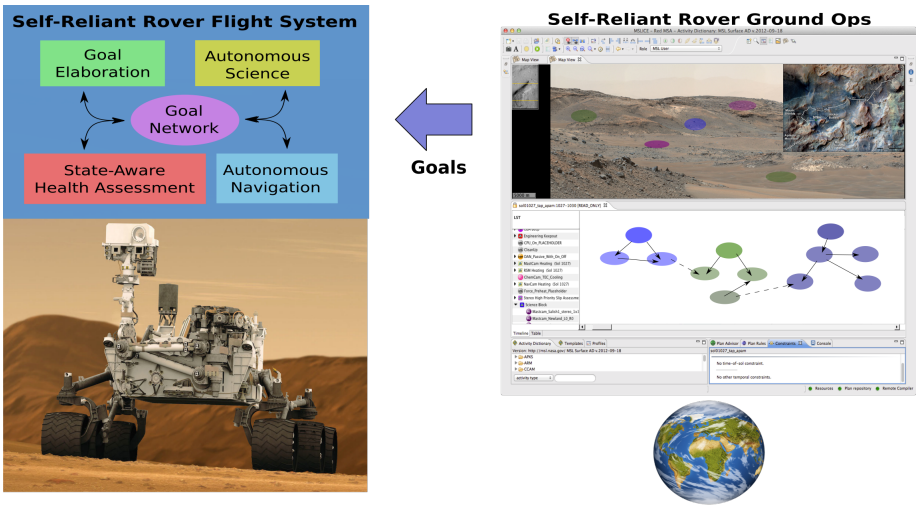


Figure 7: Key technologies for a Self-Reliant Rover.

Therefore, in the Self-Reliant Rover concept, the primary mode of interfacing with the vehicle is the specification of the team's goals. The rover will use goal elaboration to decide how to accomplish these goals given its current state. Further, the team can safely provide a set of goals that would potentially over-subscribe the available resources. The rover will use its up-to-date knowledge of its state along with ground-provided goal priorities to select a subset of goals that can be safely accomplished with available resources. The rover will still perform resource predictions to estimate resource availability for future activity, but these predictions will be continually updated with the latest vehicle state.

This approach, of course, requires more up-front effort in the flight software development process, to develop and validate the onboard planning models used by the rover. But we anticipate that it will result in a large payoff in the form of significant reductions in tactical development and validation of command products.

There exists a large body of work in planning and execution from which we will draw including (Chien et al. 2014; Worle and Lenzen 2013; Fratini et al. 2013; Rajan, Py, and Barreiro 2013; Ceballos et al. 2011; Dvorak, Amador, and Starbird 2008).

Autonomous Science

Many of the science activities performed by the rover require extensive knowledge of the state of the rover and its surroundings at the time the activities will be performed. The result is a significant reduction in science productivity when that information is not available. As discussed previously, this situation arises with restricted sols, weekends and holidays. In addition, it can occur with unfavorable downlink windows or due to unexpected downlink disruptions.

We are interested in exploring autonomous science capabilities to provide a means for scientists to express their intent to the vehicle without requiring up-to-date knowledge of the vehicle's state. The role of autonomous science is not to replace scientists, but to enable scientists to guide the collection of science data in situations in which they would otherwise be unable to do so. The AEGIS system, deployed on the MER and MSL rovers rovers, is an example of this approach to autonomous science (Estlin et al. 2012; Francis et al. 2015). AEGIS allows scientists to specify the types of targets they are interested in acquiring data on following rover drives. We are using the case study to help identify additional opportunities for onboard science to enable scientists to guide rover activity.

The autonomous science component will interact with the rest of the system by posting new goals into the goal network. Goal elaboration will select among these new goals based on available resources.

State-Aware Health Assessment

The traditional approach to health assessment in spacecraft has been to create fault monitors that detect pre-defined failure conditions, typically by detecting when measured values exceed some defined threshold. When a fault monitor is tripped, a pre-defined response is taken in an attempt to isolate the fault. This could involve precluding further use of an instrument or putting the entire spacecraft into a safe-mode, until ground can intervene.

In general, health assessment has been restricted to identifying and responding to pre-defined off-nominal behavior. We are interested in incorporating health assessment into the nominal operation of the vehicle. In our design, health assessment is an integral part of onboard evaluation of the performance of activity as it is executed. This feedback will enable the task executive to monitor ongoing activity, enabling it to make decisions about continuing the activity.

If problems arise, health assessment will enable the vehicle to identify faults and assess their impact in order to determine how to continue to achieve mission goals while ensuring vehicle safety. State-aware heath assessment will integrate with goal elaboration by updating states to reflect the health of various vehicle devices and subsystems. This in turn will pose new goals for longer-term fault response and

impose constraints on how goals can be implemented. The system will maintain a high level of productivity in the face of faults by seeking alternative means of accomplishing impacted goals, if appropriate, or substituting alternative goals that previously did not fit within available resources.

Relevant work in this area from which the project will draw includes (Fesq et al. 2002; Robertson, Effinger, and Williams 2006; Hayden, Sweet, and Christa 2004; Mikaelian, Williams, and Sachenbacher 2005).

Autonomous Navigation

Autonomous navigation is one of the major areas in which rovers are provided intent and allowed to decide how to accomplish tasks (Maimone, Leger, and Biesiadecki 2007). However, when autonomous navigation is used, the operators typically set conservative constraints on the conditions under which the vehicle is allowed to continue autonomous navigation. For example, tight limits may be set on the amount of slip or yaw the rover is allowed to tolerate based on expectations the operators have of the terrain the rover will encounter. During restricted time periods of the mission, when the rover may go multiple sols without interaction with the ground, this conservative strategy can result in a significant reduction in productivity.

We will be exploring ways to make the scheduling of autonomous navigation plans more robust to unexpected terrain conditions. When terrain conditions deviate from expectations, the rover will assess if it is still safe to pursue its current route, if it should plan an alternate course, if it should halt and wait for ground interaction, or if it should give up on its current driving objective and choose a different goal. Decisions will be fed back to the rest of the system to enable coordination with the overall goals of the rover. For example, if the rover chooses to continue with the current path, an update on the time and energy required to complete the traverse will be made. This will be used to make an updated resource projection to see if this expenditure of resources is still consistent with the priorities of goals and required resource margins.

Ground Operations

In addition to changes to flight software, we are also investigating changes to ground operations that will enable future surface missions to address productivity challenges. The scope of these changes includes how the operations team communicates their intent to the vehicle and how command products are validated. Giving the vehicle authority on the goals it accomplishes and the ways in which it accomplishes those goals results in less certainty at planning time on what exactly the vehicle will be doing. This is a significant shift in how the operations team currently validates command products. We will explore ways to give operations personnel expectations on the behavior the vehicle will perform. We will also look for ways in which the ground team can constrain or guide the vehicle's behavior when desired.

Simplifying the interface with vehicle will reduce the time required to generate new plans. This will result in fewer restricted sols as the team will not require as much time to make uplink deadlines.

A major challenge in this work will be relaxing the reliance on up-to-date vehicle knowledge when developing command products. The goal is to make it natural for the team to not know the state the vehicle will be in when the command products are received, but still be able to express intent and guidance to the vehicle so that it can effectively carry out the teams' objectives.

Conclusions

We are in the early stages of a multi-year project to study and address productivity challenges of future surface missions. We have identified campaigns from the MSL mission for study which we believe will yield valuable information about the nature of surface mission productivity challenges. Based on preliminary analysis from the data collected we anticipate that the lessons from these case studies will help develop and mature our concepts for changes to flight and ground systems to address these challenges.

While the focus of our work is on Mars rover missions, we believe the concepts in the work will be applicable to a variety of in-situ explorers, including Venus, and Titan, as well as orbital missions, such as the Europa orbiter. These missions will also benefit from the ability to adapt and respond to the latest state of the spacecraft and its environment.

Acknowledgments

This research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. This work was funded by the Jet Propulsion Laboratory Research and Technology Development program. We are extremely grateful to the MSL team for their support of this project. The generous assistance they provided in enabling access to MSL data, answering our many questions and supporting our interviews was invaluable to this study.

References

[Bresina, Ari K.J̃nsson, and Rajan 2005] Bresina, J. L.; Ari K.J̃nsson, P. H. M.; and Rajan, K. 2005. Activity planning for the mars exploration rovers. In *Fourteenth International Conference on Automated Planning and Scheduling*.

[Ceballos et al. 2011] Ceballos, A.; Bensalem, S.; Cesta, A.; Silva, L. D.; Fratini, S.; Ingrand, F.; Ocon, J.; Orlandini, A.; Py, F.; Rajan, K.; Rasconi, R.; and Winnendael, M. V. 2011. A goal-oriented autonomous controller for space exploration. In *Proceedings of the 11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*.

[Chattopadhyay et al. 2014] Chattopadhyay, D.; Mishkin, A.; Allbaugh, A.; Cox, Z. N.; Lee, S. W.; Tan-Wang, G.; and Pyrzak, G. 2014. The Mars Science Laboratory supratactical process. In *Proceedings of the SpaceOps 2014 Conference*.

[Chien et al. 2014] Chien, S.; Bue, B.; Castillo-Rogez, J.; Gharibian, D.; Knight, R.; Schaffer, S.; Thompson, D. R.; and Wagstaff, K. L. 2014. Agile science: Using onboard autonomy for primitive bodies and deep space exploration. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space*.

[Dvorak, Amador, and Starbird 2008] Dvorak, D. L.; Amador, A. V.; and Starbird, T. W. 2008. Comparison of goal-based operations and command sequencing. In *Proceedings of SpaceOps*.

[Estlin et al. 2012] Estlin, T.; Bornstein, B.; Gaines, D.; Anderson, R. C.; Thompson, D.; Burl, M.; Castano, R.; and Judd, M. 2012. AEGIS automated targeting for the mer opportunity rover. *ACM Transactions on Intelligent Systems and Technology* 3(3).

[Fesq et al. 2002] Fesq, L.; Ingham, M.; Pekala, M.; Eepoel, J.; Watson, D.; and Williams, B. 2002. Model-based autonomy for the next generation of robotic spacecraft. In *Proceedings of the 53rd International Astronautical Congress of the International Astronautical Federation*.

[Francis et al. 2015] Francis, R.; Estlin, T.; Gaines, D.; Bornstein, B.; Schaffer, S.; Verma, V.; Anderson, R. C.; Burl, M.; Chu, S.; Castano, R.; Thompson, D.; Blaney, D.; de Flores, L.; Doran, G.; and Wiens, R. 2015. AEGIS autonomous targeting for the curiosity rovers chemcam instrument. In *Proceedings of the Applied Imagery Pattern Recognition (AIPR) Workshop*.

[Fratini et al. 2013] Fratini, S.; Martin, S.; Policella, N.; and Donati, A. 2013. Planning-based controllers for increased levels of autonomous operations. In *Proceedings of the 12th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2013)*.

[Grotzinger et al. 2015] Grotzinger, J. P.; Gupta, S.; Malin, M. C.; Rubin, D. M.; Schieber, J.; Siebach, K.; Sumner, D. Y.; Stack, K. M.; Vasavada, A. R.; Arvidson, R. E.; III, F. C.; Edgar, L.; Fischer, W. F.; Grant, J. A.; Griffes, J.; Kah, L. C.; Lamb, M. P.; Lewis, K. W.; Mangold, N.; Minitti, M. E.; Palucis, M.; Rice, M.; Williams, R. M. E.; Yingst, R. A.; Blake, D.; Blaney, D.; Conrad, P.; Crisp, J.; Dietrich, W. E.; Dromart, G.; Edgett, K. S.; Ewing, R. C.; Gellert, R.; Hurowitz, J. A.; Kocurek, G.; Mahaffy, P.; McBride, M. J.; McLennan, S. M.; Mischna, M.; Ming, D.; Milliken, R.; Newsom, H.; Oehler, D.; Parker, T. J.; Vaniman, D.; Wiens, R. C.; and Wilson, S. A. 2015. Deposition, exhumation, and paleoclimate of an ancient lake deposit, Gale crater, Mars. *Science* 350(6257).

[Hayden, Sweet, and Christa 2004] Hayden, S. C.; Sweet, A. J.; and Christa, S. E. 2004. Livingstone model-based diagnosis of Earth Observing One. In *Proceedings of AIAA Intelligent Systems*.

[Jet Propulsion Laboratory Press Release 2015] Jet Propulsion Laboratory Press Release. 2015. Quick detour by NASA Mars rover checks ancient valley. `http://mars.jpl.nasa.gov/msl/news/whatsnew/index.cfm?FuseAction=ShowNews&NewsID=1809`.

[Johnson et al. 2015] Johnson, J.; III, J. B.; Hayes, A.; Deen, R.; Godber, A.; Arvidson, R.; Lemmon, M.; Kuhn, S.; Carsten, J.; and Kennedy, M. 2015. Recent Mastcam and MAHLI visible/near-infrared spectrophotometric observations: Kimberley to Hidden Valley. In *Proceedings of the 46th Lunar and Planetary Science Conference*.

[Lewis, Peters, and Gonter 2016] Lewis, K. W.; Peters, S. F.; and Gonter, K. A. 2016. First gravity traverse on the Martian surface from the Curiosity rover. In *Proceedings of the 47th Lunar and Planetary Science Conference*.

[Maimone, Leger, and Biesiadecki 2007] Maimone, M. W.; Leger, P. C.; and Biesiadecki, J. J. 2007. Overview of the Mars Exploration Rovers' autonomous mobility and vision capabilities. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Space Robotics Workshop*.

[Mikaelian, Williams, and Sachenbacher 2005] Mikaelian, T.; Williams, B. C.; and Sachenbacher, M. 2005. Model-based monitoring and diagnosis of systems with software-extended behavior. In *Proceedings of the 20th National Conference on Artificial Intelligence*.

[Milliken et al. 2016] Milliken, R.; Hurowitz, J.; Bish, D.; Grotzinger, J.; and Wiens, R. 2016. The chemical and mineralogical stratigraphy of lower Mt. Sharp: Relating rover observations to orbital predictions. In *Proceedings of the 47th Lunar and Planetary Science Conference*.

[Mishkin et al. 2006] Mishkin, A. H.; Limonadi, D.; Laubach, S. L.; and Bass, D. S. 2006. Working the Martian night shift: The MER surface operations process. *IEEE Robotics and Automation Magazine* 13(2):46–53.

[Rajan, Py, and Barreiro 2013] Rajan, K.; Py, F.; and Barreiro, J. 2013. *Marine Robot Autonomy*. Springer. chapter Towards deliberative control in marine robotics, 91–175.

[Robertson, Effinger, and Williams 2006] Robertson, P.; Effinger, R.; and Williams, B. C. 2006. Autonomous robust execution of complex robotic missions. In *Proceedings of the 9th International Conference on Intelligent Autonomous Systems*, 595–604.

[Stack et al. 2015] Stack, K. M.; Grotzinger, J. P.; Gupta, S.; Kah, L. C.; Lewis, K. W.; McBride, M. J.; Minitti, M. E.; Rubin, D. M.; Schieber, J.; Sumner, D. Y.; Beek, L. M. T. J. V.; Vasavada, A. R.; and Yingst, R. A. 2015. Sedimentology and stratigraphy of the Pahrump Hills outcrop, lower Mount Sharp, Gale Crater, Mars. In *Proceedings of the 46th Lunar and Planetary Science Conference*.

[Vasavada et al. 2014] Vasavada, A. R.; Grotzinger, J. P.; Arvidson, R. E.; Calef, F. J.; Crisp, J. A.; Gupta, S.; Hurowitz, J.; Mangold, N.; Maurice, S.; Schmidt, M. E.; Wiens, R. C.; Williams, R. M. E.; and Yingst, R. A. 2014. Overview of the mars science laboratory mission: Bradbury landing to yellowknife bay and beyond. *Journal of Geophysical Research* 119(6):1134–1161.

[Worle and Lenzen 2013] Worle, M. T., and Lenzen, C. 2013. VAMOS: verification of autonomous mission planning onboard a spacecraft. In *Proceedings of the International Workshop on Planning and Scheduling for Space*.

[Yingst et al. 2015] Yingst, R.; Berger, J.; Cohen, B.; Hynek, B.; and Schmidt, M. 2015. A test of two field methods: Determining best practices in reconnoitering sites for habitability potential using a semi-autonomous rover. In *Proceedings of the 46th Lunar and Planetary Science Conference*.

Safe Motion Planning for Human-Robot Interaction

Jae Sung Park and **Chonhyon Park** and **Dinesh Manocha**
Department of Computer Science
University of North Carolina at Chapel Hill
{jaesungp, chpark, dm}@cs.unc.edu

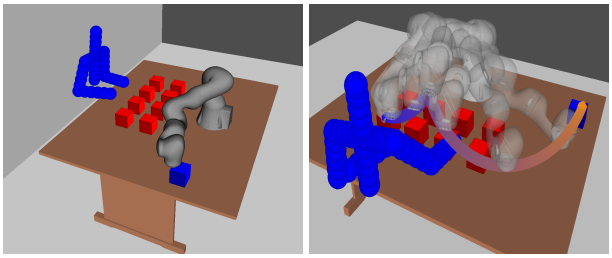
Abstract

We present a motion planning algorithm to compute collision-free and smooth trajectories for robots cooperating with humans in a shared workspace. Our approach uses offline learning of human actions and their temporal coherence to predict the human actions at runtime. This data is used by an intention-aware motion planning algorithm that is used to compute a reliable trajectory based on these predicted actions. This representation is combined with an optimization-based trajectory computation algorithm that can handle dynamic, point-cloud representations of human obstacles. We highlight the performance of our planning algorithm in complex simulated scenarios with a 7-DOF KUKA arm operating in a workspace with a human performing complex tasks. We demonstrate the benefits of our intention-aware planner in terms of computing safe trajectories.

1 Introduction

Motion planning algorithms are used to compute collision-free paths for robots among obstacles. Most of the techniques have been designed for static environments with known obstacle positions. As robots are increasingly used in workspaces with moving or unknown obstacles, it is important to develop reliable planning algorithms that can handle environmental uncertainty and the dynamic motions. In particular, we address the problem of planning safe and reliable motions for a robot that is working in environments with humans. As the humans move, it is important for the robots to predict the human actions and motions from sensor data and to compute appropriate trajectories.

In order to compute reliable motion trajectories in such shared environments, it is important to gather the state of the humans as well as predict their motions. There is considerable work on realtime tracking of human motion in computer vision and related areas (Plagemann et al. 2010; Shotton et al. 2013). However, the current state of the art in collecting such motion data results in many challenges. First of all, there are errors in the data due to the sensors (e.g., point-cloud sensors) or poor sampling. Secondly, human motion can be sudden or abrupt and this can result in various uncertainties in the environment representation. One way to overcome some of these problems is to use predictive or estimation techniques for human motion or actions, such as using various filters like Kalman filters or particle filters (Madhavan and Schlenoff 2003; Vasquez, Fraichard, and Laugier 2009). Most of these prediction algorithms use a motion model that can predict future motion based on the prior positions of human body parts or joints, and corrects the error between the estimates and actual measurements. In practice, these approaches work well when there is sufficient information about prior motion that can be accurately modelled by the underlying motion model. However, these estimates can be inaccurate where there is not sufficient data about human motions, or the prior motions of humans can't be accurately captured by the underlying motion models.



(a) (b)

Figure 1: **Evaluation scenario:** (a) The human (blue spheres) assembles an object by taking parts from eight positions (red cubes). The robot delivers the parts from the blue position to the red positions. (b) The robot motion and the trajectory of robot's end-effector are illustrated for performing human and robot actions simultaneously. In this case, the human is taking a part from position 2, the robot is delivering other part, while avoiding collisions with the human.

In some scenarios, it is possible to infer high-level human intent using additional information, and thereby to perform a better prediction of future human motions (Bandyopadhyay et al. 2013; Turnwald et al. 2016). These techniques are used to predict the pedestrian positions for autonomous vehicles based on environmental information, such as the location of crosswalks or traffic light signals. However, these techniques are limited to the 2D trajectories of the pedestrians on the ground.

In our case, it is important to develop intention-aware techniques that can predict the actions of humans that corre-

spond to high-DOF models. Furthermore, we need to incorporate these human intentions into a motion planning framework that can avoid collisions with the humans and other obstacles in the scene, and still generate smooth trajectories that also satisfy other kinematic and dynamic constraints.

Main Results: In this paper, we present a novel motion planning algorithm to compute safe and collision-free trajectories for robots operating in workspaces involved in human-robot cooperating scenarios, where the robot and a human perform manipulation tasks simultaneously. We use offline training in workspaces with humans. Our approach can used in scenarios where the robot needs to treat each human as a dynamic obstacle and avoid collisions with it. Or it can be used in human-robot cooperating scenarios, where they jointly perform some tasks. We use offline training techniques to generate a database of human motions and actions. These actions are represented using the positions of various joint and our algorithm learns the temporal coherence between different actions for a given task. At runtime, our approach uses the trained human actions and their temporal coherence to predict the future motion from the captured point-cloud data. The robot repeatedly chooses a task that is most demanded for the inferred human action. We use a realtime algorithm to perform probabilistic collision detection between the point cloud representation of a human and the robot, as the robot computes the trajectory for the high level task. These predicted human actions and probabilistic cost functions are integrated into a trajectory optimization algorithm that tends to compute smooth paths as well as satisfy other kinematics and dynamics constraints. We highlight the performance of our planner in a simulator with a 7-DOF KUKA arm operating in a workspace with a moving human and performing cooperative tasks. We demonstrate the benefits of our intention-aware planner in terms of avoiding collisions with the humans.

2 Intention-Aware Motion Planning

Goals of our planner are: (1) planning high-level tasks for a robot by anticipating the most likely next human action and (2) computing a robot trajectory that reduces the collision probability between the robot and human by predicting the human motions. In order to achieve an accurate prediction for human intentions and motions, we use offline learning from a database of actions and motions previously performed to complete some tasks. For high-level task planning, sequences of human actions for completing the whole task are learned based on temporal relations of different subtasks, in order to choose the best action for the robot to take during task planning in real-time.

In the high-level task planning step, we use Markov Decision Processes (MDP), which gives the best action policies for each state. The state of an MDP graph denotes the progress of the whole task. The best action policies are determined through reinforcement learning with Q-learning. Then, the best action policies are updated within the same state. The probability of choosing the action increases or decreases according to the reward function. Our reward computation function is affected by the prediction of intention and the delay caused by predictive collision avoidance.

We also estimate the short-term future motion from learned information. From the joint position information, motion features are extracted based on human poses and surrounding objects related to human-robot interaction tasks, such as joint positions of humans, relative positions from a hand to other objects, etc. We use a Dynamic Time Warping (DTW) (Müller 2007) kernel function for incorporating the temporal information. Given the human motions database of each action type, we train the future motion using a hierarchical k-means clustering and Gaussian Process regression (Rasmussen 2006). The final future motion is computed as the weighted sum over different action types weighed by the probability of each action type that could be performed next.

After deciding which robot task will be performed, the robot motion trajectory is then computed that tends to avoid collisions with humans. An optimization-based motion planner (ITOMP) (Park, Pan, and Manocha 2012) is used to compute a locally optimal solution that minimizes the objective function subject to many types of constraints such as robot related constraints (e.g., kinematic constraint), human motion related constraints(e.g., collision free constraint), etc. Because future human motion is uncertain, we can only estimate the probability distribution of the possible future motions. Therefore, we perform probabilistic collision checking to reduce the collision probability in future motions. We also continuously track the human pose and update the predicted future motion to re-plan safe robot motions.

For more implementation details, please refer to (Park, Park, and Manocha 2016).

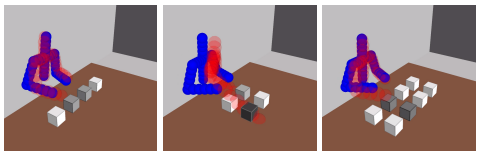
3 Results

We highlight the performance of our algorithm in a situation where the robot is performing a collaborative task with a human and computing safe trajectories. The benchmark scenario is shown in Figure 1. The human is sitting in front of a desk and assembling an object (e.g., Lego). In this case, the robot arm helps the human by delivering the parts from one position that is far away from the human to another position that is closer to the human. The human waits till the robot delivers that part. As different tasks are performed in terms of picking the parts and their delivery to the goal position, the temporal coherence is used to predict the actions.

We use a 7-DOF KUKA-IIWA robot arm model in our simulated environment. The human motion is captured by a Kinect sensor operating with a 15Hz frame rate, and only upper body joints are tracked. The lower body joint positions are ignored as they do not cause any collisions with the robot arm. The action set for a human is $A^h = \{Take_0, Take_1, \cdots\}$, where $Take_i$ represents an action of taking part i from its current position to the new positoin. The action set for the robot arm is defined as $A^r = \{Fetch_0, Fetch_1, \cdots\}$. The robot action $Fetch_i$ should precede the human action $Take_i$, so we set the reward function to positive values in the MDP. Because of the collaborative nature of the tasks, the order of human actions is different from that of robot actions.

Our algorithm has been implemented on a PC with 8-core i7-4790 CPU. We used OpenMP to parallelize the computation of probabilistic collision checking, which is the main bottleneck in the overall planner. Table 1 highlights the performance of our algorithm in three different variations of this scenario: *arrangements of blocks*, *task order* and *confidence level*.

Arrangements of Blocks on the Desk

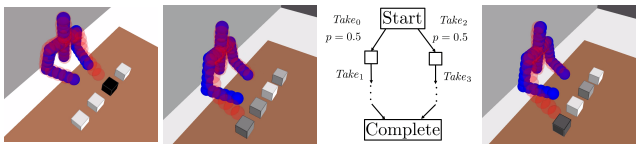


(a) (b) (c)

Figure 2: **Different block arrangements:** Different arrangements in terms of the positions of the blocks, results in different human motions and actions. Our planner computes their intent for safe trajectory planning. The different arrangements are: (a) 1×4. (b) 2×2. (c) 2×4.

In the *Block* scenarios, the position and layout of the blocks changes. Figure 2 shows three different arrangements of the blocks: 1×4, 2×2 and 2×4. In the other two cases, where positions are arranged in two rows unlike 1×4 scenario, the human arm blocks a movement from a front position to the back position. As a result, the robot needs to compute its trajectory accordingly.

Temporal Coherence

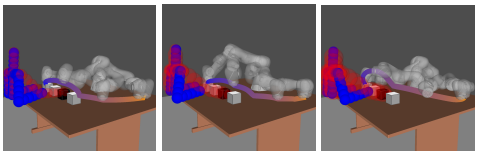


(a) (b) (c) (d)

Figure 3: **Temporal relationships exploited to complete the tasks:** (a) Sequential order; the object parts are selected in a sequential order. (b) Random order; there is no temporal coherence between the human actions. The human randomly selects the next action. (c) (d) Personal order; although there is no temporal coherence, the human selects the object parts according to his or her habit.

Depending on the temporal coherence present in the human tasks, the human intention prediction may or may not improve the performance o the task planner. In the sequential order coherence, the human intention is predicted accurately with our approach with 100% certainty (Figure 3(a)). In the random order, however, the human intention prediction step is not accurate until the human hand reach to the position (Figure 3(b)). The personal order varies for each human, and reduces the possibility of predicting the next human action. When the right arm moves forward a little, $Fetch_0$ is predicted as the human intention with a high probability whereas $Fetch_1$ is predicted with low probability, even though position 1 is closer than position 0.

Confidence Level



(a) (b) (c)

Figure 4: **Probabilistic collision checking with different confidence levels:** A robot trajectory is considered to be collision free with respect to human obstacle if the collision probability is less than or equal to $(1 - \delta_{CD})$. The current pose (i.e., blue spheres) and the predicted future pose (i.e. red spheres) are shown. The robot's trajectory avoids these collisions before the human performs its action. (a) $\delta_{CD} = 0.90$. (b) $\delta_{CD} = 0.95$. (c) $\delta_{CD} = 0.99$.

For a given confidence level δ_{CL}, we compute a trajectory that its probability of collision is upper-bounded by $(1 - \delta_{CL})$. In the *confidence level* scenarios, we analyze the effect of confidence level δ_{CD} on the trajectory computed by the planner, the average task completion time, and the average motion planning time. As the confidence level becomes higher, the robot may not take the smoothest and shortest path so as to compute a collision-free path that is consistent with the confidence level.

4 Conclusions and Future Work

We present a novel intention-aware planning algorithm to compute safe robot trajectories in dynamic environments with human performing different actions. Our approach uses offline learning of human motions, decomposes different high-DOF human actions, and learns about their temporal coherence in the given environment. At runtime, our approach uses the learned human actions to predict and estimate the future motions. Moreover, we perform probabilistic collision checking to compute safe trajectories. We highlight the performance of our planning algorithm in complex benchmarks for human-robot cooperation in simulated scenarios with a 7-DOF robot.

Our approach has some limitations. Our probabilistic collision checking formulation does not take into account robot control errors, which also affect to the collision probability. The performance of motion prediction algorithm depends on the variety and size of the learned data. Currently, we use supervised learning with labelled action types, but it will be useful to explore unsupervised learning based on appropriate action clustering algorithms.

Scenarios	Arrangement	Task Order	Confidence Level	Average Task Completion Time	Average Planning Time
Block 1	1×4	$(0, 1) \rightarrow (2, 3)$	0.95	46.2 s	52.0 ms
Block 2	2×2	$(1, 5) \rightarrow (2, 6)$	0.95	47.8 s	72.4 ms
Block 3	2×4	$(0, 4) \rightarrow (1, 5) \rightarrow (2, 6) \rightarrow (3, 7)$	0.95	109 s	169 ms
Temporal Coherence 1	1×4	$0 \rightarrow 1 \rightarrow 2 \rightarrow 3$	0.95	42.5 s	52.1 ms
Temporal Coherence 2	1×4	Random	0.95	53.6 s	105 ms
Temporal Coherence 3	1×4	Habitual (Figure 3 (d))	0.95	46.5 s	51.7 ms
Confidence Level 1	1×4	$0 \rightarrow (1, 2) \rightarrow 3$	0.90	44.5 s	47.2 ms
Confidence Level 2	1×4	$0 \rightarrow (1, 2) \rightarrow 3$	0.95	45.1 s	50.7ms
Confidence Level 3	1×4	$0 \rightarrow (1, 2) \rightarrow 3$	0.99	63.7 s	155 ms

Table 1: Performance of our planner in different scenarios. We take into account different arrangement of blocks as well as the confidence levels used for probabilistic collision checking.

References

Bandyopadhyay, T.; Jie, C. Z.; Hsu, D.; Ang Jr, M. H.; Rus, D.; and Frazzoli, E. 2013. Intention-aware pedestrian avoidance. In *Experimental Robotics*, 963–977. Springer.

Madhavan, R., and Schlenoff, C. I. 2003. Moving object prediction for off-road autonomous navigation. In *AeroSense 2003*, 134–145. International Society for Optics and Photonics.

Müller, M. 2007. Dynamic time warping. *Information retrieval for music and motion* 69–84.

Park, C.; Pan, J.; and Manocha, D. 2012. ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of International Conference on Automated Planning and Scheduling*.

Park, J. S.; Park, C.; and Manocha, D. 2016. Intention-aware motion planning using learning based human motion prediction. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill.

Plagemann, C.; Ganapathi, V.; Koller, D.; and Thrun, S. 2010. Real-time identification and localization of body parts from depth images. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 3108–3113. IEEE.

Rasmussen, C. E. 2006. Gaussian processes for machine learning.

Shotton, J.; Sharp, T.; Kipman, A.; Fitzgibbon, A.; Finocchio, M.; Blake, A.; Cook, M.; and Moore, R. 2013. Real-time human pose recognition in parts from single depth images. *Communications of the ACM* 56(1):116–124.

Turnwald, A.; Althoff, D.; Wollherr, D.; and Buss, M. 2016. Understanding human avoidance behavior: Interaction-aware decision making based on game theory. *International Journal of Social Robotics* 1–21.

Vasquez, D.; Fraichard, T.; and Laugier, C. 2009. Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion. *The International Journal of Robotics Research*.

Planning Competition for Logistics Robots in Simulation

Tim Niemueller
RWTH Aachen University
niemueller@kbsg.rwth-aachen.de

Erez Karpas
Technion - Israel Institute of Technology
karpase@technion.ac.il

Tiago Vaquero
Massachusetts Institute of Technology
California Institute of Technology
tvaquero@mit.edu

Eric Timmons
Massachusetts Institute of Technology
etimmons@mit.edu

Abstract

Robots gain more capabilities every year, yet the use of planning methods to determine the overall behavior is still the exception rather than the norm. A robotics planning competition (in 2017) could foster mutual and closer cooperation between the planning and robotics communities. A first domain could be based on the RoboCup Logistics League in simulation. We propose this as a challenge to the planning and robotics community. We will organize a *half-day tutorial* in 2016 to introduce the scenario, explain how to use the simulation, and characterize the planning domain for potential participants. The purpose of this paper is to explain the challenge, and serve as grounds for discussion.

Introduction

With robots gaining ever more capabilities, both in terms of perception and manipulation, and with the desire to solve tasks of increasing complexity and higher relevance, the design and composition of robot behavior becomes more complex and tedious. The goal is to automate this process as much as possible, which would improve longevity, extensibility, and robustness of integrated robot systems.

Planning systems would be a natural component for developing complex robot behavior. However, such systems are still the exception rather than the norm in robotics applications. This is, in part, due to the fact that robotic systems are often used to develop, demonstrate, and evaluate specific capabilities like perception or manipulation. On the other hand, the planning community often use robotics as a motivation, rather than as a full evaluation testbed. System integration and actual execution of plans and typical time constraints encountered in robotic domains are not considered. An effort is required to ease the integration and make the communities more accessible towards each other.

To foster closer cooperation among the communities, we envision a new robotic planning competition in simulation. Starting out with a specific scenario, we can pave the way for the development of more integrated systems. The planning community would benefit from a recent and readily prepared evaluation robotics testbed to show the relevance and performance of their work. For the robotics community, planning systems would become more accessible building on or introspecting existing integrated systems in a robotics context.

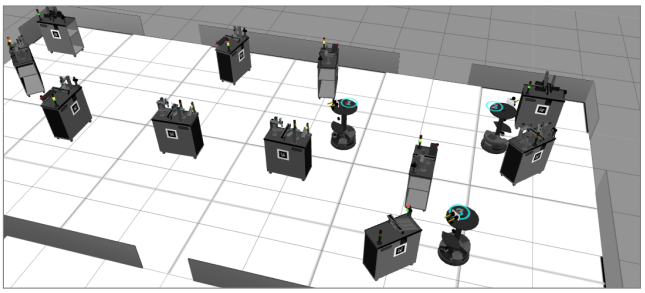


Figure 1: RoboCup Logistics League Simulation

As development and maintenance of actual robots is costly and tedious, the competition will be held in simulation.

This competition builds on the industry-inspired scenario of the RoboCup Logistics League (RCLL) (Niemueller, Lakemeyer, and Ferrein 2015). Manufacturing industries are on the brink of widely accepting a new paradigm for organizing production by introducing perceiving, active, context-aware, and *autonomous systems*. This is often referred to as Industry 4.0 (Kagermann, Wahlster, and Helbig 2013), a move from static process chains towards more automation and autonomy. The corner stones for this paradigm shift are *smart factories* (Lucke, Constantinescu, and Westkämper 2008), which is a context-aware facility in which manufacturing steps are considered as services that can be combined efficiently in (almost) arbitrary ways allowing for the production of various product types and variants cost-effectively even in small lot sizes, rather than the more traditional chains which produce only a small number of product types at high volumes. Flexible and efficient logistics is crucial in such a scenario. The RCLL models this very task at a comprehensible and manageable scale. Methods and techniques for planning and reasoning on a factory, robot fleet, and individual robot levels are highly relevant in this context. The chosen domain may therefore also allow for more interest from industrial partners.

As a preparation for this competition to take place (in 2017), we will hold a half-day tutorial at ICAPS 2016 to present the idea, introduce the integrated open-source base system, and to kickstart interested teams. The purpose of this paper is to briefly present the challenge and foster discussion about the best way to proceed with the challenge.

RoboCup Logistics League (RCLL)

The RCLL (Niemueller, Lakemeyer, and Ferrein 2015) is an industry-inspired league in RoboCup (Kitano et al. 1997), an initiative to foster research in the field of robotics and artificial intelligence. The goal is to organize an efficient workflow in a simplified virtual factory environment.

The game starts with the *exploration phase*, lasting four minutes, where the group of three robots must roam the environment and identify the machines and their positions on the field. To introduce uncertainty and foster the development of robust and failure tolerant systems, two teams operate on a common field at the same time. Each team has an exclusive set of six machines of four different types of Modular Production System (MPS) stations. They are assigned randomly to 24 zones on the field (position and orientation within zone is randomized). The playing field is symmetric along the shorter middle axis to ensure fairness for both teams. In the *production phase* the robots need to coordinate to efficiently operate their machines to refine workpieces to deliverable products according to randomized orders that are posted dynamically at run-time. A product consists of a cylindrical base element, zero or up to three colored rings, and a gray or black cap (order of colors is relevant). These elements can be obtained at different machine types. For further details we refer to Niemueller, Lakemeyer, and Ferrein (2015) and RCLL Technical Committee (2016). Figure 2 shows the finals of the RoboCup 2015 competition in Hefei, China.

The game is controlled by the *referee box (refbox)*, a software component which provides agency to the environment. The simulation (Figure 1) uses the exact same controller and therefore provides reactions similar to the real world. After the game is started, no manual interference is allowed, robots receive goals only from the refbox and must act completely autonomously. The robots communicate among each other and with the refbox through WiFi. Communication delays and interruptions are common and must be handled gracefully – they are also modeled in the simulation.

Task-Level Executive

For a robot to fulfill a certain task, a component is necessary that composes basic skills or actions to form a coherent behavior – this is the task-level executive. Typical approaches can be roughly divided in three categories (Niemueller, Lakemeyer, and Ferrein 2015): state machine based controllers like SMACH (Bohren and Cousins 2010), reasoning systems from Procedural Reasoning Systems (Ingrand et al. 1996) or rule-based agents (Niemueller, Lakemeyer, and Ferrein 2013) to more formal approaches like GOLOG (Levesque et al. 1997), and finally planning systems with varying complexity and modeling requirements. Hybrid systems may integrate aspects of more categories, e.g. PDDL-based planning and GOLOG (Hofmann et al. 2016).

The task-level executive is usually composed of a modeling framework to design the domain model, constraints, and mission goals; a generative planner to perform mission planning or a reasoning system for action selection; a dispatcher to send task commands to robots; an execution monitoring system to overview progress and detect disturbance; and a state estimation system to identify the current state of the world. The technology used in each of these elements varies and the literature is vast. The integration of these components in a robotics application is an interesting and real challenge in both planning and robotics communities.

Figure 2: RoboCup Logistics League Finals 2015

Robotics Planning Competition (RPC)

To bring the robotics and planning communities closer and to foster the integration of planning approaches in more robotic systems, we propose to organize a Robotics Planning Competition. Acquiring and maintaining one or more robots is often prohibitively costly and particularly for many members of the planning community undesirable. Therefore, this robotics competition should take place in simulation. An open source (base) system will ensure that participants can re-create the used infrastructure with reasonable effort.

The idea for the competition was conceived during a discussion in the Workshop on Planning in Robotics at ICAPS 2015. It was observed that even in this focused workshop, papers used robotics mostly as a motivation, rather than as an integration and evaluation testbed, therefore not producing results necessary to attract more members of the robotics community. Based on the presentation of the RCLL (Niemueller, Lakemeyer, and Ferrein 2015) it was deemed to be a suitable candidate. The authors do not claim that this is and should be the one and only simulated robotics domain for such a competition. However, it is readily available and a good match for a start. The topic is also relevant for future industrial production, as demonstrated by Festo as the industrial partner for the RCLL (Niemueller et al. 2013).

Challenges

The main challenges in the Logistics Robot Planning Competition (LRPC) are short-term planning, multi-robot coordination, run-time integration, and dynamic adaptation. As outlined in (Niemueller, Lakemeyer, and Ferrein 2015) various options for planning exist. The domain can be modeled towards *local-scope* (single robot) or *global-scope* (overall fleet) planning, in a *centralized* or *distributed* fashion, e.g., allowing to employ something like plan merging (Alami et al. 1998; Joyeux et al. 2009), and producing either *incremental* or *complete* plans. The most successful strategy in 2014 and 2015 was based on local-scope, distributed, incremental reasoning system (Niemueller, Lakemeyer, and Ferrein

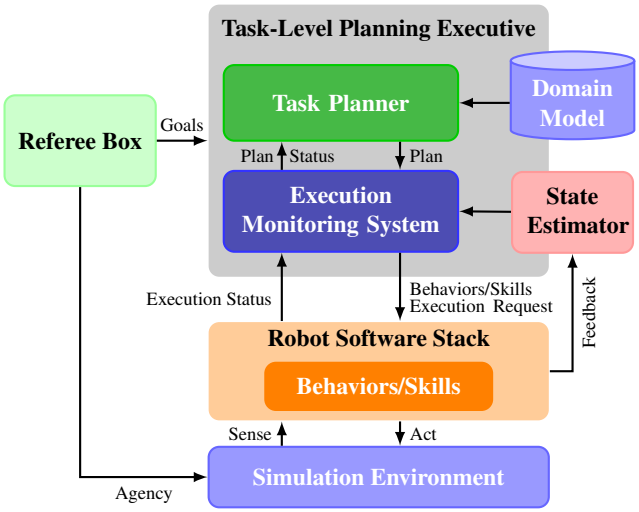


Figure 3: Simulation Architecture

2013). However, achieving cooperation among robots must be encoded explicitly which may cause inefficient interactions and resource usage. Therefore, globally optimizing planning with intrinsic cooperation seems desirable.

Supply chains describe logistic networks which comprise interlinked logistic actors, locally in a factory as in the RCLL, or more generally on a larger scale among factories and industries. Supply chain optimization (SCO), especially as observed at the larger scale, is a hard problem (Radzi, Fox, and Long 2007) showing that the competition will not only serve as a testbed for integration and evaluation, but may rather also pose new challenges. Also, the competition will evolve along the requirements. An alignment with the RCLL is desirable, for one to encourage members of the planning community to participate with real robots or collaborate with existing teams, for another to foster the acceptance of planning techniques in the robotics community. But the planning competition may choose to alter the scenario, for example to operate more robots in a larger environment to scale the problem. This is highly relevant, for example to look at scenario sizes expected in real industrial contexts.

System Architecture and Components

The competition requires the use of a modular system architecture that integrates key components for planning, monitoring, and adaptation. Figure 3 depicts the architecture that will be used to operate the robots in the simulation environment to fulfill the given tasks. In what follows we describe the main components.

Simulation Environment A complete simulation of the RCLL environment, depicted in Figure 1, is readily available (Zwilling, Niemueller, and Lakemeyer 2014) as open source software.[1] The simulation is connected to the Referee Box (refbox) that provides fully autonomous agency.

Domain model We will provide at least one default domain model based on PDDL. Models might have different requirements in terms of fidelity (e.g., temporal or non-temporal) or language (depending on the planner). There is potential for cooperation with the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS).[2]

Task Planner This is obviously a crucial part for the challenge and the center piece which we expect teams to modify or replace. We will provide a reference implementation.

Execution Monitoring System This component is responsible for executing a plan and handling action failure, unexpected events etc. We will provide a default executive based on Enterprise Pike (Levine and Williams 2014).

Behaviors/Skills We will provide a set of standard behaviors — implemented using the Lua-based Behavior Engine (Niemueller, Ferrein, and Lakemeyer 2009) — required to play the game, such as go to a certain place, pick-up a workpiece, or operate a machine.

Robot Software Stack The base robot system is implemented using the Fawkes Robot Software Framework based on the publicly released software stack of the Carologistics RoboCup team (Niemueller, Reuter, and Ferrein 2015). It is available as open source software.[3] Functional components for navigation, self-localization, and perception will be provided. This includes a component path-planning (replaceable in track 2 and 3, see below). Perception will mostly be provided through ground-truth from the simulation.

Middleware The provided infrastructure is based on the Fawkes Robot Software Framework. We will additionally provide a full Robot Operating System (ROS) integration to allow interfacing with all the components of the architecture, having Fawkes then only working in the "engine room".

Competition Tracks

The basic competition will consist of the RoboCup Logistics League task (RCLL Technical Committee 2016) that has to be solved with three robots. Games can be played with a single or two teams at a time (the latter being the official RCLL game increasing uncertainty). The competition will provide great flexibility in terms of software and methods used to perform the task and mission planning. Some basic tracks provide some rough separation to ensure a proper comparison of solutions.

For the time being, we will focus on global centralized planning, that is planning is performed on a single host for the overall fleet of robots. We will focus on providing state-of-the-art planners (e.g., PDDL and RMPL planners) and existing planning architecture (e.g., Enterprise, ROSPlan (Cashmore et al. 2015)). A readily integrated base system will be provided as an example. We envision a number of different tracks, distinguished by the amount of influence the team has on the overall software stack and what components may be replaced. These tracks are explicitly up for discussion with interested teams.

Track 1: Planner Teams may replace the domain model, task planner, execution monitoring system, and/or the state estimator.

If the default execution monitoring system is used an appropriate translation of the planner output to the execution

[1] https://www.fawkesrobotics.org/p/rcll-sim

[2] https://ickeps2016.wordpress.com/

[3] https://www.fawkesrobotics.org/p/rcll2015-release

monitor input is required and must be provided by the teams. The interface to the software stack are either Fawkes blackboard interfaces or ROS topics for information retrieval. To execute actions (if the execution monitor is replaced), a Fawkes interface or ROS action are provided that takes specific skill strings for execution.[4]

We are considering running both a classical and a temporal track, depending on participant interest.

Track 2: Behaviors and Motion Planning In addition to the items of track 1, teams may extend or replace the existing skills (including a different execution engine), and the motion planning component. If a different skill execution engine is used, it must either implement the Behavior Engine interface if the default executive is used, or it must provide a custom executive as well.

The default motion planning component is based on a global dynamically generated graph-based planner and a local planner implementing collision avoidance (Jacobs et al. 2009). Either component may be replaced by the team in this track. Note, that the positions of the machines are not fixed and positions have to be explored during the game.

Track 3: Free Style The team may apply any modification to the robot software, except anything related to the simulation (this denies at least modifications to models, plugins, configuration, or refereeing). We intend to provide a simplified Gazebo-based API that teams can use to integrate with a software stack of their choice. This track is specifically targeted towards interested parties from the robotics community that wants to participate in the challenge.

Conclusion

We have proposed a competition for logistics robots in simulation to bridge the gap between the planning and robotics communities. We will hold a half-day tutorial at ICAPS 2016 to introduce the simulation scenario and outline its planning challenges. The base software and further information is provided on the project website.[5] This paper and the tutorial are meant to start the discussion with interested parties to implement the competition in 2017.

Acknowledgments. T. Niemueller is supported by the German National Science Foundation (DFG) research unit *FOR 1513* on Hybrid Reasoning for Intelligent Systems (`http://www.hybrid-reasoning.org`).

We thank the Carologistics RoboCup Team for their efforts in developing the base system and in particular Frederik Zwilling for countless hours developing the simulation.

We gratefully acknowledge travel funding and equipment by Festo Didactic SE for the tutorial at ICAPS 2016.

References

Alami, R.; Fleury, S.; Herrb, M.; Ingrand, F.; and Robert, F. 1998. Multi-Robot Cooperation in the MARTHA project. *Robotics & Automation Magazine, IEEE* 5(1).

Bohren, J., and Cousins, S. 2010. The SMACH High-Level Executive. *Robotics & Automation Magazine, IEEE* 17(4).

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the robot operating system. In *25th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Hofmann, T.; Niemueller, T.; Claßen, J.; and Lakemeyer, G. 2016. Continual Planning in Golog. In *30th Conference on Artificial Intelligence (AAAI)*.

Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *Int. Conf. on Robotics and Automation (ICRA)*.

Jacobs, S.; Ferrein, A.; Schiffer, S.; Beck, D.; and Lakemeyer, G. 2009. Robust collision avoidance in unknown domestic environments. In *RoboCup Symposium 2009*.

Joyeux, S.; Alami, R.; Lacroix, S.; and Philippsen, R. 2009. A Plan Manager for Multi-Robot Systems. *The International Journal of Robotics Research* 28(2).

Kagermann, H.; Wahlster, W.; and Helbig, J. 2013. Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final Report, Platform Industrie 4.0.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1997. RoboCup: The Robot World Cup Initiative. In *Proc. 1st Int. Conf. on Autonomous Agents*.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3).

Levine, S. J., and Williams, B. C. 2014. Concurrent plan recognition and execution for human-robot teams. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Lucke, D.; Constantinescu, C.; and Westkämper, E. 2008. Smart factory – a step towards the next generation of manufacturing. In *Manufacturing Systems and Technologies for the New Frontier, The 41st CIRP Conference on Manufacturing Systems*.

Niemueller, T.; Lakemeyer, G.; Ferrein, A.; Reuter, S.; Ewert, D.; Jeschke, S.; Pensky, D.; and Karras, U. 2013. Proposal for Advancements to the LLSF in 2014 and beyond. In *Proc. of 1st Workshop on Developments in RoboCup Leagues at IEEE ICAR*.

Niemueller, T.; Ferrein, A.; and Lakemeyer, G. 2009. A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In *RoboCup Symposium 2009*.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2013. Incremental Task-level Reasoning in a Competitive Factory Automation Scenario. In *AAAI Spring Symposium - Designing Intelligent Robots: Reintegrating AI*.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *WS on Planning and Robotics (PlanRob) at Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Niemueller, T.; Reuter, S.; and Ferrein, A. 2015. Fawkes for the RoboCup Logistics League. In *RoboCup Symposium 2015 – Development Track*.

Radzi, N. H. M.; Fox, M.; and Long, D. 2007. Planning in Supply Chain Optimization Problem. In *26th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG-07)*.

RCLL Technical Committee. 2016. RoboCup Logistics League – Rules and Regulations 2016.

Zwilling, F.; Niemueller, T.; and Lakemeyer, G. 2014. Simulation for the RoboCup Logistics League with Real-World Environment Agency and Multi-level Abstraction. In *RoboCup Symposium*.

[4] Refer to `https://trac.fawkesrobotics.org/wiki/Carologistics/skills` for a list of available skills.

[5] `http://www.robocup-logistics.org/sim-comp`

Spatio-Temporal Planning for a Reconfigurable Multi-Robot System

Thomas M. Roehr[1] and **Frank Kirchner**[1,2]
[1] DFKI GmbH Robotics Innovation Center, Bremen, Germany
{thomas.roehr, frank.kirchner}@dfki.de
[2] Robotics Group, Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany

Abstract

In this paper we introduce a spatio-temporal planning and scheduling approach for collaborative multi-robot systems. In particular, we are targeting an application to physically reconfigurable systems in order to take advantage of morphological changes. The planning approach relies on an ontology to model the functionalities individual physical agents offer within the multi-robot system, while an implicit domain representation is given. An inference layer on top of a knowledge-based system allows to account for superadditive effects from physically combining two or more robots. We present a formulation of the domain-specific planning problem and outline our spatio-temporal planning approach. This approach combines the use of constraint-based satisfaction techniques with linear optimization to solve a multicommodity min-cost flow problem to deal with the transportation of immobile robotic systems. The paper describes the findings after implementing core features and evaluates the approach based on a fictitious scientific mission. We close with a discussion of the current limitations of the illustrated approach.

1 Introduction

Robotic space missions rely on highly specialized robotic systems to perform scientific missions. Despite the existing capabilities of these systems, the communication delay in a space-exploratory mission has a significant influence on operations and is a major reason for an overall slow mission progress (NASA 2016). While automation and introduction of techniques from artificial intelligence could mitigate some of these effects, there is a reluctance of increasing autonomy in such systems. One of the reasons for this reluctance is keeping the operational and financial risk to a minimum.

The application of reconfigurable multi-robot systems (Roehr, Cordes, and Kirchner 2014) offers an approach to reduce these types of risk in multiple ways. Firstly, the modularity of such system allows for an incremental mission design, which can include systems with different degrees of specialization. That means, that after an initial deployment phase where a limited number of robotic systems is used, subsequent system development and deployments can take advantage of reconfigurability and extend the functionality of the existing systems. Secondly, the additional degree of freedom that arises from the flexibility of reorganising resources (Evans 1991) within the multi-robot system can be used at the time of mission planning as well as at runtime. Thus, the inherent flexibility (DeLoach and Kolesnikov 2006) of a physically reconfigurable multi-robot system offers a benefit for operations regarding safety, efficiency and efficacy. A reconfigurable multi-robot system encompasses all the benefits of a multi-robot system so that it can mitigate the issue of single-point-of-failure. As an additional benefit, resources can be dynamically shifted within a physically reconfigurable system, so that functionalities and redundancies can be created where they are needed the most. Meanwhile, efficiency can be increased by distributing tasks according to the level of capability of individual robots and while traditional multi-robot systems can increase efficacy by applying general cooperation schemas, reconfiguration adds the possibility of creating physical coalitions aside from creating virtual ones. This not only results in a morphological change, but allows to take advantage of any superadditive effect, e.g., using abilities that are not available within individual systems, but only when two or more join.

The motivation for developing a planner for reconfigurable multi-robot systems comes from the idea of taking full advantage of the available flexibility in an automated way. While will not present results of a fully completed implementation that includes the anticipated multi-objective optimization for redundancy and efficiency, but we illustrate an overall integrated planning approach that accounts for the full flexibility of a reconfigurable multi-robot system. Hence, our main contribution is the outline of an integrated multi-robot planning approach, which allows us to limit the combinatorial explosion that comes with a cooperative multi-robot system. In the following we therefore introduce related work in this area in Section 2. Subsequently, we outline our planning approach in Section 3 and provide details on the implementation and validation in Section 4. The final section 5 provides our conclusions and an outlook of future work.

2 Background

Real reconfigurable multi-robot systems have been successfully developed in the past years, though the main focus of these efforts has been swarm-like systems, i.e. modules that can be almost arbitrarily combined, but are limited with

Figure 1: The multi-robot system which represents the initial motivation and target platform for the planning algorithm.

respect to their final applicability for complex tasks. The work by Roehr et al. (Roehr, Cordes, and Kirchner 2014) presents a recent achievement in this area by allowing more complex and more capable robots to combine and dynamically form coalitions. The project TransTerrA (Sonsalla et al. 2014) builds upon these experiences and develops a reconfigurable multi-robot system that comprises of mobile and immobile robotic agents such as the mobile rover SherpaTT, the mobile shuttle Coyote III, payload-items and so-called base camps; the robotic systems are depicted in Figure 1.

Leading to the progress of developing these kind of reconfigurable multi-robot systems has been the development of interfaces that allow for a reliable electro-mechanical coupling of two physical agents. While the development of such hardware is progressing, the automated exploitation of this additional degree of freedom has yet received little attention.

Theoretical work for organization modelling has been collected by Dignum (Dignum 2009) and OperA and LOA (logic for agent organizations) are two examples providing a formal foundation. Dignum also stresses the importance of dealing with reconfigurable systems as yet another dimension for organizations. Closer oriented towards the multi-agent and multi-robot domain are MOISE+ (Hübner, Sichman, and Boissier 2002) and OMACS (Zhong and DeLoach 2011), which provide models to describe goal oriented organizations to consider restructuring (of task assignments) of such multi-agent systems as error response and based on a metric, which quantifies the ability to fulfil specific tasks. In order to quantify the benefit of a specific configuration of an organization the work of DeLoach et al. (DeLoach and Kolesnikov 2006) is a rare example and relies on a static analysis at design stage.

Looking at the safety property a reconfigurable multi-robot system could be viewed as the so-called superadditive game (Weiss 2009) - a subclass of characteristic function games; the best coalition of agents is a monolith composite agent, i.e. merging all available agents leads to a maximum degree of redundancy. However, a reconfigurable multi-robot system is restricted in the way composite agents can be constructed, i.e. physical interfaces might be already used or are incompatible with each other, thus possible combinations are limited. Rahwan et al. (Rahwan et al. 2011) provide an approach to find the best possible combination for this restricted case, however, the given approach already requires knowledge about compatibility, which becomes impractical even for medium sized teams when agents can be connected via multiple interfaces. Meanwhile, approaches such as ModRED (Baca et al. 2014), and coalition structure generation (Rahwan et al. 2009),(Rahwan et al. 2011) provide means to identify (near) optimal configurations for unrestricted coalition building. Since the search space is the space of all so-called coalition structures $\mathcal{P}^A = 2^{|A|}$, where A is the set of available agents; the cost for this computation can be significant: Rahwan et al. show that it is (Rahwan et al. 2009) $O(2^{|A|})$ and limiting even for a small team size; stronger assumptions allow to reduce this restriction, e.g., Baca et al. (Baca et al. 2014) show a reduction to $O(log|A|)$ but they also assume a constant utility of two linked agents irrespective of the total size the coalition they are in.

Tenorth and Beetz (Tenorth and Beetz 2013) illustrate that ontologies offer a scalable approach to model robotic systems and plan with this information, e.g., they rely on a set of ontologies to describe actions, capabilities and interdependencies. However, they use their ontological description to parametrize so-called action-recipes and focus on single robotic systems, whereas Cashmore et al.(Cashmore et al. 2015) embed ontologies into a full planning approach.

Planning for reconfigurable multi-robot systems is challenging due to their high degree of flexibility and marsupial-like robotic teams (Murphy, Ausmus, and Bugajska 1999) are only one of multiple possible expressions of such reconfigurable multi-robot systems. Wurm et al. (Wurm et al. 2013) apply temporal planning in the context of a multi-robot carrier service which enhances robotic exploration. They use the PDDL-based temporal-planning system TFD/M (Eyerich, Mattmüller, and Röger 2012) which offers the use of so-called semantic attachments or rather external evaluation functions for better context integration, e.g., allowing to use a path-planning component to compute action costs. Similarly, Eich et al. (Eich et al. 2014) perform coordination of a multi-robot team using hierarchical task networks (HTNs), but without explicitly using temporal planning. A combination of (meta-)constraint-based solvers, temporal planning and HTN planning can be found in CHIMP (Stock et al. 2015), which has also been successfully applied to single and multi-robot problems. While all mentioned planning systems are suitable to multi-robot planning, none of these systems takes into account superadditive effects of combining two or more robots.

Numerous timeline-based approaches have been gathered for space-related applications by Chien et al. (Chien et al. 2012) and in our approach we also use a timeline-based representation and Temporal Constraint Networks (Nau, Ghallab, and Traverso 2004),(Dechter 2003) for qualitative temporal reasoning. In addition, we adopt the PLASMA planner approach (Maio et al. 2015) of resource driven planning.

3 Planning with a reconfigurable multi-robot system

The challenge of planning with reconfigurable multi-robot system is directly related to characteristic function games (Weiss 2009). A configuration of the multi-robot system can be viewed as a so-called coalition structure $CS \in \mathcal{P}^A$, where $\mathcal{P}^A$ represents the space of all coalitions structures that can be created from the set of actors A. Rahwan et al. formalize in (Weiss 2009) a coalition structure over A as $CS = \{C_1, \ldots, C_{|CS|}\}$ such that $\bigcup_{i=1}^{|CS|} C_i = A$. By definition coalitions within a coalition structure cannot overlap, i.e., $C_i \cap C_j = \emptyset, i, j = 1 \ldots |CS|, i \neq j$. That means, that each agent of A is part of one and only one coalition in a coalition structure.

Planning needs to consider possible transitions between these coalition structures, while finding the optimal coalition structure is already $O(2^{|A|})$ for a set of agents A (Rahwan et al. 2009).

The planning goal is to provide a feasible plan which outlines core actions to perform a robotic mission, i.e., the mission specification can be seen as the goal specification. The initial intention, however, it not to produce a fully detailed action plan, e.g., when and how manipulation takes place. Instead, the goal lies in exploiting reconfigurability to form composite agents while guaranteeing that resources for specific functionality (and thus actions) will be available at a specific location and time. Thus, a solution will represent a rather coarse grid for a robotic mission (but fulfilling necessary preconditions for functionalities) which can be used by more specialized planners, e.g., manipulation planner or navigation planner, to provide a detailed plan.

In the following we introduce the basic notation, definitions and our assumptions regarding reconfigurable multi-robot systems such as implemented in Roehr et al. (Roehr, Cordes, and Kirchner 2014):

Definition 3.1. *An **atomic agent** $a \in A$ represents a monolithic physical robotic system, where $A = \{a_0, \ldots, a_n\}$ is the set of all atomic agents. An atomic agent cannot be separated into two or more physical agents.*

Definition 3.2. *A mechanically coupled system of two or more atomic agents is denoted a **composite agent** $CA = \{a_i, \ldots, a_j\}$, where $a_i, \ldots, a_j \in A, |A| \geq |CA| > 1$.*

Definition 3.3. *The type of an atomic agent a is denoted $\widehat{a}$ and equivalently for a composite agent CA the type is denoted $\widehat{CA}$. The set of all agent types is denoted $\widehat{A} = \{\widehat{a_1}, \ldots, \widehat{a_n}\}$.*

Definition 3.4. *A (general) agent type $\widehat{GA}$ is represented as a tuple set of agent type and type cardinality: $\widehat{GA} = \{(\widehat{a}_0, c_0), \ldots, (\widehat{a}_n, c_n)\}$, where $a_i \in A$ and $0 \leq c_i \leq |A|$. $\widehat{GA} \supset \widehat{GA'} \iff \forall (a_i, c_i) \in \widehat{GA}, (\widehat{a}_i, c_i') \in \widehat{GA'} : c_i > c_i'$, where $i = 1 \ldots |A|$. Such a tuple set will be denoted an **agent pool**.*

Definition 3.5. *A **reconfigurable multi-robot system** (RMRS) is a set of fully cooperative atomic agents. It can temporarily form composite agents from two or more atomic agents.*

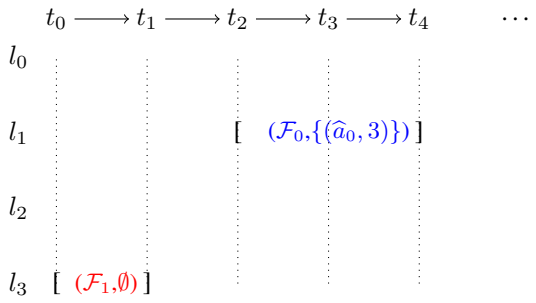


Figure 2: A mission specification example consisting of two spatio-temporal requirements $(\mathcal{F}_1, \emptyset)@(l_3, [t_0, t_1])$ and $(\mathcal{F}_0, \{(\widehat{a}_0, 3)\})@(l_1, [t_2, t_4])$, where $l_0, \ldots, l_3$ are location variables and $t_0, \ldots, t_4$ are timepoint variables.

Assumption 3.1. *Each agent can be mapped to a single agent type only.*

Assumption 3.2. *Each atomic and composite agent comprises a central controller.*

Assumption 3.3. *A mechanical coupling between two atomic agents can only be established through two compatible coupling interfaces.*

Formally, a **robotic mission** is a tuple $\mathcal{M} = (A_a, STR, \mathcal{X})$, where $A_a = \{a_0, \ldots, a_n\}$ is the set of available atomic agents, STR is a set of spatio-temporally qualified expressions (*steqs*) and $\mathcal{X}$ is a set of (temporal) constraints.

A spatio-temporally qualified expression in this context is an expression of the form: $(\mathcal{F}, \mathcal{A}_r)@(l, [t_s, t_e])$, where $\mathcal{F}$ is a set of functionality constants, $\mathcal{A}_r$ is a set of required (general) agent types, l is a location variable, and t_s, t_e are temporal variables describing a temporal interval with the implicit constraint $t_s < t_e$. Currently, we use qualitative timepoints and favour the notation of time slots by start and end time over the specification of a duration, since this (a) allows a future addition of quantitative timepoints (and mix between quantitative and qualitative time), and (b) can be directly translated to the problem solver for constraint-satisfaction problems (CSPs). Figure 2 illustrates a mission specification example. A mission specification can contain partially or fully temporally ordered requirements, i.e., constraints between all qualitative timepoints can remain incomplete in this specification.

Organization model

Reconfigurable multi-robot systems come with great flexibility and as already mentioned come with the possibility of increasing efficacy through cooperation. One of the challenges in planning with such a system lies in accounting for superadditive effects of composite agents, e.g., in our work we want to account for functionality that becomes only available when two or more agents join together - an example: a robot 'mobile' that is mobile, can provide power to external modules, but has no inbuilt camera, can pick up an

(unpowered) camera module 'cam'. In contrast to the atomic agents ('mobile' and 'cam') the newly formed composite agent 'mobile_cam' can take images and take them from any location the robot can reach. We will later refer to this ability of this composite system as 'LocationImageProvider'.

Furthermore, in order to use planning to improve the safety of operations, we have to provide holistic metrics, i.e. we have to account for the state of the multi-robot system as a whole. The introduction of holistic metrics will allow to compare different states and characterise a multi-robot system, e.g., using the overall redundancy level with respect to the required functionality.

To model a multi-robot system, allow for inferring capabilities of agent coalitions and eventually use the model for planning we bring the ideas of Tenorth and Beetz (Tenorth and Beetz 2013) and Cashmore et al. (Cashmore et al. 2015) together. Both our requirements for planning with reconfigurable multi-robot systems, i.e. reasoning for composite agents and providing a holistic metric, are tackled by using a knowledge-base which we denote *organization model*.

The organization model is an ontology that can be augmented with domain-specific and system-specific information; it allows to encode basic functionalities of agents as well as the dependence of an agent's functionality towards the availability of other resources. We account for two resource concepts to describe functionality of an agent: capability and service. A functionality can have resource requirements which are defined using qualified cardinality constraints on the property has; this quantifies ownership of a resource. Figure 3 is an excerpt of an organization model and shows the basic modeling approach. Requirements are defined using *minimum* cardinality constraints in order to allow accounting for redundancies. Meanwhile, resources associated with an agent are defined as *maximum* cardinalities; this allows to encode resource outages for individual agents in the organization model. The organization model can further be augmented with more agent specific data, e.g., transport capacities or power consumption [1].

To identify functional dependencies of agents and infer the availability of a functionality f of a composite agent type $\widehat{CA}$ due to superadditive effects, the following problem definition can be used: searching for the assignment matrix M which maps available resources (described by a vector U that is derived from the given composite agent type $\widehat{CA}$) to required resources (represented by a vector L and derived from the given functionality):

$$L - M \cdot I = \vec{0}$$

subject to

$$u_j - \sum_{i=1}^{\dim(L)} m_{i,j} \geq 0, j = 1 \dots \dim(U)$$

$$m_{i,j} = \begin{cases} \geq 0 & \iff c_{U,j} \sqsubseteq c_{L,i} \\ 0 & otherwise \end{cases}$$

[1]Example ontologies: `https://github.com/2maz/rmrs-pub`

$$\begin{array}{lll}
Capability & \sqsubseteq Functionality \sqsubseteq Resource \sqsubseteq \top \\
Service & \sqsubseteq Functionality \sqsubseteq Resource \sqsubseteq \top \\
MoveTo & \sqsubseteq Capability \\
ImgProvider & \sqsubseteq Service \\
MoveTo & \equiv \geq 1.has.Locomotion \\
 & \quad \sqcap \geq 1.has.Localization \\
 & \quad \sqcap \geq 1.has.Mapping \\
 & \quad \sqcap \geq 1.has.Power \\
ImgProvider & \equiv \geq 1.has.Camera \\
 & \quad \sqcap \geq 1.has.Power \\
LocImgProvider & \equiv \geq 1.has.ImgProvider \\
 & \quad \sqcap \geq 1.has.MoveTo \\
ARobot & \equiv Agent \\
 & \quad \sqcap \leq 1.has.Locomotion \\
 & \quad \sqcap \leq 1.has.Localization \\
 & \quad \sqcap \leq 1.has.Mapping \\
 & \quad \sqcap \leq 4.has.Camera \\
 & \quad \sqcap \leq 1.has.Power
\end{array}$$

Figure 3: Organization model excerpt of a Description Logic (DL) for an atomic agent concept $ARobot$. This illustrates an example formulation for a service named $LocImgProvider$ which reflects the functionality to provide images from specific locations.

$L = [l_1, \dots, l_n]$ represents a vector of minimum cardinalities of required resources for the functionality f, $U = [u_1, \dots, u_n]$ represents the vector of maximum available resources (provided by the set of agents forming the composite agent); n represents the number of different required resource classes. I is the all-ones matrix $\dim(U) \times 1$, M is the assignment matrix $\dim(L) \times \dim(U)$ with entries $m_{i,j}, i = 1 \dots dim(L), j = 1 \dots dim(U)$ which are restricted to 0 or positive integer values, and $c_{V,k}$ is the concept (here: resource class) belonging to entry $1 \leq k \leq dim(V)$ in a vector V. The value $m_{i,j}$ indicates how many instances of a particular resource class j are available to satisfy the request for resources of class i.

The following relationship between the existence of the functionality f and the existence of a solution to the stated problem exists:

$$ARobot \equiv 1.has.f \iff \exists M \; L - M \cdot I = \vec{0}$$

Thus, if an assignment matrix M can be found for a function f and a given agent concept, then the functionality is available for this concept.

Similarly, if the availability of a set of functionalities has to be tested, then the combined resource requirements of individual functionalities have to be considered. We define the required set of resources L_f for a functionality f as:

$$L_f = [l_{f,1}, \dots, l_{f,n}]$$

The required resources for the set of functionalities $F = \{f_1, \dots, f_{|F|}\}$ can then be represented as:

$$L_F = [\max(l_{f_1,1}, \dots, l_{f_{|F|},1}), \dots, \max(l_{f_1,n}, \dots, l_{f_{|F|},n})]$$

At present, the organization model does not directly account for negative effects that might come with forming a composite agent; this is a limitation we intend to tackle in future revisions.

To reduce the search space we apply a functional saturation bound to a set of composite agent types $CT = \{\widehat{CA}_0, \ldots, \widehat{CA}_n\}$ which provides a minimal set of coalition types $CT_{min} \subseteq CT$ that fulfil the functional requirements, so that for any other coalition type $\widehat{CA}_v \notin CT_{min}$ the following holds:

$$\exists C \in CT_{min} : \widehat{CA}_v \supset C$$

To give a more intuitive interpretation of this formulation: we intend to remove any composite agent type from the search space when a subset of its embodied agents is sufficient to provide a requested functionality. We encode this problem as integer-linear program and solve it with a standard solver (cf. Section 4) once for a given mission specification, i.e. initially when required functionalities and the available agent pool are known.

Metrics and Heuristics

To estimate travel cost for an agent we take the same approach as Wurm et al. (Wurm et al. 2013) and estimate the travel time between two locations; we define a nominal speed as a default property for mobile agents, so that based on this information the duration estimate can be computed,

A mapping from location symbols to actual coordinates can be added to the mission specification and as long as no better information is available, e.g., from a path planner (Wurm et al. 2013), the distance between locations will be the basis for cost computations. Robotic systems consume electrical energy and thus all agents come with a nominal power consumption. Hence, duration of actions is not selected directly as cost measure, but total energy consumption. Although the power consumption can vary over time with the type of activity, we assume a constant consumption and leave this improvement as future enhancement. Furthermore, we assume (though not true in all cases) that reducing energy consumption will be a primary goal for optimizing multi-robot plans since it can be mapped to the efficiency of the mission plan.

As mentioned in the introduction, safety is another criteria that has to be taken into consideration. Reconfigurable multi-robot systems can take advantage of their flexibility to exchange resources (by forming new coalitions) in order to adjust the level of redundancy in a composite agents. This leaves the options to deliberately increase redundancy for prioritized tasks or maintain a minimum level of redundancy in general. Demanding a high level of redundancy in a mission tends towards a monolithic system that performs all tasks, while a low level of redundancy thrives for a maximum number of parallel tasks, i.e. maximising efficiency. This means that in the optimization function of the planner a (user-defined) balance has to be established to trade-off redundancy and efficiency.

We will base our safety heuristic on the standard modelling of parallel and serial component-based systems. Each component will be associated with a probability of survival, leading to an overall measure of probability of survival. This information can only be extracted by measuring the performance of the real systems, and the relation to individual components. The reliability R_f of a functionality f can be computed by accounting for parallel components, i.e., resources that are not strictly required but which can serve as replacement:

$$R_f(t) = \begin{cases} 1 - \prod_{i=1}^{n}(1 - p_i(t)) & \text{parallel system} \\ \prod_{i=1}^{n} p_i(t) & \text{serial system} \end{cases}$$

where $p_i(t)$ is the time-dependant probability of survival with $0 \leq p_i(t) \leq 1$, e.g. influenced by component degrading.

Overall, the metric can be seen as characteristic function of a characteristic function game (Weiss 2009). However, here it is a multi-objective optimization function to trade-off safety and efficiency, and the problem does not fit into any of the existing four major subclasses of monotone, superadditive, convex or simple games.

The integration of metrics remains work in progress, however, computing the reliability of a plan will based on a critical path analysis and tracing the dependency of individual functionalities on specific agents.

Planning algorithm

A dedicated domain definition is not provided to the planner, yet a planning domain is implicitly given by using the organization model and accounting for recombination of agents.

A full domain definition can be provided, e.g., by translating the organization model and inference results into Planning-Domain Definition Language (PDDL). Core actions considered for such an encoding are (cf. Figure 4): (i) move, (ii) join, and (iii) split. The *move* action represents a typical change of location and requires three parameters: agent, start location and target location, while split and join refer to a composite agent instance which is uniquely identifiable by the combination of agents.

The required set of predicates: (i) *atomic(a)*: an agent a is atomic, (ii) *operative(a)*: a composite agent a is currently assembled or an atomic agent a is operative (and thus not part of any composite agent), (iii) *at(a,l)*: an agent a is at location l, (iv) *embodies(c,a)*: a composite agent c embodies an atomic agent a, (v) *mobile(a)*: an agent a is mobile (can move by itself), and (vi) *provides(a,f)*: an agent a provides a functionality f.

The action models in Figure 4 have been tested with and transcoded from a PDDL-based representation and to facilitate the transcoding into PDDL, we initially prohibit mixing of composite and atomic agents in a new composite agent. However, this does not limit the modelling capabilities: instead we assume a complete separation of a composite agent into atomic agents when a reconfiguration takes place to form a new composite agent. The corresponding transition can be easily optimized before performing it with the real robots. Furthermore, an atomic agent is either embodied by a composite agent (and becomes a virtual instance which cannot be directly associated with a location), or it is operative and physically present at a location.

$moveto(a, l_s, l_t)$ – move actor a from start l_s to target l_t

$$\begin{aligned} precond: &\quad mobile(a) \wedge operative(a) \wedge at(l_s) \wedge \neg at(l_t) \\ effects: &\quad at(l_t) \end{aligned}$$

$join(c, l)$ – construct the composite actor c at location l

$$\begin{aligned} precond: &\quad \forall z \in A : \neg atomic(c) \wedge \neg operative(c) \wedge \\ &\quad ((embodies(c, z) \wedge operative(z) \wedge at(z, l)) \\ &\quad \vee (\neg embodies(c, z)) \\ effect: &\quad \forall z \in A : at(c, l) \wedge operative(c) \wedge \\ &\quad (\neg embodies(c, z) \vee (\neg at(z, l) \wedge \neg operative(z)) \end{aligned}$$

$split(c, l)$ – split the composite actor c at location l

$$\begin{aligned} precond: &\quad operative(c) \wedge at(c, l) \\ effect: &\quad \forall z \in A : \neg operative(c) \wedge \neg at(c, l) \wedge \\ &\quad (\neg embodies(c, z) \vee (operative(z) \wedge at(z, l)) \end{aligned}$$

Figure 4: Operations as part of the domain definition, for a set of atomic agent A and location variables l, l_s,l_t

While this approach offers the possibility to reuse existing PDDL-based planners in a similar as done by Wurm et al. (Wurm et al. 2013) and Cashmore et. al (Cashmore et al. 2015), we found the need for translating the organization model into this intermediate representation restrictive and counter-productive for our problem. Therefore, we employ a planning and scheduling approach that uses the organization model as an integral part. The algorithm consists of the following main steps: (1) generation of a fully specified qualitative temporal constraint network (2) typing for satisficing assignment also referred to as model assignment (3) role assignment, (4) timeline construction, (5) time-expanded network construction (6) flow optimization, (7) solution evaluation, and eventually (8) generation of multi-robot plan(s).

The core structure of the planning algorithm is illustrated in Algorithm 1, i.e. up to the flow optimization step. This variant is a simple high-level search strategy and illustrates the main ideas of this paper to tackle planning with a reconfigurable multi-robot system.

Temporal constraint network The planning algorithm starts by taking all temporal constraints of the mission and generating a temporal constraint network. Based on this input $nextQualTCN$ in Algorithm 1 computes a qualitative temporal constraint network which is consistent and has no timeline gaps – such a temporal constraint network eventually contains one constraint out of $>, <, =$ between any two qualitative timepoints.

Model assigment A least-commitment principle is applied as part of the model assignment process in order to reduce the search space of coalition structures $\mathcal{P}^A$; this is done by limiting the search to composite agents that satisfy the functional requirements for each spatio-temporal expression while ignoring supersets of such composite agents. Time overlapping requirements with the same location will be merged into one requirement.

The model assignment process requires the quantification

Algorithm 1: TemPl Version 0.1

Data: $\mathcal{M}$: mission spec, $minNumS$: min number of solutions
Result: timeline-based solutions

```
1  begin
2      S = ∅;
       // model assignment conflict resolvers
3      MCR = ∅;
       // role assignment conflict resolvers
4      RCR = ∅;
5      while tcn = nextQualTCN(M) do
6          while nextModelAssignment(M, tcn) do
7              while ra = nextRoleAssignment(M, ma) do
8                  rt = computeRoleTimelines(M, ra) ;
9                  ten = computeTempExpNetwork(M,
                   rt);
10                 mcf = computeMinCostFlow(M, ten);
11                 if conflictFree(mcf) then
12                     s = renderSolution(M, mcf);
13                     S = S ∪ s;
14                     if |S| ≥ minNumS then
15                         return S
16                 else
17                     RCR =getRAConflictResolvers(mcf);
                       MCR =getMAConflictResolvers(mcf);
                       if RCR ≠ ∅ then
18                         r = popResolver(RCR);
19                         applyResolver(M, r);
20                         goto 7;
21             if MCR ≠ ∅ then
                   // no role assignment found
22                 r = popResolver(MCR);
23                 applyResolver(M, r);
24                 goto 6;
25     return S
```

of $support$ of a functionality for an atomic agent type $\widehat{a}$ with respect to a resource class c:

$$support(\widehat{a}, c, f) = \frac{card_{max}(c, \widehat{a})}{card_{min}(c, f)}$$

The functions $card_{min}$ and $card_{max}$ return the minimum and maximum required cardinality for an instance of a resource class, leading to the following definition of support of a function f with respect to a resource class c:

$$support(\widehat{a}, c, f) = \begin{cases} 0 & \text{no support} \\ \geq 1 & \text{full support} \\ > 0 \text{ and } < 1 & \text{partial support} \end{cases}$$

We define the *functional saturation bound* for an atomic agent type $\widehat{a}$ with respect to functionality f using the inverse of $support$:

$$FSB(\widehat{a}, f) = \max_{c \in \mathcal{C}} \frac{1}{support(\widehat{a}, c, f)},$$

where $\mathcal{C}$ is a set of resource classes and $\forall c \in \mathcal{C} : card_{min}(c, f) \geq 1$ to account only for relevant resource

classes. Similarly, the bound for a set of functions $\mathcal{F}$ is defined as:

$$FSB(\widehat{a}, \mathcal{F}) = \max_{f \in \mathcal{F}} FSB(\widehat{a}, f)$$

Two main interpretations of the functional saturation bound exist. First, it is a lower bound on the number of required instances of an atomic agent type to achieve a functionality (if these instances are the sole contributors). Second, it is also an upper bound for the number of instances of an atomic agent type which are actually contributing to achieve this functionality; any excess availability of this agent type is not strictly necessary, but will increase the level of resource redundancy. Hence, the functional saturation bound defines the boundary between supporting functionality and introducing redundancy.

Applying the functional saturation bound allows to reduce the number of agent types that needs to be considered for a satisficing assignment, which is subsequently solved as a CSP. Each variable in this CSP corresponds to a spatio-temporal expression defined in a mission $\mathcal{M}$; each spatio-temporal expression represents a joint requirement of functionality that needs to be fulfilled by an agent and agent (type) availability. Hence, each CSP-variable has a finite domain $D = \{\widehat{CA}_k, \ldots, \widehat{CA}_l\}$ consisting of composite agent types. The solution of this CSP contains the minimum assignments of agent types for each spatio-temporal expression.

Role assignment While the model assignment leaves unsolved what agent (instance) has to be assigned to a requirement, this will be detailed by the subsequent role assignment step. This allows to deal with concurrent activities. A role $role = (i, \widehat{a})$, where $0 < i < N$ is a tuple that represents an identifiable instance of an agent type; N is the maximum number of available agents of the type $\widehat{a}$. The role assignment process takes into account the number of available agents (per type) and introduces unification constraints to allow only feasible concurrent activities. Again, this role assignment is solved as a CSP, where the domain of a variable is the set of all available roles, although this domain will be further limited for individual requirements based on the required number and type of agents. Furthermore, all roles associated with concurrent requirements have to be distinct; this is enforced by introducing inequality constraints between time overlapping requirements. A solution to the role assignment is either empty or contains (correct-by-construction) conflict-free assignments of agent roles for each spatio-temporal expression, i.e. an agent role is associated with one location at a time only.

Having conflict free role assignments is a necessary prerequisite to produce timelines in a subsequent step, i.e. after the role assignment or unification process a timeline is computed for each role (cf. Figure 5).

Logistic network At this stage of the processing it has not been considered, whether an agent is mobile and able to change the location by its own means. Hence, for the next planning step, the algorithm starts to distinguish between mobile and immobile agents. The sub-problem of dealing with mobile and immobile systems resembles the

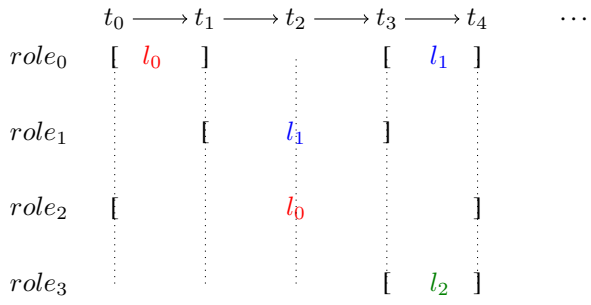


Figure 5: Timelines of multiple roles, representing the spatio-temporal requirement for each role; a role can be mapped to an instance of an agent type.

one in (Wurm et al. 2013), but can be interpreted as network flow problem (Ahuja, Magnanti, and Orlin 1993) or more specifically as a so-called *transshipment* problem.

Mobile agents can offer transport capacities, which can be used to transfer immobile agents; to distinguish between mobile and immobile systems the process relies on checking the availability of two (domain) specific functionalities: (a) 'MoveTo' describes a systems ability to perform location changes by its own means, and (b) 'TransportService' describes a system's ability to carry payload and this capability can be further detailed by specification of transport capacities. The information about available functionalities will also be defined in the organization model for each robotic agent type and clearly, this is a domain specific addition, but we are dealing with a planning problem of physical agents.

Based on the information about mobile and immobile agents, a logistic network can be modelled and solved using flow optimization techniques. We model the flow optimization problem using a time-expanded network. The time-expanded network is a directed graph $G(V, E)$. Each vertex $v \in V$ represents a tuple (l, t), where l is a location variable and t represents a (qualitative) timepoint. Each directed edge $e \in E$, $e = (v, v')$ with $v = (l, t)$ and $v' = (l', t')$ has to fulfil the temporal constraint: $t < t'$, and each role's timeline corresponds to a path in this graph (under the mentioned assumption of an underlying strongly connected temporal network). The multicommodity min-cost flow problem has been formulated as integer programming problem based on the most-general formulation with commodity dependant upper and lower bounds on edges (Kennington 1978); a commodity represents a resource type that can be transported across an edge in the graph. Each edge e has an upper bound for the overall capacity ub_e and a lower and upper bound for each commodity k: $0 \leq lb_e^k \leq ub_e^k \leq ub_e$. We extend this formulation to provide some control on the flow and allow to define a *trans-flow* constraint for a commodity on a vertex, i.e. setting a minimum inflow for a vertex that has no supply or demand for this commodity. Due to balancing constraints, i.e. inflow and outflow need to match supply or demand a valid solution will contain a symmetric outflow of the commodity from the given node.

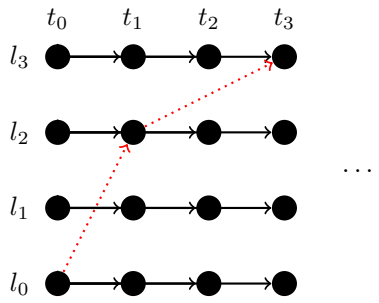


Figure 6: The initial time-expanded network with temporal constraints: $t_0 < t_1 < t_2 < t_3$ and upper capacity bound of $\leq \infty$. The dotted path represents the timeline-based path of a mobile agent a, for which an upper capacity bound UB_a can be set - given the transport capacity of this mobile agent.

In order to perform flow optimization the mission specification has to contain information about the initial location of all resources. The initial location can be interpreted as *supply* node, whereas subsequently the final destination is defined as *demand* node. There might be even unused resources that remain at the initial location as part of the solution. In order to consider the timeline for immobile systems the corresponding spatio-temporal requirements are expressed by setting supply on the start vertex, the demand on the final vertex, and the trans-flow constraint for the role, i.e. commodity it maps to, on all intermediate vertices.

Flow violations Finding a valid minimum cost flow might fail and the failed state of the min cost flow graph can be analysed to identify flow violations. Two types of violation are possible: (i) *trans-flow violation*: a commodity is not routed through a location at a specific timepoint, or (ii) *min-flow violation*: a commodity is not supplied to a location at a specific timepoint.

Both violation types can be addressed by the following resolution strategies: (a) *role distinction*: increasing the role distinction (for the amount of the missing resources) between two spatio-temporal constraints where the violation is found, or if the system is immobile (b) *transport request*: requesting the presence of a mobile system with transport capability

Plan rendering If a solution to the flow optimization problem has been found, it can be translated into a plan for the multi-robot system (or a plan per role). This solution can be characterised regarding safety and efficiency, e.g., computing the associated level of redundancy and the expected energy required to execute this solution.

4 Implementation and validation

In the following, we detail the implementation of our planning and scheduling system and while we aim at a scalable approach our application targets medium-sized reconfigurable robotic teams consisting of about 25 member agents.

We target an application on small board computers and thus try to maintain a consistent C++-based code-basis, e.g., for the ontology based organization modelling we created a C++-variant of owlapi (Horridge and Bechhofer 2011). This allows us to take advantage of ontology modelling capabilities including support for datatypes and qualified cardinality constraints. Reasoning on ontologies is based on the Description Logic (DL) reasoner FACT++ (Tsarkov and Horrocks 2006). A custom reasoner is introduced for the organization model (cf. Section 3) and relies on solving integer-linear programs. For dealing with constraint-satisfaction problems we rely on Gecode (Schulte and Tack 2012). All our graph-related subproblems such as temporal constraint networks and min-cost flow optimization share the same graph library implementation.

The specification of temporal constraints in a mission is based on qualitative relations and despite its more limited expressiveness point algebra (PA) has been selected as main representation due to better computational characteristics. Qualitative reasoning and generation of qualitative temporal constraint networks without time gaps relies on GQR (Gantner, Westphal, and Wölfl 2008) which is combined with a custom implementation of qualitative reasoning to facilitate the future combination of qualitative with quantitative temporal reasoning (Meiri 1996).

The mission specification is given by a user in XML [2] and allows to associate location constants either with tuples of latitude-longitude or Cartesian 3D coordinates; this allows the computation of metric distances between two locations (an additional radius parameter allows to map latitude-longitude coordinates to metric distances) to identify overall plan cost.

Tests have been performed on an PC equipped with an Intel CORE i7-4600U 2.1 GHz with 12 GB of memory.

A mission example

To illustrate the working of the core features of our approach we present here two example missions that are driven by some scientific goals. The following section describes one mission example using a more abstracted representation since it allows a more compact illustration. Subsequently, we will provide a more concrete and slightly more complex example, including the corresponding solution graph.

Abstracted mission We assume an initially given set of robots implicitly given by a composite actor type $\widehat{A}_I$; all agents are available to perform a mission illustrated in Figure 7. Agents of types $\widehat{a}, \widehat{b}$, and $\widehat{c}$ are mobile, where $\widehat{a}$ has a transport capacity for eight immobile systems and the remaining for one immobile system; agents of type $\widehat{d}$ and $\widehat{e}$ are immobile and as such require to join with any of the mobile agents to change their location. All or a subset of these robots can be used to produce a satisficing solution.

We assume a space-exploration mission, where all resources are initially deployed at a landing site denoted l_0. Starting from this landing site a set of tasks has to be carried

[2] Mission scenarios: `https://github.com/2maz/rmrs-pub`

out at three further locations which are denoted l_1, l_2, and $l3$. The mission specification comprises 9 locations symbols and 14 qualitative timepoints in total. The functionalities required to perform these tasks are $\mathcal{F}_0 = \{f_0, f_1\}$ and $\mathcal{F}_1 = \{f_0, f_1, f_2\}$. Functionality f_2 is only supported by immobile agents of type $\widehat{e}$. The organization model encodes the information on (superadditive) capabilities and the functional saturation bound is applied once before the planning step, i.e. leading to a mapping between composite agent types and available functionalities based on the existing set of agents.

The first iteration of the planning algorithm will lead to an incomplete solution, since the flow optimization step cannot satisfy all constraints. The problem arises through the use of a single agent of type $\widehat{e}$; after the role distribution step has taken place, a single timeline is generated for an agent of type $\widehat{e}$ which supports the functionality $(\mathcal{F}_1,\{(\widehat{d},5)\})$ and $(\emptyset,\widehat{A}_I)$. Initially the role distribution only takes temporal constraints into account, hence both requirements can be served by a single agent. Therefore, the flow optimization suffers from a trans-flow constraint violation on $(\mathcal{F}_1,\{(\widehat{d},5)\})$ after the first iteration, i.e. the required inflow and outflow for a commodity is not fulfilled. To refine the partial plan a resolver introduces additional constraints for role assignment or model distribution, e.g., to fix a role assignment the respective timeline is split, by increasing the distinction of agents between the two affected spatio-temporal requirements. If that repair action does not lead to a feasible plan, resolution has to backtrack to the level of model distribution and add a functional requirement for a mobile system.

For this particular mission example the mincommodity flow optimization requires to encode a linear problem of the following size: 4643 rows, 4290 columns and 12870 nonzeros. Columns correspond to the number of edges times the number of commodities. The problem instance is solved in about $4\,s$ including fixing one flaw in the plan. Applying the linear optimization problem does not result in a primal feasible solution and checking the first solution for constraint violations on trans-flow constraints lead to the application of a resolver. The second iteration still has no primal feasible solution, though there will be no violations on trans-flow constraints. This is due to the fact, that the excess resources exist at the starting point, i.e. a solution might leave resources unused, so that the total demand supply balance constraint will be violated. The degree of this violation can also be quantified using the internal solver results, e.g. Figure 8 illustrates output of the internally used linear program solver; the cost for routing one commodity across an edge is uniformly set to 1. Hence, the result shows that finally 28 commodities can be moved between edges while 20 resources overall remain at their initial position, i.e. are unused. The final assignment is illustrated in Figure 9, where the n^{th} instance of an agent type $\widehat{a}$ will be denoted $i^{\widehat{a}}_n$.

Eventually, splitting and joining of agent groups can be mapped to the implicit actions *join* and *split* (cf. Figure 4), while transitions between different locations map to the *move* action. These implicit action center around a set of

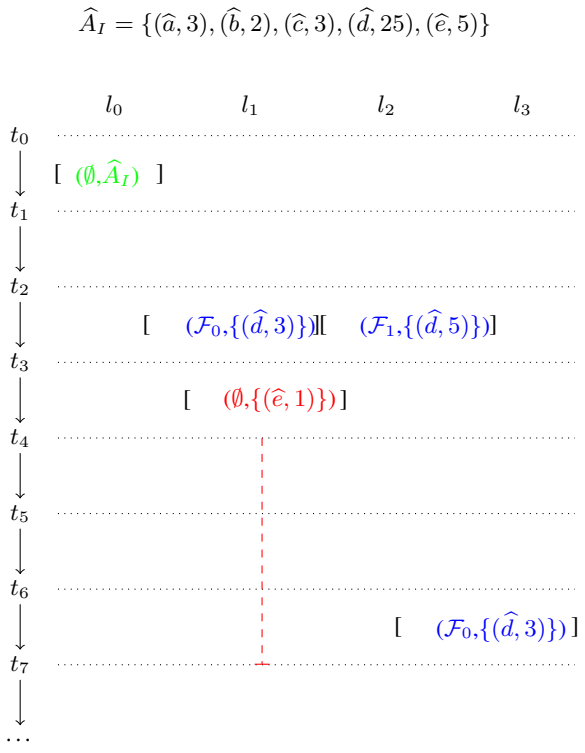


Figure 7: Mission example to describe a robotic mission: $\widehat{A}_I$ is the set of initially available resources, and the first requirement is making all resources available at location $l0$, i.e. setting the starting point. $\mathcal{F}_0 = \{f_0, f_1\}$ is a set of functionalities that needs to be available for a limited time interval.

```
GLPK Simplex Optimizer, v4.52
4643 rows, 4290 columns, 12870 non-zeros
    0: obj =    0.0  infeas =  39.0 (4500)
  500: obj =   26.0  infeas =  21.0 (4000)
 1000: obj =   26.0  infeas =  21.0 (3500)
 1356: obj =   26.0  infeas =  21.0 (3144)
LP HAS NO PRIMAL FEASIBLE SOLUTION

GLPK Simplex Optimizer, v4.52
4643 rows, 4290 columns, 12870 non-zeros
    0: obj =    0.0  infeas =  40 (4500)
  500: obj =   28.0  infeas =  20 (4000)
 1000: obj =   28.0  infeas =  20 (3500)
 1356: obj =   28.0  infeas =  20 (3144)
LP HAS NO PRIMAL FEASIBLE SOLUTION
```

Figure 8: Output of the GLPK simplex optimization (for formatting reasons we have manually shortened this output) for two subsequent planning steps: the first optimization results in an invalid solution where on 26 out of 39 commodities can be moved and triggering repairing the initial flaw by requesting the addition of another mobile agent; in the following iteration 28 out of 40 commodities be moved.

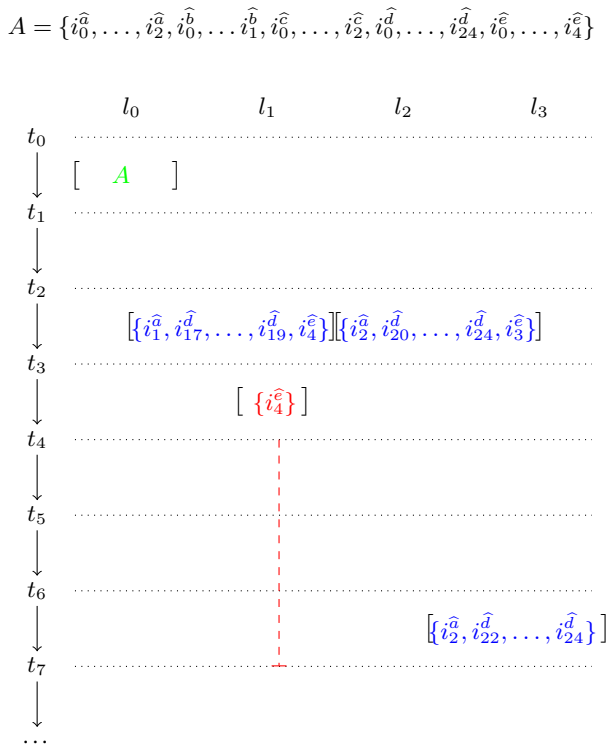


Figure 9: Computed solution for the mission example. The initial set of agents is split into three groups, one remaining at the initial position l_0, and two groups moving to $(l_1, [t2, t3])$ and $(l_2, [t2, t3])$ respectively. An additional split allows a subgroup to continue to $l3$.

atomic agents, which can be combined to form composite agents.

Concrete mission For the concrete mission we assume that all robotic systems are initially available at the location 'lander' and a mission designer outlines the requirement based on activities that might have to be performed at certain location and in some kind of general order. The set of robot types available for this mission is based on the real systems available in (Roehr, Cordes, and Kirchner 2014) and (Sonsalla et al. 2014): the exploration rover Sherpa, the legged crater explorer CREX, the star-wheeled scout Coyote III, 25 Payloads, and 5 BaseCamps - which can be used as logistic hubs to store payloads. In this mission only a Sherpa can carry payloads and up to 8 of them. Payloads and BaseCamps are immobile units, while all other systems are mobile. The following locations are defined: $lander, b1, \ldots, b7$ and qualitative timepoints $t1 < t2 < \cdots < t14$.

The following spatio-temporal qualified expressions define the requirements for the mission:

1. ({},{(Sherpa,3),(CREX,2),(CoyoteIII,3),(Payload,25),(BaseCamp,5)} @(lander,[t0,t1])
2. ({},{(Payload,3})@(lander,[t4,t10])
3. ({LocationImageProvider, EmiPowerProvider},{(Payload,3)})@(b1,[t2,t3])
4. ({},{(Payload,1)})@(b1,[t3,t14])
5. ({},{(BaseCamp,1)})@(b1,[t4,t7])
6. ({LogisticHub, LocationImageProvider, EmiPowerProvider},{(Payload,3)}) @(b2,[t2,t3])
7. ({LocationImageProvider, EmiPowerProvider},{(Payload,6)}) @(b4,[t6,t7])
8. ({},{(Payload,3})@(b4,[t8,t9])
9. ({},{(BaseCamp,3)})@(b4,[t11,t14])
10. ({},{(Payload,1)})@(b6,[t10,t14])

The functionalities *LocationImageProvider* are available on Sherpa, while *EmiPowerProvider* is available on all mobile systems. On a single core a solution is computed in 57.56 ± 9.8 s (averaged over 10 runs), a solution is illustrated in Figure 10; the linear problem to solve the transshipment problem has the following size: 9100 rows, 4320 columns and 21536 nonzeros.

Limitations This paper illustrates first results of a prototype of the planning approach outlined in the previous sections. As such, it requires further improvement and assessment to analyse computational properties and completeness. The optimization for performance has so far focused on the model-based planning approach and the use of the functional saturation bound and performance is expected to be improved further.

A detailed quantification of functionalities is not yet part of the modelling, i.e. a transport capacity can be requested in general, but not the transport capacity of a certain number of some agent type. Currently, this has to be solved by introducing additional spatio-temporal requirements which are nearby to the original one in time and space.

5 Conclusions and Future Work

This paper illustrates an approach towards automated planning and scheduling for reconfigurable multi-robot systems that accounts for the embodiment of agents and reconfigurability. In this work we bring knowledge engineering and temporal planning together to find a practical and scalable solution of dealing with reconfigurable multi-robot systems.

The organization model is a key element to allow reasoning with capabilities of atomic and composite agents. Since the organization model is encoded in Web Ontology Language (OWL), which is a specification of the World Wide Web Consortium (W3C), it offers a well defined interface to users for modelling as well as for interoperation and exchange of information within the multi-robot system. By this and similar to reconfigurable multi-robot systems it supports incremental mission design since it can grow with the system, e.g., by introducing new functionalities and agent descriptions.

Our planning approach uses the newly introduced functional saturation bound to limit the effects of combinatorial explosion. While the functional saturation bound cannot prevent combinatorial explosion, it can reduce the planning problem even when hundreds of agents are available for the mission since handling of redundant coalitions of agents is avoided. The core planning approach relies on temporal networks that have no timeline gaps, but we use a CSP-based

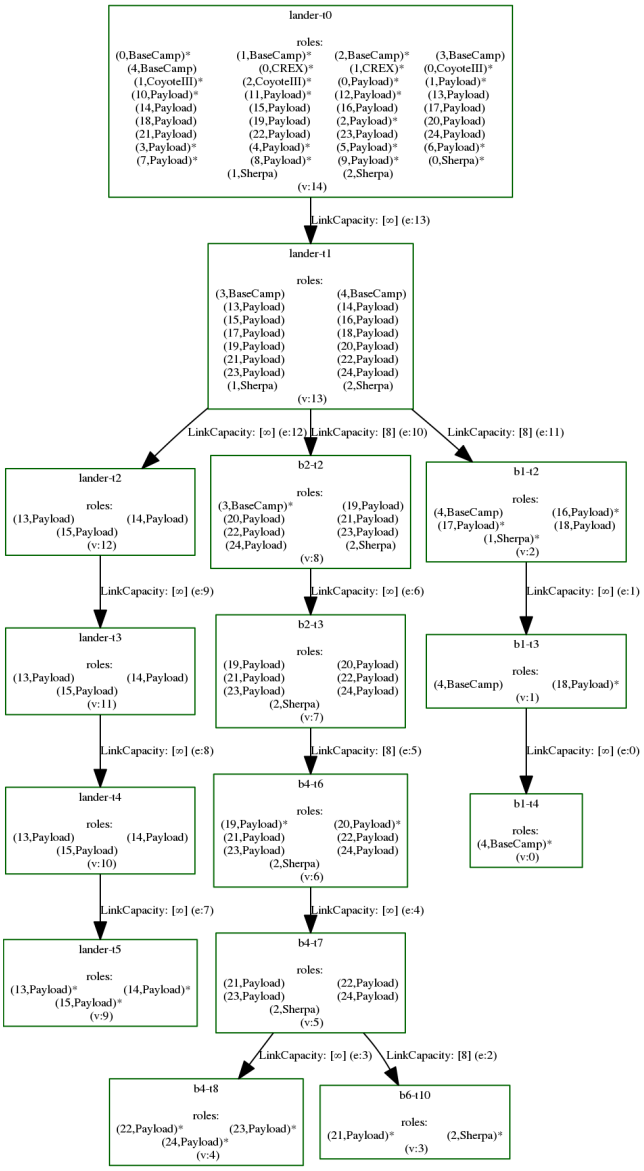


Figure 10: A final mission outline computed by our planning system from which individual plans can be computed from. The asterisk (*) marks roles that have reached their final destination. Link capacities depend on the robotic agents transferring from one location to another; link capacities to transfer to, i.e. remain at, same location are infinite.

generation of temporal constraint networks to satisfy this requirement. Furthermore, we adopt of a flaw-based plan repair strategy similar to (Maio et al. 2015). The planning approach has been validated using a set of example mission specifications.

This paper currently only mentions the use of metrics to analyse and optimize multi-robot plans, but our main motivation of the planning approach aims at using the information about the organizational, i.e. multi-robot system's, state to optimize resource usage and distribution. In general, our future work will be focused on completion of the full planning approach including the integration of the multi-objective optimization and an application of the planning system to the real reconfigurable multi-robot system (cf. Figure 1).

Since planning can only operate on an abstraction of the real world, any resulting plan will leave potential for optimization. However, to augment the multi-robot planning approach we suggest to strengthen the ability for local collaboration, e.g., in terms of annotating a solution plan with potential for local optimization. While the multi-robot plan identifies two or more agents that join at some location and point in time, this information should be used by this subset of agents to join at an even earlier stage, e.g., by sharing knowledge about target positions, leading to an embedded, online and local optimization.

6 Acknowledgments

This work was supported by the German Space Agency (DLR) under grant agreement 50RA1301 and by the Federal Ministry of Education and Research under grant agreement 01IW15001.

References

Ahuja, R. K.; Magnanti, T. L.; and Orlin, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Michigan, US: Prentice Hall.

Baca, J.; Hossain, S.; Dasgupta, P.; Nelson, C. a.; and Dutta, A. 2014. ModRED: Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration. *Robotics and Autonomous Systems* 62(7):1002–1015.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *Proceedings of International Conference on AI Planning and Scheduling (ICAPS)*, 333–341.

Chien, S. A.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *Proceedings of the 12th International Conference on Space Operations*, 1–17. Reston, Virigina: American Institute of Aeronautics and Astronautics.

Dechter, R. 2003. Temporal Constraint Networks. In Dechter, R., ed., *Constraint Processing*, The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann. chapter 12, 333–362.

DeLoach, S. A., and Kolesnikov, V. A. 2006. Using Design Metrics for Predicting System Flexibility. In Baresi, L., and Heckel, R., eds., *Fundamental Approaches to Software Engineering*, volume 3922. Springer Berlin Heidelberg. 184–198.

Dignum, V., ed. 2009. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global.

Eich, M.; Hartanto, R.; Kasperski, S.; Natarajan, S.; and Wollenberg, J. 2014. Towards coordinated multirobot missions for lunar sample collection in an unknown environment. *Journal of Field Robotics* 31(1):35–74.

Evans, J. S. 1991. Strategic flexibility for high technology manoeuvers: A conceptual framework. *Journal of Management Studies* 28:69–89.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2012. Using the context-enhanced additive heuristic for temporal and numeric planning. *Springer Tracts in Advanced Robotics* 76(STAR):49–64.

Gantner, Z.; Westphal, M.; and Wölfl, S. 2008. GQR-A fast reasoner for binary qualitative constraint calculi. In *Proceedings of the AAAI'08 Workshop on Spatial and Temporal Reasoning*, volume 8, 24–29.

Horridge, M., and Bechhofer, S. 2011. The OWL API: A Java API for OWL ontologies. *Semantic Web* 2(1):11–21.

Hübner, J. F.; Sichman, J. S. a.; and Boissier, O. 2002. MOISE+. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems - Part 1*, AAMAS '02, 501. New York, USA: ACM Press.

Kennington, J. L. 1978. A Survey of Linear Cost Multicommodity Network Flows. *Operations Research* 26(2):209–236.

Maio, A. D.; Fratini, S.; Policella, N.; Donati, A.; and Agency, E. S. 2015. Resource driven planning with plasma: the plan space multi-solver application. In *Proceedings of the 13th Symposium on Advanced Space Technologies in Robotics and Automation*.

Meiri, I. 1996. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence* 87(1-2):343–385.

Murphy, R.; Ausmus, M.; and Bugajska, M. 1999. Marsupial-like Mobile Robot Societies. *Proceedings of the Third Annual Conference on Autonomous Agents* 1–14.

NASA. 2016. Mission Timeline: Surface Operations, retrieved May 3, 2016, from `http://mars.nasa.gov/mer/mission/tl_surface_nav.html`.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Rahwan, T.; Ramchurn, S. D.; Jennings, N. R.; and Giovannucci, A. 2009. An Anytime Algorithm for Optimal Coalition Structure Generation. *Journal of Artificial Intelligence Research* 34:521–567.

Rahwan, T.; Michalak, T. P.; Elkind, E.; Faliszewski, P.; Sroka, J.; Wooldridge, M.; and Jennings, N. R. 2011. Constrained Coalition Formation. In *Proceedings of Twenty-First AAAI Conference on Artificial Intelligence*, 719–725.

Roehr, T. M.; Cordes, F.; and Kirchner, F. 2014. Reconfigurable integrated multirobot exploration system (rimres): Heterogeneous modular reconfigurable robots for space exploration. *Journal of Field Robotics* 31(1):3–34.

Schulte, C., and Tack, G. 2012. View-based propagator derivation. *Constraints* 18(1):75–107.

Sonsalla, R.; Cordes, F.; Christensen, L.; Planthaber, S.; Albiez, J.; Scholz, I.; and Kirchner, F. 2014. Towards a Heterogeneous Modular Robotic Team in a Logistic Chain for Extraterrestrial Exploration. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Hierarchical Hybrid Planning in a Mobile Service Robot. In Hölldobler, S.; Krötzsch, M.; Penaloza, R.; and Rudolph, S., eds., *KI 2015 – Advances in Artificial Intelligence*. Springer International Publishing. 309–315.

Tenorth, M., and Beetz, M. 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research* 32(5):566–590.

Tsarkov, D., and Horrocks, I. 2006. FaCT++ Description Logic Reasoner: System Description. In Furbach, U., and Shankar, N., eds., *Automated reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg. 292–297.

Weiss, G., ed. 2009. *Multiagent Systems*. MIT Press, 2nd edition.

Wurm, K. M.; Dornhege, C.; Nebel, B.; Burgard, W.; and Stachniss, C. 2013. Coordinating heterogeneous teams of robots using temporal symbolic planning. *Autonomous Robots* 34(4):277–294.

Zhong, C., and DeLoach, S. 2011. Runtime Models for Automatic Reorganization of Multi-Robot systems. In *Proceedings of the 6th Symposium of Software Engineering for Adaptive and Self-Managing Systems*, 20 – 29. Honolulu, HI, USA: ACM Press.

Path Planning for Unmanned Vehicles Operating in Time-Varying Flow Fields

Brual C. Shah and Petr Švec
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742, USA
{brual,petrsvec}@umd.edu

Atul Thakur
Department of Mechanical Engineering
Indian Institute of Technology Patna
Bihta, Bihar, 801103, India
athakur@iitp.ac.in

Satyandra K. Gupta
Department of Aerospace and Mechanical Engineering
University of Southern California
Los Angeles, CA 90089, USA
skgupta@usc.edu

Abstract

Significant fluid medium flows such as strong currents may influence the maximum velocity and energy consumption of an unmanned vehicle. Weather forecast reports provide an estimate of the medium flow and can be utilized to generate low-cost paths that exploit the flow to aid the motion of the vehicle and conserve energy. Conserving energy has also an indirect benefit of extending the range of operation. This paper presents a discrete search based path planning approach that can utilize an arbitrary cost function for computing energy efficient, collision-free paths in complex scenarios with dynamic obstacles. Traditional admissible heuristics that are based on shortest distance or time are not suitable for this problem as exploiting the medium flow to propel the vehicle forward often requires a longer time or distance to reach the goal. We have developed new admissible heuristics for estimating the cost-to-go by taking into account flow characteristics. The proposed method also selects the start time for commencing the mission by waiting for the favorable medium flow conditions.

1 Introduction

Many unmanned vehicles (also called robotic vehicles) interact with the underlying fluid medium in which they operate. The medium may exhibit a significant fluid flow. For example, an aerial vehicle may encounter significant winds and an underwater vehicle may encounter strong water currents. The performance of the vehicle such as its maximum velocity and also the energy consumption per unit distance traveled is affected by the medium flow.

Usually, a global path planner is used for finding the shortest collision-free path to a specified goal location. The waypoints generated by the global planner are often followed using a feedback controller in order to compensate for environmental disturbances. The rejection of the disturbances via feedback controlled actuators may consume significant energy if the vehicle travels against a strong medium flow. From the operational efficiency point of view, the vehicle should exploit the fluid flow instead of attempting to overcome it.

Weather forecast reports provide an estimate of the medium flow as a function of time. This information can be exploited to generate low-cost paths that utilize the flow to aid the motion of the vehicle. Such paths can save energy and hence be much lower in cost. Conserving energy has also an indirect benefit of extending the range of operation. This is especially important in missions where long term operation is desired. In many cases, the vehicle has a window of opportunity for completing a given mission and thus the mission manager can select the mission start time such that the vehicle experiences the most favorable medium flow during the operation.

Let us consider the following scenarios to understand the implications of the fluid flow on the path:

- *Scenario 1:* The vehicle takes a straight line path to the goal and does not account for the influence of the medium flow. It encounters a strong medium flow during the travel that impedes its motion. The vehicle then needs to use a significant energy to traverse the path against the flow, which increases the cost of the path.
- *Scenario 2:* The vehicle waits for the flow to become more favorable. Once the flow is in a favorable direction, the vehicle utilizes it to advance itself and thus uses less energy as it does not need to generate thrust to propel forward. The vehicle only dissipates energy when performing minor corrective actions to mitigate spatial disturbances. The path of the vehicle exploiting the medium flow may be curved and longer than a straight line path. The flow velocity is small compared to the velocity of the vehicle that uses its thrusters to propel forward, so the vehicle takes much longer time to reach the goal compared to the first scenario. Despite the longer path length and travel time, the vehicle consumes significantly less energy as compared to the first scenario. Therefore, the cost of travel is much lower. The decrease in the energy consumption means that the vehicle can do several more missions without the need for refueling.

An unmanned surface vehicle on a long voyage (see Fig. 1) will need to employ the following three types of planners:

- A global path planner to compute a sequence of waypoints from the start to the goal location which form a collision-free path (Koenig and Likhachev 2002; Likhachev et al. 2005; Lavalle and Kuffner Jr 2000; Shah and Gupta 2016). Often, only static obstacles are handled by this planner.

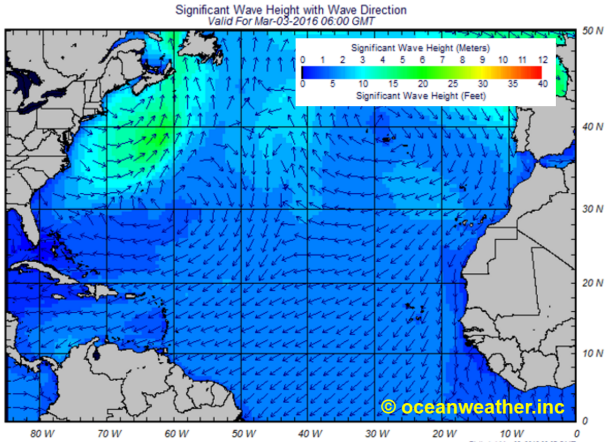


Figure 1: Surface currents in the Atlantic ocean.

- A trajectory planner to compute a risk-aware, dynamically feasible trajectory (Shah et al. 2014; Švec et al. 2011) via the waypoints generated by the global path planner.
- A reactive planner for locally avoiding highly dynamic obstacles (Fiorini and Shiller 1998; Fox, Burgard, and Thrun 1997; Martinez-Gomez and Fraichard 2009).

This paper deals with the global path planning under the influence of medium flows. The main contribution of this paper is the incorporation of the influence of the flow field into the search for an optimal path.

Path planning for vehicles operating in the presence of flow fields has been previously studied in (Reif and Sun 2004; Ceccarelli et al. 2007). Computing energy-efficient paths for a vehicle operating in a large environment with flow fields present several challenges such as:

- Utilizing a model of a time-varying flow field during path planning to predict the impact of the flow on the motion of the vehicle and hence compute paths that are energy-efficient as well as compliant with the vehicle's dynamics.
- Integrating the uncertainty in the prediction model of the flow field and the vehicles' spatio-temporal uncertainty arising due to its interaction with obstacles.
- Navigating around complex obstacles in the environment during the computation of an optimal collision-free path.

A majority of previous approaches attempted to address the aforementioned challenges separately. For example, graph search-based algorithms are efficient at computing paths in complex obstacle fields. Model predictive control-based techniques (Smith and Huynh 2014) are good at computing paths in the presence of flow fields. Stochastic mathematical programming-based techniques (Russell and Norvig 1995; Likhachev, Thrun, and Gordon 2004; Sanner et al. 2009) are good at computing paths in the presence of uncertainty.

We believe that an integrated approach that can address all the aforementioned challenges consists of two steps. First, a discrete graph search technique is needed to compute a global path that not only avoids obstacles but also exploits the medium flow. Second, a stochastic mathematical programming-based techniques (Fathpour et al. 2014) or model predictive control-based techniques (Huynh, Dunbabin, and Smith 2015) are needed to compute trajectories between intermediate goals lying on the computed global path.

In this paper, we present a heuristic-based search technique for computing an energy-efficient global path for a vehicle moving in a flow field. The technique is relatively easy to implement, can incorporate intention models (if available) of dynamic obstacles, and enforce safety constraints (Švec et al. 2013; Shah et al. 2014; Švec et al. 2014). The proposed algorithm can also determine the cost optimal start time of a given mission based on the model of the fluid flow. The computed path can be further locally altered by the mathematical programming or MPC-based techniques to account for spatial uncertainties.

Traditional distance and time-based admissible heuristics, used for estimating cost-to-go by discrete graph search algorithms, are not suitable for this domain because curved paths are needed to exploit available flows. Moreover, using the medium flow to propel the vehicle forward often requires a longer time to reach the goal. Hence, we have developed new admissible heuristics for estimating cost-to-go while taking into account the medium fluid flow.

2 Related Work

Several path planning approaches to realize energy-efficient, autonomous operations of robotic systems in non-linear, time-varying fields were developed in the past. In particular, a purely local path optimization technique was developed by Kruger et al. (Kruger et al. 2007) to allow autonomous guidance of an autonomous underwater vehicle (AUV) in a fast flowing tidal river. The technique employs a gradient based approach to locally modify an initial straight path between two given locations according to a predefined cost function. Similarly, the technique developed in (Witt and Dunbabin 2009) is used for searching paths in a time-extended state space that balances the path execution time and energy requirements. The approach searches over a predefined set of global static paths represented as splines and is combined with a local random, simulated annealing-inspired search. This choice of search techniques, however, does not allow a systematic exploration of complex search spaces.

Thompson et al. (Thompson et al. 2010) developed a wavefront based path planning algorithm for an UAV to follow paths that ensure fastest arrival of the vehicle to given locations through uncertain, time-varying current fields. The algorithm, however, does not explicitly balance the energy expenditure of the vehicle with the path execution time, prune the search space, or account for the uncertain dynamics of the field (contrary to the claims in the paper). Similarly, the approach developed by Lolla et al. (Lolla et al. 2012) is also based on the forward evolution of a wavefront from the initial to the goal vehicle states, determining a series of states along the evolving wavefronts that optimize a given objective (in this case, travel time). In contrast to this approach, the technique presented in this paper

prioritizes states during the expansion, which increases its computational performance. In addition, the use of the free-flow action allows a variable resolution search. Soulignac (Soulignac 2011) developed a sliding wavefront expansion algorithm that computes physically controllable, globally optimal and feasible paths for a vehicle operating in an environment with strong currents (i.e., currents that may overcome the physical capabilities of the vehicle). Although theoretically sound, the algorithm is based on the classical wavefront expansion and as such does not consider the evolution of currents in time.

Garau et al. (Garau, Alvarez, and Oliver 2005) evaluated several heuristic functions as candidate components of the A* algorithm. The developed approach, however, is suitable only for static fields. Similarly, Isern-Gonzalez et al. (Isern-González et al. 2012) developed a method based on the A* and Nearest Diagram (ND) algorithms. The method finds an initial path that is further locally optimized. The A* algorithm was also combined with the fast marching algorithm into the FM* algorithm (Petres et al. 2007) that has the capability of computing smooth paths in continuous environments. However, the work does not explicitly address the energy-efficiency as well as time-varying fields.

Al-Sabban et al. (Al-Sabban et al. 2013) developed an energy-efficient path planning algorithm for an unmanned aerial vehicle (UAV) operating in an uncertain wind field. The problem was defined as a Markov Decision Process (MDP) to consider the local, stochastic nature of the field vectors. The algorithm, however, does not explicitly account for a time-varying field. The work was further adapted for path planning of AUVs (Al-Sabban, Gonzalez, and Smith 2012).

Sampling-based methods were also used for energy-efficient, probabilistic-complete path planning. For example, a path planner based on the Rapidly Exploring Random Trees (RRT) (Rao and Williams 2009) is used for computing paths to realize a long-term, autonomous operation of underwater gliders.

Long-term path planning in time-varying fields with obstacles is computationally as well as spatially expensive due to the large size and complexity of the search space. Most recently, the approach developed by Fathpour et al. (Fathpour et al. 2014; Kuwata et al. 2009) computes paths or navigation functions for autonomous guidance and reachability analysis of a hot-air balloon in time-invariant, time-varying, and stochastic wind fields. The approach allows spatially and computationally efficient planning through decomposition of the planning problem into subproblems, and solving each of them sequentially.

In this paper, we use a heuristics-based search technique to enable users to easily incorporate their mission-specific considerations into the path generation process. Our main contributions are novel heuristic functions for prioritizing processing of search nodes and thus decreasing computation time for finding optimal paths in environments with complex, time-varying medium flows and obstacles.

3 Problem Formulation

3.1 Terminology

The continuous state space $\mathcal{X} = \mathcal{X}_\eta \times \mathcal{X}_\nu \times \mathcal{T}$ consists of states $\mathbf{x} = [\eta^T, \nu^T, t]^T \in \mathcal{X}$, where $\eta = [x, y, \psi]^T \in \mathcal{X}_\eta \subset \mathbb{R}^2 \times \mathbb{S}^1$ is the vehicle's pose, $\nu = [u, v, r]^T \in \mathcal{X}_\nu \subset \mathbb{R}^3$ is the vehicle's velocity consisting of the surge speed u, sway speed v, and angular speed r about the z axis, and t is the time. The approximated lower dimensional, discrete 4D version of the continuous state space $\mathcal{X}$ is represented by $\mathcal{S}$, where each state $\mathbf{s} = [x, y, u, t]^T$ contains position, surge speed, and time variables.

The continuous, state-dependent motion primitive space of the vehicle is defined as $\mathcal{U} = \{\mathcal{U}_a, \mathbf{u_f}\} \subset \mathbb{R}^2 \times \mathbb{S}^1$. Here, $\mathcal{U}_a$ is the set of the vehicle's thrust-producing actions in which each action $\mathbf{u_a} = [u_d, \psi_d, \delta t]^T$ consists of the desired surge speed u_d, the desired heading ψ_d, and the execution time δt. The velocity vector of the vehicle at state $\mathbf{x}$ is given by $\mathbf{v^r_x} = [u_d, \psi_d]^T$. A special free-flow action is defined as $\mathbf{u}_f = [u_m, \psi_m, \delta t]$ allows the vehicle to travel freely with the flowing medium along the current flow vector $\mathbf{v^m_x}$ for the time interval δt (see Section 3.2). The discrete set of vehicle's thrust-producing actions is given by $\mathcal{U}_{a,d}$. Each discrete thrust-producing action of the vehicle is given by $\mathbf{u}_{a,d}$.

3.2 Medium Flow Model

In a real world scenario, it is difficult to have a continuous forecast of natural phenomena (e.g., wind, ocean currents, etc.). The available forecast is usually discrete and it is assumed to hold for a specific interval of time. On similar lines, we have modeled the medium flow in the environment to be discrete and is assumed to vary temporally but not spatially. In other words, the flowing medium for any discrete time t is constant for the time interval δt (see Figure 2). The simulation time interval of motion primitives $\mathcal{U}_d$ is kept to be the same as the discrete time interval δt of the medium flow model.

The time-varying model m_f of the flowing medium outputs a velocity vector $\mathbf{v^m_s} = [u_m, \psi_m]^T$ at every state $\mathbf{s} \in \mathcal{S}$. Here, u_m is the magnitude and ψ_m is the direction of the flowing medium.

3.3 Motion Model

The motion of the vehicle in an environment with a flowing medium is dependent upon the direction and the magnitude of the flow. The transition of the vehicle from the current state $\mathbf{s}$ to the next state $\mathbf{s}'$ is determined by the vehicle's thrust-producing action $\mathbf{u}_{a,d} \in \mathcal{U}_{a,d}$ and the velocity vector $\mathbf{v^m_s}$ of the flowing medium at the current state $\mathbf{s}$. We assume that the low level controller of the vehicle is capable of maintaining its heading along the direction ψ_d of the thrust-producing action $\mathbf{u}_{a,d}$.

Depending upon the medium flow, the forward velocity of the vehicle may be boosted or hindered. The magnitude of the forward velocity $|\mathbf{v}^f_\mathbf{s}|$ is determined by the vector sum of $\mathbf{v}^m_\mathbf{s}$ and $\mathbf{v}^r_\mathbf{s}$, with an assumption of the vehicle being a point mass (see Figure 3). The velocity of the vehicle $\mathbf{v}^r_\mathbf{s}$ can be resolved into two components: the magnitude of the

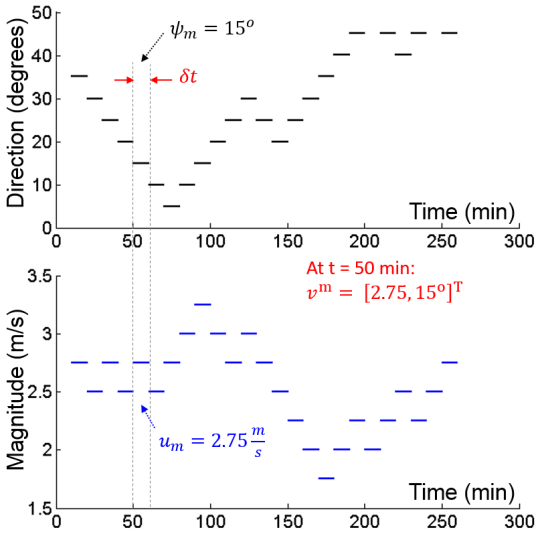


Figure 2: Model of the flowing medium.

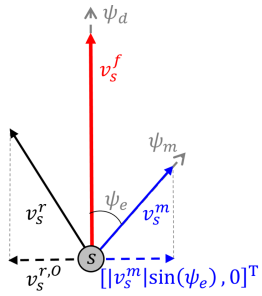


Figure 3: Computation of vehicle's forward velocity under medium flow.

component in the direction orthogonal to the desired direction $|\mathbf{v_s}^{r,O}| = |\mathbf{v_s}^m| sin(\psi_e)$ and the magnitude of the component along the desired direction $|\mathbf{v_s}^{r,D}| = \sqrt{|\mathbf{v_s}^r|^2 - |\mathbf{v_s}^{r,O}|^2}$, where $\psi_e = \psi_d - \psi_m$, and ψ_m is the orientation of the flowing medium. Thus, the resultant forward velocity of the agent along the direction of the thrust-producing action $\mathbf{u}_{a,d}$ is given by $|\mathbf{v_s}^f| = |\mathbf{v_s}^{r,D}| + |\mathbf{v_s}^m| cos(\psi_e)$. The position of the next state $\mathbf{s}'$ generated by the thrust-producing action $\mathbf{u}_{a,d}$ is determined by $[x', y']^T = [x, y]^T + |\mathbf{v_s}^f| \cdot \delta t \cdot [cos(\psi_d), sin(\psi_d)]^T$. Similarly, the position of the next state $\mathbf{s}'$ generated by the free-flow $\mathbf{u}_{f,d}$ action is determined by $[x', y']^T = [x, y]^T + |\mathbf{v_s}^m| \cdot \delta t \cdot [cos(\psi_m), sin(\psi_m)]^T$.

3.4 Cost Model

Let the cost of executing a vehicle's thrust-producing action per unit time be given by C_a^t and the cost of the special free-flow action per unit time be given by C_m^t, where $C_m^t < C_a^t$. The values of C_m^t and C_a^t are constant and provided by the user. Let $c(\mathbf{s}, \mathbf{s}')$ denote the traversal cost from the state $\mathbf{s}$ to state $\mathbf{s}'$. The traversal cost $c(\mathbf{s}, \mathbf{s}')$ equals $C_a^t \delta t$ if the vehicle arrives at state $\mathbf{s}'$ by using the thrust-producing action $\mathbf{u}_{a,d} \in \mathcal{U}_{a,d}$. Similarly, the traversal cost $c(\mathbf{s}, \mathbf{s}')$ equals $C_m^t \delta t$ if the vehicle arrives at state $\mathbf{s}'$ by using the free-flowing action $\mathbf{u}_{f,d}$. Finally, let the optimal cost of the computed path τ from the initial state $\mathbf{s_I}$ to the goal state $\mathbf{s_G}$ at start time t_{start} be denoted by $c^*_{start}(\mathbf{s_I}, \mathbf{s_G})$ (see Section 4.3). Here t_{start} is the time when the vehicle starts the mission, i.e., leaves from the initial state $\mathbf{s_I}$ and proceed towards the goal state $\mathbf{s_G}$.

3.5 Problem Statement

We are interested in designing an energy-efficient path planning algorithm for computation of collision-free paths between the initial and the goal states of a vehicle operating in an environment with a flowing medium. The developed planner searches for a path that minimizes the energy cost by exploiting the medium flow.

Given,

- the discrete state space $\mathcal{S}$ of the vehicle,
- the initial $\mathbf{s_I}$ and the goal $\mathbf{s_G}$ states of the vehicle,
- the discrete model of the medium flow m_f,
- the cost C_a^t of the vehicle's thrust-producing action per unit time and the cost C_m^t of the free-flowing action per unit time,
- the map of the environment with the geometric regions occupied by static obstacles $\mathcal{O}_s = \bigcup_{k=1}^{K} o_{s,k} \subset \mathbb{R}^2$, and
- the maximum time duration $t_{mission}$ in which the vehicle should complete the current mission and reach the goal $\mathbf{s}_G$.

Compute:

- The start time of the mission $t_{start} < t_{mission}$ that minimizes the cost incurred by the vehicle to travel from $\mathbf{s_I}$ to $\mathbf{s_G}$.
- A collision-free, dynamically feasible trajectory $\tau : [t_{start}, t_{finish}] \rightarrow \mathcal{S}$ such that $\tau(t_{start}) = \mathbf{s_I}$, $\tau(t_{finish}) = \mathbf{s_G}$ and its travel cost is minimized. The value of t_{finish} should not exceed $t_{mission}$ and $\mathbf{s}(t) \notin \mathcal{O}_s$ Each state $\mathbf{s}(t)$ along τ belongs to the free state space.

In this paper we assume that the actuators of the vehicle are able to overcome the medium flow and the velocity of the agent $\mathbf{v^r_s}$ is greater than the medium velocity $\mathbf{v^m_s}$.

4 Approach

4.1 Overview

The deliberative path planner described in Section 4.2 searches in a discrete 4D state space for a collision free, lattice-based path $\tau : [0, t_{finish}] \rightarrow \mathcal{S}$ from a given initial state $\mathbf{s_I}$ to a goal state $\mathbf{s_G}$. The path is optimized not only with respect to its travel cost but also with respect to the vehicle's start time t_{start} (see Section 4.3).

The medium flow forecast may have uncertainty associated with it. The A* algorithm does not handle this uncertainty. This uncertainty can be handled by refining the path by using forward value iteration-based stochastic dynamic programming in the vicinity of the computed path (LaValle 2006). This post-processing can refine the paths to reduce

the probability of collision with obstacles due to the uncertainty in the medium flow by optimizing the expected costs of paths. This step is outside the scope of this paper and will not be discussed further.

4.2 Path Planning

The global path planner is designed based on the lattice-based A* heuristic search (Pivtoraiko, Knepper, and Kelly 2009). The search for a path τ with the minimum cost is performed by expanding states in the least-cost fashion according to the cost function $f(\mathbf{s}) = g(\mathbf{s}) + h(\mathbf{s})$, where $g(\mathbf{s})$ is the cost-to-come at state $\mathbf{s}$ from the initial state $\mathbf{s_I}$, and $h(\mathbf{s})$ is the cost-to-go from the state $\mathbf{s}$ to the goal state $\mathbf{s_G}$. The cost-to-come $g(\mathbf{s})$ is computed by summing a traversal cost of each action (see Section 3.4) executed to reach the current state $\mathbf{s}$ from the initial state $\mathbf{s_I}$. In Section 5, we compute three different types of the cost-to-go (h-cost) to be used in the cost function described above. During the search, the neighboring state $\mathbf{s}$ is determined by the motion model described in Section 3.3. We have defined a desired goal state region S_G in close proximity around the goal state $\mathbf{s_G}$. The search is terminated when any of the expanded states $\mathbf{s}$ lies in S_G.

4.3 Start Time Optimization

The optimal cost $c^*_{start}(\mathbf{s_I}, \mathbf{s_G})$ of the path $\tau : [t_{start}, t_{finish}] \rightarrow \mathcal{S}$ computed by the path planner is highly dependent on the medium flow encountered by the vehicle and the start time t_{start}. In some situations, it is beneficial for the vehicle to wait at the initial location $\mathbf{s_I}$ and start its journey only when the medium flow becomes favorable. The maximum time $t_{mission}$ the vehicle is allowed to wait and complete its mission is predefined by the user.

We find the resolution-optimal start time t_{start} of the path between $\mathbf{s_I}$ to $\mathbf{s_G}$ by adaptively sampling the time interval $\{0, t_{mission}\}$. We iteratively call the deliberative path planner to compute the cost $c^*_{start}(\mathbf{s_I}, \mathbf{s_G})$ of a path for different values of t_{start}. The bounding constraints for t_{start} are given by $0 < t_{finish} \leq t_{mission}$, where $t_{finish} = t_{start} + t_{execution}$ and $t_{execution}$ is time taken by the vehicle to execute the path. The sampling method begins with large interval steps and adaptively reduces the time step in the promising regions.

5 Design of Heuristics

5.1 Heuristic #1

Let $\mathbf{s}$ be the current state. We are interested in estimating the cost to reach the goal state $\mathbf{s}_G$ from the current state $\mathbf{s}$. Let $c^*(\mathbf{s}, \mathbf{s}_G)$ be the optimal cost of reaching $\mathbf{s}_G$ from $\mathbf{s}$. We will refer to this cost as optimal cost-to-go. Let heuristic $h(\mathbf{s})$ be a function that provides an estimate of $c^*(\mathbf{s}, \mathbf{s}_G)$. $h(\mathbf{s})$ will be called *admissible heuristic* if, $c^*(\mathbf{s}, \mathbf{s}_G) \geq h(\mathbf{s})$.

A simple way to compute $h(\mathbf{s})$ would be to assume that the flow will be most favorable during the the vehicle operation. This will enable us to achieve the smallest possible cost per unit distance traveled. Hence estimate of cost-to-go $h(\mathbf{s})$ cannot exceed the actual optimal cost to goal state $\mathbf{s}_G$.

Let $dist(\mathbf{s}, \mathbf{s_G})$ be the Euclidean distance between states $\mathbf{s}$ and $\mathbf{s_G}$. If the flow is assumed to be at maximum velocity $\mathbf{v^m}_{max}$ and directly flowing towards the goal, then the total cost of travel using free flow is given by Equation 1.

$$C^1 = \frac{dist(\mathbf{s}, \mathbf{s_G}) \cdot C^t_m}{|\mathbf{v^m}|_{max}} \tag{1}$$

If the vehicle uses its actuators and travels at maximum vehicle velocity $\mathbf{v^r}_{max}$ in addition to taking the advantage of the flow, then the total cost of travel is given by Equation 2.

$$C^2 = \frac{dist(\mathbf{s}, \mathbf{s_G}) \cdot C^t_a}{|\mathbf{v^m}|_{max} + |\mathbf{v^r}|_{max}} \tag{2}$$

Minimum of the two estimates C^1 and C^2 can be determined to calculate $h(\mathbf{s})$.

$$h(\mathbf{s}) = min(C^1, C^2) \tag{3}$$

5.2 Heuristic #2

Although admissible, the heuristic presented in Section 5.1 significantly underestimates the cost, so we have designed a better heuristic that utilizes the medium flow information. In order to utilize the flow information, we need to first determine the relevant time window. We do this by first estimating the upper bound t_{bound} on the time associated with the optimal path. Any medium flow available at time greater than t_{bound} will not be available during the execution of the path.

We compute the cost $C_{straight}$ incurred by the vehicle to travel in a straight path to the goal state $\mathbf{s_G}$ using its thrust-producing action in the presence of medium flow. $C_{straight}$ is the upper bound on the optimal cost. Let C' be the cost of any arbitrary path to the goal state $\mathbf{s_G}$, and can be represented by $C' = t_f \cdot C^t_m + t_a \cdot C^t_a$, where t_f is the total time consumed by the free flow action and t_a is the total time consumed by the vehicle's thrust-producing action. We are only interested in paths $C_{straight} \geq t_f \cdot C^t_m + t_a \cdot C^t_a$. We can compute t_{bound} by maximizing the objective function $t_f + t_a$, where $t_f = (C_{straight} - t_a \cdot C^t_a)/C^t_m$. We assume that $C^t_m < C^t_a$. Hence, we can maximize total time by selecting $t_a = 0$ and $t_f = C_{straight}/C^t_m$. Thus, the upper bound on time is given by $t_{bound} = C_{straight}/C^t_m$. The value of $C_{straight}$ will change with the scenarios having different medium flows.

As mentioned in Section 3.2, the estimated flow conditions are available as an ordered sequence. Each flow condition holds for a time interval of δt. In each time interval δt, we have the direction and the magnitude of the flowing medium in terms of velocity vector $\mathbf{v^m_s}$.

We can view each flow condition as a performance altering condition that lasts for a duration of δt. We are interested in exploiting these conditions that lower the cost of travel. For each flow condition that we plan to utilize, we need to make a decision to either execute the free-flow action or use the thrust-producing action. In computation of the heuristic cost $h(\mathbf{s})$, we select the action that has the lower cost incurred per unit distance advancement towards the goal.

The cost incurred per unit projected distance traveled using the free-flowing action $\mathbf{u}_{f,d}$ is calculated using Equa-

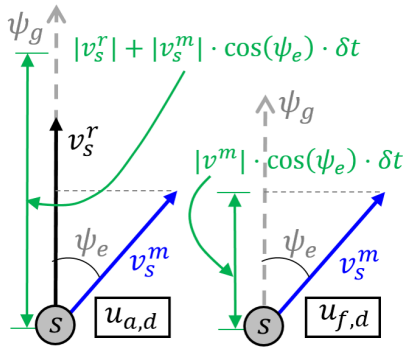


Figure 4: Calculation of heuristic #2.

tion 4.

$$C_f^l = \frac{C_m^t}{|\mathbf{v^m_s} \cdot |cos(\psi_e)} \tag{4}$$

The cost incurred per unit projected distance traveled using the vehicle's thrust-producing action $\mathbf{u}_{a,d} \in \mathcal{U}_{a,d}$ is calculated using Equation 5.

$$C_a^l = \frac{C_a^t}{|\mathbf{v^r_s}| + |\mathbf{v^m_s}| \cdot cos(\psi_e)} \tag{5}$$

In Equations 4 and 5, the angle ψ_e is the angle between the desired direction ψ_g to the goal state $\mathbf{s_G}$ from the current state $\mathbf{s}$ and the direction of the flowing medium ψ_m (see Figure 4). We can choose the appropriate action for each discrete time interval δt starting from the current time t to t_{bound}. The selected actions for each discrete time interval $\delta t \in \{t, t_{bound}\}$ are sorted and stored into the priority queue E_O according to the cost incurred per unit distance advancement towards the goal with the least cost action on the top. The actions are sequentially popped out of the priority queue E_O and are integrated for time interval δt, until the summation of the projected distance traveled by all actions is equal to the projected distance to the goal $\mathbf{s_G}$ from the state $\mathbf{s}$.

This heuristic uses actions that have the lowest per unit length cost towards the goal from the available time window. It selects actions without requiring them to be contiguous in time. The optimal path will have either the same action as used by the heuristic or will be forced to use actions that have higher per unit length cost towards the goal. Therefore, it is not possible for the optimal path to exceed the cost estimated by this heuristic. Therefore, this heuristic is admissible.

5.3 Heuristic #3

In Heuristic #2 described above, each selected action is assigned a cost-to-go based on the cost incurred per unit projected distance traveled towards the goal. This is a tight lower bound on cost for thrust producing actions. However, when the free flow action is used, unless $\psi_e = 0$, the vehicle does not go directly towards the goal (see Figure 5). If free flow conditions do not exist within the time bound that can provide ψ_e of opposite sign, a thrust-producing action is needed to bring the vehicle towards the goal. If using the

Algorithm 1 COMPUTEHEURISTIC2$(\mathbf{s}, t, \psi_g, t_{bound}, m_f)$

Input: The current node $\mathbf{s}$, current time of arrival t, the desired direction ψ_g from the current state $\mathbf{s}$ to the goal state $\mathbf{s_G}$, the maximum bound on travel time t_{bound}, and the model of medium flow m_f.

Output: An estimated cost-to-go $h(\mathbf{s})$ from current state $\mathbf{s}$ to goal state $\mathbf{s}_G$.

Let $t_i = t$ be the forward simulation time and δt be the simulation time step.
Let $\mathbf{v}_i$ be the velocity vector along the desired direction achieved by executing action $\mathbf{u}_i$ at time $t_i \in \{t, t_{bound}\}$.
Let E_O be a priority queue containing selected actions $\mathbf{u}_i$ at each discrete time $t_i \in \{t, t_{bound}\}$
while $t_i \leq t_{bound}$ **do**
    Let the current velocity vector of the medium flow at state $\mathbf{s}_i$ be denoted by $\mathbf{v^m}_{\mathbf{s}_i}$
    Cost incurred by per unit length advanced towards the goal while executing free-flow action $\mathbf{u}_{f,d}$ and thrust-producing action $\mathbf{u}_{a,d}$ are given by Equation 4 and 5 and denoted as C_f^l and C_a^l respectively.
    if $C_f^l < C_a^l$ **then**
        $\mathbf{v}_i = |\mathbf{v^m}_{\mathbf{s}_i}|cos(\psi_e)$ and $C_i^l = C_f^l$
    else
        $\mathbf{v}_i = |\mathbf{v^r}_{\mathbf{s}_i}| + |\mathbf{v^m}_{\mathbf{s}_i}|cos(\psi_e)$ and $C_i^l = C_a^l$
    end if
    Insert vector $[\mathbf{v}_i, C_i^l]^T$ into/in E_O
    $t_i = t_i + \delta t$
end while
Let $d_G = dist(\mathbf{s}, \mathbf{s_G})$ be the distance of the state $\mathbf{s}$ from the goal state $\mathbf{s_G}$.
$d_{travel} = 0$ and $C_{incur} = 0$
while E_O not empty **do**
    $[\mathbf{v}_i, C_i^l] \leftarrow E_O.First()$
    $d_{travel} = d_{travel} + |\mathbf{v}_i| \cdot \delta t$
    $C_{incur} = C_{incur} + C_i^l \cdot d_{travel}$
    if $d_{travel} \geq d_G \cdot cos(\psi_g)$ **then**
        $h(\mathbf{s}) \leftarrow C_{incur}$
        return $h(\mathbf{s})$
    end if
end while
return $h(\mathbf{s}) = \infty$ (not enough time to reach the goal state, thus the node $\mathbf{s}'$ does not lie on optimal path τ^*.

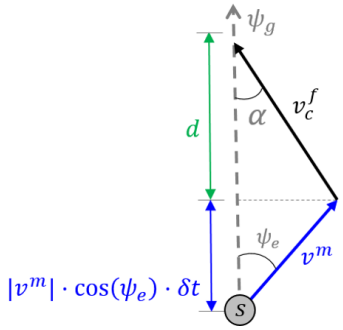


Figure 5: Calculation of additional compensation cost-to-go for free-flowing action $\mathbf{u}_{f,d}(\mathbf{s})$.

thrust-producing action becomes necessary in conjunction with the free flow action, then the lower bound computed on the cost in heuristic #2 significantly underestimates the cost and can lead to expansion of a large number of states. We have devised an improvement over heuristic #2, by increasing the per unit length cost associated with free flow actions to account for use of the thrust producing actions. Let us consider a free flow action shown in Figure 5. Without the loss of generality, let us assume that ψ_e is positive and no free flow action is available with negative value of ψ_e until time t_{bound}. We will, therefore, have to use a thrust-producing action to bring the vehicle towards the goal.

In the Figure 5, the free flowing action $\mathbf{u}_{f,d}(s')$ along the flowing medium with velocity $\mathbf{v^m}$ makes an angle ψ_e with the desired direction of motion . Now, the corrective distance the vehicle has to travel to get back to the desired path is given by $d_c = \sqrt{d^2 + (|\mathbf{v^m}| \cdot sin(\psi_e) \cdot \delta t)^2}$, where d is the projected distance traveled along the desired direction towards the goal.

To compute the lower bound on the cost, we want to use the fastest possible velocity for the vehicle. Let us assume that there will be flow available that will provide the maximum possible assistance to the vehicle. We will only apply this correction if there is no flow available with negative ψ_e. The best that we can hope for is that the flow is going along ψ_g as shown in Figure 5. Let us assume that the magnitude of the flow velocity is the maximum possible $|\mathbf{v^m}|_{max}$ within the available time window. Under these conditions the vehicle's forward velocity while performing the corrective action can be calculated as:

$$\begin{aligned} |\mathbf{v^f}_c| &= |\mathbf{v^r}| + |\mathbf{v^m}|_{max} \cdot cos(\alpha), \\ \text{where } \alpha &= \tan^{-1}[(|\mathbf{v^m}|_{max} \cdot sin(\psi_e) \cdot \delta t)/d] \end{aligned} \tag{6}$$

The time taken to perform the corrective action can be calculated as $t_c = d_c/|\mathbf{v^f}_c|$. Thus, the cost incurred per unit distance advanced towards the goal by using a combination of free-flow and thrust-producing action can be given by:

$$C_a^l = \frac{t_c \cdot C_a^t + C_m^t \cdot \delta t}{d + |\mathbf{v^m}| \cdot cos(\psi_e) \cdot \delta t}. \tag{7}$$

In order to compute the lower bound on the cost given in Equation 7, we need to select d so that the C^l is minimized. Solving the above function analytically is not possible and requires application of numerical techniques. Please note that this function depends on $|\mathbf{v^m}|$, $|\mathbf{v^m}|_{max}$, and ψ_e. We have optimized the above function for different combinations of these values using off-line computation. A meta-model (e.g., lookup table) has been developed that allows us to quickly access the lower bound on the value of C^l for free flow actions. Please note that these optimized values of C^l are usually higher compared to the values provided by Equation 4.

If there is no free flow available within the available time window with the opposite sign of ψ_e that will take the vehicle back towards the goal, then we use the modified value of C^l in line 6 of Alg. 1. The use of this value is expected to produce a much better estimate of the cost-to-go and hence improve the computational performance.

6 Results and Discussion

6.1 Simulation Setup

We chose an action set comprising of seventeen actions, out of which 16 actions are thrust-producing $\mathbf{u}_{a,d} = [u_d, \psi_d, \delta t]$ having desired direction ψ_d equally spaced from 0 to 360 degrees and constant surge speed of 10 m/s with respect to the medium. We assume that the maximum magnitude of the medium flow is 6 m/s. We assigned the cost of executing each thrust-producing action to be $C_a^t = 6$ per minute while the cost of executing a free-flow action to be $C_m^t = 1.2$ per minute. We discretized the time with 10 min intervals, i.e., $\delta t = 10$ min, which means that a motion primitive is executed for δt duration before another motion primitive can be commanded. This is mainly because the weather predictions available in practice are seldom more frequent than $\delta t = 10$ min. The medium flow model as described in Section 3.2 has a discrete magnitude and direction profile. The designed scenarios used for performance evaluation of the developed heuristics (see Section 5) have medium profiles that either vary in magnitude, direction or both.

The first set of scenarios uses medium with a constant magnitude profile. The magnitude of the medium flow is held constant at 6 m/s. Specific test cases are:

- Constant flow directions along 30^o and 90^o.
- Rotating medium flow at the rate of 0.1^o per minute with initial direction of 330^o and rotating medium flow of 0.2^o per minute with initial direction of 310^o.

The second set of scenarios uses medium with a randomly generated magnitude profile. The magnitudes are randomly generated in a range of 0 to 6 m/s with the rate of change of 0.5 m/s between two consecutive discrete time steps. Specific test cases are:

- Constant flow direction of 30^o and 90^o
- Randomly generated direction profile changes by 5^o in each discrete time step. We use two scenarios having initial medium flow directions of 5^o and 45^o
- Rotating medium flow at the rate of 0.1^o per minute with initial direction of 330^o and rotating medium flow of 0.2^o per minute with initial direction of 310^o

Table 1: Comparison of the number of states expanded by the path planner using the heuristic $\#2$ and $\#3$ with respect to the heuristic $\#1$ in scenario having medium flows with (a) constant magnitude and (b) random magnitude.

Scenarios with Constant Magnitude of Flow		# States Expanded (in order of 1000s)			% Reduction in States Expanded	
		# 1	# 2	# 3	# 2 with # 1	# 3 with # 1
Constant Direction	30°	196.32	71.32	1.51	63.67	99.23
	170°	459.86	9.04	8.80	98.03	98.09
Rotating Direction	Rate 1°/s	48.86	15.59	5.40	68.10	88.96
	Rate 2°/s	58.89	14.65	5.05	75.13	91.43

Scenarios with Random Magnitude		# Avg. States Expanded (in order of 1000s)			% Reduction in States Expanded			
					# 2 with # 1		# 3 with # 1	
		# 1	# 2	# 3	Mean	Std Dev	Mean	Std Dev
Constant Direction	30°	186.62	56.59	3.83	59.33	23.13	92.17	15.45
	170°	363.71	17.73	7.14	95.13	13.84	98.04	12.83
Random Direction	Init at 0°	323.08	53.75	32.37	81.03	19.40	91.19	11.62
	Init at 45°	270.45	48.63	37.23	87.94	12.33	89.61	11.28
Rotating Direction	Rate 1°/s	336.75	52.01	24.13	84.54	16.27	92.84	12.68
	Rate 2°/s	215.75	28.45	13.32	86.81	17.28	93.82	16.85

6.2 Comparison of Heuristics

The results presented in Table 1(a) compares the performance of all the three heuristics in test scenarios having medium flow of constant magnitude. The reduction in number of states by heuristic $\#2$ with respect to heuristic $\#1$ is lower for scenarios with constant and rotating medium flow with lower values of ψ_e (i.e., favorable medium flows) because it does not account for the cost of thrust-producing action to reach the goal after executing the free-flow action. Heuristics $\#3$ corrects for this problem.

The results presented in Table 1(b) shows the performance of all the three heuristics in test scenarios having medium flows of random magnitude. The performance of heuristic $\#3$ is lower in the scenario having random direction, because in this case it uses best case correction for the deviations caused by the free flow actions.

Table 2 shows the ratio of the cost C_1 to C_2, where C_1 is the cost incurred by the vehicle while using the shortest distance path, and C_2 is the cost incurred by the vehicle while using the developed path planner and the vehicle starts its mission at time $t_{start} = 0$. Higher the ratio of C_1/C_2, the vehicle saves more energy by using the developed planner as compared to the shortest distance-based path planner. The results in Table 2 are computed by randomly generating 100 scenarios for each occupancy value ranging from 10-40%. The results show that with the increase in occupancy of the scenario, the performance of the developed path planner degrades. The primary reason for this decline is the lack of free space for executing long free-flowing actions. Secondly, the vehicle has to execute its thrust-producing action to overcome large number of obstacles in the environment.

6.3 Results on Example Scenarios

The results presented in Figure 6 show the paths generated by the deliberative path planner in the scenario *A* at different start times of the mission. The scenario presented in the figure has a medium flow of constant magnitude, rotating clockwise at the rate of 0.2^o/min. The initial direction of the medium flow at $t_{start} = 0$ is pointing towards the west

Table 2: Performance of the developed energy-efficient planner in randomly generated scenarios with varying occupancy. Cost C_1 is the cost incurred while using the shortest distance path planner and cost C_2 is the cost incurred while using the developed path planner at time $t_{start} = 0$.

Occupancy (in %)	Ratio of C_1 and C_2	
	Mean	Std Dev
10.00	1.94	0.15
20.00	1.72	0.21
30.00	1.42	0.30
40.00	1.12	0.37

(i.e., 270^o). The blue actions are the free-flowing actions and the black actions are the thrust-producing action. Also, the green circle represents the initial location and the red circle indicates the goal location of the vehicle.

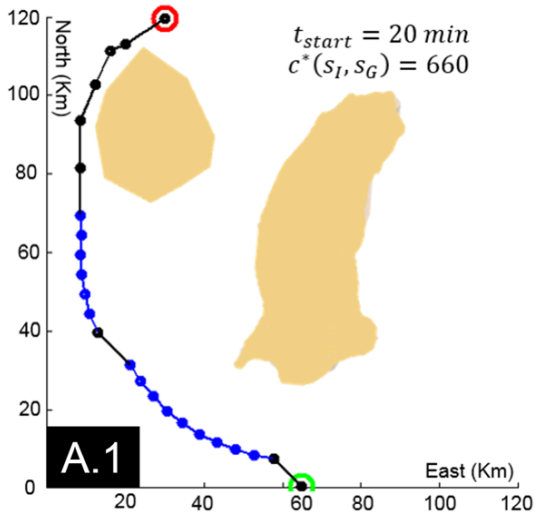
Now, if the vehicle decides to start the mission early at $t_{start} = 20$ min (see Figure 6(a.1)), the planner generates the path by initially using the free-flow action along the medium direction and moves the vehicle far west. In the latter half of the path, the vehicle has to use its thrust-producing action to avoid the obstacle and to reach the goal. On the other hand, if the vehicle prefers to start the mission late (see Figure 6(b)), then it can just use the free-flow action in the middle portion of the path. Finally, Figure 6(c) shows the lowest-cost path produced by the path planner when the vehicle decides to start the mission at the optimal start time.

The paths shown for scenarios *B*, *C* and *D* in Figure 7, are computed at the optimal start time produced by the optimizer. Scenario *B* has medium flows similar to scenario *A*, but rotating at the rate 0.4^o/min. Scenario *C* and *D* have the same medium flow with constant magnitude of 6 m/s, rotating counterclockwise at the rate of 0.6^o per minute. The initial direction of the medium flow at start time $t_{start} = 0$ is pointing east (i.e., 90^o).

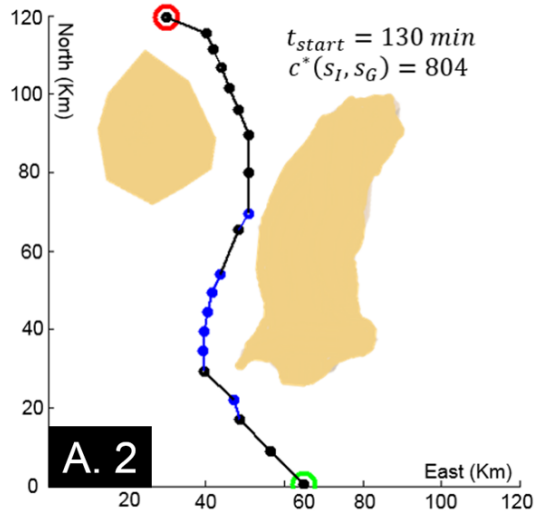
Similar to the results shown in Table 2, Table 3 shows the comparison between cost ratio C_1/C_2 and C_1/C_3 in example scenarios (A-D). Here, the costs C_1, C_2 are the same as described in Section 6.2, cost C_3 is the cost incurred by the vehicle while using the developed energy-efficient path planner and the vehicle starts its mission at optimal time time $t_{start} = t_{optimal}$. The table compares the energy efficiency of the developed algorithm with and without start time optimization.

7 Conclusion and Future Work

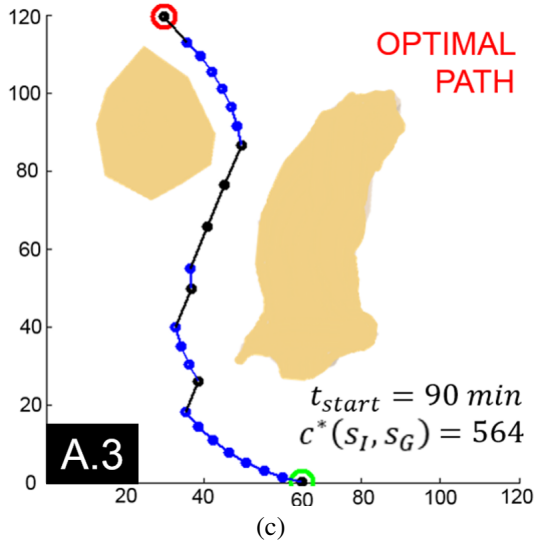
This paper presents a new approach for generating paths for unmanned vehicles in time-varying flow fields. Generated paths show significant improvement in terms of energy cost compared to the shortest distance paths. This has been accomplished by selecting an optimal start time to exploit the flow conditions and using free flow actions that propel the vehicle forward instead of using thrust produced by the actuators. We have developed new admissible heuristics to estimate the cost-to-go in the A* algorithm. These heuristics



(a)

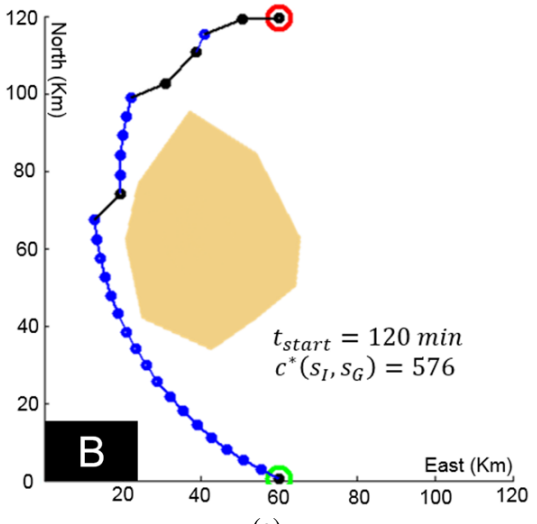


(b)

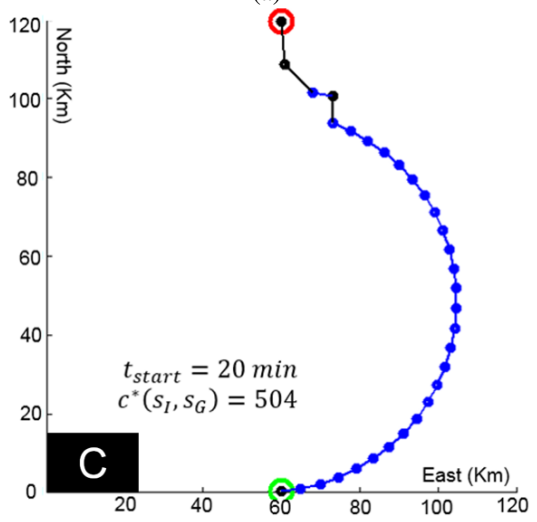


(c)

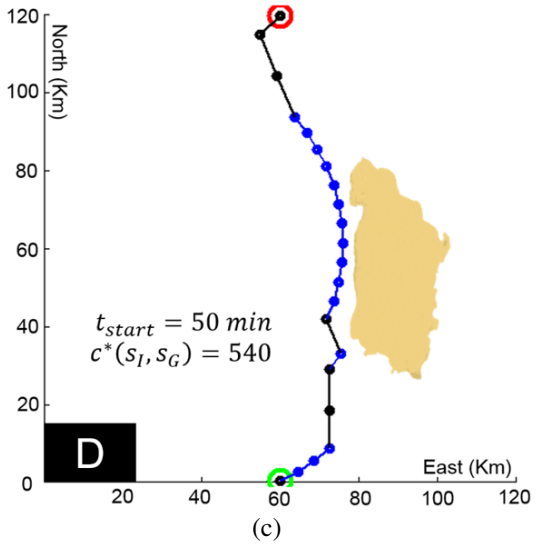
Figure 6: Comparison of paths for the scenario *A* with different start times. The green circle represents the initial location and the red circle represents the goal location of the vehicle. Each blue segment is a free-flow action and each black segment is a thrust-producing action.



(a)



(b)



(c)

Figure 7: Optimal path produced by the path planner for scenarios *B*, *C*, and *D* at optimal start times produced by the optimizer.

Table 3: Comparison between the energy-efficiency provided by the developed path planner without start time optimization (i.e., ratio C_1/C_2) and with start time optimization (i.e., ratio C_1/C_3). Cost C_1 is the cost incurred while using the shortest distance path planner, cost C_2 is the cost incurred while using the developed path planner at time $t_{start} = 0$, and cost C_3 is the cost incurred while using the developed path planner at time $t_{start} = t_{optimal}$.

Scenarios	Start at t = 0 min	Start at optimal time
	Ratio of C_1 and C_2	Ratio of C_1 and C_3
A	1.38	1.39
B	1.11	1.67
C	1.35	1.55
D	1.21	1.33

work effectively to reduce the number of expanded states in a wide variety of flow conditions.

We plan to extend the work presented in this paper as follows. First, we will extend the approach to account for uncertainties in the medium flow forecast by refining paths generated using the A* algorithm. Second, we will use an adaptive control action space to increase angular resolution when necessary to improve the path quality with minimal effect on computational efficiency (in this paper, we use a fixed control action space during the search for a path). Finally, we will incorporate spatial variation in the flow field at a given time instance into the search. This capability will become important when the vehicle needs to travel over very large distances.

References

Al-Sabban, W. H.; Gonzalez, L. F.; Smith, R. N.; and Wyeth, G. F. 2013. Wind-energy based path planning for unmanned aerial vehicles using markov decision processes. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*, 784–789.

Al-Sabban, W. H.; Gonzalez, L. F.; and Smith, R. N. 2012. Extending persistent monitoring by combining ocean models and markov decision processes. In *Oceans*, 1–10. IEEE.

Ceccarelli, N.; Enright, J. J.; Frazzoli, E.; Rasmussen, S. J.; and Schumacher, C. J. 2007. Micro UAV path planning for reconnaissance in wind. In *American Control Conference (ACC '07)*, 5310–5315. IEEE.

Fathpour, N.; Blackmore, L.; Kuwata, Y.; Assad, C.; Wolf, M. T.; Newman, C.; Elfes, A.; and Reh, K. 2014. Feasibility studies on guidance and global path planning for wind-assisted montgolfière in titan. *IEEE Systems Journal* 8(4):1112–1125.

Fiorini, P., and Shiller, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research* 17(7):760–772.

Fox, D.; Burgard, W.; and Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* 4(1):23–33.

Garau, B.; Alvarez, A.; and Oliver, G. 2005. Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an A* approach. In *IEEE International Conference on Robotics and Automation*, 194–198.

Huynh, V. T.; Dunbabin, M.; and Smith, R. N. 2015. Predictive motion planning for AUVs subject to strong time-varying currents and forecasting uncertainties. In *IEEE International Conference on Robotics and Automation (ICRA '15)*, 1144–1151.

Isern-González, J.; Hernández-Sosa, D.; Fernández-Perdomo, E.; Cabrera-Gámez, J.; Domínguez-Brito, A. C.; and Prieto-Marañón, V. 2012. Obstacle avoidance in underwater glider path planning. *Journal of Physical Agents* 6(1):11–20.

Koenig, S., and Likhachev, M. 2002. D* Lite. In *AAAI/IAAI*, 476–483.

Kruger, D.; Stolkin, R.; Blum, A.; and Briganti, J. 2007. Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments. In *IEEE International Conference on Robotics and Automation (ICRA'07)*, 4265–4270.

Kuwata, Y.; Blackmore, L.; Wolf, M.; Fathpour, N.; Newman, C.; and Elfes, A. 2009. Decomposition algorithm for global reachability analysis on a time-varying graph with an application to planetary exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'09)*, 3955–3960.

Lavalle, S. M., and Kuffner Jr, J. J. 2000. Rapidly-Exploring Random Trees: Progress and Prospects. In *Algorithmic and Computational Robotics: New Directions*.

LaValle, S. M. 2006. *Planning algorithms.* Cambridge, U.K.: Cambridge University Press. Available at http://planning.cs.uiuc.edu.

Likhachev, M.; Ferguson, D. I.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2005. Anytime Dynamic A*: An anytime, replanning algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS '05)*, 262–271.

Likhachev, M.; Thrun, S.; and Gordon, G. J. 2004. Planning for Markov decision processes with sparse stochasticity. In *Advances in neural information processing systems*, 785–792.

Lolla, T.; Ueckermann, M.; Yigit, K.; Haley Jr, P.; and Lermusiaux, P. F. 2012. Path planning in time dependent flow fields using level set methods. In *IEEE International Conference on Robotics and Automation (ICRA'12)*, 166–173.

Martinez-Gomez, L., and Fraichard, T. 2009. Collision avoidance in dynamic environments: an ICS-based solution and its comparative evaluation. In *IEEE International Conference on Robotics and Automation (ICRA'09).*, 100–105.

Petres, C.; Pailhas, Y.; Patron, P.; Petillot, Y.; Evans, J.; and Lane, D. 2007. Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics* 23(2):331–341.

Pivtoraiko, M.; Knepper, R. A.; and Kelly, A. 2009. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics* 26(3):308–333.

Rao, D., and Williams, S. B. 2009. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA '09)*.

Reif, J. H., and Sun, Z. 2004. Movement planning in the presence of flows. *Algorithmica* 39(2):127–153.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A modern approach*. Pearson.

Sanner, S.; Goetschalckx, R.; Driessens, K.; Shani, G.; et al. 2009. Bayesian real-time dynamic programming. In *International Joint Conference on Artificial Intelligence (IJCAI '09)*, 1784–1789.

Shah, B., and Gupta, S. K. 2016. Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles. In *Internation Conference on Automated Planning and Scheduling (ICAPS' 16)*.

Shah, B. C.; Švec, P.; Bertaska, I. R.; Klinger, W.; Sinisterra, A. J.; Ellenrieder, K. v.; Dhanak, M.; and Gupta, S. K. 2014. Trajectory planning with adaptive control primitives for autonomous surface vehicles operating in congested civilian traffic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '14)*.

Smith, R. N., and Huynh, V. T. 2014. Controlling buoyancy-driven profiling floats for applications in ocean observation. *IEEE Journal of Oceanic Engineering* 39(3):571–586.

Soulignac, M. 2011. Feasible and optimal path planning in strong current fields. *IEEE Transactions on Robotics* 27(1):89–98.

Thompson, D. R.; Chien, S.; Chao, Y.; Li, P.; Cahill, B.; Levin, J.; Schofield, O.; Balasuriya, A.; Petillo, S.; Arrott, M.; et al. 2010. Spatio-temporal path planning in strong, dynamic, uncertain currents. In *IEEE International Conference on Robotics and Automation (ICRA'10)*, 4778–4783.

Švec, P.; Schwartz, M.; Thakur, A.; and Gupta, S. K. 2011. Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'11)*.

Švec, P.; Shah, B. C.; Bertaska, I. R.; Alvarez, J.; Sinisterra, A. J.; Ellenrieder, K. v.; Dhanak, M.; and Gupta, S. K. 2013. Dynamics-aware target following for an autonomous surface vehicle operating under COLREGs in civilian traffic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*.

Švec, P.; Thakur, A.; Raboin, E.; Shah, B. C.; and Gupta, S. K. 2014. Target following with motion prediction for unmanned surface vehicle operating in cluttered environments. *Autonomous Robots* 36(4):383–405.

Witt, J., and Dunbabin, M. 2009. Go with the flow: optimal AUV path planning in coastal environments. In *Proceedings of the 2008 Australasian Conference on Robotics & Automation*, 1–9. Australasian Robotics and Automation Association (ARAA).

A Bayesian Effort Bias for Sampling-based Motion Planning

Scott Kiesel and **Wheeler Ruml**

Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Abstract

Recent advances in sampling-based motion planning have exploited concepts similar to those used in the heuristic graph search community, such as computing heuristic cost-to-go estimates and using state-space abstractions to derive them. Following this trend, we explore how the concept of search effort can be exploited to find plans quickly. Previous work in motion planning attempts to find plans quickly by preferring states with low cost-to-go. Recent work in graph search suggests that following search-effort-to-go estimates can yield faster planning. In this paper, we demonstrate how this idea can be adapted to the context of motion planning. Our planner, BEAST, uses on-line Bayesian estimates of effort to guide the expansion of a motion tree toward states through which a plan is estimated to be easy to find. We present results in five simulated domains (Kinematic and Dynamic Car, Hovercraft, Blimp and Quadrotor) indicating that BEAST is able to find solutions more quickly and has a higher success rate than previous methods. We see this work as further strengthening the algorithmic connections between motion planning and heuristic graph search.

Introduction

We address the problem of single-query kinodynamic motion planning: given a start state, description of the obstacles in the workspace, and a goal region, find a dynamically feasible continuous trajectory (a sequence of piece-wise constant controls) that takes the robot from the start state to the goal region without intersecting obstacles (Choset et al. 2005; LaValle 2006). We work within the framework of motion trees, popularized by sampling-based motion planning, in which the planner grows a tree of feasible motions from the start state, attempting to reach the goal state. This approach is appealing because it applies to any vehicle that can be forward simulated, allowing the planner to respect realistic constraints such as acceleration limits. Examples of algorithms taking this approach include RRT (LaValle and Kuffner 2001), KPIECE (Şucan and Kavraki 2009), and P-PRM (Le and Plaku 2014).

Although the figure of merit on which these algorithms are usually compared is the time taken to find a (complete and feasible) solution. Close examination of these algorithms reveals that their search strategies are not explicitly designed to optimize that measure. RRT uses sampling with a voronoi bias to encourage rapidly covering the entire state space. KPIECE uses more sophisticated coverage estimates to achieve the same end. Focusing on regions of the state space with low motion tree coverage helps to grow the tree outward, but is not focused on reaching the goal. Coverage promotes probabilistic completeness but not necessarily finding a solution quickly.

In artificial intelligence, a central principle for exploring large state spaces is to exploit heuristic information to focus problem-solving in promising regions. The A* heuristic graph search algorithm serves as the central paradigm. In motion planning, the P-PRM algorithm exploits heuristic cost-to-go information to guide growth of its motion tree, with the aim of finding solutions faster than unguided methods. While focusing on low cost regions directs sampling toward the goal, it ignores the effort that can be required for a motion planner to thread a trajectory through a cluttered area. In this way, cost-to-go estimates can encourage the search to focus on challenging portions of the state space, slowing the search. Fundamentally, optimizing solution cost is not the same as optimizing planning effort.

Recent work in heuristic graph search has recognized the separate roles of cost and effort estimates in guiding search, particularly when solutions are desired quickly (Thayer and Ruml 2009; Thayer and Ruml 2011). In this paper, we show how to exploit that idea in the context of motion planning. We propose an algorithm, Bayesian Effort-Aided Search Trees (BEAST), that biases tree growth through regions in the state space believed to be easy to traverse. If motion propagation does not go as anticipated, effort estimates are updated online based on the planner's experience, and used to redirect planning effort to more fruitful parts of the state space. We implement this method in the Open Motion Planning Library (OMPL) (Sucan, Moll, and Kavraki 2012) and evaluate it in five different simulated domains (Kinematic and Dynamic Car, Hovercraft, Blimp and Quadrotor). The results suggest that BEAST successfully uses effort estimates to efficiently allocate planning effort: it finds solutions faster than RRT, KPIECE, and P-PRM and is the only method able to solve all benchmark instances. We see this work as a further demonstration of how ideas from heuristic graph search can be useful in motion planning.

Previous Work

There has been much previous work on biases for sampling-based motion planners. The two most prominent types in the recent literature have been to bias toward less explored portions of the state space or to bias exploration toward regions of the state space believed to contain low cost solutions. Both of these biases have shown strong results in finding solutions quickly. The two state of the art algorithms considered in this paper are KPIECE (Şucan and Kavraki 2009) and P-PRM (Le and Plaku 2014).

KPIECE

Kinodynamic Planning by Interior-Exterior Cell Exploration, or KPIECE (Şucan and Kavraki 2009), is an algorithm that uses a multi-level projection of the state space to estimate coverage in the state space. It then uses these coverage estimates to reason about portions of the state space to explore next. Expansive Space Trees (EST) (Hsu, Latombe, and Motwani 1999) and Path-Directed Subdivision Tree (PDST) (Ladd and Kavraki 2005) also focus on less explored portions of the state space but have been shown to be outperformed by KPIECE. The general all-around good performance of KPIECE has led to its selection as the default motion planner in OMPL.

KPIECE is focused on quickly covering as much of the state space as possible. It always gives priority to less covered areas of the state space. When an area of low coverage is discovered it attempts to extend the motion tree into that area. It uses a coarse resolution initially to find out roughly which area is less explored. Within this area, finer resolutions can then be employed to more accurately detect less explored areas.

While KPIECE targets exploring unvisited areas of the state space, this may not always be the fastest approach to finding the goal. Certainly targeting exploration toward the goal could help improve performance.

P-PRM

P-PRM (Le and Plaku 2014) is based on ideas from an earlier planner called Synergistic Combination of Layers of Planning (SyCLOP) (Plaku, Kavraki, and Vardi 2010). It shares the intuition that information from a discrete abstraction of the workspace can be used to identify low level paths that may lead to the goal. While SyCLOP was shown to be very successful, in recent work P-PRM has been shown to outperform SyCLOP in a variety of planning problems.

P-PRM uses the geometric component of the state space to construct a Probablistic Roadmap (PRM) (Kavraki et al. 1996). It generates random states in the geometric space, then connects each state to its nearest neighbors via an edge, forming a graph. The edges in the graph are collision checked and removed from the graph if a collision along them is found. The graph vertices represent regions of geometric space and the edges summarize the connectivity of the regions.

P-PRM runs a Dijkstra search out from the abstract region containing the concrete goal to compute h-values, or heuristic estimates of cost to the goal. It then uses these heuristic values, and the associated shortest paths from the goal to each abstract node, to bias sampling.

It searches by maintaining a queue of abstract states in the graph sorted by increasing scores (initially their h-value, see the paper for details). At each search iteration the abstract state with the lowest score is selected. An abstract state along the cheapest precomputed path rooted at the currently selected state is chosen. This state is then used to create a random concrete state within some pre-specified state radius. This is now the "target" state used similarly to when plain RRT chooses a state uniformly at random. That means that the nearest state in the existing motion tree is chosen as the root for the new propagation which is steered (if possible) toward the random state. Any new abstract states touched by the propagation attempt are added to the queue if not previously enqueued.

P-PRM tries to pursue the completion of low cost paths by following its heuristic estimates in the abstract space. It tries to avoid getting stuck during planning by penalizing the score of abstract states when they are examined.

Speedy Search

While RRT and KPIECE are often the reliable workhorses of motion planning, the success of heuristically-informed graph search algorithms such as A* (Hart, Nilsson, and Raphael 1968) in artificial intelligence would suggest that brute-force expansion into all unexplored regions of the state space (in a manner similar to Dijkstra's algorithm) is not an optimal strategy. P-PRM has been shown to provide state of the art performance by exploiting heuristic cost-to-go guidance. Yet recent results in the heuristic graph search community show that exploring the state space based on cost often does not give the best speedup.

In the context of discrete graphs, Greedy search, which focuses on nodes with low heuristic cost-to-goal, is often surpassed by 'Speedy search', which focuses on nodes with a low estimated number of hops (or graphs edges) to the goal (Thayer and Ruml 2009; Wilt and Ruml 2014). In this paper, we present one attempt at adapting this idea to motion planning, in which the state and action spaces are continuous and there is no predefined graph structure.

Exploiting Effort Estimates

While there is not a direct translation of the "number of edges to the goal" concept, there is still a notion of search effort. In heuristic search, the fewer expansions needed to find the goal, typically the quicker a solution is found. In sampling-based motion planning, the unit of measure would be the number of samples, or propagation attempts in the motion tree. Each forward propagation of the system state requires collision checking, which is computationally expensive. The fewer propagation attempts made before finding the goal, typically the faster a solution is found (assuming reasonable iteration overhead).

Overview

Bayesian Effort-Aided Search Trees (BEAST) is a novel method that tries to find solutions as quickly as possible by

constructing solutions which it estimates require the least effort to build. It maintains online Bayesian estimates of the effort of connecting abstract regions of the state space and allocates its search effort to the region of the state space that is estimated to require the lowest effort to connect to the abstract goal region.

BEAST exploits a discrete abstraction of the state space. In the experiments reported below, we use a PRM workspace abstraction very similar to the one used by P-PRM. We begin by identifying the geometric component of the state space. The abstraction will exist only in this subspace. We generate uniformly random states in the abstract space (1000 in the experiments below). As in P-PRM, these states induce a division of the state space into abstract regions (by associating any concrete state with the nearest abstract state). Neighboring abstract states (the 5 nearest in the experiments below) are connected by directed edges, forming a directed graph. (If the abstract start and goal regions remain unconnected, additional samples are taken until they are.)

As just discussed, for each edge e, BEAST maintains an effort estimate, $ee(e)$, of how many propagation attempts would be required on average to take a concrete state contained in the abstract region represented by the source vertex of the edge to a concrete state contained in the abstract region represented by the end vertex. These estimates are initialized by a geometric collision check along the abstract edge. However, BEAST explicitly acknowledges that this quick check in geometric space is only a rough approximation of a robot's ability to steer from one region to the other. We represent our uncertain belief about each edge in a Bayesian style: we regard a propagation attempt as sampling a Bernoulli variable and we maintain a beta distribution (with parameters α, β) over its success probability. The initial geometric collision check provides some evidence about this probability, and then each propagation attempt during planning provides additional evidence. In the experiments reported below, an edge with a detected collision is initialized to $\alpha = 1, \beta = 10$, and all other edges are initialized to $\alpha = 10, \beta = 1$. Successful attempts increase α by one and unsuccessful attempts increase β by one. Based on our belief, we estimate the number of propagation attempts that will be necessary in order to have a successful one as $(\alpha + \beta)/\alpha$.

BEAST uses the abstract graph as a metareasoning tool to decide where it should spend its time growing the motion tree. We only consider abstract regions touched by the motion tree and each edge from the corresponding vertex in the abstract graph represents a possible propagation attempt. We compute, for each directed edge e, the expected total effort $te(e)$ required to reach the abstract goal if we start propagating a state from its start region through its end region and onward to the goal. For 'exterior' edges, whose start region has not yet had a successful propagation into its end region, this is straightforward: the effort to cross that edge plus the total effort-to-goal from the end vertex. More formally: if, for every vertex in the abstract graph, we let $te(v)$ be the minimum over its outgoing edges e of $te(e)$, then $te(e) = ee(e) + te(e.end)$. 'Interior' edges are more complex. Unless the goal region has been reached, any interior edge will lead to an exterior edge that has a lower total effort estimate, so such edges may not appear to be useful for propagation. However, recall that our state space abstraction might be very rough, and not all concrete states falling in the same abstract region are necessarily equivalent. We may well want to propagate along an interior edge in order to add additional states to the end region, in the hopes that this will increase the probability of being able to propagate onward from there. We model this by assuming that an additional state in the destination region will raise its α by $1/n$, where n is the number of states already in the region. (We want this bonus to depend inversely on the number of existing states, to reflect the decreasing marginal utility of each additional state.) So for an interior edge e with a destination vertex d that contains n states in its abstract region,

$$te(e) = ee(e) + \min_{e_2 \in e.out} \frac{e_2.\alpha + 1/n + e_2.\beta}{e_2.\alpha + 1/n} + te(e_2.dest).$$

BEAST$(Abstraction, Start, Goal)$

```
1.  AbstractStart = Abstraction.Map(Start)
2.  AbstractGoal = Abstraction.Map(Goal)
3.  Abstraction.PropagateEffortEstimates()
4.  Open.Push(AbstractStart.GetOutgoingEdges())
5.  While NotFoundGoal
6.    Edge = Open.Pop()
7.    StartState = Edge.Start.Sample()
8.    EndState = Edge.End.Sample()
9.    ResultState = Steer(StartState, EndState)
      // Or Propagate With Random Control
10.   Success = Edge.End.Contains(ResultState)
11.   If Success
12.     Edge.UpdateWithSuccessfulPropagation()
13.     If Edge.End == AbstractGoal
14.       Open.Push(GoalEdge)
          // Goal Region To Goal State
15.   Else
16.     Edge.UpdateWithFailedPropagation()
17.   Abstraction.PropagateEffortEstimates()
18.   Open.Push(Edge)
19.   If Success
20.     Open.Push(Edge.End.GetOutgoingEdges())
```

Figure 1: Pseudocode for the BEAST algorithm.

Details

Pseudocode for BEAST is presented in Figure 1. The algorithm is passed an abstraction of the workspace, concrete start state and a concrete goal state. BEAST first begins by propagating effort estimates through the abstract graph outward from the region containing the concrete goal state (line 3). For efficiency, the collision checking and beta distribution initialization can be done lazily.

We use the pseudocode in Figure 2 to estimate the number of propagation attempts needed if the planner were to start by propagating along a specific edge. For exterior edges, this effort value is straightforward (line 22).

On Line 25 for interior edges, we examine each of the

```
GetEffort(Edge)
21.  If Not Edge.interior
22.     Return ee(Edge) + te(Edge.End)
23.  Else
24.     Child_Edges = Edge.End.GetOutgoingEdges()
25.     Return ee(Edge) +
          min_{Child∈Child_Edges} OptimisticBenefit(Child) +
                                  te(Child.End)

OptimisticBenefit(Edge)
26.  PositiveEffect = 1. / Edge.Start.NumStates
27.  Optα = Edge.α + PositiveEffect
28.  Return (Optα + Edge.β) / Optα
```

Figure 2: Pseudocode for calculating an edge effort value.

children of the current edge to see which child edge would require the least effort to arrive at the goal if it were provided another state in its start region. We take the minimum effort over the children and add in the estimated effort of propagating along the current edge.

If effort estimates were static, a single pass of Dijkstra's algorithm would suffice to compute te values. In our case, edge effort estimates change over time so we use an incremental best-first search called D* Lite (Koenig and Likhachev 2002) to avoid replanning from scratch. D* Lite updates the heuristic estimates for cost to go to the goal at each vertex in the graph, in our case we are using effort (te) to go instead. While propagating effort at each vertex we also store an effort estimate at each edge which is calculated using $GetEffort$.

To reiterate, this value can be seen as an estimate of how many samples will be required to reach the goal if you were to choose to propagate along an edge and then choose the minimum effort edges thereafter. A queue called $Open$ is then initialized with outgoing edges from the abstract region containing the concrete start state(line 4). $Open$ is sorted in increasing order of edge effort. The search always considers the least effort edge first.

The algorithm proceeds by popping the edge off $Open$ with the lowest estimated effort (line 6). This edge is then sampled at its start abstract region and its end abstract region in lines 7-8. In our implementation, the concrete state in the edge's start region that has been selected the fewest number of times is chosen as the $StartState$. A concrete state is chosen from the edge's abstract end region uniformly at random within some radius centered around the region's centroid.

An attempt is made to grow the tree from $StartState$ to $EndState$ using a steering function (line 9). In our implementation if no steering function was available in OMPL, we instead generated 10 random controls, applied each to $StartState$ and the resulting motion that got closest to $EndState$ was chosen. [1]

If the newly propagated motion at any point reached the target abstract region (the selected edge's end region), the edge is updated with a successful trial (line 10-12). This simply adds one to the α value of the beta distribution associated with this edge.

If the target region is not reached, the β bucket is incremented (line 16). With each trial to propagate along an edge we update our belief about the effort required to reach the goal by using the edge. This effectively changes the edge "cost" in the abstract graph and we use our incremental search to update the effort estimates throughout the graph based on this local update (line 17).

If the edge was successfully propagated along, we also add its child edges (outgoing edges from the current edge's end region) to the $Open$ list (line 20). We re-add the current edge to the $Open$ in all outcomes (line 18).

There is also a special case (line 13) added which enables us to use sparse abstractions. With sparse abstractions we can compute our effort values more efficiently during each iteration. However, when the goal abstract region is reached, with a sparse abstraction, it may cover a large portion of the state space. Growing the tree into a possibly large goal region may not be focused enough to find a state close to the goal state. To combat this we add a special $GoalEdge$ to $Open$ (line 14). This is an edge that when expanded will return a $StartState$ from the goal abstract region and an $EndState$ focused around the actual concrete goal state.

Experiments

All experiments were run using control algorithms from the OMPL framework where available (KPIECE and RRT). P-PRM was implemented following closely along with the description and pseudo code included in the paper. Experiments also used OMPL's implementation of a Kinematic Car, Dynamic Car, Blimp and Quadrotor vehicle, as detailed below. We implemented a Hovercraft in OMPL following Lynch (1999).

Kinematic Car

The mesh used for the Kinematic Car vehicle is shown in Figure 3 panel (a). The equations defining the Kinematic Car's motion and control inputs in OMPL are as follows:

$$\dot{x} = u_0 \cdot cos(\theta),$$
$$\dot{y} = u_0 \cdot sin(\theta),$$
$$\dot{\theta} = \frac{u_0}{L} \cdot tan(u_1)$$

where the control inputs (u_0, u_1) are the translational velocity and the steering angle, respectively, and L is the distance between the front and rear axle of the car which is set to 1.

Dynamic Car

The mesh used for the Dynamic Car vehicle is shown in Figure 3 panel (a). The equations defining the Dynamic Car's

[1] This functionality was implemented at the control sampler level in OMPL for each domain so any algorithm using "sampleTo" provided by the domain's control sampler received equal benefit.

motion and control inputs in OMPL are as follows:

$$\begin{aligned}\dot{x} &= v \cdot cos(\theta),\\ \dot{y} &= v \cdot sin(\theta),\\ \dot{\theta} &= \frac{v \cdot m}{L} \cdot tan(\phi),\\ \dot{v} &= u_0,\\ \dot{\phi} &= u_1\end{aligned}$$

where v is the speed, ϕ the steering angle, the controls (u_0, u_1) control their rate of change, m is the mass of the car (set to 1), and L is the distance between the front and rear axle of the car (also set to 1)

Hovercraft

The mesh used for the Hovercraft vehicle is shown in Figure 3 panel (a). The equations defining the Hovercrafts's motion and control inputs from Lynch (1999) are as follows:

$$\begin{aligned}\dot{x} &= \frac{F}{M} cos(\theta) - \frac{B_t}{M} x,\\ \dot{y} &= \frac{F}{M} sin(\theta) - \frac{B_t}{M} y,\\ \dot{\theta} &= \frac{\tau}{0.5 \cdot M \cdot R^2} - \frac{B_r}{M} \cdot \theta\end{aligned}$$

where F is the force exerted by the thrusters and τ is the torque exerted by the thrusters. B_t and B_r are the translational and rotational friction coefficients (both set to 0). M is the mass of the robot and R is the radius of the robot (both set to 1).

Blimp

The mesh used for the Blimp vehicle is shown in Figure 3 panel (b). The equations defining the Blimp's motion and control inputs in OMPL are as follows:

$$\begin{aligned}\ddot{x} &= u_f \cdot cos(\theta),\\ \ddot{y} &= u_f \; sin(\theta),\\ \ddot{z} &= u_z,\\ \ddot{\theta} &= u_\theta\end{aligned}$$

where (x, y, z) is the position, θ the heading, and the controls (u_f, u_z, u_θ) control their rate of change.

Quadrotor

The mesh used for the Quadrotor vehicle is shown in Figure 3 panel (c). The equations defining the Quadrotor's motion and control inputs in OMPL are as follows:

$$\begin{aligned}m\ddot{p} &= -u_0 \cdot n - \beta \cdot \dot{p} - m \cdot g,\\ \alpha &= (u_1, u_2, u_3)^T,\end{aligned}$$

where p is the position, n is the Z-axis of the body frame in world coordinates, α is the angular acceleration, m is the mass, and β is a damping coefficient. The system is controlled through $u = (u_0, u_1, u_2, u_3)$.

In the Kinematic and Dynamic Car domains the goal radius was set to 0.1, the remaining domains each used a goal radius of 1. The goal distance of a state was based only on the distance in the XY or XYZ dimensions. Other parameters that were used included a propagation step value of 0.05, min and max control durations of 1 and 100 respectively, and intermediate states were included during planning. The workspace was bounded by $-30 \leq x \leq 30$, $-30 \leq y \leq 30$ and $-5 \leq z \leq 5$.

KPIECE and RRT were run using their default parameters. P-PRM was also run using its suggested parameters described in the paper. The state radius size for sampling was shared between P-PRM and BEAST. This value was set to 6, which gave good visible coverage over the abstract regions and the best performance over those state radii tried: {2,4,6}.

The obstacle mesh used for the experiments is presented in Figure 3 panel (d). For each vehicle, 5 start and goal pairs were used, and for each start and goal pair 50 different random number generator seed values were used. This provided 250 runs for each of the domains. The start states were biased toward the center of the workspace while the goal was biased toward the lower center of the workspace. This setup tends to generate problems in which the optimal solution threads its way carefully between the obstacles, but it is much easier to take a more costly route around the obstacles. This wide diversity of planning time/solution cost trade-offs directly tests the ability of BEAST to estimate planning effort and adjust its behavior accordingly. A motion planning that explicitly tries to find plans quickly ought to exhibit superior performance. A planning timeout of 60 seconds was used.

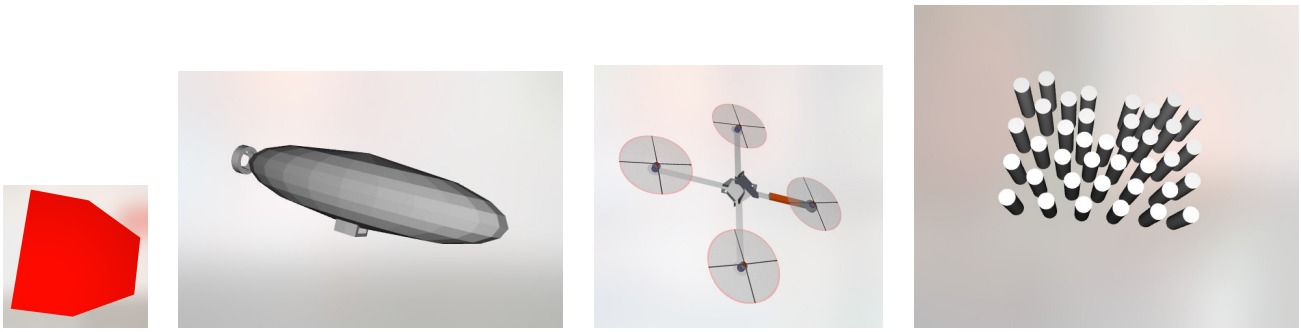
Results

The results of the experiments are presented in Figure 4. Each box represents the middle 50% of the data, with a horizontal line at the median. Whiskers extend to the furthest point within 1.5 times the interquartile range. The remaining outliers are plotted with circles. The 95% confidence interval around the mean is depicted with a gray rectangle. The plots in each panel are sorted according to their means. In order to have enough data points to create plots, algorithm runs that timed out without providing a valid solution are still included in the plot. These runs are represented by the time collected by OMPL after the timeout was issued. Several of the plots have been clipped at the top so that the top two performers remain legible.

In the Blimp domain, BEAST has the lowest mean planning time as well as the lowest variance in its performance. In the Quadrotor domain, BEAST again has the lowest mean, but P-PRM appears to have slightly lower variance.

A video of the sampling and tree growth of each of the algorithms considered in this paper can be found at `https://www.youtube.com/watch?v=Or8sQBOrVh4`. It is a top down visualization of a Quadrotor planning instance. It illustrates RRT's slow coverage of the entire state space, KPIECE's rapid coverage of the state space, P-PRM's focus on estimated low cost paths and BEAST's focus on finding low effort solutions.

The number of runs where each algorithm was unable to solve an instance is provided in Figure 5. In the Blimp do-



(a) (b) (c) (d)

Figure 3: The car, blimp and quadrotor vehicles used in the experiments, and the forest environment.

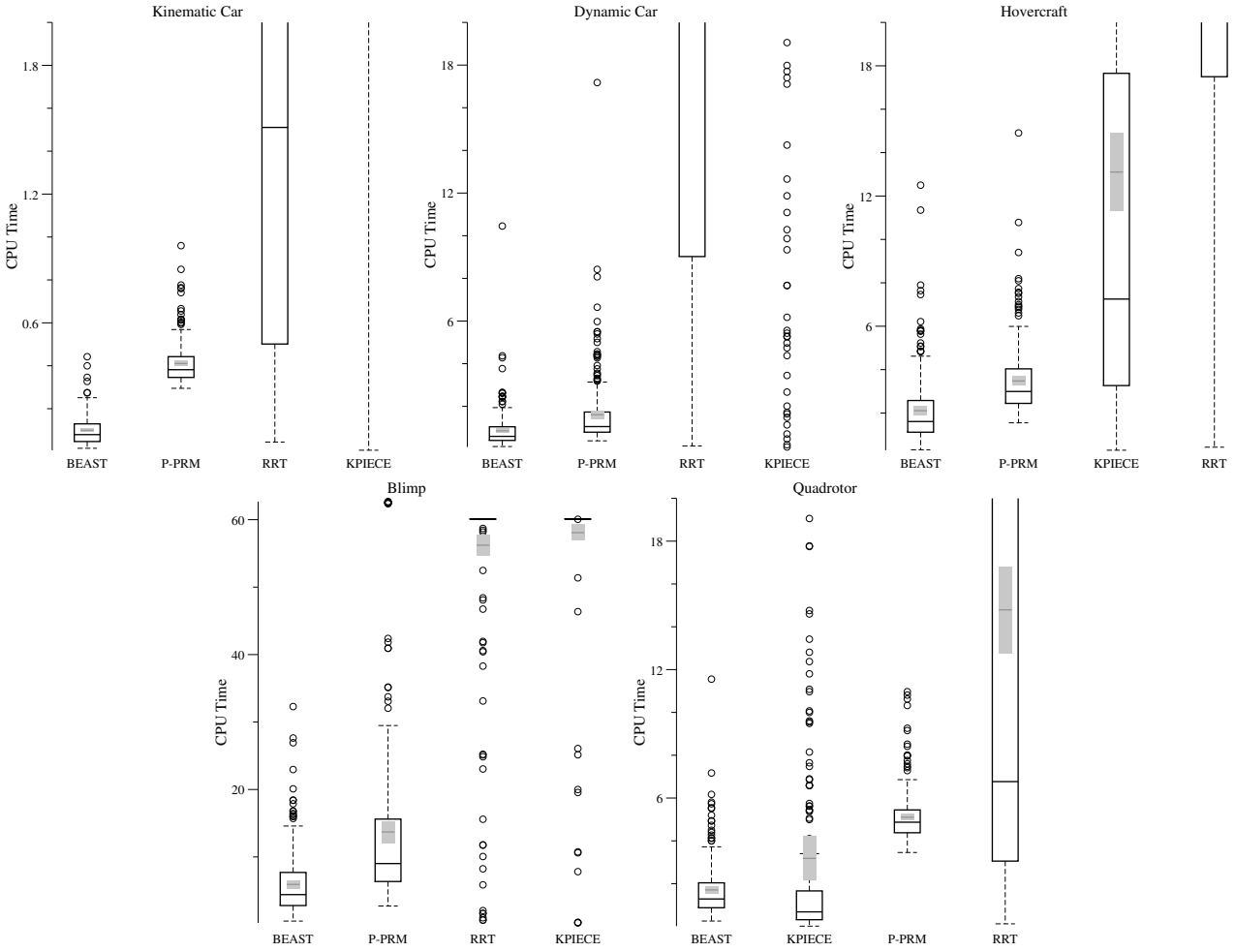


Figure 4: Computation time for 5 start goal pairs and 50 random seeds (250 instances).

	RRT	KPIECE	P-PRM	BEAST
Kinematic Car	0	99	0	0
Dynamic Car	108	189	0	0
Hovercraft	116	8	0	0
Blimp	221	238	11	0
Quadrotor	12	2	0	0

Figure 5: Number of unsolved instances for 5 start goal pairs and 50 seeds (250 instances).

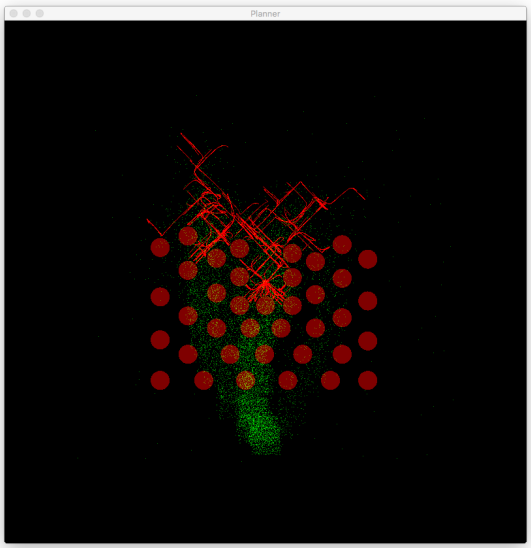


Figure 6: P-PRM sampling and tree growth example in the Quadrotor domain (top down view).

main, BEAST is the only algorithm that is able to find a solution to all the instances within the timeout. In the Quadrotor domain, BEAST and P-PRM are both able to find solutions to all instances while KPIECE and RRT are not able to within the timeout.

Discussion

One of the major benefits of BEAST is that it explicitly focuses on areas of the state space that it believes will be easy to traverse while heading toward the goal. KPIECE will eventually explore the same regions of the state space but does so without focusing on paths toward the goal. P-PRM does focus on paths leading to the goal, but focuses on paths associated with low cost. These paths can be arbitrarily difficult to find given obstacle configurations.

This is shown in Figure 6 where many P-PRM generates samples (green dots) along abstract paths to the goal, but it is challenging to grow the motion tree (red lines) toward them. Eventually from the uniform random sampling and increasing cost estimates for the states it has selected many times, search begins to spill around and through the obstacles (red circles).

Another feature of BEAST that helps it construct its tree more efficiently is that it focuses its tree growth either internal to the existing tree or directly along the fringe of the existing tree. This focus on the boundary of the motion tree is very similar to that of KPIECE, yet the two methods allocate their exploration effort very differently. P-PRM does not focus its sampling near the existing tree and can generate samples arbitrarily far away, which are less helpful when growing the tree through tight spaces.

There are other motion planners that leverage heuristic cost-to-go, but in ways very different from BEAST. Informed RRT* (Gammell, Srinivasa, and Barfoot 2014) uses ellipsoidal pruning regions to ignore areas of the state space that are guaranteed not to include a better solution. BIT* (Gammell, Srinivasa, and Barfoot 2015) uses heuristic cost estimates directly in its search strategy, but for kinodynamic planning it requires a boundary value problem solver to rewire trajectories between sampled states, making it inapplicable to many problems.

Finding solutions quickly is an important feature in many applications, but convergence to an optimal solution is also highly desirable. In future work, we plan to combine our effort based planner BEAST with heuristic cost estimates, yielding an anytime planner which quickly finds a solution and then spends its remaining planning time improving its incumbent solution cost.

Conclusion

We have presented a new algorithm called Bayesian Effort-Aided Search Trees. BEAST exploits and updates Bayesian estimates of propagation effort through the state space to find solutions quickly. Results on a variety of domains showed that BEAST on average found solutions the fastest and was the only algorithm to find solutions to every instance in the benchmark set. We see this work as reinforcing the current trend toward exploiting ideas from AI graph search in the context of robot motion planning, and providing further evidence that searching under time pressure is a distinct activity from searching for low-cost solutions.

Acknowledgments

We gratefully acknowledge support from NSF (grant 1150068).

References

[Choset et al. 2005] Choset, H.; Lynch, K.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2005. *Principles of robot motion: theory, algorithms, and implementation*. MIT Press.

[Gammell, Srinivasa, and Barfoot 2014] Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Gammell, Srinivasa, and Barfoot 2015] Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2015. Batch informed trees (BIT*): Sampling-based optimal planning via the

heuristically guided search of implicit random geometric graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 3067–3074. IEEE.

[Hart, Nilsson, and Raphael 1968] Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

[Hsu, Latombe, and Motwani 1999] Hsu, D.; Latombe, J.-C.; and Motwani, R. 1999. Path planning in expansive configuration spaces. *International Journal of Computational Geometry & Applications* 9(04n05):495–512.

[Kavraki et al. 1996] Kavraki, L. E.; Švestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on* 12(4):566–580.

[Koenig and Likhachev 2002] Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI/IAAI*, 476–483.

[Ladd and Kavraki 2005] Ladd, A. M., and Kavraki, L. E. 2005. Motion planning in the presence of drift, underactuation and discrete system changes. In *Robotics: Science and Systems*, 233–240.

[LaValle and Kuffner 2001] LaValle, S. M., and Kuffner, J. J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5):378–400.

[LaValle 2006] LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

[Le and Plaku 2014] Le, D., and Plaku, E. 2014. Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 212–217. IEEE.

[Lynch 1999] Lynch, K. M. 1999. Controllability of a planar body with unilateral thrusters. *Automatic Control, IEEE Transactions on* 44(6):1206–1211.

[Plaku, Kavraki, and Vardi 2010] Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2010. Motion planning with dynamics by a synergistic combination of layers of planning. *Robotics, IEEE Transactions on* 26(3):469–482.

[Şucan and Kavraki 2009] Şucan, I. A., and Kavraki, L. E. 2009. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*. Springer. 449–464.

[Sucan, Moll, and Kavraki 2012] Sucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The open motion planning library. *Robotics & Automation Magazine, IEEE* 19(4):72–82.

[Thayer and Ruml 2009] Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*, 382–385.

[Thayer and Ruml 2011] Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

[Wilt and Ruml 2014] Wilt, C. M., and Ruml, W. 2014. Speedy versus greedy search. In *Seventh Annual Symposium on Combinatorial Search*.

Scheduling Pick-and-Place Tasks for Dual-arm Manipulators using Incremental Search on Coordination Diagrams

Andrew Kimmel and **Kostas E. Bekris**
Computer Science Department
Rutgers, the State University of New Jersey
110 Frelinghuysen Road, Piscataway NJ, USA
email:{andrew.kimmel,kostas.bekris}@cs.rutgers.edu

Abstract

A premise of dual-arm robots is increased efficiency relative to single-arm counterparts in manipulation challenges. Nevertheless, moving two high-dimensional arms simultaneously in the same space is challenging and care must be taken so that collisions are avoided. Given trajectories for two arms to pick two objects, velocity tuning over a coordination diagram can resolve collisions. When multiple objects need to be moved, a scheduling challenge also arises. It involves finding the order with which objects should be manipulated. This paper considers two ways to approach this combination of scheduling and coordination challenges: (i) a "batch" approach, where an ordering of objects is selected first; for the given ordering, velocity tuning is performed over a matrix of coordination diagrams that considers all pairs of pick-and-place trajectories; and (ii) an incremental approach, where the ordering of objects is discovered on the fly given the subset of coordination diagrams that arise depending on which object one of the arms is currently manipulating. Simulated experiments for a Baxter robot show that both methods return significantly more efficient trajectories relative to the naive "round-robin" schedule, where only one arm moves at a time. Furthermore, the incremental approach is computationally faster, it implicitly provides a good schedule for picking objects and can be used effectively when objects appear dynamically.

1 Introduction

One broad category of manipulation tasks, which appear frequently in manufacturing setups, correspond to pick and place challenges. For instance, a robot may need to grasp multiple products from a tabletop and place them in bins so that they are packaged. An example of such a scenario is shown in Figure 1. Alternatively, a robot could be tasked with stocking and retrieving items from shelves in a warehouse. It has been argued that dual-arm humanoid manipulators can be appropriate solutions for such tasks. The reasoning is that they can be easily used in facilities that have been constructed for human workers as they exhibit similar reachability to people and bi-manual skills.

This work focuses on pick-and-place tasks where multiple objects need to be manipulated and those objects

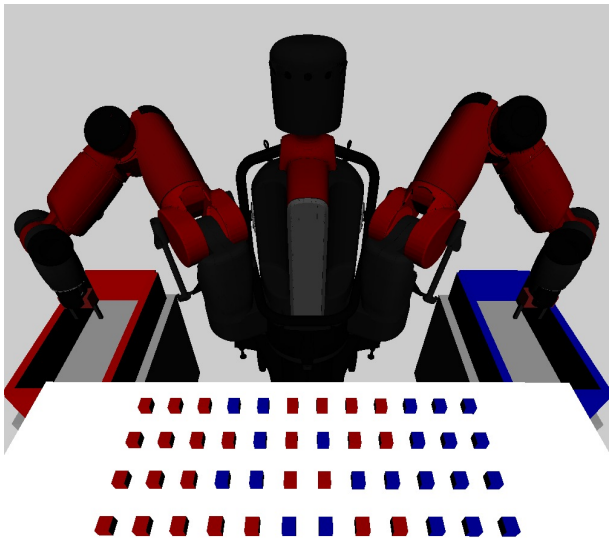


Figure 1: A dual-arm Baxter robot tasked with picking two types of objects from a tabletop and placing them into two distinct storage bins.

are separated in two groups, where a different arm needs to be used for each group. This can be useful in the case of different end-effectors, appropriate for different types of objects, and in general of separation tasks where two types of objects need to be moved to different target areas. Such tasks introduce a scheduling subproblem together with the dual-arm coordination challenge, where the ordering of the pick-and-place tasks affects the overall completion time and the conflicts arising between the two arms. For instance, the trajectory for a particular task could cause the arm to occupy the majority of the shared workspace, e.g., one of the arms reaching across the robot. Completing such a task could delay, or even obstruct, the other arm from carrying out its task depending on the placement of the objects. Consequently, scheduling the tasks for the two arms becomes a new aspect of dual-arm coordination in this context.

A naïve yet straightforward method for solving pick-and-place tasks with a humanoid robot corresponds to moving one arm at a time in a "round-robin" fashion, guaranteeing that only one of the arms operates in the workspace shared by both arms. This solution provides

fairness when different objects need to be grasped by different arms by alternating task completion. It is also simple to implement, since the coordination is minimal. The downside is that it doesn't take full advantage of the dual-arm capabilities of a humanoid robot relative to simultaneous arm motion.

Nevertheless, simultaneous arm motion corresponds to a high dimensional planning challenge, which for many popular dual-arm systems, such as a Baxter robot by Rethink Robotics, involves at least 14 DoFs. Given the complexity of motion planning, operating in the composite configuration space is computationally expensive, especially for high-quality paths. A practical alternative is to follow a decoupled methodology. For instance, paths for each arm can be computed independently and then coordination can be achieved by finding the velocity along the given paths that allows the arms to execute their task without collisions and deadlocks. This velocity tuning approach makes use of a representation known as a coordination diagram (O'Donnell and Lozano-Perez (1989); Siméon, Leroy, and Laumond (2002)). Although the coordination diagram approach is incomplete in the general case, it is computationally efficient to search, and in practice produces good quality paths for most realistic setups.

This paper first considers a "batch" scheduling approach, which is a straightforward extension of velocity tuning for multiple pick-and-place tasks. This involves a two-step process, where an ordering of the tasks for each arm is decided first. Given this ordering, the set of all coordination diagrams is implicitly searched so as to find the best coordination between the arms for that order. An alternative approach proposed here is an "incremental" scheduling method, which achieves online coordination of the two arms effectively. Tasks are assigned one at a time to each arm, and coordination diagrams are used to produce solution trajectories for a pairwise task assignment. The search method is defined so that a solution trajectory prioritizes the completion of a single task, thus "freeing" up one of the arms and allowing for a new task to be assigned.

To evaluate the effectiveness of both the "batch" and "incremental" methods, a series of simulated experiments on a Baxter robot were conducted. First, an experiment in a "tabletop" environment with a randomized distribution of objects in the scene showed that both methods produced solution trajectories that were nearly twice as fast compared to the "round robin" method, where only one arm moves at a time. This is close to the best that can be achieved by a simultaneous motion solution. The proposed "incremental" method, however, utilized much less computing resources - both processor time and memory - compared to the "batch" approach. If the coordination diagrams can be precomputed, the online computation time for both methods can be improved by a couple of orders of magnitude.

Another weakness of the "batch" scheduling approach beyond the high computational requirements is that the robot must compute the full schedule for both arms ahead of time. When a new task is added dynamically to the scene, this can result in additional cost for the "batch" approach. Simulations in a "shelf" environment, where objects appear dynamically in the scene, indicate that the "incremental" method produces even better results than the "batch" method in dynamic setups giving its capability to adapt to the scene.

Both the "batch" and "incremental" algorithms make a series of assumptions. First, the possible placements of the target objects are known during the precomputation step so as to allow the generation of grasping configurations and manipulation plans offline. Furthermore, the target objects are all geometrically similar, and are positioned in such a way so as to not prevent other objects from being grasped. Finally, the manipulator has access to perfect sensing, trajectory tracking, and grasping capabilities since the focus of the paper is on the combinatorial aspects of the problem.

2 Related Work

Manipulation Planning There are many ways to achieve manipulation planning, including tree sampling-based planners (Berenson, Srinivasa, and Kuffner (2012)), heuristic search (Cohen, Chitta, and Likhachev (2010)), constraint satisfaction (Lozano-Pérez and Kaelbling (2014)), and optimization approaches (King et al. (2013)). The underlying methodology for this work corresponds to searching solutions over a "manipulation graph" that contains "transit" and "transfer" paths (Alami, Siméon, and Laumond (1989); Siméon et al. (2004)). The graph itself can be built using sampling-based roadmaps (Kavraki et al. (1996)). One way to take advantage of multiple arms is through the use of handoffs (otherwise known as regrasps) (Vahrenkamp et al. (2009); Cohen, Phillips, and Likhachev (2014)), which involves passing an object from one arm to another. Grasp planning for multiple robots can be achieved with distributed constraint satisfiers (Panescu and Pascal (2014)).

A prototypical application of manipulation planning is in the area of bin-picking. Due to the structured nature of this domain, it is possible to take advantage of precomputed paths to speed up the online execution time. Such paths can be blended and further optimized (Ellekilde and Petersen (2013)) to account for noisy actuation and state uncertainty. The focus of such methodologies is on the optimality of trajectories generated for a single-arm and not on the effects of task ordering or dynamically appearing tasks. This work focuses on establishing a baseline framework which can effectively utilize dual-arm manipulators, and provides an adaptive online schedule which can be created in the event of dynamically appearing tasks.

Multi-Robot Coordination Multi-robot motion planning is a difficult problem due to the increased dimensionality. Coupled, complete approaches typically do not scale well with additional robots, despite the fact that there are methods which reduce the number of effective DOFs (Aronov et al. (1999)). A decoupled

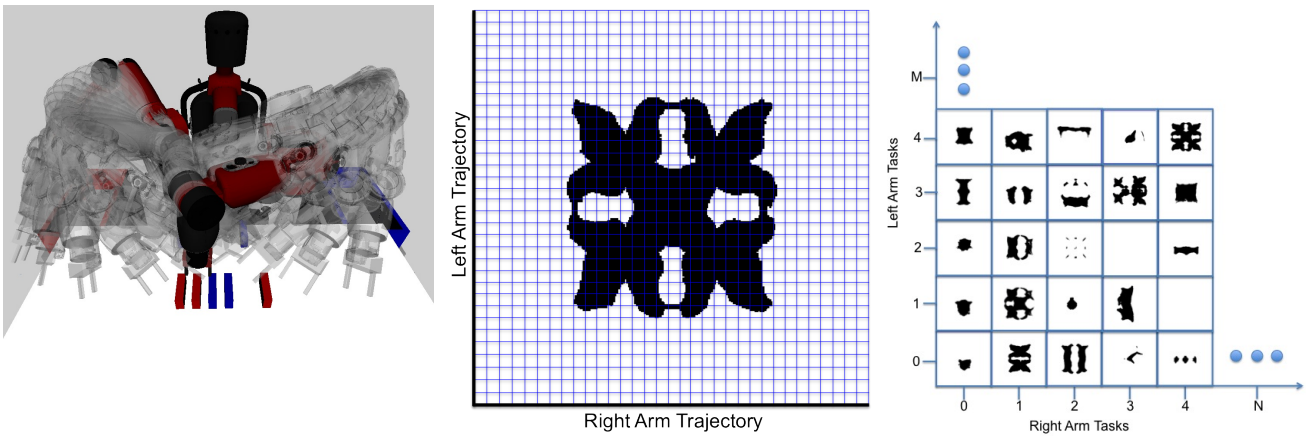


Figure 2: (Left) Given a pair of pick-and-place tasks for the two arms, the collisions between the two corresponding trajectories are computed. (Middle) The results of the pairwise collision checking define a *coordination diagram*. (Right) Building coordination diagrams for all pairs of trajectories defines a *coordination matrix*. Precomputing the entire matrix is useful in scheduling approaches as it is then possible to quickly query the relevant *coordination diagram* given a specific pair of designated tasks for each arm.

approach, which aims to best coordinate the individual paths of robots, can be used instead. Decoupling can be accomplished in several ways: by discretizing the common workspace and adopting a master-slave prioritization scheme (Zurawski and Phang (1992)), by using prespecified velocity profiles and varying the start times of the robots (Akella and Hutchinson (2002)), or by precomputing the collision volumes between the arms to form coordination diagrams (O'Donnell and Lozano-Perez (1989); Siméon, Leroy, and Laumond (2002)). This paper makes use of such coordination diagrams to achieve simultaneous, collision and deadlock-free movements of both arms.

Scheduling There is a very rich literature on scheduling challenges. One way to schedule a set of tasks is through the use of "batch" scheduling, which computes schedules over a set of assigned tasks (Maheswaran et al. (1999)). Genetic algorithms can be used to find the time-optimal batch schedules in the context of robotic manipulation (Xidias, Zacharia, and Aspragathos (2010)).

In contrast to batch scheduling, online scheduling methods are able to handle dynamic allocations of new tasks. Some work on online schedulers makes use of swarm optimization techniques (Xu, Hou, and Sun (2003); Yan et al. (2005); Chang, Chang, and Lin (2009)). While the proposed method here does not use such algorithms, it is also an online, incremental scheduling approach.

There has also been significant work towards addressing the issue of finding the optimal task assignment for multiple robots (Gerkey and Matarić (2004); Tang and Parker (2007); Zhang and Parker (2013)). This paper does not focus on the task assignment component of similar challenges. It instead focuses on the *ordering* of the tasks, given that they have already been assigned to a particular robot.

3 Problem Setup

Informally, a dual-arm manipulator is given a set of objects that can appear in known locations in the workspace to retrieve and place at specific goals. Each arm is assigned its own set of objects to be grasped and placed. The arms need to move the objects to their assigned goals as fast as possible while conflicts (i.e., collisions and deadlocks) need to be avoided. Thus, the problem involves finding both the order in which the objects are retrieved and placed as well as achieving collision-free and time efficient coordination. Consider a 3D workspace that contains obstacles and:

- **Two manipulators** M_{left} and M_{right} capable of picking and placing objects in the workspace.
- A set of **movable rigid-body objects** $\mathcal{O}_{left}$, where each object $o^i_{left} \in \mathcal{O}_{left}$ can acquire a **pose** in $SE(3)$ and can be grasped by the left arm M_{left}. Similarly for $\mathcal{O}_{right}$ and the right arm M_{right}.
- Two **goal regions**, which are the destinations for picked objects to be placed in. Each goal region corresponds to one of the two arms. Once an object is moved in a goal region, it is removed from the workspace.

An arm M_{arm} (i.e., either the left or the right one) is assigned an individual set of tasks $\mathcal{T}_{arm} = \{t^1_{arm}, \ldots, t^k_{arm}\}$, where a task $t^i_{arm} \in \mathcal{T}_{arm}$ corresponds to a single arm following a trajectory so as to pick a specific object $o^i_{arm} \in \mathcal{O}_{arm}$ from an initial pose and place it in the corresponding goal region.

Given an assignment of tasks $\{\mathcal{T}_{right}, \mathcal{T}_{left}\}$ to the two arms, a "manipulation coordination schedule" $\{S(\mathcal{T}_{right}), S(\mathcal{T}_{left})\}$ defines both the order of tasks with which each arm will execute the pick-and-place motions, as well as the corresponding *velocity profile* along the corresponding trajectories that the assigned arm will follow.

Overall, the problem is to find the solution schedules for both arms $S(T_{left})$ and $S(T_{right})$, which minimize *completion time* for all tasks, such that no *conflicts* arise between the arms. A *conflict* occurs both from collisions between the arms and deadlocks. A deadlock here corresponds to either arm reaching a configuration such that it is no longer possible for one of the arms (or, potentially both arms) to make progress towards completing assigned tasks.

This paper makes a few assumptions about the objects in the scene. The problem definition requires that each arm retrieves a different set of objects $\mathcal{O}_{left} \cap \mathcal{O}_{right} = \emptyset$. An example task would involve each arm equipped with a unique tool for manipulating objects. This assumption allows the results to focus on the effects of the selected schedule, rather than being an effect of different task assignments. Furthermore, each object's initial pose belongs in a set of poses that are known a priori to the robot. The rationale behind this is that the robot will be manipulating objects whose placements in the scene is predictable and in this way precomputation can be used to speed up the online solution times.

4 Method

Section 4.1 presents an overview of fundamental methodologies used throughout the rest of the paper. Then, Sections 4.2 and 4.3 describe the two different types of coordination modes, "batch" and "incremental" correspondingly, while Section 4.4 gives an overview of the precomputation utilized in this work so as to improve the running time of the proposed solutions.

4.1 Fundamentals

Given a pair of paths for the two arms, processing the collisions between all pairs of states along each trajectory can produce a *coordination diagram* (O'Donnell and Lozano-Perez (1989); Siméon, Leroy, and Laumond (2002)) as shown in Figure 2 (middle). Each axis corresponds to a path for one of the two arms, i.e., the range is [0,1], corresponding to where along each path each arm is located. Given the maximum velocity of each arm and the length of each path, each axis is discretized into intervals along which each arm is moving an equal distance.

A vertex in the resulting *coordination diagram* corresponds to a placement of both arms at configurations along the corresponding paths. The configurations which result in collisions between the arms are invalidated and marked as black in the diagrams. Searching such a diagram involves finding a collision-free path that starts from the bottom left vertex and reaches the top right vertex and does not go through black regions. Horizontal or vertical steps in the *coordination diagram* correspond to a single arm moving with maximum velocity, while a diagonal direction means that both arms are moving simultaneously with their maximum velocity. Extracting a solution from this search will produce a coupled trajectory that moves both arms to their respective goals without collisions.

4.2 Batch Coordination

This work considers the set of all coordination diagrams, which form a *coordination matrix* as shown in Figure 2 (right). The length of the trajectories does not have to be the same, in contrast to what is shown in the figure. This matrix can be constructed by computing and storing the *coordination diagrams* for all pairs of trajectories. Once this matrix is available, it allows for the framework to quickly check different orderings of tasks and find the corresponding solution trajectories for a particular schedule.

The *batch* mode coordination approach considers the orderings of tasks for both arms and then searches the resulting coordination matrix. In particular, the approach can be broken down into two main steps:

- `COMPUTE_TASK_ORDERING`(T_{left}, T_{right})
- `SOLVE_FULL_COORDINATION`$(S(T_{left}, S(T_{right}))$

The function `COMPUTE_TASK_ORDERING` takes as parameters the set of tasks for both arms. The tasks are then ordered to create the sequence of objects that each arm should aim to pickup. This step does not consider interactions between the arms. Instead, this ordering can be determined randomly, or could correspond to optimizing certain heuristic functions, such as minimizing execution time ignoring interactions between the arms. This ordering determines which pairs of coordination diagrams are tiled together and searched by the A*.

Once the ordering of the tasks has been fixed, the method calls `SOLVE_FULL_COORDINATION` to retrieve the solution trajectory for both arms. Given the order with which the tasks have been assigned, this method combines the corresponding pairwise coordination diagrams retrieved from the coordination matrix. Running an A* in this composed diagram, which will have a similar structure to the coordination matrix shown in Figure 2 (Right), will produce the coupled trajectory that moves both arms towards their goals.

For the A* search on the coordination matrix, a vertex in this space is valid as long as it does not occupy a cell in the diagram that is marked as a collision region (colored black in the figures). The cost of expanding a vertex is expressed in terms of time and corresponds to an additional "time step" for the arms to move. This cost is therefore equivalent in all directions of expansion. Since the tasks for both arms have already been ordered, the goal of the A* is to have each arm reach the end of their task schedule. Thus, the heuristic used in the search corresponds to the summation of each arm's distance to its goal. When both arms have finished all of their assigned tasks, the A* has successfully expanded the goal vertex and the solution trajectory is returned.

The benefit of using the *batch* coordination is that the A* returns the optimal path given the computed schedule. At the same time, there is a caveat. In particular,

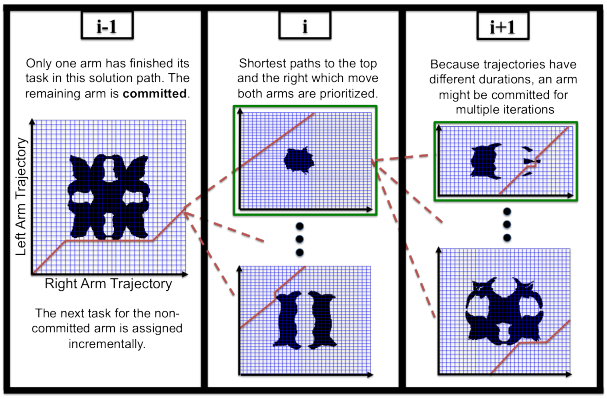


Figure 3: Initially, an arm might be committed to carrying out a pick-and-place task from a previous assignment. The non-committed arm, which has already finished its task, must be assigned a new task. This assignment is found by searching over all possible available tasks for the arm, and finding the assignment which produces the fastest solution time.

the schedule itself may not correspond to the optimal solution. Additionally, if any new tasks are introduced after the solution trajectory has been produced, there is no easy way to incorporate these tasks dynamically, i.e., until the robot has finished executing a trajectory. This would force an arm that has already completed its tasks to wait until the other arm is also finished before computing a new schedule.

4.3 Incremental Coordination

Rather than coordinating over the full coordination matrix, the *incremental* scheduling algorithm assigns a single task to each arm per iteration. An outline of an iteration for the approach is given in Algorithm 1. This method makes a distinction between a **committed** and a **non-committed** arm. If an arm has only partially completed its assigned task when a new iteration starts, it is **committed** to completing its current task, and is labeled as the *master* arm in the corresponding algorithmic. The remaining arm is labeled as the *slave* arm, which is the only arm that must be assigned a new task. This process continues as long as there are tasks remaining for either arm.

Line 1: The method `CHECK_PARTIAL_EXECUTION` determines if one of the arms is still in the middle of picking and placing an assigned object. This arm is set to be the master arm $M_{\texttt{MASTER}}$.

Lines 2-4: If both arms have finished their previously assigned tasks, a new master arm needs to be found. The method `ASSIGN_NEW_MASTER` searches the remaining tasks for both arms, and finds a suitable assignment. This can either be done by randomly selecting one of the arms to be the master, or it can be done by optimizing certain criteria. For this paper, random and *minimum conflict* assignments were tested. The *minimum conflict* assignment finds the pairwise task assignment which has the smallest amount of collision

Algorithm 1: Incremental Scheduling Algorithm

Data: $T_{\texttt{left}}$, $T_{\texttt{right}}$: left/right tasks
$\tau_{\texttt{left}}(t), \tau_{\texttt{right}}(t)$: current left/right trajectories
Result: Assigns a single task to each arm, returning the corresponding coupled trajectory to pick and place the two objects.

1 `CHECK_PARTIAL_EXECUTION`$(\tau_{\texttt{left}}(t), \tau_{\texttt{right}}(t), M_{\texttt{MASTER}})$
2 **if** $M_{\texttt{MASTER}} = \emptyset$ **then**
3 $\quad t_{\texttt{MASTER}} :=$ `ASSIGN_MASTER`$(T_{\texttt{left}}, T_{\texttt{right}}, M_{\texttt{MASTER}})$
4 $\quad$ `REMOVE_TASK`$(t_{\texttt{MASTER}}, T_{\texttt{MASTER}})$
5 $M_{\texttt{SLAVE}} := \{M_{\texttt{left}}, M_{\texttt{right}}\} - \{M_{\texttt{MASTER}}\}$
6 **if** $T_{\texttt{SLAVE}} \neq \emptyset$ **then**
7 $\quad t_{\texttt{SLAVE}} :=$ `ASSIGN_SLAVE`$(T_{\texttt{left}}, T_{\texttt{right}}, M_{\texttt{SLAVE}})$
8 $\quad$ `REMOVE_TASK`$(t_{\texttt{SLAVE}}, T_{\texttt{SLAVE}})$
9 $\quad$ `SOLVE_PARTIAL_COORDINATION`$(t_{\texttt{MASTER}}, t_{\texttt{SLAVE}})$

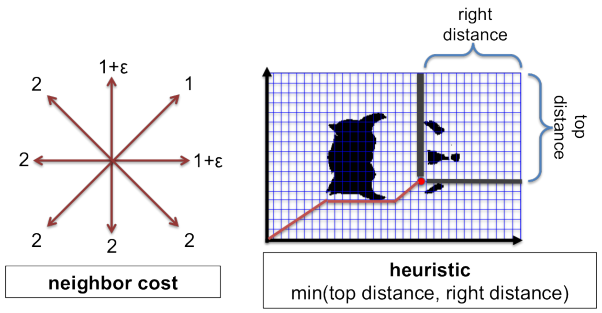


Figure 4: The neighbor cost and heuristic used by the A* in the incremental search is shown. Motions which involve one or both arms moving backwards along their trajectory are penalized during vertex expansion. The heuristic used here guides the search to allow one of the arms to finish as quickly possible.

area as determined by the *coordination diagrams*, and uses this as a heuristic to select the master arm.

Lines 5-6: The arm which is not the master arm is assigned to be the slave $M_{\texttt{SLAVE}}$. If the slave arm $M_{\texttt{SLAVE}}$ has any tasks remaining in its task set $T_{\texttt{SLAVE}}$, then an appropriate assignment must be made.

Lines 7-8: This assignment is computed by searching the remaining set of tasks for the slave arm and finding which pairwise task assignment for both arms produces the fastest solution time. An illustration of this is shown in Figure 3. The assigned task is then removed from the remaining set of tasks.

Line 9: The method `SOLVE_PARTIAL_COORDINATION` runs an A* on the coordination diagram corresponding to the pairwise task assignment of the arms. Since the *incremental* method does not make a complete ordering of the objects, and is therefore only operating on a single coordination diagram, it is no longer possible to guarantee that an optimal path will be returned by the A*. To promote trajectories that allow both arms to move simultaneously, the A* used in the *incremental*

approach was altered as shown in Figure 4. The heuristic used in the A* was the minimum distance for either arm to finish, while the goal condition was that either arm finished their currently assigned task.

The path returned by the A* is constructed such that the solution trajectory promotes the movement of both arms towards their goals, while penalizing any regressive movements. Single arm movements where the other arm is stationary is slightly penalized with a cost of $1+\epsilon$, with $\epsilon << 1$.

4.4 Precomputation

Computing pick and place motions for an individual object can be accomplished by building and querying a *manipulation graph* (Alami, Laumond, and Siméon (1997)). The graph contains roadmaps for transfer and transit paths, which are connected at grasping configurations. Querying the roadmap for a pick-and-place task corresponding to an individual object and arm will produce trajectories similar to the ones shown in Figure 2 (Left).

In many industrial settings, such as a factory or warehouse, it is reasonable to assume that a) the robot has knowledge of the static and immovable workspace, and b) movable objects can appear in poses out of a set of known ones.

To take advantage of such structured setups, the proposed framework performed the majority of the computationally expensive functions offline. This includes building the manipulation roadmap ahead of time for the known static scene and the known poses where objects can appear. In particular, the following data structures were precomputed and stored:

1. The *manipulation roadmap* corresponding to the static workspace.
2. The trajectories for each arm to pick and place objects that can appear in the predetermined poses, which are computed with the aid of the manipulation roadmap.
3. The *coordination diagrams* storing the collision checking information between pairs of trajectories for the two arms, as shown Figure 2 (middle).
4. The *coordination matrix* storing each computed *coordination diagram*, as shown Figure 2 (right).

5 Experimental Evaluation

Simulated experiments were performed, using a model of the dual-arm Baxter robot, where the robot needs to pick two sets of objects and place them into different bins, one on each side of the robot. Two separate problem scenarios, shown in Figure 5, were considered, where the objects are placed either on a tabletop or inside a shelving unit. The following methods were evaluated:

- **Round Robin (RR)**: This base case comparison point corresponds to only a single arm operating in the workspace at any given time. The other arm stays stationary at a pre-specified safe position. Each arm then alternates taking turns completing their tasks, until all tasks have been completed.
- **Batch (BA)**: This method corresponds to the *batch* approach described in Section 4.2. The ordering of the tasks is determined randomly at the beginning of each experiment.
- **Incremental (IN)**: This method corresponds to the incremental approach as described in Section 4.3. The initial assigned task is determined randomly.

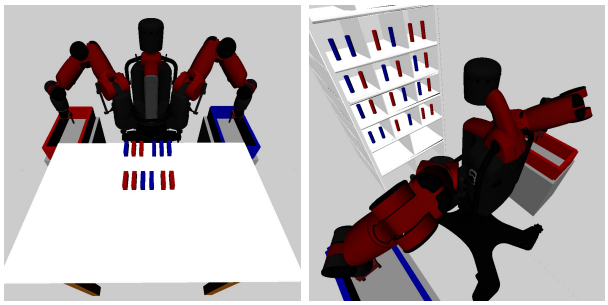


Figure 5: The objects colored red must be deposited in the red goal bin on the robot's right side. Similarly, the blue objects must be deposited in the blue bin on the left side. (Left) Table scenario with an example arrangement of objects. (Right) Shelf scenario with an example arrangement of objects.

The methods are then evaluated in terms of:

1) efficiency: measured as average online computation time and memory requirements, as well as
2) solution quality: measured as average duration of execution for solution trajectories.

Each simulated experiment was run on a single computer with an Intel Xeon E5-4650 2.8 gHz processor and 8GB of RAM.

Tabletop Results: The "tabletop" scenario, as shown in Figure 5 (Left), had 50 different randomized task assignments, i.e., potential object placements and corresponding arm assignments. A comparison of the average solution and computation time for all methods is shown in Figure 6. Since the schedule for the *batch* method was randomized, and since the initial task selection for the *incremental* method is also randomized, the numbers presented are averaged over a total of 1000 runs. The *round robin* method indicates both an upper and a lower limit for the performance of the methods considered in this paper. In particular, the best possible execution time that *batch* and *incremental* can return corresponds to half the duration of the round robin solution. And they cannot possibly exceed the time of the round robin solution.

The results show that in terms of execution time, both the batch and the incremental approach achieve

	4 objects		8 objects		12 objects	
	Batch	Incremental	Batch	Incremental	Batch	Incremental
# collision checks	655,306	409,566	2,544,669	1,431,376	5,686,158	3,080,021
computation time (s) without precomputation	114.2	71.2	443	261.65	1,000.6	532.2
computation time (s) with precomputation	0.08	0.05	0.35	0.18	0.7	0.25
execution time (s)	21.5	20.9	40.5	39.75	59.4	59

Table 1: Cost of Online Collision Checking in the Table Scenario

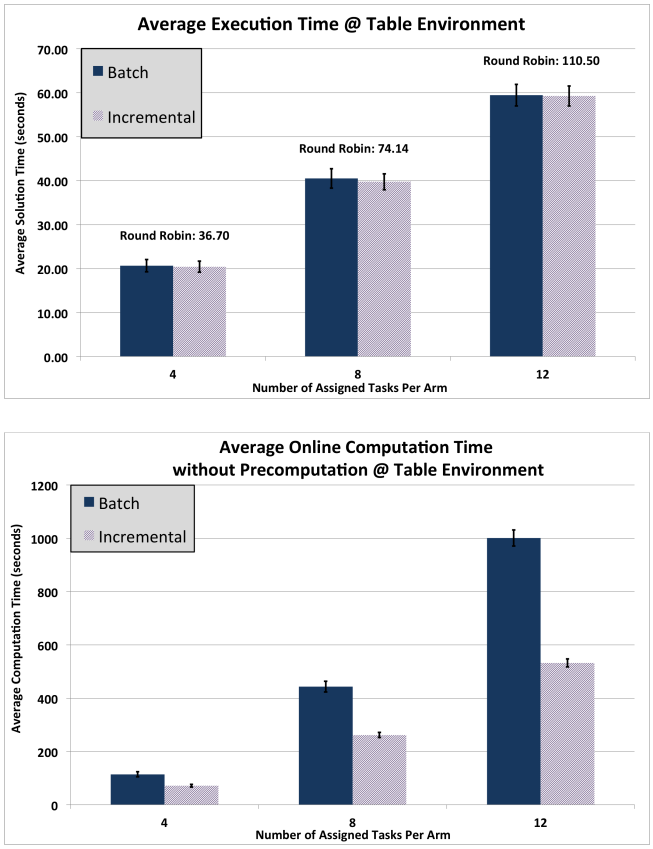


Figure 6: Comparison of average execution and computation times for 4, 8, and 12 tasks assignments per arm in the Table environment. Each bar is averaged over 50 different object placements, with the randomized approaches averaged over 1000 randomized orders per object placement. The 95% confidence interval for the randomized approaches is displayed.

times close to half of that of the round robin schedule. In terms of computation time, the *incremental* approach is faster than the *batch* one. In the results of Figure 6, the coordination diagrams of pairs of arm trajectories were not assumed precomputed. Thus, it is necessary to spend online computation time to identify the collision-free solution in the coordination space. Nevertheless, it is possible to achieve orders of magnitude faster solutions if the coordination diagrams are precomputed ahead of time. This can be done when the possible locations of objects which can appear are known ahead of time.

Table 1 provides a more detailed comparison of the computation cost of the two alternatives. An interesting statistic is how often the *batch* and *incremental* approaches queried the coordination diagram. In the case that a trajectory needs to be verified online, these calls correspond to collision checks, which are computationally expensive. The *batch* method makes significantly more collision checks than the *incremental* method. When the coordination diagrams are not precomputed, this is directly reflected in the online computation time of the two approaches. With precomputation, both methods can be computed orders of magnitude faster. In this case, the incremental method remains the faster out of the two.

Obviously preprocessing helps but at the cost of significant memory requirements in the case of the batch approach. In the above results, the number of objects considered was up to 12. It is interesting to evaluate what happens when a significantly higher number of objects is involved, which can happen in some industries where many small objects (i.e., electronic components) may need to be picked up by a robot. Figure 1 shows an example placement of a larger number of objects, while Table 2 shows the resulting RAM usage by all three methods for increasing numbers of objects in this scene. The *batch* method showed an enormous difference in the amount of memory used, and at 64 objects it was no longer able to solve the problem given a maximum of 6 GB of RAM. The memory requirements of the *batch* approach arise from the need to store the frontier nodes of the A* search over a significantly larger space.

	30 objects	**50 objects**	**64 objects**
RR	30.5 MB	41 MB	45 MB
BA	1.51 GB	4.21 GB	5.92+ GB
IN	43.7 MB	46 MB	47.1 MB

Table 2: RAM usage in the Table scenario for increasing numbers of objects and the three methods (RR: round robin, BA: batch, IN: incremental).

Shelf Results: The second setup, shown in Figure 5 (Right), considers an environment, which involves tight spaces. Rather than randomizing the placement of objects in the scene, the objects were instead placed in such a way so as to maximize the interactions between the two arms. To accomplish this, two objects were placed in the same bin side by side, and each was assigned to a different arm. A comparison of the average solution and computation time for all methods is shown in Figure 7. In this experiment, the online computation time considers the case where precomputed coordination diagrams are available. Examining these results shows that the *incremental* method provides improvements in both solution and computation time relative to the *batch* method and for precomputed coordination diagrams it is possible to achieve very fast solutions.

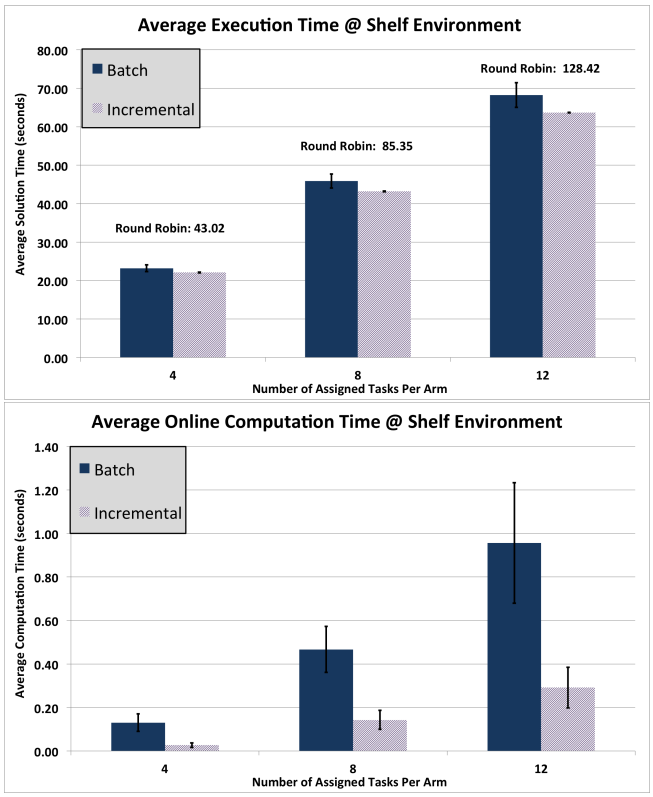


Figure 7: Comparison of average execution and computation times for 4, 8, and 12 tasks assignments per arm in the Shelf environment. To promote interaction between the arms, two objects are placed in the same bin. The numbers are averaged over 1000 different orderings. The 95% confidence interval for the approaches is displayed.

Dynamically Appearing Objects: Another important aspect is the performance of the methods when objects are allowed to appear dynamically during the execution of a schedule. To analyze this, the Shelf scenario was utilized so that once an object was placed at its goal bin, another object assigned to the same arm would appear in one of the empty shelves. The *batch* method was altered in the following way so as to support dynamically appearing objects: every time a new object appeared in the scene, both arms would finish placing their currently assigned objects and the schedule for the arms would be recomputed for the new set of objects. The *incremental* method did not have to be altered as it directly accommodates the presence of new objects. The results shown in Figure 8 measure total time, which takes into account both execution time and the amount of time each method takes to compute a schedule once a new object appears. From these results, the *incremental* method shows better performance in total time compared to the *batch* method. This is due to the fact that the *incremental* method computes a schedule on the fly, and can directly accommodate the dynamically appearing tasks.

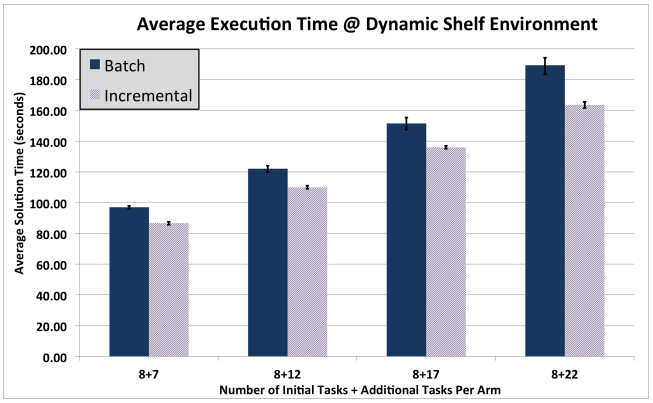


Figure 8: Comparison of average execution times in the Shelf scenario with increasing total number of dynamic objects appearing in the scene. The 95% confidence interval for each approach is displayed.

Precomputation Statistics: In the "tabletop" scenario, a roadmap with 4,600 vertices and 103,466 edges was generated. In the "shelf" scenario, a roadmap with 5,800 vertices and 104,138 edges was generated. The amount of time spent computing the trajectories and coordination diagrams is given in Table 3. Since the *round robin* method did not utilize the coordination graphs, this precomputation cost can be seen as the trade-off for achieving two orders of magnitude faster online computation times for *batch* and *incremental* scheduling.

	computation time (seconds)
12 Shelf Trajectories	15.7
24 Table Trajectories	23.4
"shelf" coord. graphs	2,176.7
"table" coord. graphs	5,024.2

Table 3: Trajectory and Coordination Matrix Computation Time

For picking tasks that appear infrequently, or only once, precomputation is not warranted and instead the summation of computation and execution time provides the best measure of effort for solving the tasks. When tasks appear repeatedly, whether in a static scene or in a dynamically appearing scene, precomputation can be taken advantage of to reduce the amount of time spent online. Once precomputation is utilized, the online computation time becomes significantly smaller than the execution time, and has a smaller impact on the overall time spent solving the task schedule. Nevertheless, in dynamic scenes, the batch method will need to recompute the schedule, which significantly increases its online execution time, whereas the incremental method can directly incorporate the newly added task into its existing search.

6 Discussion

This work examines the coordination and scheduling of dual-arm manipulators in the context of pick-and-place tasks. Three different methods for accomplishing this were evaluated. First, a naíve round-robin approach was examined that defines bounds for achieving coordination between the arms by ensuring that only one arm moves at any given time. Then, two approaches for allowing simultaneous arm movements are considered. A *batch* method, which searches over all pairs of *coordination diagrams* given an ordering of pick-and-place tasks. And an *incremental* approach, which dynamically assigns tasks to each arm, producing solutions of similar quality to the *batch* method for static scenes and improved performance in the presence of dynamically appearing objects. The *incremental* approach exhibits significant benefits, in terms of memory requirements as well as computation time, especially when precomputation cannot be taken advantage of.

In the current work none of the objects occludes any other object in the scene. Thus, the task ordering can be freely defined by the scheduler so as to improve performance. An interesting direction is to extend the method for solving problems where objects occlude one another, where some form of object rearrangement will also be required. This would define an implicit, potentially partial, ordering of the tasks, such that certain tasks must be completed before other tasks would even be feasible. Similarly, it is interesting to consider challenges where the arms need to perform handoffs in order to solve the problem (Cohen, Phillips, and Likhachev (2014)), when objects and their target placements cannot be reached by both arms.

Another interesting direction is scheduling in the context of multiple humanoid robots with overlapping tasks. In this case, an object might be graspable by more than one robot. Solving such a challenge would involve a task assignment sub-challenge for multiple robots and reasoning over a *generalized coordination diagram* (Siméon, Leroy, and Laumond (2002)). It should still be possible to apply the methodologies presented here to obtain an incremental schedule in this context.

Accounting for uncertainty due to sensing and noisy actuation are essential steps to consider when implementing the method on real robots. In this context, it is important to consider in terms of schedules that provide robust solutions. One potential way to adopt the coordination diagrams to account for noisy actuation is to artificially expand the collision region, thereby producing more conservative trajectories.

References

Akella, S., and Hutchinson, S. 2002. Coordinating the motions of multiple robots with specified trajectories. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, 624–631. IEEE.

Alami, R.; Laumond, J.-P.; and Siméon, T. 1997. Two Manipulation Planning Algorithms. In Laumond, J.-P., and Overmars, M., eds., *Algorithms for Robotic Motion and Manipulation*. Wellesley, MA: A. K. Peters.

Alami, R.; Siméon, T.; and Laumond, J.-P. 1989. A Geometrical Approach to Planning Manipulation Tasks. In *Proc. of International Symposium on Robotics Research*, 113–119.

Aronov, B.; de Berg, M.; van den Stappen, A. F.; Svestka, P.; and Vleugels, J. 1999. Motion Planning for Multiple Robots. *Discrete and Computational Geometry* 22(4):505–525.

Berenson, D.; Srinivasa, S. S.; and Kuffner, J. J. 2012. Task Space Regions: A Framework for Pose-Constrained Manipulation Planning. *The International Journal of Robotics Research (IJRR)* 30(12):1435–1460.

Chang, R.-S.; Chang, J.-S.; and Lin, P.-S. 2009. An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems* 25(1):20–27.

Cohen, J. B.; Chitta, S.; and Likhachev, M. 2010. Search-based Planning for Manipulation with Motion Primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Cohen, B.; Phillips, M.; and Likhachev, M. 2014. Planning single-arm manipulations with n-arm robots. In *Proceedings of Robotics: Science and Systems*.

Ellekilde, L.-P., and Petersen, H. G. 2013. Motion planning efficient trajectories for industrial bin-picking. *The International Journal of Robotics Research* 32(9-10):991–1004.

Gerkey, B. P., and Matarić, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23(9):939–954.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

King, J.; Klingensmith, M.; Dellin, C.; Dogar, M.; Velagapudi, P.; Pollard, N.; and Srinivasa, S. S. 2013. Pregrasp Manipulation as Trajectory Optimization. In *Robotics: Science and Systems (RSS)*.

Lozano-Pérez, T., and Kaelbling, L. P. 2014. A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS*

2014), 2014 IEEE/RSJ International Conference on, 3684–3691. IEEE.

Maheswaran, M.; Ali, S.; Siegal, H.; Hensgen, D.; and Freund, R. F. 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, 30–44. IEEE.

O'Donnell, P. A., and Lozano-Perez, T. 1989. Deadlock-free and collision-free coordination of two robot manipulators. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, 484–489. IEEE.

Panescu, D., and Pascal, C. 2014. A constraint satisfaction approach for planning of multi-robot systems. In *System Theory, Control and Computing (ICSTCC), 2014 18th International Conference*, 157–162. IEEE.

Siméon, T.; Laumond, J.-P.; Cortés, J.; and Sahbani, A. 2004. Manipulation Planning with Probabilistic Roadmaps. *International Journal of Robotics Research (IJRR)* (23).

Siméon, T.; Leroy, S.; and Laumond, J.-P. 2002. Path coordination for multiple mobile robots: A resolution-complete algorithm. *Robotics and Automation, IEEE Transactions on* 18(1):42–49.

Tang, F., and Parker, L. E. 2007. A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3351–3358. IEEE.

Vahrenkamp, N.; Berenson, D.; Asfour, T.; Kuffner, J.; and Dillmann, R. 2009. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2464–2470. IEEE.

Xidias, E.; Zacharia, P. T.; and Aspragathos, N. 2010. Time-optimal task scheduling for two robotic manipulators operating in a three-dimensional environment. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 224(7):845–855.

Xu, Z.; Hou, X.; and Sun, J. 2003. Ant algorithm-based task scheduling in grid computing. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, volume 2, 1107–1110. IEEE.

Yan, H.; Shen, X.-Q.; Li, X.; and Wu, M.-H. 2005. An improved ant algorithm for job scheduling in grid computing. In *Proc. of 2005 Int. Conf. on Machine Learning and Cybernetics, 2005.*, volume 5, 2957–2961. IEEE.

Zhang, Y., and Parker, L. E. 2013. Multi-robot task scheduling. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2992–2998. IEEE.

Zurawski, R., and Phang, S. 1992. Path planning for robot arms operating in a common workspace. In *Industrial Electronics, Control, Instrumentation, and Automation, 1992. Power Electronics and Motion Control., Proceedings of the 1992 International Conference on*, 618–623. IEEE.

A Risk-Based Framework for Incorporating Navigation Uncertainty Into Exploration Strategies

Jason Gregory
University of Maryland, College Park, MD, USA
jgregory@umd.edu

Jonathan Fink
United States Army Research Laboratory, MD, USA
jonathan.r.fink3.civ@mail.mil

John Rogers
United States Army Research Laboratory, MD, USA
john.g.rogers59.civ@mail.mil

Satyandra K. Gupta
University of Maryland, College Park, MD, USA
skgupta@umd.edu

Abstract

We present a novel framework for incorporating navigation uncertainty into exploration strategies, specifically for autonomous ground robots operating in environments where there exists a nonzero probability of complete failure. In real-world applications, autonomous navigation is negatively influenced by uncontrollable factors, such as terrain complexity and dynamic obstacles, which require the robot to spend additional time or energy during navigation that is not accounted for in conventional path planners. Using a Monte-Carlo simulation-based scheme, we preserve the stochasticity of the environment and allow for any type of model for navigation cost to be used. We show that the expected rate at which an unknown environment is discovered can be increased in a series of simulations.

1 Introduction

A fundamental area of research in the realm of robotics is autonomous exploration; that is, the iterative process of a robot observing it's surroundings using onboard sensors, selecting from a set of candidate locations, planning a feasible route, and autonomously navigating to the desired location. In general, the objective of exploration is to discover some specific information as efficiently as possible. This information that a robot, or team of robots, searches for may be mission-specific and require prioritization. For example, the goal may be to determine topological structure in the minimal amount of time (Burgard et al. 2000), to locate a radio-frequency source (Twigg et al. 2012), or to search a disaster site (Visser and Slamet 2008), (Gregory et al. 2015). The exploration problem is inherently difficult, however, due to the lack of *a-priori* information. Given a known map of the environment, planning an optimal route under some constraints is a well-understood problem (LaValle 2006). Without knowledge of the operating environment, the principal question for efficient exploration is: where should the robot move to next?

One of the fundamental approaches to exploration is *frontier-based* (Yamauchi 1997), (Yamauchi, Schultz, and Adams 1998). In this approach, it is assumed a robot has some mapping capability in order to build an *occupancy grid*, which represents an environment by discretizing space into cells that are assigned one of three values: *free*, *occupied*, or *unknown* based on sensor readings, e.g., 2D or 3D laser data. A *frontier* is defined as a region of free cells that are adjacent to unknown cells. By navigating to a frontier, the robot would expect to obtain measurements for the unknown cells and gain some new information. At any given instance during exploration, we have a collection of frontiers to choose from and the efficiency of our exploration solution depends on our ability to chose our next location intelligently, as it relates to our specific exploration objective. A critical distinction to note is that both the number and locations of frontiers are continuously changing as the robot moves through the environment and receives new sensor data. This presents a unique challenge for route planning in that the candidate locations are dynamically changing across each iteration of exploration.

A number of efforts have been made to improve the performance of frontier detection (Keidar and Kaminka 2014) as well as the selection of frontiers for maximizing efficiency. This includes multi-agent exploration strategies, which consider the allocation of frontiers to a team of robots. The authors of (Burgard et al. 2000) evaluate a collaborative strategy, in simulation and real-world experiments, for assigning target positions to a team of robots with the intention of reducing the overall exploration time. Similarly, the work in (Faigl, Simonin, and Charpillet 2014) characterizes the performance of the greedy assignment, Traveling Salesman Problem (TSP) assignment, Hungarian assignment, Iterative assignment, and MinPos assignment for allocating frontiers amongst a team of robots.

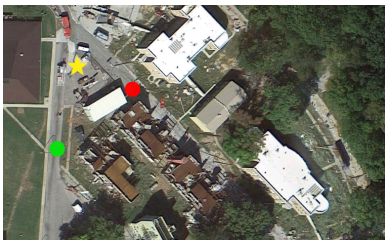
As we start to consider applications of autonomous exploration in the natural world, we introduce stochastic events that negatively impact the navigation of a planned path. For example, imperfect sensor data, faulty sensor hardware, complex terrain, and dynamic obstacles are all sources of uncertainty that ultimately cause navigation to differ from the output of a path planner. In other words, a path planner can provide a lower bound on the traversal cost to a frontier in the form of an optimal route, but this typically ignores factors that complicate, and in some cases prevent, real-world navigation. To address these issues, research efforts focus on incorporating map and localization uncertainty by de-

termining salient positions and routes in the environment as it relates to localization error. The system described in (Stachniss, Hahnel, and Burgard 2004) performs active loop closures, i.e., traversal of paths to previously visited locations, in a frontier-based exploration strategy to reduce pose estimation uncertainty. Other approaches use decision-theoretic frameworks to consider the expected information that could be gained for each frontier in addition to both map and localization uncertainty (Stachniss, Grisetti, and Burgard 2005), (Carrillo et al. 2015). Similarly, the work presented in (Makarenko et al. 2002) offers an integrated exploration strategy based on a utility function that considers information gain, localization quality, and navigation cost. Our work differs from these methods in that we assume that stochasticity plays a central role in the calculation of navigation cost and, therefore, should be considered in the decision making process. We do not feel that this approach should necessarily replace existing techniques that consider other types of uncertainty, but rather could be integrated along side methods in an effort to develop more robust systems. More specifically, we present a general framework that accounts for navigation-based uncertainty and then characterize properties of this approach in the specific use case of autonomous exploration. We conjecture that deriving an accurate utility function, and the associated assumptions, for a risk-based decision making framework as it relates to navigation-based uncertainty is difficult because of the mission-specific preferences that may or may not be generalizable. We note, however, that if the mapping between navigation uncertainty costs and utility values were known, our framework could be converted into an approach that follows classical decision theory principles (Sage 1992), (French 1986), (Raiffa 1974).

Previous work that is most relevant to this framework include (Murphy 2010), where the authors present a Gaussian-process-based approach for generating probabilistic cost maps using overhead imagery so that uncertainty in terrain classification and spatial variations in terrain costs are considered by path planners. Their approach is evaluated on aerial imagery of suburban- and rural-type environments and assumes access to both labelled training data for predefined terrain classes as well as cost data for navigating each terrain class.

In this work, specifically, we apply our framework to the problem of selecting navigable locations during exploration as it relates to unmanned ground vehicles completing search and reconnaissance missions in complex environments, e.g., Army-relevant or post-disaster settings, where the terrain is more difficult to model than conventionally-studied scenarios. Unique to this problem space, we note that there exists a nonzero probability of terminal failure in some, or all, regions of the operational environment - a factor that has not been considered in previous works. The success of an exploratory mission is, therefore, correlated to the robot's ability to take into consideration the uncertainty associated with navigating paths due to the discerpancy between planned and observed performance. Our framework can be applied to any exploration strategy that generates and chooses from a set of candidate locations; however, to present and evaluate our approach we consider incorporating navigation uncertainty into a frontier-based exploration strategy, as this type of exploration has been well studied and extensively used in literature.

Without loss of generality, we can consider exploration as a graph search, where frontiers are represented by nodes and the navigation cost between frontiers are edges. It follows, autonomous exploration is closely related to the Orienteering Problem (OP) (Chao, Golden, and Wasil 1996), (Vansteenwegen, Souffriau, and Oudheusden 2011) and Vehicle Routing Problem (VRP) (Laporte and Martello 1992), with the underlying difference that the complete problem involving the entire environment is not known at any given instance during exploration. The operations research community has studied the OP and VRP extensively, both with and without stochasticity (Gendreau, Laporte, and Séguin 1996), (Li, Tian, and Leung 2010), (Tang and Miller-Hooks 2005); however, the findings for these types of problems have not yet been applied to frontier-based exploration in the context of navigation uncertainty.



(a)

(b)

Figure 1: A satellite image of a hypothetical environment where the yellow star represents the robot's location and the red and green disks represent two waypoint options for exploration (a). The actual environment that is observed during mission execution if the robot navigated to the red disk (b).

To motivate this work, we begin with a canonical example that illustrates the importance of incorporating uncertainty with respect to navigation costs. In Figure 1(a), an overhead satellite image of a hypotehtical disaster site is shown, which could potentially be used as *a-priori* information. Here, the yellow star represents the starting location of a robot and the red and green disks are two potential waypoints to choose from. The distance, Euclidean or planned path length, from the robot's location to the red disk is shorter than from the robot's location to the green disk so it would be selected in the traditional greedy-based exploration strategy. However, Figure 1(b) shows the potential environment that the robot

might encounter if it successfully navigated to the waypoint indicated by the red disk. As in the real-world, there exists obstacles that are not captured in a satellite image, which pose considerable challenges for autonomous navigation. In this case, both the task of navigating to this waypoint, and any additional exploration thereafter, may be complicated or impossible because this navigation uncertainty was not taken into consideration. While the robot would be required to navigate to both locations in an exploration mission, we desire a strategy that collects information quickly and safely, so that a greater understanding of the environment can be acquired before a potential navigation failure occurs.

Although this level of *a-priori* information is not always known, having some awareness of navigation uncertainty through out the environment could directly impact the planning process. If, for example, the robot has performed exploration in a similar environment previously, it can use this initial model for navigation uncertainty in the new environment. Furthermore, the robot may be able to learn an improved navigation model for the environment in an online fashion by comparing it's planned paths with it's observed navigation performance (Papadakis 2013), (Silver, Bagnell, and Stentz 2010),(Martin, Murphy, and Peter 2013), (Howard et al. 2007). For this work, we are less concerned with determining what precipitates navigation uncertainty or how to detect it, but rather how to effectively incorporate this uncertainty into existing exploration strategies as well as characterizing its effect on the performance of collecting information about the environment.

The contribution of this work is a general framework for incorporating navigation uncertainty to a novel application, namely autonomous exploration in extreme, real-world environments. Additionally, we show that incorporating navigation uncertainty leads to an increased, expected rate of information acquired over the course of exploration due to an improved decision-making process that actively avoids operation-ending navigation failures.

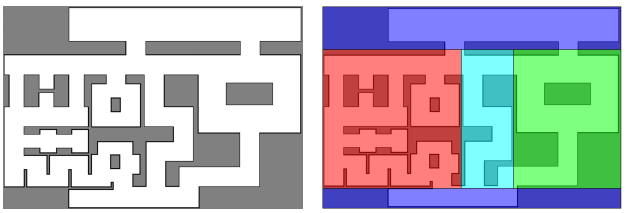
2 Methodology

We present a framework by which navigation uncertainty is considered when choosing locations to explore. The specific objective for autonomous exploration that we are interested in is maximizing the amount of the environment discovered while minimizing operating time.

Incident Modeling

While navigating in a real-world environment, a robot can encounter any number of internal or external factors that cause it to spend additional time or energy to traverse a planned path. We refer to these factors as *incidents* and the associated cost incurred when handling this type of event as the *incident cost*.

In general, the existence of incidents is environment dependent, e.g., terrain complexity, and the magnitude of incident costs vary through out an environment. To model this, we assume there exists a set, Ξ, of *incident polygons*, each of which represents a bounded region with a specific distribution of incident costs per meter. This formulation is illustrated in Figure 2 where five polygons are used to denote four regions that cover an environment and have different incident costs. Note, our framework does not limit the number of incident polygons over the environment; in fact, one could define an incident polygon for every grid cell in the occupancy grid, if desired.



(a) (b)

Figure 2: An arbitrary environment that a robot could be tasked to explore (a) and that same environment with five regions, contained within the colored polygons, that have distinct incident cost distributions (b).

For each polygon $\xi_i \in \Xi$, there exists some random variable, $\mathbf{D}_i$, that represents the incident cost incurred per meter driven, where $i = 1...N$ and N is the number of polygons defined for the environment. Our framework makes no assumption on the type of probability distribution used for each random variable, and allows for distribution functions to differ across the set of polygons. It is important to note that any of distribution functions could have a point mass at infinite cost, which corresponds to the possibility of catastrophical failure while navigating through an incident polygon.

Updating Navigation Costs

Given a well-defined distribution, or set of distributions, it may be possible to generate statistics on the required navigation costs and use traditional methods for accounting for uncertainty, such as relying on the mean and variance of the distribution. However, modeling natural phenomena in the field of robotics often times requires less-structured distributions, or distributions where there exists no expected value, e.g., a model with a nonzero probability of infinite cost. To overcome this issue and allow for any experimentally-derived distribution to be used, we propose a method by which we sample the distributions multiple times to preserve the stochastic nature of incidents.

Our process for updating navigation costs begins by computing initial plans to each candidate location, g_i in the set of all of the candidates, Φ, generated by the desired exploration strategy. In the case of the greedy, frontier-based exploration strategy, for example, we require plans from the robot's current pose to every frontier's pose. Next, we perform a Monte-Carlo simulation where each trial is a new instantiation of the problem generated using different incident costs. In other words, we update the initial navigation cost by sampling the probability distributions that represent the potentially-incurred incident costs, for each path in each simulation iteration. Mathematiically, let ω_a and ω_b be the poses of a segment of the planned path such that the segment

is completely contained within an incident polygon, then we update the navigation cost using

$$C(\omega_a, \omega_b) = C_o(\omega_a, \omega_b) + \mathbf{D}_i * l_p \tag{1}$$

where $C(\omega_a, \omega_b)$ is the updated cost between poses ω_a and ω_b, $C_o(\omega_a, \omega_b)$ is the initially-computed cost, $\mathbf{D}_i$ is the random value representing the incidence cost generated using the probability distribution from the corresponding incident polygon, and l_p is the length of the path between ω_a and ω_b. To update the navigation cost for the entire planned path, Γ, we require each segement generate a finite cost, i.e., a successfully traversed path, at which point we accumulate the sampled navigation costs. In other words,

$$\Omega(\omega_r, \omega_f) = \sum_{\gamma \in \Gamma} C(\gamma_a, \gamma_b) \tag{2}$$

where $\Omega(\omega_r, \omega_f)$ is the final, updated navigation cost for the planned path, Γ, from pose ω_r to pose ω_f, and γ_a and γ_b are the poses of the two endpoints of the path segment γ, which is completely contained within an incident polygon. Note, if the random variable for any of these path segments takes the value ∞, then the entire path is deemed a failure. An example of this calculation is illustrated in Figure 3.

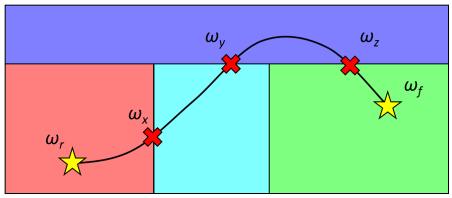


Figure 3: A planned path, Γ, from pose ω_r to pose ω_f has four segments, contained within four incident polygons. The updated navigation cost for the entire path is $\Omega(\omega_r, \omega_f) = C(\omega_r, \omega_x) + C(\omega_x, \omega_y) + C(\omega_y, \omega_z) + C(\omega_z, \omega_f)$, assuming navigation for each path segment is successful.

All of the updated navigation costs for each candidate location are stored in a single trial of the Monte-Carlo simulation and the entire process repeats for $M = |\Phi|^3$ iterations so that we probabilistically estimate the cost of navigating to each candidate.

Incorporating Stochasticity Into Exploration Strategies

After the Monte-Carlo simulation has been executed, we have a distribution of navigation costs for each candidate and we are required to choose a single solution, π, to execute. We seek the solution that has a high probability of successful navigation, in an effort to manage the threat of fatal operations, and is easier to traverse, i.e., minimal incurred incidence costs, relative to the other solutions. To achieve this, we approximate a probability density function for each candidate location, g_i, by normalizing the histogram of navigation costs computed in the associated Monte-Carlo trials by the total number of iterations. Then, using these density functions, we solve

$$P(\Omega(\omega_r, \omega_f) \leq c_\tau) \geq \alpha \tag{3}$$

for c_τ, where $\alpha \in [0, 1]$ is the user-defined confidence level and $P(\Omega(\omega_r, \omega_f) \leq c_\tau)$ is equivalently the approximated cumulative distribution function representing the path costs between poses ω_r and ω_f. Analytically, we are atleast α-% confident that navigation to solution g_i will cost less than or equal to c_τ. We note that a candidate location is infeasible if the sampled probability of infinite cost is greater than $1 - \alpha$. In the event that two candidates have equal cost, i.e., $c_{\tau_i} = c_{\tau_j}$ for $i \neq j$, we select the candidate that has the lower probability of infinite cost to minimize the likelihood of failure. It follows that if $\alpha = 0.5$, we would solve for the median cost for each candidate. Our final solution chosen for execution is then

$$\pi = \underset{g_i \in \Phi}{\arg\min} \quad c_\tau \tag{4}$$

By defining our framework in this way, we effectively compare the risk of failure and incurred incidence costs for every candidate location.

3 Experiments Using the Greedy Approach and Accounting for Uncertainty

In general, the greedy approach consists of selecting the frontier with the maximum utility cost in hopes of an immediate payoff. This has historically been the most common strategy for frontier-based exploration due to it's simplistic implementation and minimal computational requirement. For a navigation-based greedy approach this equates to selecting the frontier that has the shortest distance to the robot's current location. As a proof of concept, we evaluate the greedy approach using our framework to represent the anticipated effect of uncertainty.

To incorporate navigation uncertainty into the greedy approach we seek the frontier that has the lowest cost that includes stochasticity using some model for incident costs. That is,

$$\underset{g_i \in \Phi}{\arg\min} \quad \Omega(\omega_r, \omega_{g_i}) \tag{5}$$

where ω_r is the robot's pose and ω_{g_i} is the pose of a single frontier in the set of all frontiers, Φ.

We performed a series of simulations using the 59m $\times$ 40m environment shown in Figure 2 to characterize the greedy approach, both with and without incorporating navigation uncertainty. We choose to define each random variable representing incidence cost using a lognormal distribution, i.e., $\mathbf{D}_i \sim \ln\mathcal{N}(\mu, \sigma^2)$ where $\mu \in \mathbb{R}$ and $\sigma > 0$. The selection of this distribution is based on the existing work in the field of transportation engineering that suggests this is an accurate model for traffic delays, which we assert are analogous to stochastic-events that negatively effect planned navigation for autonomous robots (Garib, Radwan, and Al-Deek 1997), (Sullivan 1997). Additionally, to account for the possibility of navigation having infinite cost, our implementation also assigns a Bernoulli random variable, $\mathbf{B}_i$, to each incident polygon, ξ_i; in doing so, we effectively add a delta function at ∞ such that there exists probability, p, that navigation is successful, and probability, $q = 1 - p$, that navigation fails. It follows, we only sample from the random

variable $\mathbf{D}_i$, if our sample of $\mathbf{B}_i$ corresponds to successful navigation.

We take a minimal approach to effectively simulate the phenomena affecting the application of route-planning algorithms to real-world robotics scenarios, namely the evolution of the observed map and uncertainty in control, which affects the time to perform autonomous navigation. We assume a two-dimensional environment and simulate a LiDAR sensor with a 240° field of view, 0.5° angular resolution, and maximum range of 40m. The robot is restricted to a maximum velocity of 1.0 m/s and the differential-drive kinematics are given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \mathcal{N}(0, \sigma_\omega(x, y)) \end{bmatrix} \tag{6}$$

where the noise-parameter $\sigma_\omega(x, y)$ is known *a-priori* and approximately induces a log-normal distortion when performing closed-loop path-following control. Table 1 shows the specific parameters for all of the random variables, $\mathbf{B}_i$ and $\mathbf{D}_i$, as well as the noise parameters, for each incident polygon used in our simulations.

	$\mathbf{B}_i$	$\mathbf{D}_i$		
Polygon Color	p	μ	σ^2	$\sigma_\omega(x, y)$
Blue	0.98	-0.60	0.25	0.00
Green	0.95	0.00	0.10	0.00
Cyan	0.90	0.70	0.05	0.96
Red	0.80	1.10	0.03	1.44

Table 1: The simulation parameters for each incident polygon that were used in our simulations.

For each iteration of exploration, we analyze the occupancy grid to form a set of frontiers. To ensure frontiers are navigable, the cells within the robot's inscribed radius around the centroid of the convex hull for every polygon are verified to contain only free cells. We use ARA* to plan kinematically feasible paths to every frontier and the cost for a planned path is equal to the path length. We force our planner to heavily favor planning through known space by assigning near-lethal costs to unknown space so that costs to frontiers are more accurately estimated. After a frontier is selected, the robot assigns a waypoint and autonomously navigates to the desired location. In an effort to improve efficiency, we use the immediate replanning method (Faigl, Simonin, and Charpillet 2014). That is, if during navigation, the selected frontier vanishes, i.e., there no longer exists any unknown cells contained within the polygon that defines the frontier, the robot initiates the next iteration of exploration. Regardless of achieving a frontier or determining that the frontier has vanished, the robot always halts navigation and plans from a stationary position so that the occupancy grid does not change during the planning process. Finally, in the event our framework determines that all frontiers are infeasible due to the probability of failure, we choose to terminate exploration as a safety mechanism.

As a baseline, we executed 15 trials using the greedy strategy without considering uncertainty, referred to as *naïve greedy*, which is representative of the current method by which many existing systems use. We then performed 15 trials using our framework to evaluate the performance of an exploration strategy where navigation uncertainty is considered, referred to as *cautiously greedy*. We performed simulations using $\alpha = 0.5$ and $\alpha = 0.8$ to capture the effect of accepting more or less risk when selecting frontiers in the presence of uncertainty. In all cases, tests were limited to 10 minutes to emphasize the desire for obtaining information as quickly as possible.

To evaluate the rate at which information about the environment is acquired during exploration, we evaluate each method based on the amount of uncertainty, i.e., *entropy*, that exists in the robot's map of the environment. Given an occupancy grid, G, entropy is computed using

$$E(G) = -\sum_{c \in G} P(c) \log P(c) + (1 - P(c)) \log (1 - P(c)) \tag{7}$$

where $P(c)$ represents the probability of occupancy of grid cell c. This measure of entropy represents the amount of the environment that the robot has knowledge of at a particular instance in time.

4 Results

To illustrate each strategy's rate of acquiring new information, we normalize the measure of entropy by the maximum amount of entropy, i.e., the size of the environment if all cells were unknown, shown in Figure 4 as the proportion of the environment still unexplored.

For each trial we compute the area under the normalized entropy curve, referred to as the *exploration burden*, to quantify the overall performance of exploration. To do this, we fit polynomials to the entropy curves, in 10 second segments to minimize fitting error, and numerically integrate over the interval. The final exploration burden is the accumulation of all integrated segments for the duration of the entire mission, and measured in natural unit of information (nat)-seconds. Intuitively, one would desire an exploration strategy with a lower exploration burden as this indicates that unknown space is discovered quicker. The 25%-quantile, median, and 75%-quantile exploration burdens for each strategy are shown in Figure 5.

We note that the median exploration burden for the naïve greedy strategy is 441.19 nat-seconds while the exploration burden for the cauitiously greedy strategy for $\alpha = 0.5$ and $\alpha = 0.8$ are 384.73 and 392.54 nat-seconds, respectively - a decrease in exploration burden by 14.68% and 12.39%. From this we conclude that, by incorporating uncertainty into the frontier selection process, we effectively improve exploration by increasing the expected rate of information gain. Using the statistics of the exploration burdens, we performed unpaired T-tests to vaidate the statistical significance of our trials, which are presented in Table 2. Each p-value is within the acceptable limit of 5.0×10^{-2} and rejects the null hypothesis that states the sample of exploration burdens are drawn from the same population and have equal distribution shapes.

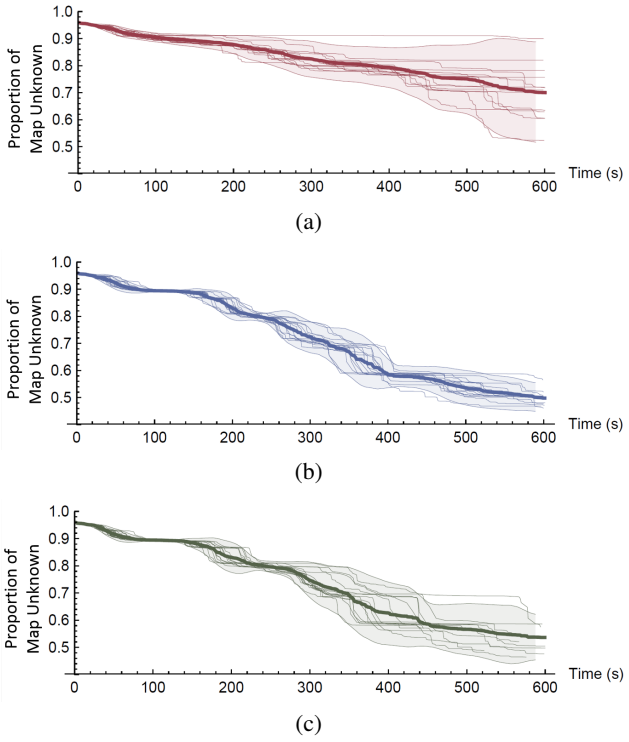


Figure 4: The normalized entropy over the duration of simulated exploration for the *naïve greedy* approach (a), the *cautiously greedy* approach using a confidence level of $\alpha = 0.5$ (b), and with $\alpha = 0.8$ (c). Thin lines indicate individual trials, thick lines are time-varying averages over the trials, and the shaded regions represent the standard deviation.

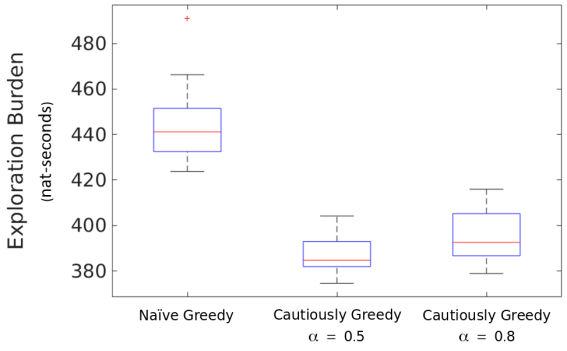


Figure 5: Analysis of *exploration burden* as a measure for performance.

Greedy Strategy 1	*Greedy Strategy 2*	*p-value*
Naïve	*Cautiously* $\alpha = 0.5$	2.27×10^{-12}
Naïve	*Cautiously* $\alpha = 0.8$	6.63×10^{-10}
Cautiously $\alpha = 0.5$	*Cautiously* $\alpha = 0.8$	3.23×10^{-2}

Table 2: The unpaired T-test p-values as an evaluation of statistical significance.

5 Discussion and Future Work

We present and evaluate a general framework for incorporating navigation uncertainty in the goal location selection of autonomous exploration, specifically for real-world environments where there exists a possibility of operation-ending failure. Through a series of simulations, presented in Figure 4, we show that the average rate of obtaining new knowledge of the environment during the exploration process can be improved by including information regarding navigation uncertainty and failure.

In the simulations of the naïve greedy exploration strategy, specifically for our constructed environment, the robot consistently enters the regions with the highest noise and probability of failure because these regions are the closest to the initial location. As a result, the robot's ability to plan and navigate through the environment is negatively impacted because uncontrollable navigation noise forces the robot to accidentally drive into, or unnecessarily close to, obstacles. Additionally, the robot accumulates non-negligible map degradation, which in some cases prohibits progress because the robot perceives it's location, or the goal location, to be in occupied space. Ultimately, these factors cause the naïve greedy approach to produce the worst mean rate of entropy reduction. A screenshot of a selected trial is shown in Figure 6(a) to collectively represent the map generated using the naïve greedy approach.

The evaluation of the cautiously greedy approach with a confidence threshold of $\alpha = 0.5$ reveals that the robot intelligently selects goal locations in areas of less noise and prolongs entering regions of dangerous navigation so that it is able to continue making progress. This, in turn, results in the highest expected rate of entropy reduction and lowest average exploration burden of the tested strategies. Furthermore, when the confidence threshold is increased to $\alpha = 0.8$, we notice the robot effectively chooses to navigate through even fewer areas of risky navigation and improves map quality. It is important to note, however, that because the confidence threshold is relatively high, the robot sometimes terminates exploration prematurely because no candidate locations are deemed safe enough for navigation, resulting in zero entropy reduction. This is represented by the lesser mean entropy reduction rate, larger entropy reduction standard deviation, and larger average exploration burden, with respect to $\alpha = 0.5$, as shown in Figures 4 and 5. An example illustrating the effect of the confidence threshold on the exploration and mapping properties of the cautiously greedy approach can be seen in Figures 6(b) and 6(c).

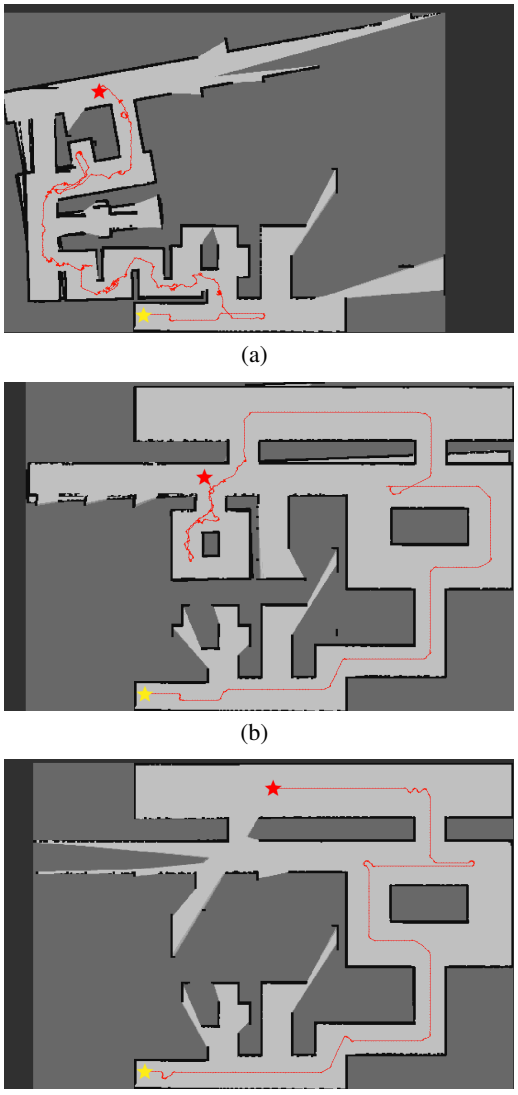


Figure 6: Selected screenshots of the final maps for the *naïve greedy* approach (a), *cautiously greedy* approach using a confidence level of $\alpha = 0.5$ (b), and with $\alpha = 0.8$ (c). Here, the robot started exploring from the location indicated by a yellow star and stopped exploring at the location represented by a red star.

In our simulations, we found that the primary factors that directly effected the rate of exploration are the structure of the environment and the associated navigation noise because these determine the level of difficulty for autonomous navigation. It should be noted that we constructed a testing environment such that the regions with the most difficult structure and highest navigation noise are located closest to the robot's initial location. This was chosen to capture a known drawback of the naïve greedy approach, which is that it chooses goals based on distance calculations and could realistically encounter scenarios where this has a long-term, detrimental effect on exploration. We do not feel that this environment necessarily represents the large majority of operational environments; however, it still provides valuable insight to the characterization of navigation uncertainty as it relates to autonomous exploration. It is our belief that because the cautiously greedy approach accounts for navigation uncertainty, it will always choose a goal location that is considered at least as "safe" or "safer", in terms of navigation uncertainty and failure, than the location chosen by the naïve greedy strategy - which, in turn, could improve the long-term performance of autonomous exploration.

In the future, we would like to more thoroughly characterize the effect of navigation uncertainty, and the associated confidence threshold α, in both the greedy approach as well as other existing exploration strategies. This includes executing more simulations with randomized initial locations and randomly-generated environments. Also, mathematically proving the number of samples required for the Monte-Carlo simulation to probabilistically estimate navigation costs could help optimize the computational requirements of our framework. A more detailed analysis of the distribution of solutions produced from the Monte-Carlo simulation might offer some valuable information in terms of contingency planning. Finally, an integrated system that 1) learns the incident cost distribution models during autonomous navigation and/or 2) incorporates empirically-derived distributions for probabilistic path planning could improve the performance of our framework when applied to single or multi-robot exploration.

References

Burgard, W.; Moors, M.; Fox, D.; Simmons, R.; and Thrun, S. 2000. Collaborative Multi-Robot Exploration. *International Conference on Robotics and Automation (ICRA)* 1.

Carrillo, H.; Dames, P.; Kumar, V.; and Castellanos, A. 2015. Autonomous Robotic Exploration Using Occupancy Grid Maps and Graph SLAM Based on Shannon and R enyi Entropy. *2015 IEEE International Conference on Robotics and Automation (ICRA) Washington State Convention Center Seattle, Washington, May 26-30, 2015*.

Chao, I. M.; Golden, B. L.; and Wasil, E. a. 1996. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* 88(3):475–489.

Faigl, J.; Simonin, O.; and Charpillet, F. 2014. Comparison of Task-Allocation Algorithms in Frontier-Based Multi-Robot Exploration. *Multi-Agent Systems* 101–110.

French, S. 1986. Decision theory: an introduction to the mathematics of rationality. Halsted Press.

Garib, a.; Radwan, a. E.; and Al-Deek, H. 1997. Estimating Magnitude and Duration of Incident Delays. *Journal of Transportation Engineering* 123(4):459–466.

Gendreau, M.; Laporte, G.; and Séguin, R. 1996. Stochastic vehicle routing. *European Journal of Operational Research* 88(1):3–12.

Gregory, J.; Fink, J.; Stump, E.; Twigg, J.; Rogers, J.; Baran, D.; Fung, N.; and Young, S. 2015. Application of Multi-Robot Systems to Disaster-Relief Scenarios with Limited Communication. *Field and Service Robotics* 1–14.

Howard, A.; Turmon, M.; Matthies, L.; Tang, B.; Angelova, A.; and Mjolsness, E. 2007. Towards learned traversability for robot navigation: From underfoot to the far field. *Journal of Field Robotics* 23:1005–1017.

Keidar, M., and Kaminka, G. A. 2014. Efficient frontier detection for robot exploration. *The International Journal of Robotics Research* 33(2):215–236.

Laporte, G., and Martello, S. 1992. The selective travelling salesman problem. *Discrete Applied Mathematics* 26(2-3):193–207.

LaValle, S. M. 2006. Planning Algorithms. *Methods* 2006:842.

Li, X.; Tian, P.; and Leung, S. C. H. 2010. Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm. *International Journal of Production Economics* 125(1):137–145.

Makarenko, A.; Williams, S.; Bourgault, F.; and Durrant-Whyte, H. 2002. An experiment in integrated exploration. *IEEE/RSJ International Conference on Intelligent Robots and System* 1:534–539.

Martin, S.; Murphy, L.; and Peter, C. 2013. Building Large Scale Traversability Maps Using Vehicle Experience. In *13th International Symposium on Experimental Robotics*, Springer Tracts in Advanced Robotics, 891–905.

Murphy, E. 2010. *Planning and Exploring Under Uncertainty*. Ph.D. Dissertation, University of Oxford.

Papadakis, P. 2013. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence* 1373–1385.

Raiffa, H. 1974. Applied Statistical Decision Theory.

Sage, A. 1992. Systems Engineering. John Wiley & Sons.

Silver, D.; Bagnell, A.; and Stentz, A. 2010. Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain. *International Journal of Robotics Research*.

Stachniss, C.; Grisetti, G.; and Burgard, W. 2005. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. *Robotics: Science and Systems*.

Stachniss, C.; Hahnel, D.; and Burgard, W. 2004. Exploration with active loop-closing for FastSLAM. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)* 2:1505–1510.

Sullivan, E. 1997. New Model for Predicting Freeway Incidents and Incident Delays. *Journal of Transportation Engineering* 123(4):267–275.

Tang, H., and Miller-Hooks, E. 2005. Algorithms for a stochastic selective travelling salesperson problem. *Journal of the Operational Research Society* 56(4):439–452.

Twigg, J. N.; Fink, J. R.; Yu, P. L.; and Sadler, B. M. 2012. RSS gradient-assisted frontier exploration and radio source localization. *2012 IEEE International Conference on Robotics and Automation* 889–895.

Vansteenwegen, P.; Souffriau, W.; and Oudheusden, D. V. 2011. The orienteering problem: A survey. *European Journal of Operational Research* 209(1):1–10.

Visser, A., and Slamet, B. A. 2008. Balancing the information gain against the movement cost for multi-robot frontier exploration. *European Robotics Symposium 2008* 43–52.

Yamauchi, B.; Schultz, A.; and Adams, W. 1998. Mobile Robot Exploration and Map-Building with Continuous Localization. *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)* 4(May):3715–3720.

Yamauchi, B. 1997. A frontier-based approach for autonomous exploration. *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'* 146–151.

ACTORSIM: A toolkit for studying Goal Reasoning, Planning, and Acting

Mark Roberts[1] **Ron Alford**[2] **Vikas Shivashankar**[3] **Michael Leece**[4] **Shubham Gupta**[5] **David W. Aha**[6]

[1]NRC Postdoctoral Fellow; Naval Research Laboratory (Code 5514); Washington, DC | mark.roberts.ctr@nrl.navy.mil
[2]MITRE; McLean, VA | ralford@mitre.org
[3]Knexus Research Corp.; Springfield, VA | vikas.shivashakar@knexusresearch.com
[4]Dept. of Computer Science; Univ. of California Santa Cruz; Santa Cruz, CA | mleece@soe.ucsc.edu
[5]Thomas Jefferson High School for Science and Technology; Alexandria, VA
[6]Naval Research Laboratory (Code 5514); Washington, DC | david.aha@nrl.navy.mil

Abstract

Goal reasoning is maturing as a field, but lacks a unified model and common implementation that researchers can build from. This paper presents three contributions. First, it formalizes goal reasoning with crisp semantics by extending a recent formalism that blends goal-network and task-network planning. Second, it describes an open source package, called the Actor Simulator (ACTORSIM), that has been used in activity planning for robotics and partially implements the semantics of the formal model. Third, it presents a new study applying ACTORSIM and goal reasoning to the game of Minecraft. The study examines the role of learning from experience to improve goal selection and reveals that simple mechanisms for capturing experience are adequate for the problem we study.

1 Introduction[1]

Goals are a unifying structure for designing and studying intelligent systems, including robotic systems, which may perform goal reasoning to manage long-term behavior, anticipate the future, select among priorities, commit to action, generate expectations, assess tradeoffs, resolve the impact of notable events, or learn from experience. If a goal is an objective a robot wishes to achieve or maintain, then planning is deliberating on what action(s) best accomplish the objective, acting is deliberating on how to perform each action of a plan, and goal reasoning is deciding what goal(s) to pursue given trade-offs in dynamic, possibly adversarial, environments. Thus, goal reasoning is a critical component for enabling more responsive and capable autonomy.

Researchers have examined a variety of goal reasoning topics (Vattam et al., 2013), including studies on Goal-Driven Autonomy (Klenk et al., 2013; Munoz-Avila et al., 2010; Dannenhauer et al., 2015), goal formulation (Wilson, Molineaux, and Aha 2013), goal motivators (Munoz-Avila, Wilson, and Aha 2015), goal recognition (Vattam and Aha 2015), goal prioritization (Young and Hawes 2012), explanation generation (Molineaux and Aha 2014), and agent-oriented programming (Thangarajah et al., 2010; Harland et al., 2014; De Giacomo et al., 2016). Some studies have proposed models for specific aspects of goal reasoning, namely planning and acting (Thangarajah et al., 2010, Harland et al., 2014, Cox et al. 2016), while one study by Roberts et al. (2015b) adds goal formulation and goal selection to complete the entire lifecycle but lacks semantics . Four workshops[2] provide a more complete survey of the area. As this area of research matures, it can be enriched by more comprehensive studies using publicly available systems that implement a clear semantics.

We describe the Actor Simulator, ACTORSIM, which is a general platform for conducting studies of goal reasoning in simulated environments. Although goal reasoning has strong ties to planning, acting, and robotics, it is an understudied area of research partly because there exists no publicly available language, definition, and generic implementation. We draw inspiration from the literature in planning, where 15 years of international competitions has blossomed into a research ecosystem of nearly 100 open source planning systems and hundreds of planning benchmarks in a standardized language. Despite this progress, many planning systems focus on a single actor operating within a static environment and with static objectives. ACTORSIM can facilitate studies in which these constraints are relaxed.

ACTORSIM implements the goal reasoning model of Roberts et al. (Roberts et al. 2015a). We view goal reasoning as leveraging the work of Ghallab, Nau, and Traverso (Ghallab, Nau, and Traverso 2014; Nau, Ghallab, and Traverso 2015), wherein deliberation takes place on (1) descriptive models of *what* to accomplish (e.g., a goal to be in room_a might be decomposed into the subgoals to be near a door for room_a, achieve $\text{open}(\text{door}_{to-A})$ if applicable, and then perform the task $\text{enter}(\text{room}_a)$), and (2) operational models of *how* to perform a task (e.g., opening the door is a sequence of subtasks such as: determine the type of door handle, grasp the handle, and push (or pull) the door). Thus, the deliberation in such systems resembles a task network with goals interspersed. Similarly, we argue that goal reasoning is a kind of hybrid task-goal planning that supports the larger cycle of Planning and Acting by allowing an actor to determine and prioritize its goals dynamically.

Contributions. Our objective in this paper is to foster studies of goal reasoning by presenting:

[1]A more recent version of this paper will appear in the Proceedings of the 4th Annual Conference on Advances in Cognitive Systems; see Roberts et al. (to appear) for details.

[2]The latest workshop is described at http://makro.ink/ijcai2016grw/

A **formal model** of goal reasoning and its semantics. This model extends previous work by Roberts et al. (2015) and builds on a hybrid model of planning called Goal-Task Network (GTN) planning (Anonymous, under review), which blends Hierarchical Task Network (HTN) planning with Hierarchical Goal Network (HGN) planning. This hybrid model allows us to seamlessly intermix task and goal networks with state-based planning concerns, which is critical in a system that performs goal reasoning and deliberation (Ghallab, Nau, and Traverso 2014).

An **open source platform**, called ACTORSIM[3], that implements this formal model. ACTORSIM's initial design spurred from work on robotic applications to Foreign Disaster Relief operations (Roberts et al. 2015b) and has since been extended to several other domains. We briefly summarize how ACTORSIM has supported these studies and our future plans for integration with more sophisticated simulators such as ROS or Gazebo.

The **application** of ACTORSIM to Minecraft, with preliminary results showing that (1) learning from structured experience to select subgoals improves behavior for a simple navigation task, (2) gathering evidence showing expert knowledge is useful but not essential for effective decision making in this task, and (3) costly random knowledge gathering, as typically performed in unsupervised learning, is best used only to broaden structured knowledge. Our results complement existing studies on how to gather and learn from experience and demonstrate that goal networks can overcome some limitations of action selection approaches.

The paper proceeds in two parts. The first part establishes a model of goal reasoning. We briefly describe our notation (Section 2) and a formalism called *Goal-Task Network* (GTN) planning (Section 2.1) that we will apply to define the semantics of goal reasoning. Section 3 formalizes the model and semantics that ACTORSIM implements.

The second part provides a snapshot of ACTORSIM (Section 4) and then describes an implementation of ACTORSIM for the game of Minecraft (Section 5). We present a pilot study that highlights the benefits of applying learning to goal selection in a simple maze problem (Section 6). Finally, we describe other ACTORSIM connectors we have developed (Section 7), related work (Section 8), and conclude with future work objectives.

2 Preliminaries

Ghallab et al. (2014) and Nau et al. (2015) point out that planning and acting systems must often deliberate about both descriptive and operational models. Descriptive models detail *what* actions would accomplish a goal (e.g., "plans"), while operational models detail *how* to accomplish it; (e.g., "tasks" or "procedures"). Thus, a hybrid model that combines state-based planning and hierarchical planning is needed.

Let $\mathcal{L}$ be a propositional language. We partition $\mathcal{L}$ into *external state* $s \subset \mathcal{L}_{external}$ relating to an agent's belief about the world, where the set of all external states is $S = 2^{\mathcal{L}_{external}}$, and *internal state* $z \subset \mathcal{L}_{internal}$ relating to internal decisions and processes of the agent, where the set of all internal states is $Z = 2^{\mathcal{L}_{internal}}$. $\mathcal{L} = \mathcal{L}_{external} \cup \mathcal{L}_{internal}$, where $\mathcal{L}_{external} \cap \mathcal{L}_{internal} = \emptyset$.

Let $\mathcal{T}$ be a set of task names represented as propositional symbols not appearing in $\mathcal{L}$ (i.e., $\mathcal{L} \cap \mathcal{T} = \emptyset$), and let O and C be a partition of $\mathcal{T}$ ($O \cup C = \mathcal{T}$, $O \cap C = \emptyset$). O denotes the set of *primitive tasks* that can be executed directly, while C represents compound or *non-primitive* tasks that need to be recursively decomposed into primitive tasks before they can be executed.

We augment the model of online planning and execution by Nau (2007) with a goal reasoning loop (cf. Figure 1). The world is modeled as a state transition system $\Sigma = (S, A, E, \delta)$ where S is a set of states that represent facts in the world as above, $A = (a_1, a_2, ...)$ are the allowed actions of the Controller, $E = (e_1, e_2, ..)$ is a set of exogenous events, and $\delta : S \times (A \cup E) \rightarrow S$ is a state transition function. Let s_{init} denote the initial state and S_g denote the set of allowed goal states. The *classical* planning problem is stated: Given $\Sigma = (S, A, \delta)$, s_{init} and S_g, find a sequence of actions $\langle a_1, a_2, .., a_k \rangle$ such that $s_1 \in \delta(s_{init}, a_1), s_2 \in \delta(s_1, a_2), .., s_k \in \delta(s_{k-1}, a_k)$ and $s_k \in S_g$. Thus, the actor seeks a set of transitions from s_{init} to one of a set of goal states $S_g \subset S$.

We call the goal reasoner in Figure 1 the GRPROCESS and assume the Controller only executes one action x_j at a time, returning PROGRESS$_j$ to update progress, SUCCESS$_j$ for completion, and FAIL$_j$ for failure. A goal memory stores goals that transition through the goal lifecycle, which we will define more fully in §3.2. We simplify the discussion by considering only achievement goals even though the model and ACTORSIM can represent maintenance goals.

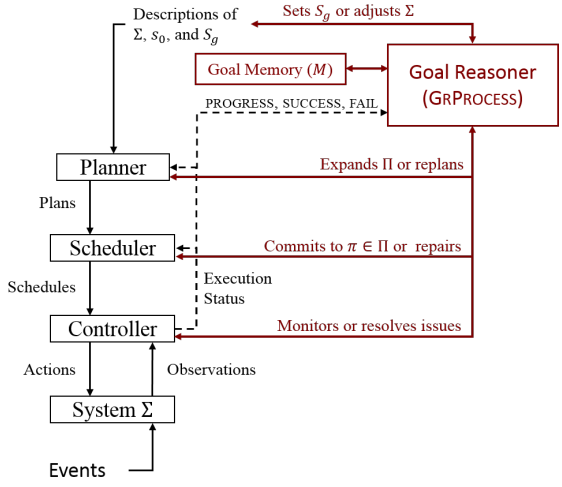


Figure 1: Relating goal reasoning with online planning, where the GRPROCESS works with a goal memory to monitor and modify the goals or planning model of the system. The goal memory stores goal nodes that transition according to the goal lifecycle described later.

[3]Available at http://makro.ink/actorsim

2.1 Goal-Task Network (GTN) Planning

Alford et al (2016) model both hierarchical task and goal planning in a single framework called Goal-Task Network (GTN) planning, which was partly inspired by conversations with Ghallab, Nau, & Traverso following their position paper on Planning and Acting (Ghallab, Nau, and Traverso 2014). GTN planning augments the notation of (Geier and Bercher 2011) with goal decomposition from HGN planning (Shivashankar et al., 2012) and SHOP2-style method preconditions (Nau et al., 2003). While HTN planning is over partially-ordered multisets of *task names* from $\mathcal{T}$ and HGN planning is over totally-ordered subgoals in $\mathcal{L}$, GTN elegantly models both. The rest of this section summarizes Alford et al. (2016) as it relates to the goal reasoning model we introduce.

A *goal-task network* is a tuple $(I, \prec, \alpha)$ where I is a set of instance symbols that are placeholders for task names and goals, $\prec \subset I \times I$ is a partial order on I, and $\alpha : I \rightarrow \mathcal{L} \cup \mathcal{T}$ maps each instance symbol to a goal or task name. An instance symbol i is *unconstrained* if no symbols are constrained to be before it ($\forall_{i' \in I}\ i' \not\prec i$) and *last* if no symbols are constrained to be after it ($\forall_{i' \in I}\ i' \prec i$). A symbol i is a *task* if $\alpha(i) \in \mathcal{T}$ and is a *goal* if $\alpha(i) \in \mathcal{L}$; recall that $\mathcal{L}$ and $\mathcal{T}$ are disjoint.

Methods We distinguish the methods of a GTN by the kind of symbol they decompose. A *task method* m_t is a tuple (n, χ, gtn) where $n \in C$ is a non-primitive task name, $\chi \in \mathcal{L}$ is the precondition of m_t, and gtn is a goal-task network over $\mathcal{L}$ and $\mathcal{T}$. m_t is *relevant* to a task i in $(I, \prec, \alpha)$ if $n = \alpha(i)$. m_t is a specific decomposition of a task n into a partially-ordered set of subtasks and subgoals, and there may be many such methods. A *goal method* m_g, similarly, is a tuple (g, χ, gtn) where $g, \chi \in \mathcal{L}$ are the goal and precondition of m_g and gtn is a goal-task network. m_g is *relevant* to a subgoal i in $(I, \prec, \alpha)$ if at least one literal in the negation-normal form (NNF) of g matches a literal in the NNF of $\alpha(i)$ (i.e., accomplishing g ensures that part of $\alpha(i)$ is true). By convention, $gtn = (I, \prec, \alpha)$ has a last instance symbol $i \in I$ with $\alpha(i) = g$ to ensure that m_g accomplishes its own goal.

Operators An *operator* o is a tuple (n, χ, e) where $n \in O$ is a primitive task name (assumed unique to o), χ is a propositional formula in $\mathcal{L}$ called o's precondition (or $prec(o)$), and e is a set of literals from $\mathcal{L}$ called o's effects. We refer to the set of positive literals in e as $add(o)$ and the negated literals as $del(o)$. An operator is *relevant* to primitive task i_t if $n = \alpha(i_t)$ and to a subgoal i_g if the effects of o contain a matching literal from the NNF of $\alpha(i_g)$. A set of operators $\mathcal{O}$ forms a transition (partial) function $\gamma : 2^{\mathcal{L}} \times \mathcal{O} \rightarrow 2^{\mathcal{L}}$ as follows: $\gamma(s, o)$ is defined iff $s \models prec(o)$ (the precondition of o holds in s), and $\gamma(s, o) = (s \setminus del(o)) \cup add(o)$.

GTN Nodes and Progression Operations Let $N = (s, gtn)$ be a *gtn-node* where s is a state and gtn is a goal-task network. A *progression* transitions a node N by applying one of four progression operations: operator application (A), task decomposition (D_t) goal decomposition (D_g), or release (G). Let $P = \{A, D_t, D_g, G\}$ represent any of these four operations (when the context is clear we write D for either D_t or D_g). Then $N \rightarrow_P N'$ denotes a single progression operation from N to N', while $N \rightarrow_P^* N''$ denotes a progression sequence from N to N''. Here we only summarize these operations, although their semantics are defined by Alford et al. (2016).

Operator application, $(s, gtn) \xrightarrow{i,o}_A (s', gtn')$, applies an operator o to a node (s, gtn), with $gtn = (I, \prec, \alpha)$ and is defined if $s \models prec(o)$ and o is relevant to an unconstrained instance symbol i in gtn. If i is a primitive task with task name n, then this corresponds to primitive task application in HTNs. If i is instead a relevant goal task, this corresponds to primitive task application in HGNs; in this case, $gtn' = gtn$, and the subgoal remains while the state changes.

Goal decomposition, $(s, gtn) \xrightarrow{i,m}_D (s, gtn')$, for an unconstrained subgoal i by a relevant goal method $m = (g_m, \chi, gtn_m)$ is defined whenever $s \models \chi$. It *prepends* i with gtn_m.

Task decomposition, $(s, gtn) \xrightarrow{i,m}_D (s, gtn')$, for an unconstrained task i by a relevant task method $m = (c, \chi, gtn_m)$ is defined whenever $s \models \chi$. It expands i in gtn, replacing i with the network gtn_m.

Goal release, $(s, gtn) \xrightarrow{i}_G (s, gtn')$, for an unconstrained subgoal i is defined whenever $s \models \alpha(i_g)$. It can remove a subgoal whenever it is satisfied by s.

GTN Planning Problems and Solutions A *gtn-problem* is a tuple $P = (\mathcal{L}, \mathcal{O}, \mathcal{M}, N_0)$, where $\mathcal{L}$ is propositional language defining the operators ($\mathcal{O}$) and methods ($\mathcal{M}$), N_0 is the initial node consisting of the initial state s_0, and gtn_0 is the initial goal-task network. O and C are implicitly defined by $\mathcal{O}$ and $\mathcal{M}$. A problem P is *solvable* under GTN semantics iff there is a progression $N_0 \rightarrow_P^* N_k$, where $N_k = (s_k, gtn_\emptyset)$, s_k is any state, and $gtn_\emptyset$ is the empty network.

Solutions are distinguished by two kinds of plans that depend on whether the world state is changed via operator application. The subsequence of *operator applications* of a progression sequence is a *plan* for P, since such operations modify world state. A *gtn-plan* for P is consists of all progression operators, since this sequence captures the entire set of progressions that must occur for a valid solution. The GRPROCESS produces gtn-plans as explained in the following section.

3 A Goal Reasoning Model

To arrive at a goal reasoning model, we blend GTN semantics with the goal lifecycle in Figure 2 to define a semantics for the GRPROCESS we have partially implemented in ACTORSIM. We begin by extending the online planning model of §2 to model the GR actor as a state transition system $\Sigma_{gr} = (M, R, \delta_{GR})$, where M is the goal memory, R is a set of refinement strategies, and $\delta_{gr} : M \times R \rightarrow M'$ is the goal-reasoning transition function. We next define these components.

3.1 Nodes, Progression, and the Goal Memory

The goal memory stores goal nodes. A *goal node* is a tuple $\mathcal{N} = (g_i, N, C, o, X, x, q)$ where: $g_i \subset \mathcal{L}$ is the goal to be achieved; $N = (s, gtn)$ is a gtn-node for g_i; Con is the set of constraints on g_i and gtn; o is the current mode of g_i, defined below; X is the set of expansions that could achieve g_i, defined below; $x \in X$ is the committed expansion along with any applicable execution status; and q is a vector of quality metrics. Metrics could be domain-dependent (e.g., priority, cost, value, risk, reward) and are associated with achieving g_i. An important domain-independent metric, *inertia*, stores the number of refinements applied to $\mathcal{N}$. Dotted notation indicates access to $\mathcal{N}$'s components, e.g., $\mathcal{N}.N := (s, gtn')$ indicates that the gtn-node gtn of $\mathcal{N}$ has been updated to gtn'.

Similar to GTN planning, progressions modify components of $\mathcal{N}$; we call these the refinement strategies R. Let χ be a set of preconditions and $r \in R$ denote a progression operator for $\mathcal{N}$. Then a refinement $r = (\mathcal{N}, \chi)$ transitions one or more components of $\mathcal{N}$ to $\mathcal{N}'$ and is written $\mathcal{N} \xrightarrow{N,C,o,X,q}_R \mathcal{N}'$. Refinement sequences from $\mathcal{N}$ to $\mathcal{N}''$ are written $\mathcal{N} \rightarrow^*_R \mathcal{N}''$. Preconditions χ come from either the goal lifecycle discussed below or domain-specific requirements for a specific world state or specific events before a refinement can transition.

The *goal memory* $M = \{\mathcal{N}_1, \mathcal{N}_2, .., \mathcal{N}_m\}$ for $m \geq 0$ holds the active goal nodes for the GRPROCESS. Most refinements modify the goal memory by modifying a node within memory, in which case we write $M \rightarrow_R M'$ for a single strategy application resulting in M' and $M \rightarrow^*_R M''$ for a sequence of applied strategies resulting in M''.

3.2 Operations and Semantics: Refinement Strategies (R)

Figure 2 displays the possible refinement strategies, where an actor's decisions consist of applying one or more refinements from R (the arcs) to transition $\mathcal{N}$ between modes (rounded boxes). Strategies are denoted using small caps (e.g., FORMULATE) with the modes in monospace (e.g., `FORMULATED`). For the remainder of this section, we detail semantics for many of these strategies. We shorten the discussion by omitting quality metrics $\mathcal{N}.q$ but leave the q above the progression to indicate that at least inertia is modified. For example, every refinement $\mathcal{N} \xrightarrow{q}_R \mathcal{N}'$ results in $\mathcal{N}'.q.inertia \mathrel{+}= 1$ indicating increased refinement effort on $\mathcal{N}$. GRPROCESS may favor nodes with higher inertia by pushing them toward completion or limiting further processing on them.

For the remainder of this section, we detail the forward sequence of a single goal node through the lifecycle of Figure 2, which consists of goal formulation (via FORMULATE), goal selection (SELECT), planning (EXPAND and COMMIT), plan execution (DISPATCH, MONITOR, EVALUATE), and resolving execution events (FINISH and the suite of RESOLVE-BY strategies, which are displayed as annotations to dashed lines). Finally, we explain two strategies that can be applied at any time: DROP and PROCESS (not shown in this figure).

Goal formulation and Goal Selection Two important decisions for GRPROCESS concern determining which goals to create (i.e., FORMULATE) and which to pursue (i.e., SELECT).

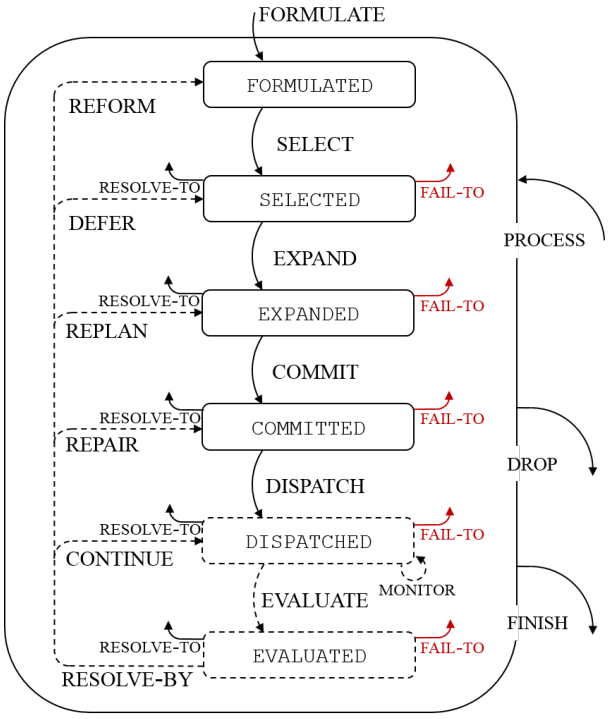


Figure 2: The goal lifecycle. Refinement strategies (arcs) denote possible decision points of an actor, while modes (rounded boxes) denote the status of a goal (set) in the goal memory.

FORMULATE adds a new goal to the goal memory, written $M \xrightarrow{g,\mathcal{N}}_{\text{FORM}} M'$ for a new goal g, its corresponding node $\mathcal{N}$, the goal memory M before the application, and M' the revised memory. The result of applying FORMULATE is: $\mathcal{N}.g = g$; $\mathcal{N}.N = (s_{current}, gtn_g)$; $\mathcal{N}.Con = \emptyset$; $\mathcal{N}.o =$ `FORMULATED`; $\mathcal{N}.X = \emptyset$; $\mathcal{N}.x = nil$; $\mathcal{N}.q.inertia = 1$; and $M' = M \cup \mathcal{N}$.

SELECT transitions $\mathcal{N}.o$ from `FORMULATED` to `SELECTED`, written $\mathcal{N} \xrightarrow{o,q}_{\text{SEL}} \mathcal{N}'$. It allows GRPROCESS to determine which goal nodes move forward and which remain `FORMULATED`. In a GRPROCESS where $|M| \leq k$ is bound to no more than k goals, SELECT can limit extensive processing on nodes. Many nodes trivially transition: $\mathcal{N}'o :=$ `SELECTED`.

Planning Classical planning systems often make strong assumptions about the kind of plan required (i.e., the optimal plan), the number (i.e., usually one), and the nature of execution (i.e., actions are deterministic and atomic). In contrast, a GRPROCESS may explore alternative plans and commit to one after further deliberation. We define an *expansion* to mean any kind of plan to achieve a goal. While we focus on state transitions in Σ or Σ_{gr}, expansions more generally include motion planning, trajectory planning, reactive planning, etc., as often used in robotics applications.

EXPAND, written $\mathcal{N} \xrightarrow{o,X,q}_{\text{EXP}} \mathcal{N}'$, generates expansions (i.e., gtn-plans) via operator application, task decomposition, and goal decomposition from §2.1. Consider a progression $\pi = N_0 \rightarrow_P^* N_k$, where $N_k = (s_k, gtn_\emptyset)$, s_k is any state, and $gtn_\emptyset$ is the empty network. Recall from §2.1 that such a progression is a solution to a GTN problem and was called a *gtn-plan*. EXPAND generates k expansions such that $x^1, x^2, .., x^k \in X$, $|X| > 0$, and $x^1 = \pi^1, .., x^k = \pi^k$ are the available expansions. The result is: $\mathcal{N}'.o \coloneqq$ EXPANDED and $\mathcal{N}'.X \coloneqq \{x^1, .., x^k\}$.

COMMIT chooses one expansion from $\mathcal{N}.X$ for Controller execution and is written $\mathcal{N} \xrightarrow{o,q,x}_{\text{COM}} \mathcal{N}'$. The result is: $\mathcal{N}'.o \coloneqq$ COMMITTED and $\mathcal{N}'.x \coloneqq x^c$ for some $1 \leq c \leq k$.

Plan Execution The Controller executes the steps in $\mathcal{N}.x$ until no more steps remain or a step fails; $\mathcal{N}$ is DISPATCHED during this progression. Some expansions (e.g., goal or task decomposition) are internal to the goal memory and do not result in external actions of the actor. In the case of decomposition, a node remains DISPATCHED until its subgoals or subtasks are completed. Other expansions (e.g., operator application) result in external actions by the Controller during execution. Plan execution consists of DISPATCH, MONITOR, and EVALUATE.

DISPATCH, written $\mathcal{N} \xrightarrow{o,N,q}_{\text{DISP}} \mathcal{N}'$, applies the steps of the progression within $\mathcal{N}.x$. First, the goal node transitions: $\mathcal{N}'.o \coloneqq$ DISPATCHED. Then, the GRPROCESS steps through the expansion $\mathcal{N}.x$. Suppose that $\mathcal{N}.x$ points to the expansion $x = N_0 \rightarrow_P^* N_k$ and that an index $0 < j \leq k$ indicates the step of the progression such that $N_{j-1} \rightarrow_P^j N_j$. For k steps in x and each step x_j for $0 < j \leq k$, the result is: $\mathcal{N}.N_{j-1} \rightarrow_P^j \mathcal{N}'.N_j$. How the GRPROCESS applies x_j depends on specified operation (cf. §2.1): *Operator Application* applies operator o to the instance symbol i. This application results the Controller executing i. *Task Decomposition* applies method m to a compound task i, written $\xrightarrow{i,m}_D$, such that $\mathcal{N}.N.gtn$ is progressed. *Goal Decomposition* applies method m to a goal i, written $\xrightarrow{i,m}_D$, such that $\mathcal{N}.N.gtn$ is progressed, resulting in new subgoals being added to the goal memory M. Let there be t new subgoals resulting from applying m to i, labeled $(g_{i1}, g_{i2}, .., g_{it})$. For goal g_{ij} where $0 < j \leq t$, then FORMULATE(g_{ij}) is called, resulting in $M \xrightarrow{g_{ij},\mathcal{N}}_{\text{FORM}} M'$.

MONITOR, if enabled, proactively checks on the status of $\mathcal{N}.x_j$. If the status is FAIL or is not meeting expectations, then EVALUATE is called. Nominal status only modifies the inertia.

EVALUATE, written $\mathcal{N} \xrightarrow{o,q}_{\text{EVAL}} \mathcal{N}'$, processes events that impact $\mathcal{N}$ during execution, which might include execution updates or unanticipated anomalies. This strategy allows a goal node to signal track that its execution is impacted: $\mathcal{N}'.o \coloneqq$ EVALUATED.

Resolving Notable Events A *notable event* is one that impacts $\mathcal{N}$. A number of possible strategies relate to such events and some are relevant from particular modes. Often the goal determines for itself whether an event is noteworthy, which simplifies the encoding of strategies for a domain. However, in more complex cases another deciding process may arbitrate this determination. Resolution strategies can roughly be divided into those that occur during execution (shown as dashed lines in Figure 1), those that are related to error conditions, and those that occur outside of executions or errors.

PROCESS may be called in any node. It is the means by which external processes or the GRPROCESS notify a goal about an event and allow the goal to determine whether the event is significant. In many cases, an event can be disregarded and the only the inertia is incremented. If the node is DISPATCHED then the event may impact the execution of a step x_j. The impact of the event may be positive (e.g., completion of x_j), neutral (e.g., x_j is progressing as expected) or negative (e.g., the imminent or detected failure of x_j). In this case, $\mathcal{N}$ transitions to EVALUATED and there are several possible resolutions from this mode, as shown by the dashed RESOLVE-BY strategies of Figure 1.

RESOLVE-BY can only be called from EVALUATED and consists of a suite of strategies, which we only briefly describe. These strategies are distinct because the Controller may need to be notified. CONTINUE allows $\mathcal{N}$ to proceed without significant change to its members. ADJUST corrects the state models Σ or Σ_{GR} that would modify future planning. REPAIR modifies the current expansion x to x'. REEXPAND creates new expansions $\{x'^1, .., x'^k\}$ for the GRPROCESS to consider. DEFER returns $\mathcal{N}$ in a SELECTED mode and REFORMULATE returns $\mathcal{N}$in a FORMULATED mode. FAIL-TO is a failure mode that allows the GRPROCESS to return a goal to any previous mode for further processing. This strategy applies when a transition is attempted but fails. For example, if a plan cannot be generated then EXPAND may trigger FAIL-TO(SELECTED).

RESOLVE-TO is used when a notable event impacts a node but the impact is not deemed a failure. For example, if a plan has already been generated but the goal for a node is preempted, then the GRPROCESS may call RESOLVE-TO(FORMULATED) to unselect the goal. In contrast to RESOLVE-BY, these methods simply "park" $\mathcal{N}$ in the appropriate mode and will not otherwise modify the goal node. Such progressions may be useful for quickly pausing a goal.

DROP removes $\mathcal{N}$ from M such that $M' = M \setminus \{\mathcal{N}\}$. It is analogous to goal release (cf. §2.1).

FINISH, written $\mathcal{N} \xrightarrow{o,q}_{\text{FIN}} \mathcal{N}'$, indicates that execution is complete for this expansion. FINISH cannot complete if subgoals in gtn exist in the memory M. If x involved decomposition, then all subgoals or subtasks have been DROPed. If x involved operator application, then the Controller returned SUCCESS. This strategy does not remove $\mathcal{N}$ from M, which is performed by DROP.

3.3 Goal Reasoning Problems and Solutions

Let $P_{gr} = (\mathcal{L}, \mathcal{O}, \mathcal{M}, R_d, R_p, M_0)$ be a goal-reasoning problem where $\mathcal{L} = \mathcal{L}_{external} \cup \mathcal{L}_{internal}$ is the propositional language, $\mathcal{M}$ and $\mathcal{O}$ are defined as in Section 2.1, R_d is the default set of refinement strategies detailed in Section 3.2,

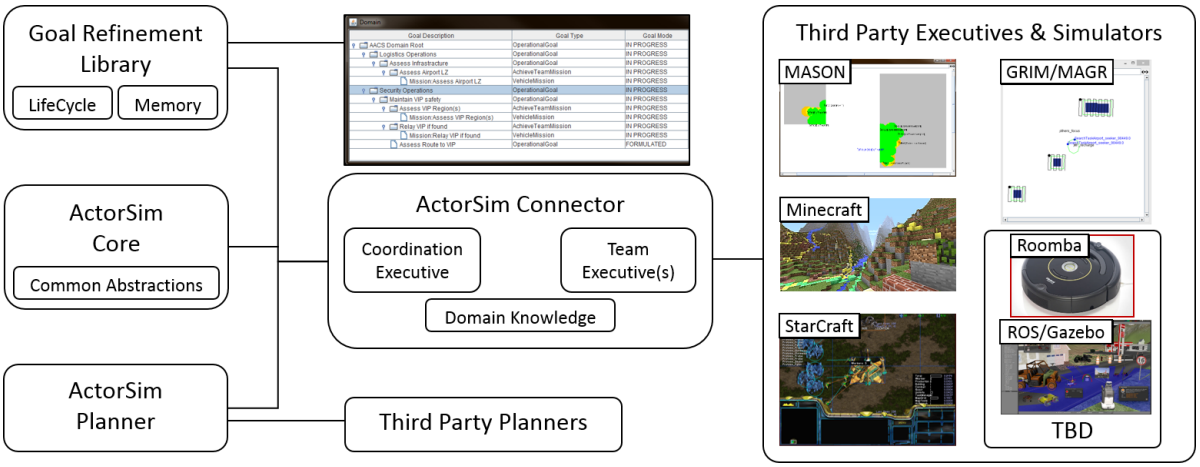


Figure 3: The Component Architecture of ACTORSIM

R_p is a set of refinement strategies provided by the domain designer, and M_0 is the initial goal memory.

We say that P_{gr} is solvable iff there is a progression $M_0 \rightarrow_R^* M_k$, where $M_k = \emptyset$.

4 The Actor Simulator

The Actor Simulator, ACTORSIM (Figure 3), is a partial implementation of the goal lifecycle of Roberts et al. (2015), which is described in Section 3.2. ACTORSIM complements existing open source planning systems with a standardized implementation of goal reasoning. It also provides links to simulators that can simulate multiple agents interacting within a dynamic environment.

ACTORSIM **Core** provides the interfaces and minimal implementations of the platform. It contains the essential abstractions that apply across any simulator. This component contains information about Areas, Locations, Actors, Vehicles, Symbols, Maps, Sensors, and configuration details.

ACTORSIM **Planner** contains the interfaces and minimal implementations for linking to existing open source planning systems. This component unifies Mission Planning, Task Planning, Path Planning, and Motion Planning. It currently includes simple, hand-coded implementations of these planners, although we envision linking this component to many open source planning systems.

ACTORSIM **Connector** links to existing simulators directly or through a network protocol. Currently supported simulators include George Mason University's MASON[4] and two computer game simulators: StarCraft and Minecraft. We envision links to common robotics simulators (e.g., Gazebo, ROS, OpenAMASE), additional game engines (e.g., Mario Bros., Atari arcade, Angry Birds), and existing competition simulators (e.g., RDDLSim). We may eventually link ACTORSIM to physical hardware.

ACTORSIM **Coordinator** (not shown in the figure) houses the interfaces that unify all the other components. This component contains abstractions for Tasks, Events, Human interface Interaction, Executives (i.e., Controllers), and Event Notifications. It uses Google's protocol buffers[5] for messaging between distributed components.

The **Goal Refinement Library** is a standalone library that is integral to ACTORSIM , but could be used on its own. It provides goal management and the data structures for transitioning goals throughout the system. This library contains the default implementations for goals, goal types, goal refinement strategies, the goal memory, domain loading, and domain design.

5 Overcoming Obstacles in Minecraft

We study goal reasoning in Minecraft, a popular game where a human player moves a character, named Steve, to explore a 3D virtual world while gathering resources and surviving dangers. Managing the complete game is challenging. The character holds a limited inventory to be used for survival. Resource blocks such as sand, dirt, wood, and stone can be crafted into new items, which in turn can be used to construct tools (e.g., a pickaxe for mining or shovel for digging) or structures (e.g., a shelter, house, or castle). Some blocks are dangerous to the character (e.g., lava or water). Hostile non-playing characters like a creeper or skeleton, generally called mobs, can damage the characters health. Steve can only fall two blocks without taking damage. We focus on the problem of navigating a much simpler subset of the world. The set of possible choices available to achieve even this *simple* goal is staggering; for navigating a 15x15 maze in Minecraft, Abel et al. (2014) estimate the state space to be nearly one million states.

Researchers have recently begun using the Minecraft game for the study of intelligent agents (Aluru et al. 2015). In previous work, researchers developed a learning architecture called the Brown-UMBC Reinforcement Learning and Planning (BURLAP) library, which they implemented in their variant of Minecraft, BURLAPCraft (Abel et al. 2015) BURLAPCraft allows a virtual player to disregard certain

[4] http://cs.gmu.edu/~eclab/projects/mason/

[5] https://developers.google.com/protocol-buffers/

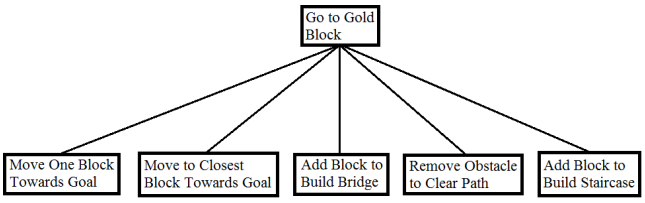


Figure 4: The gtn for the GRPROCESSin our study.

actions that are not necessary for achieving goals such as navigating a maze.

Similar to that research, we task the GRPROCESS , acting as a virtual player, with controlling Steve to achieve the goal of navigating to a gold block through an obstacle course. However, our technical approach differs from prior research. Our aim is to develop a GRPROCESS that can incorporate increasingly sophisticated goal-task networks and learned experience about when to apply them. At a minimum, this requires thinking about how to compose action primitives into tasks that the GRPROCESS can apply and linking these tasks into a gtn. Thus, we construct these tasks and build a gtn that uses them.

Figure 4 shows the gtn consisting of a top goal of moving to the gold block and the five descriptive subgoals that help the character lead to that objective. These subgoals do not contain operational knowledge. For example, preconditions on actions ensure that Steve will not violate safety by falling too far or walking into a pool of lava or water. For moving toward the goal, the block at eye level must be air, the block stepped on cannot be lava or water, and Steve cannot fall more than a height of two blocks. A staircase requires a wall with a height of two blocks and the ability to move backwards in order to place a block. Mining is only applicable if the obstacle has a height of three blocks.

The order of subgoal choice impacts performance. For example, suppose the subgoal to step forward is selected when lava is directly in front of Steve. Steve's Controller disallows this step because it violates safety and the subgoal will fail, which will require additional goal reasoning to resolve the failure.

Two features of our goal representation complement prior research in action selection (e.g., reinforcement learning or automated planning). First, we model the subgoal choice at descriptive level, assuming that committing to a subgoal results in an effective operational sequence (i.e., a plan) to achieve the goal. We rely on feedback of the Controller running the plan to resolve the subgoal. Second, the entire state space from start to finish is inaccessible to the GRPROCESSso it cannot simply perform offline planning or interleave full planning with online execution. Each obstacle course is distinct and there must be an interleaving of perception, goal reasoning, acting. Third, the operational semantics of committing to a subgoal are left to the Controller. Thus, the GRPROCESS must learn to rank the subgoals based on the current state using prior experience. Although random exploration is possible, we will present evidence that that such an approach is untenable, corroborating the findings of

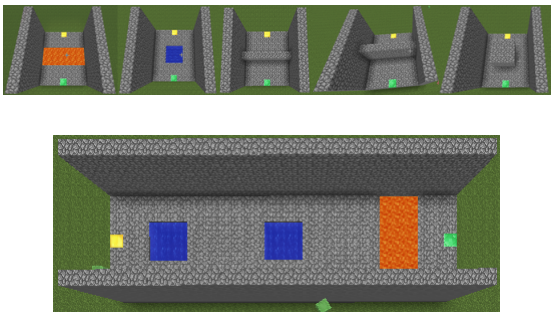


Figure 5: The section types (top) and a short obstacle course (bottom) where the GRPROCESS must traverse from the emerald block on the right to the gold block on the left while through a lava pit and two ponds.

Abel et al. (2015) that the state/action space is too large to explore without a bias.

The question, then, is how to bias the exploration in such a way as to speed up learning. *Our research hypothesis is that making effective choices at the* GTN *level can be done by learning from traces (i.e., examples) that lead to more efficient behavior, where improved efficiency is measured as getting to the goal in fewer steps or failing less frequently.* Our research focus in this paper is examining what kind of experience is most valuable. To this end, we demonstrate a pilot study that leverages three kinds of prior experience (completely random, ordered, and expert) to learning an effective subgoal selection policy.

We next describe how ACTORSIM connects to Minecraft and how we collect experience.

5.1 The Minecraft Connector in ACTORSIM

The Minecraft Connector integrates ACTORSIM abstractions with a reverse-engineered game plugin called the Minecraft Forge API (Forge), which provides methods for manipulating Minecraft. We implemented basic motion primitives such as looking, moving, jumping, and placing or destroying blocks. These motion primitives compose the operational plans for the five sub-goals: step forward, move closer, step up, mine, and bridge. Although some of this functionality was present in BURLAPCraft (Abel et al. 2015), our implementation better matches with the abstractions provided by ACTORSIM Core and ACTORSIM Coordinator.

We have simplified Steve's motions to be axis aligned. Steve always faces North and the maze is constructed such that the gold block is North of Steve in a straight line. Steve is 1.8 meters high; voxels in Minecraft are 1 meter square. So, Steve occupies roughly a 1x2 meter space. Steve interacts with a limited set of world objects: cobblestone, emerald, air, lava, water, and gold.

The Minecraft connector constructs the obstacle courses for our study. Figure 5 (top) shows the five sections the GRPROCESS may encounter: lava, pond, short wall (2 blocks high), tall wall (3 blocks high), and pillar (3 blocks high). Figure 5 (bottom) shows a maze composed of three sections. Steve begins at an emerald block on the right with

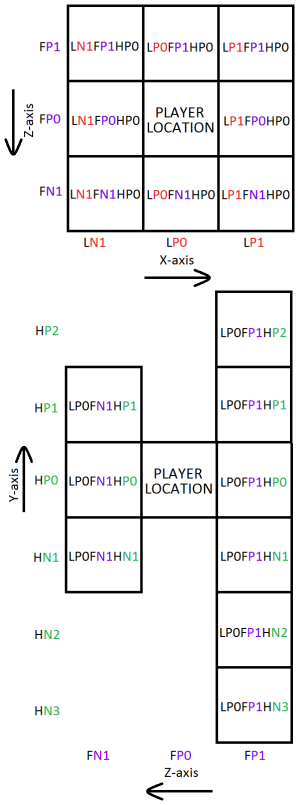


Figure 6: Observable blocks around Steve from the top view (top), where the player is facing "up" and the side view (bottom), where the player is facing to the right.

a goal of being at the gold block.

Each obstacle has an appropriate subgoal choice. For lava or pond, the best choice is a bridge; alternatively the GRPROCESS may also move closer and go around the pond. For the short wall, the best subgoal is to create a single stair and step up. For the tall wall or pillar, which are both three blocks high, the best subgoal is to mine through the wall; alternatively, the GRPROCESS may also move closer and go around the pillar.

Observations Figure 6 shows the set of states around Steve that the GRPROCESS can observe. These include the eight blocks directly around Steve's feet, the two blocks directly behind and in front of Steve, one block behind and below Steve, the block just above Steve's head to the front, and the block three down and in front of Steve. A state is labeled with a unique string using the relative position left/right (l), front/back (f), and height (h) with either a positive (p) or negative (n) offset, where zero is denoted as a positive number. Each state is assigned a unique string (shown in each box) to denote the world object in that position.

Collecting Traces of Experience We collect three kinds of traces for choosing these five subgoals that vary in how much state they consider. The **random** training procedure is worst-case baseline; it ignores state and selects a subgoal with uniform probability. The **ordered** training procedure selects the subgoals in the same order of Figure 4: step forward, move closer, bridge, mine, and step up; it also ignores state. If a subgoal is allowed by the Controller, the subgoal is applied. If not, it continues to the next subgoal in the ordering; on success selection restarts from the beginning of the order. The **expert** training procedure ensures most runs reach the gold as a best-case bound; it is hand-coded (by an author of this paper) and examines detailed state to select the best subgoal.

We collect traces from the random, ordered, and expert procedures, capturing the state, distance to the goal, sub-goal chosen, and whether the chosen subgoal succeeded. Both the random and ordered training procedures can fail to reach the gold. The expert procedure never fails to reach the gold but also represents extremely biased knowledge about which subgoal is appropriate.

6 Learning from Experience in Minecraft

We apply two learning procedures to the traces of one or more of the training procedures. The **frequentist** procedure applies simple statistical sampling from random, ordered, or expert traces to select the best choice. The frequentist procedure leverages all state knowledge even though some state may be irrelevant to decision making. To apply these traces, we then collated the results into tables that counted the subgoal chosen for a particular state where, if the subgoal was successful, we add 3 to the frequency, otherwise we subtract 1. The frequentist procedure then chooses the subgoal with the highest frequency based on the current state. If the GRPROCESS encounters a state that was not observed in any of its training traces, it fails to reach the goal location.

The **decision tree** (d-tree) procedure learns a decision tree over past experience to select the best subgoal. We used the J48 algorithm implemented in WEKA[6]. Figure 7 shows the tree learned from expert traces. The world object at that position is indicated by a single letter: cobblestone (C), emerald (E), air (A), lava (L), water (W), and gold (G). Thus the first state listed in the tree is the block immediately in front of Steve's head: left/right of 0, front/back of 1, and height of 1.

Evaluation We measured the number of subgoal choices to complete each variant using each of the three training procedures. We also counted the number of failed attempts the GRPROCESS tried before reaching the goal location or failing. We ran 10 trials for each procedure in each variant on random course lengths of 5, 10, 15, and 20 sections. The experiment times out if time exceeds 240 steps or when, during the frequentist procedure, an unknown state is encountered.

The expert and learned approaches never fail, so we focus our discussion on the runtime, which is a proxy for the number of steps taken. We measured the run time taken for the GRPROCESS to complete the obstacle course using each of the three procedures. We expected the expert procedure to have the lowest elapsed time, the random procedure to have the highest elapsed time, and ordered to be between random

[6] http://www.cs.waikato.ac.nz/~ml/weka/

```
lp0fp1hp1 = C
|   lp0fp1hp2 = C: RemoveObstacle (43.0)
|   lp0fp1hp2 = A: CreateStairs (19.0)
|   lp0fp1hp2 = E: RemoveObstacle (0.0)
|   lp0fp1hp2 = L: RemoveObstacle (0.0)
|   lp0fp1hp2 = W: RemoveObstacle (0.0)
|   lp0fp1hp2 = G: RemoveObstacle (0.0)
lp0fp1hp1 = A
|   lp0fp1hn1 = C: WalkTo (731.0)
|   lp0fp1hn1 = A: WalkTo (53.0)
|   lp0fp1hn1 = E: WalkTo (0.0)
|   lp0fp1hn1 = L: CreateBridge (42.0)
|   lp0fp1hn1 = W: CreateBridge (36.0)
|   lp0fp1hn1 = G: WalkTo (17.0)
lp0fp1hp1 = E: WalkTo (0.0)
lp0fp1hp1 = L: WalkTo (0.0)
lp0fp1hp1 = W: WalkTo (0.0)
lp0fp1hp1 = G: WalkTo (0.0)
```

Figure 7: The decision tree learned from the expert traces.

and expert. This is because the expert procedure checks the state of the environment before choosing a sub-goal, and therefore makes an informed decision. In the random and ordered procedures, the GRPROCESS relies on a random or pre-set order of sub-goals to choose from, which could lead to inefficiencies if a sub-goal is not appropriate, but still achieved. When applying learning, we expected the frequentist procedure to have an elapsed time between ordered and expert and we expected to see an effect of using different traces.

Results Table 1 shows the elapsed time as the number of sections in the obstacle course increase from 5 to 20. The left-most column indicates one or more training traces used by the learning mechanism: random (R), ordered (O), or expert (E). The number of samples is too low for statistical testing, but we plan to run a full experiment and report such testing in future revisions.

The top subtable, Training, shows the average runtime during trace collection; a dash indicates a time out and failure to reach the gold block for all runs. On average, expert performs best, random worst, and ordered in the middle. However, in looking more closely at the runs, we noted that the expert procedure does not always outperform ordered. This finding does not meet our expectation that the expert will dominate, but the results are strongly suggestive that it generally performs best. We note that the ordered procedure does not perform that much worse than the expert, which is a theme we return to several times.

We counted the number of failed subgoal attempts, which indicates how often a procedure selects an inappropriate subgoal that the Controller disallowed. We only discuss these results and do not show the data. The expert procedure never fails a subgoal. The random procedure had the most number of failed goal attempts, and the ordered procedure was in the middle of the other two. These trends were expected as the random and ordered procedures do not check the environment state before making an informed decision, and show that the expert procedure is better in terms of choosing the right sub-goals.

	5		10		15		20	
Train	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Training								
R	-	-	-	-	-	-	-	-
O	21.5	1.7	41.8	3.3	56.8	3.3	79.8	2.1
E	16.3	1.3	23.8	1.3	46.0	4.1	64.8	5.0
Frequentist								
R	-	-	-	-	-	-	-	-
O	17.5	1.7	33.3	2.1	48.8	5.1	63.5	3.5
RO	17.5	1.7	33.3	2.1	48.8	5.1	63.5	3.5
E	16.3	1.3	32.8	1.3	47.3	5.1	65.0	4.6
RE	16.3	1.3	32.8	1.3	46.7	6.1	65.0	4.6
OE	17.5	1.7	33.3	2.1	48.8	5.1	63.5	3.5
ROE	17.5	1.7	33.3	2.1	49.3	5.4	63.5	3.5
Decision Tree								
R	-	-	-	-	-	-	-	-
O	6.7	0.6	12.0	0.8	18.3	2.1	23.5	2.1
RO	6.7	0.6	12.0	0.8	18.0	2.8	23.0	0.0
E	5.8	1.0	13.3	0.5	18.3	2.4	27.0	1.2
RE	5.8	1.0	13.3	0.5	18.3	2.4	27.0	1.2
OE	6.7	0.6	12.0	0.8	18.3	2.1	23.0	1.7
ROE	6.7	0.6	12.0	0.8	18.3	2.1	23.3	1.5

Table 1: Mean runtime and standard deviation for the study.

Table 1 (middle) shows the results obtained from the frequentist procedure learning using various combinations of the training traces. Using either expert traces (E row) or ordered traces (O row) only resulted in substantially similar runtimes to the original expert traces, suggesting that either kind of trace is suitable for biasing learning. Combining the two traces (OE) did not appear to change the behavior.

Using only random traces (R) did not produce an effective policy. But it also does not appear that adding the random trace to expert (RE), ordered (RO), or their both (ROE) cause a significant degradation of the results.

Table 1 (bottom) shows the results obtained from decision tree learning. We can observe a dramatic improvement in the runtime with this procedure. Moreover, the ordered tree (O) sometimes has better average performance than the expert tree (E). Adding random traces (RO, RE, ROE) did not appear to diminish the results substantially. As seen in Figure 6, we use 15 states per observation. The ordered tree (O) makes effective decisions using between 3 and 5 states, while the expert tree (E) makes effective decisions examining at most 2 states. Clearly, there is great benefit to learning which observations matter for effective decision making.

7 Other ACTORSIM Connectors

Other projects apply, extend, or propose ACTORSIM to work with additional simulators. We present a snapshot of each project to highlight how ACTORSIM assists in studying goal reasoning.

Foreign Disaster Relief The longest-running project for ACTORSIM is Foreign Disaster Relief, where we have studied

how to perform goal reasoning to coordinate teams of robotic vehicles (Roberts et al. 2015a), estimating high-fidelity simulations using a faster, but lower-fidelity estimates, and its application to play-calling (Apker et al., 2015). The most recent extension of this work has extended Goal Reasoning with Information Metrics (GRIM) by Johnson et al. (2016). The ACTORSIM codebase, in particular the Goal Reasoning Library, had its genesis in abstractions developed during this project. Similar to the studies presented, ACTORSIM uses the MASON simulator for the scenarios of this project. However, the set of motion and path planning primitives is simplified in that it does not leverage the LTL templates or vehicle controllers mentioned in Roberts et al. (2015a).

StarCraft StarCraft:Brood War is a Real Time Strategy (RTS) game developed by Blizzard Entertainment. At an abstract level, it is an economic and military simulation. Players build an economy to gather resources, use these resources to train an army, then use this army to attempt to defeat their opponent, either in direct engagements or through disrupting their economy. It has a number of desirable properties as an artificial intelligence testbed, and has seen a good deal of research in recent years (Ontanon et al. 2013).

ACTORSIM integrates with an existing game agent developed by Churchill et al., UAlbertaBot (UAB)[7]. UAB interfaces directly with the game of Brood War using the Brood War API (BWAPI)[8], through which it can issue commands to units and monitor the observable state of the game. It is a modular agent on which researchers can build their systems.

The ACTORSIM Connector controls a subset of the behavior of the agent, letting UAB control the remainder. For example, if ACTORSIM creates a goal to attack a specific region of the map, UAB will decide the formation and specific unit commands necessary to achieve that goal. The behavior controlled by ACTORSIM is currently region-level positioning, soon to include economic growth decisions.

We have used ACTORSIM to emulate the original hand-coded behaviors of UAB, and are in the process of implementing more complex goals to demonstrate the additional expressivity of the agent using our system. In addition, we are working on automatically learning the EVALUATEfunction based on replays of professional human players, which are available online in large quantities.

8 Related Work

Researchers have applied goal reasoning to other domains, such as the Tactical Action Officer (TAO) Sandbox (Molineaux et. al., 2010). Using the Autonomous Response to Unexpected Events (ARTUE) agent, they implemented goal-driven autonomy; ARTUE can reason about what goals to achieve based on the changing environment, in this case a strategy simulation for TAOs to train in anti-submarine warfare. Goal reasoning has been used in other gaming domains such as Battle of Survival, a real-time strategy game (Klenk et al., 2013).

Goal refinement builds on the work in plan refinement (Kambhampati, Knoblock, & Yang 1995), which equates different kinds of planning algorithms in plan-space and state-space planning. Extensions incorporated other forms of planning and clarify issues in the Modal Truth Criterion (Kambhampati and Nau 1994). More recent formalisms such as Angelic Hierarchical Plans (Marthi et al. 2008) and Hierarchical Goal Networks (Shivashankar et al. 2013) can also be viewed as leveraging plan refinement. The focus on constraints in plan refinement allows a natural extension to the many integrated planning and scheduling systems that use constraints for temporal and resource reasoning.

The goal lifecycle bears close resemblance to that of Harland et al (2014) and earlier work (Thangarajah et al. 2010). They present a goal lifecycle for BDI agents, provide operational semantics for their lifecycle, and demonstrate the lifecycle on a Mars rover scenario. Recently, Cox et al. (2016) proposed a model for goal reasoning based on planning. We hope to characterize the distinction between these models in future work.

9 Summary and Future Work

In this paper, we formalized a semantics for goal reasoning and applied our implementation, called ACTORSIM, to a pilot study in Minecraft. For the task that we examined, we developed three methods to selection sub-goals: random, ordered, and expert. Using the results of these methods, we examined two learning methods. We showed that the expert selection is the most efficient based on the elapsed time and the number of failed goal attempts, and that the random selection was the least efficient followed by the ordered selection. For the frequentist approach, we showed that the expert and ordered traces yielded the best performance, and that adding random traces did not seem to cause too much harm.

In the future, we plan to incorporate more complex tasks such as having an agent protect itself against creepers and mobs. A first step in this direction will be to encode our goal network using the goal lifecycle provided in ACTORSIM, since our current implementation applies goal reasoning without using much of its functionality. This will allow us to build (or learn) more sophisticated goal networks and to take advantage of existing planning and scheduling techniques in ACTORSIM. Finally, we plan to include non-playing characters in Minecraft with our resulting goal networks.

Broader dissemination of ACTORSIM will foster deeper study and enriched collaboration between researchers interested in goal reasoning, planning, and acting. ACTORSIM complements existing open source planning systems with a standardized implementation of goal reasoning so researchers can focus on (1) designing goals and goal transitions for their system (2) linking ACTORSIM to their particular simulator, and (3) studying goals and behavior in the dynamic environment provided by the simulator. By releasing it as an open source package, we lay a foundation for advanced studies in goal reasoning that include integration with additional simulators and planning systems, formal models, and empirical studies that examine decision making in challenging, dynamic environments.

Our upcoming projects include extending ACTORSIM to actual robotic systems that include the Roomba system

[7]http://www.github.com/davechurchill/ualbertabot

[8]http://www.github.com/bwapi/bwapi

and a set of Hubo's. The architecture of ACTORSIM is now developed enough that we can also start to consider integrating it with more sophisticated robot simulators such as Gazebo or the the robocup simulators. We believe this area presents a great deal of promise for the formal model of goal reasoning we present in this paper as well as for ACTORSIM.

Acknowledgments

This research was funded by OSD and NRL. Ron Alford performed part of this work under an ASEE postdoctoral fellowship at NRL. We thank the anonymous reviewers for comments that helped improve the paper.

References

Abel, D.; Hershkowitz, D. E.; Barth-Maron, G.; Brawner, S.; OFarrell, K.; MacGlashan, J.; and Tellex, S. 2015. Goal-based action priors. In *Proc. Int'l Conf. on Automated Planning and Scheduling*.

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. to appear. Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proc. of the Int'l Joint Conf. on AI (IJCAI)*. AAAI Press.

Aluru, K.; Tellex, S.; Oberlin, J.; and Macglashan, J. 2015. Minecraft as an experimental world for AI in robotics. In *AAAI Fall Symposium*.

Apker, T.; Johnson, B.; and Humphrey, L. 2016. Ltl templates for play-calling supervisory control. In *Proc. AIAA @Infospace*.

Cox, M. T.; Alavi, Z.; Dannenhauer, D.; Eyorokon, V.; Munoz-Avila, H.; and Perlis, D. 2016. MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *AAAI*.

Dannenhauer, D., and Munoz-Avila, H. 2015. Raising expectations in gda agents acting in dynamic environments. In *Proc. of the Int'l Joint Conf. on AI (IJCAI)*. AAAI Press.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1955–1961. AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2014. The actor's view of automated planning and acting: a position paper. *Artificial Intelligence* 208:1–17.

Giacomo, G. D.; Gerevini, A. E.; Patrizi, F.; Saetti, A.; and Sardina, S. 2016. Agent planning programs. *Artificial Intelligence* 231:64–106.

Harland, J.; Morley, D. N.; Thangarajah, J.; and Yorke-Smith, N. 2014. An operational semantics for the goal life-cycle in bdi agents. *Autonomous Agents and Multi-Agent Systems* 28:682–719.

Johnson, B.; Roberts, M.; Apker, T.; and Aha, D. to appear. Goal reasoning with information measures. In *Proceedings of the Conf. on Advances in Cognitive Systems*.

Klenk, M.; Molineaux, M.; and Aha, D. 2013. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence* 29(2):187–206.

Molineaux, M., and Aha, D. W. 2014. Learning unknown event models. In *AAAI*.

Munoz-Avila, H.; Aha, D.; Jaidee, U.; Klenk, M.; and Molineaux, M. 2010. Applying goal directed autonomy to a team shooter game. In *FLAIRS*, 465–470.

Munoz-Avila, H.; Wilson, M. A.; and Aha, D. W. 2015. Guiding the ass with goal motivation weights. In *2015 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning*.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *J. of Art. Intell. Res.* 20:379–404.

Nau, D. S.; Ghallab, M.; and Traverso, P. 2015. Blended planning and acting: Preliminary approach, research challenges. In *AAAI Conf. on Artificial Intelligence*.

Nau, D. 2007. Current trends in automated planning. *Art. Intell. Mag.* 28(40):43–58.

Ontanon, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Trans. Comput. Intellig. and AI in Games* 5:293–311.

Roberts, M.; Apker, T.; Johnston, B.; Auslander, B.; Wellman, B.; and Aha, D. W. 2015a. Coordinating robot teams for disaster relief. In *International Conference of the Florida Artificial Intelligence Research Society*.

Roberts, M.; Vattam, S.; Alford, R.; Auslander, B.; Apker, T.; Johnson, B.; and Aha, D. W. 2015b. Goal reasoning to coordinate teams for disaster relief. In *Working Notes of the PlanRob Workshop at ICAPS*.

Roberts, M.; Alford, R.; Shivashankar, V.; Leece, M.; Gupta, S.; and Aha, D. to appear. Goal reasoning, planning, and acting with ActorSim, the Actor Simulator. In *Proceedings of the Conf. on Advances in Cognitive Systems*.

Shivashankar, V.; Kuter, U.; Nau, D.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. of AAMAS*, volume 2, 981–988. Int. Foundation for AAMAS.

Thangarajah, J.; Harland, J.; Morley, D. N.; and Yorke-Smith, N. 2010. Operational behaviour for executing, suspending, and aborting goals in bdi agent systems. In *DALT*.

Vattam, S., and Aha, D. W. 2015. Case-based plan recognition under imperfect observability. In *ICCBR*.

Vattam, S.; Klenk, M.; Molineaux, M.; and Aha, D. 2013. Breadth of approaches to goal reasoning: A research survey. In Aha, D.; Cox, M.; and Muoz-Avila, H., eds., *Goal Reasoning: Papers from the ACS Workshop (Technical Report CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science*, 222–231.

Wilson, M. A.; Molineaux, M.; and Aha, D. W. 2013. Domain-independent heuristics for goal formulation. In *FLAIRS*.

Young, J., and Hawes, N. 2012. Evolutionary learning of goal priorities in a real-time strategy game. In *Proc. of the AIIDE*. AAAI Press.

Sequential Quadratic Programming for Task Plan Optimization

Christopher Lin[1*], Dylan Hadfield-Menell[2*], Rohan Chitnis[1], Stuart Russell[2], and Pieter Abbeel[2*†‡]

Abstract

We consider the problem of refining an abstract task plan into a motion trajectory. Task and motion planning is a hard problem that is essential to long-horizon mobile manipulation. Many approaches divide the problem into two steps: a search for a task plan and task plan refinement to find a feasible trajectory. We apply sequential quadratic programming to jointly optimize over the parameters in a task plan (e.g., grasps, put down locations). We provide two modifications that make our formulation more suitable to task and motion planning. We show how to use movement primitives to reuse previous solutions (and so save optimization effort) without trapping the algorithm in a poor basin of attraction. We also derive an early convergence criterion that lets us quickly detect unsatisfiable constraints so we can re-initialize their variables. We present experiments in a navigation amongst movable objects domain and show substantial improvement in cost over a backtracking refinement algorithm.

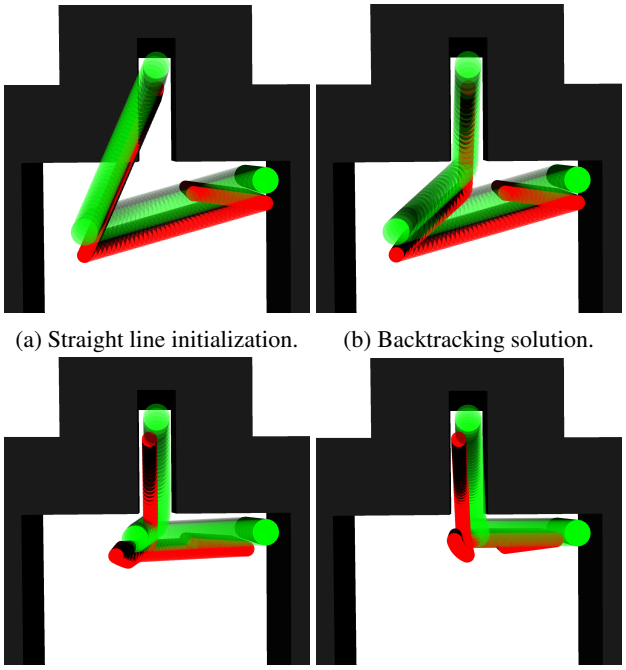
1 INTRODUCTION

Long-horizon mobile manipulation planning is a fundamental problem in robotics. Viewed as trajectory optimization, these problems are wildly non-convex and direct motion planning is usually infeasible. Viewed as a classical planning problem, there is no good way to represent the geometry of the problem efficiently in a STRIPS or PDDL representations.

The robotics and planning communities have studied the problem of *task and motion planning* (TAMP) as a way to overcome these challenges. These approaches seek to integrate classical task planning methods, that can handle long horizons, with motion planning approaches, that can handle complex geometry. In recent years, there has been substantial progress on the problem of finding feasible task and motion plans (Chitnis et al. 2016), (Garrett, Lozano-Perez, and Kaelbling 2015), (Toussaint 2015).

The approach to TAMP in (Chitnis et al. 2016) relies on three components: a black box classical planner that ignores geometry to find an abstract task plan, a black box motion planner that can determine motion plans for a given abstract action, and an interface that shares information between the two different planners. Task plans consists of bound object references (e.g., can_1) and unbound pose references (e.g., pose_1). Pose references are continuous parameters that are characterized by a set of constraints. For example, a task plan may require that pose_1 be a grasping pose for can_1.

The process of motion planning for an abstract plan is called *plan refinement*. If plan refinement for a given task plan fails, then the interface updates the task planner with information that lets it plan around the failure. In this work, we contribute a novel method for the task plan refinement component of this system. Our approach has applications to systems that use a similar decomposition and as a trajectory smoother for general TAMP algorithms.



(a) Straight line initialization. (b) Backtracking solution.

(c) Intermediate Solution. (d) Final Solution.

Figure 1: The robot, shown in red, moves a green can to the goal location. The backtracking solution samples and fixes a trajectory waypoint. This leads to an unnecessarily long path. (c) and (d) show an intermediate and final trajectory computed by running sequential quadratic programming on the task plan.

**denotes equal contribution
†1{c.l, ronuchit}@berkeley.edu
‡2{dhm, russell, pabbeel}@cs.berkeley.edu

Current approaches to task plan refinement typically on a backtracking search over the parameters of the plan and solve a sequence of independent motion planning problems. We propose an approach that jointly optimizes over all of the parameters and trajectories to implement a given abstract plan. This leads to final solutions with substantially lower cost, when compared with approaches that compute motion plans for each high level action independently. Figure 1 shows an example that compares the result from joint optimization with the result from a backtracking search.

The optimization problems we consider are highly non-convex. We rely on randomized restarts to find solutions: if we fail to converge, we determine plan parameters associated with infeasible constraints and sample new initial values for the optimization. After a fixed budget of restarts, we return to the task planning layer and generate a new task plan. We contribute two modifications to the basic algorithm to facilitate efficient randomized restarts.

The first modification uses a minimum velocity projection (Dragan et al. 2015) of the previous solution to re-initialize trajectories. This preserves the overall global structure of the trajectories without trapping new solutions in the same basin of attraction. The second modification is an early convergence criterion that checks to see if a constraint is likely to be unsatisfiable. This allows us to restart more frequently and reduces overall solution times.

Our contributions are as follows: 1) we give a formulation of task and motion planning that unambiguously specifies the associated trajectory optimization; 2) we apply sequential convex programming to jointly optimize over the trajectories and parameters in a plan refinement; 3) we show how to reuse previous solutions without trapping the optimization in a bad basin of attraction; and 4) we show how to do early convergence detection to avoid wasted effort on infeasible plans. We present experiments that compare our approach to a backtracking refinement. Our approach leads a 2-4x reduction in the total path cost of solutions at the cost of a 1.5-3x increase in running time. We verify that our proposed modifications led to reductions in refinement time.

2 TRAJECTORY OPTIMIZATION WITH SEQUENTIAL QUADRATIC PROGRAMMING

Our approach uses sequential quadratic programming to do task plan refinement. In this section, we describe the motion planning algorithm from (Schulman et al. 2013), which applies sequential quadratic programming to motion planning.

Motion Planning as Constrained Trajectory Optimization

A core problem in robotics is *motion planning*: finding a collision-free path between fixed start and goal poses. A motion planning problem is defined by:

- a *configuration space* of robot poses
- a set of obstacles O
- an initial and goal configuration.

We will define configuration spaces by a set of feasible robot poses $\mathcal{X}$ and a dynamics constraint. The dynamics constraint is a Boolean function $f : \mathcal{X} \times \mathcal{X} \rightarrow \{0, 1\}$. It takes as input a pair of poses p_1, p_2 and is 1 iff p_2 is directly reachable from p_1.

Figure 1 shows a 2D motion planning problem that will serve as the starting point for a running example. The pose of the robot is represented by a pair (x,y). We let $\mathcal{X}$ be a bounding box so $x \in [0, 7]$ and $y \in [-2, 7]$. The dynamics function ensures that the distance between subsequent states of the trajectory is always less than a fixed constant: $f(p_1,\ p_2) = (p_1 - p_2 < d_{max})$.

There are three main approaches to motion planning that are used in practice: discretized configuration space search (Cohen, Chitta, and Likhachev 2010), randomized motion planners (Kavraki et al. 1994), (Lavalle 1998), and trajectory optimization (Schulman et al. 2013), (Ratliff et al. 2009). In this work, we build on trajectory optimization approaches.

The downside of trajectory optimization approaches is that they are usually locally optimal and incomplete, while the other approaches have completeness or global optimality guarantees. The upside of trajectory optimization is that it scales well to high dimensions and converges quickly. The second property is especially useful in a task and motion planning context because it lets us quickly rule out infeasible task plans.

Trajectory optimization generates a motion plan by solving the following constrained optimization problem.

$$\begin{aligned} \min_{\tau_t \in \mathcal{X}} \quad & ||\tau||^2 \qquad (1) \\ \text{subject to} \quad & f(\tau_t, \tau_{t+1}) = 1 \\ & SD(\tau_t, o) \geq d_{safe} \ \ \forall o \in O \\ & \tau_0 = p_0, \tau_T = p_T \end{aligned}$$

We optimize over a fixed number of waypoints τ_t, with $t = 0, \ldots, T$. The objective $||\tau||^2$ is a regularizer that produces smooth trajectories. A standard choice is the *minimum velocity* regularizer

$$||\tau||^2 = \sum_t ||\tau_t - \tau_{t+1}||^2.$$

The first constraint is the dynamics constraint that ensures that the pose at time $t + 1$ is reachable from the pose at time t. The second constraint is a collision avoidance constraint. It requires that the distance[1] from any robot pose to an object be larger than a fixed safety margin. The final constraint ensures that the trajectory begins (resp. ends) at the initial (resp. final) pose.

Sequential Quadratic Programming

(Schulman et al. 2013) uses *sequential quadratic programming* (SQP) to solve the optimization problem in Equation 1. SQP is an iterative non-linear optimization algorithm that can be seen as a generalization of Newton's method. An important attribute of SQP is that it can typically solve problems with very few function evaluations. This is useful in

[1]This is actually the signed-distance, which is negative is the robot and object overlap.

trajectory optimization because collision checking is typically a computational bottleneck.

SQP minimizes a non-linear f subject to equality constraints h_i and inequality constraints g_i.

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0 \qquad i = 1, \ldots, n_{eq} \\ & g_i(x) \leq 0 \qquad i = 1, \ldots, n_{ineq} \end{aligned} \tag{2}$$

The first step is to move the constraints into the objective as an ℓ_1 penalty term:

$$\min_x f(x) + \mu \left(\sum_1^{n_{eq}} |h_i(x)| + \sum_1^{n_{ineq}} |g_i(x)|^+ \right). \tag{3}$$

As $\mu \to \infty$, this is equivalent to the original constrained problem. We repeatedly minimize this function in an outer loop that increases μ.

In the inner loop, we use an iterative algorithm. Let $x^{(i)}$ be the current solution. First, we compute a local convex approximation to Equation 3 at $x^{(i)}$. In (Schulman et al. 2013) they use a quadratic approximation to f and linear approximations to the h_i and g_i. We adopt the same approach in this work.

Once we have obtained a convex local approximation we can minimize it to get the next solution $x^{(i+1)}$. We need to ensure that the approximation is accurate so we impose a trust-region constraint. This enforces a hard constraint on the distance between $x^{(i)}$ and $x^{(i+1)}$. Let $\widetilde{f}, \widetilde{h}_i, \widetilde{g}_i$ be convex approximations to f, h_i, g_i. The optimization we solve is

$$\min_x \quad \widetilde{f} + \mu \left(\sum_1^{n_{eq}} |\widetilde{h}_i(x)| + \sum_1^{n_{ineq}} |\widetilde{g}_i(x)|^+ \right) \tag{4}$$

$$\text{subject to} \quad |x - x^{(i)}| < \delta \tag{5}$$

where δ is the trust-region size. The ℓ_1-norm to penalize constraint violations results in a non-smooth optimization, but can still be efficiently minimized by standard quadratic programming solvers. Algorithm 1 shows pseudocode for this optimization method.

As an example, consider the behavior of SQP on the motion planning problem from Figure 2. The initial pose is in the top right at location $(0, 2)$ and the target pose is around a corner at location $(3.5, 5.5)$. We initialize with an infeasible straight line trajectory. We use 20 time-steps for our trajectory. We let the x coordinate for the robot take values in $[0, 7]$ and the y coordinate take values in the range $[-2, 7]$. The corresponds to the following trajectory optimization:

$$\begin{aligned} \min_{\tau_t \in [0,7] \times [-2,7]} \quad & \sum_{t=0}^{20} ||\tau_t - \tau_{t+1}||^2 \\ \text{subject to} \quad & |\tau_t - \tau_{t+1}| \leq d_{max} \\ & SD(\tau_t, Wall) \geq d_{safe} \\ & \tau_0 = (7, 3) \\ & \tau_{20} = (3, 7) \end{aligned}$$

Algorithm 1 ℓ_1 Penalty Sequential Quadratic Programming (Nocedal and Wright 2006).

Define: SQP$(x^{(0)}, f, \{h_i\}, \{g_i\})$
Input: initial point $x^{(0)}$, the function being minimized f, a set of non-linear equality constraints $\{h_i\}$, a set of non-linear inequality constraints $\{g_i\}$.
/* increase the penalty for violated nonlinear constraints in each iteration */
for $\mu = 10^0, 10^1, 10^2 \ldots, \mu_{max}$ **do**
  for $i = 1, \ldots,$ ITER_LIMIT **do**
    /* compute a quadratic approximation for f*/
    $\tilde{f}, \{\tilde{h}_i\}, \{\tilde{g}_i\}$ = ConvexifyProblem($f, \{h_i\}, \{g_i\}$)
    for $j = 1, 2, \ldots$ **do**
      x = argmin (4) subject to (5) and linear constraints
      if TrueImprove / ModelImprove $> c$ **then**
        /* expand trust region */
        $\delta \leftarrow improve\_ratio \cdot \delta$
        break
      end if
      /* shrink trust region */
      $\delta \leftarrow decrease\_ratio \cdot \delta$
      if converged() **then**
        /* converge if trust region too small
        or current solution is a local optimum */
        return locally optimal solution x^*
      end if
    end for
  end for
end for

The first step of the algorithm makes a linear approximation to the signed distance constraint. The details of the approximation can be found in (Schulman et al. 2013). The first image shows this initialization and superimposes the local approximation to the signed distance constraint on top of it. It pushes each pose towards the outside of the walls.

The next step of the algorithm minimizes the approximation to this constraint subject to a trust region constraint. This makes progress on the objective, so we accept the move and increase the size of the trust region. After several iterations, we obtain the trajectory in the middle of the image. At termination we arrive at the motion plan in the left most image: a collision-free, locally-optimal trajectory.

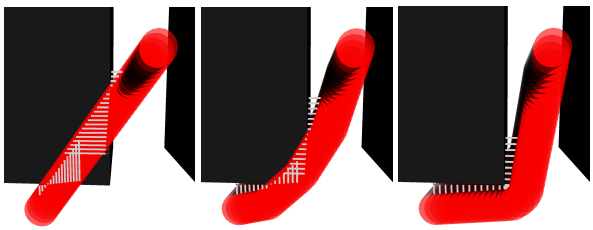
3 TASK AND MOTION PLANNING

In this section, we formulate task and motion planning (TAMP). We present an example formulation of the navigation amongst moveable objects (NAMO) as a TAMP problem. We give an overview of the complete TAMP algorithm presented in (Chitnis et al. 2016).

Problem Formulation

Definition 1 *We define a task and motion planning (*TAMP*) problem as a tuple $\langle T, O, F_P, F_D, I, G, U \rangle$:*

T *a set of object* types *(e.g., movable objects, trajectories, poses, locations).*



(a) Initialization (b) Optimization (c) Final trajectory

Figure 2: Trajectory optimization for a 2D robot. The gradient from the collision information pushes the robot out of collisions despite the infeasible initialization.

O *a set of* objects *(e.g., can$_2$, grasping_pose$_6$, location$_3$).*

F_P *a set of* primitive fluents *that collectively define the world state (e.g., robot poses, object geometry). The set of primitive fluents, together with O, defines the configuration space of the problem.*

F_D *a set of* derived fluents*, higher-order relationship between objects defined as boolean functions that depend on primitive fluents.*

I *a conjunction of primtive fluents that define the initial state.*

G *a conjunction of (primitive or derived) fluents that define the goal state.*

U *a set of* high-level actions *(e.g., grasp, move, put-down). Each high-level action $a \in U$ is parameterized by a list of objects and defined by: 1) $a.pre$, a set of* pre-conditions*, fluents that describe when an action can be taken; 2) $a.post$, a set of* post-conditions*, fluents that hold true after the action is performed; and 3) $a.mid$ a set of* mid-conditions*, fluents that must be true while the action is being executed.*

A state in a TAMP problem is defined by a set of primitive fluents. Note that this defines the truth value of all derived fluents. The solution to a TAMP problem is a plan

$$\pi = \{s^0, (a^0, \tau^0), s^1, (a^1, \tau^1), \ldots, (a^{N-1}, \tau^{N-1}), s^N\}.$$

The s^i are states, defined as a set of primitive predicates that are true. The a^i are the actions in the plan. τ^i is the trajectory for action i and is defined as a sequence of states. A valid solution satisfies the following constraints.

- The first state is the initial state: $s^0 \in I$.
- Pre-conditions are satisfied: $a^i.pre \in s^i$.
- Mid-conditions are satisfied: $a^i.mid \in \tau^i_t \; \forall t$.
- Post-conditions are satisfied: $a^i.post \in s^{i+1}$.
- Trajectories start in the states that precede them and end in the states that follow them: $\tau^i_0 = s^i$, $\tau^i_T = s^{i+1}$.
- The final state is a goal state: $G \in s^N$.

Our formulation differs from the standard formulation of TAMP in two ways. The first is that we explicitly differentiate between primitive fluents and derived fluents. We use the difference between the two types of fluents to distinguish between variables and constraints for the optimization in Section 4.

The second difference is the introduction of *mid-conditions*. These are invariants: constraints that must be satisfied on every step on of a trajectory that implements a high-level actions. Mid-conditions define the space of trajectories than can implement a given high-level action. An example mid-condition is a collision avoidance constraint.

Example Domain: Navigation Amongst Movable Objects

Here, we formulate a 2D version of the *navigation amongst moveable objects* (NAMO) problem (Stilman and Kuffner 2008). In our domain, a circular robot navigates a room full of obstructions. If the robot is next to an object, it can attach to it rigidly via a suction cup. In the top middle of our domain is a closet. The robot's goal is to store objects in, or retrieve objects from, the closet. Thus, we call the problem the 2D closet domain (CL-2D-NAMO). This domain is characterized as follows.

Object types T. There are six object types: 1) *robot*, a circular robot that can move, pick, and place objects; 2) *cans*, cylinders throughout the domain that the robot can grasp and manipulate; 3) *walls*, rectangular obstructions in the domain that the robot can not manipulate; 4) *poses*, vectors in $\mathbb{R}^2$ that represent robot poses; 5) *locs*, vectors in $\mathbb{R}^2$ that represent object poses; and 6) *grasps*, vectors in $\mathbb{R}^2$ that represent grasps as the relative position of the grasped object and robot.

Objects O. There is a single robot, R. There are N movable objects: can$_1$,...,can$_N$. There are 8 walls that make up the unmovable objects in the domain: wall$_1$,...,wall$_8$. Robot poses, object locs, and grasps make up the remaining objects in the domain. The are continuous values so there are infinitely many of these objects. Robot poses and object locs are contained in a bounding box around the room B. Grasps are restricted to the be in the interval $[-1, 1]^2$.

Primitive Fluents F_P. The primitive fluents in this domain define the state of the world. We define the robot's position with a fluent whose sole parameter is a robot pose: robotAt(?rp-pose). We define an object's loc with a similar fluent that is parametrized by an object and a loc: objAt(?o-can ?ol-loc).

Derived Fluents F_D. There are three derived fluents in this domain. The first is a collision avoidance constraint that is parametrized by an object, a loc, and a robot pose: obstructs(?obj-can ?loc-loc ?rp-pose). This is true when ?obj and the robot overlap at their respective locations and poses. It is defined as a constraint on the signed distance: SD(?obj, R) $\geq d_{safe}$.

We determine if the robot can pick up a can with isGraspPose(?obj-can ?rp-pose ?loc-loc). This is true if a robot at ?rp touches the can at location ?loc. This is implemented as an equality constraint on signed distance: SD(R, can) $= \epsilon$. We use this to determine when the robot can pick up the object, and when it can put it down.

Once the robot has picked up an object, we need to ensure that the grasp is maintained during the trajectory. We do this with inManip(?obj-can ?g-grasp), which is parametrized by a can and a grasp. It is defined by an equality constraint on the respective positions of the object and the robot: (robotAt(?rp) $\wedge$ objAt(?obj ?loc) $\Rightarrow$?rp-?loc = ?g). If the robot is holding an object (i.e., inManip is true for some object and grasp) then it is treated as part of the robot in all signed distance checks.

Dynamics. The dynamics of this problem are simple. The robot has a maximum distance it can move during any timestep. The objects remain at their previous location, otherwise they are unconstrained. The inManip fluent ensures that such objects are always in the same relative position to the robot.

High-level actions U. We have four high-level actions in our domain: MOVE, MOVEWITHOBJ, PICK, and PLACE.

The MOVE action moves the robot from one location to another, assuming it holds no object. We use ?rp$_t$ to represent the robot pose at time t within the move action's trajectory.

MOVE(?rp1-pose ?rp2-pose)
pre robotAt(?rp1)
$\wedge$ ($\forall$?obj-can, ?g-grasp $\neg$ inManip(?obj ?g)
mid ($\forall$?c-can, ?l-loc $\neg$ obstructs(?c, ?l ?rp$_t$))
post robotAt(?rp2)

The MOVEWITHOBJ action is similar to the move action. The primary difference is that the preconditions require that the robot be holding an object and that said object remain rigidly attached to the robot.

MOVEWITHOBJ(?rp1-pose ?rp2-pose ?obj-can ?g-grasp)
pre robotAt(?rp1) $\wedge$ inManip(?obj ?g)
mid ($\forall$?c-can, ?l-loc $\neg$ obstructs(?c, ?l ?rp$_t$))
$\wedge$ inManip(?obj ?g)
post robotAt(?rp2)

The final two actions pickup objects from locations and put them down. They only consist of a single timestep, so they have no mid-conditions. In order to pick up an object, the robot must be holding nothing and be next to the object. To put an object down it must be currently held and the robot has to be in the appropriate relative location.

PICK(?obj-can ?l-loc ?rp-pose ?g-grasp)
pre robotAt(?rp) $\wedge$ objAt(?obj ?l)
$\wedge$ ($\forall$?c-can, ?g-grasp $\neg$ inManip(?c ?g)
$\wedge$ isGraspPose(?obj ?rp ?l)
mid $\emptyset$
post inManip(?obj ?g)

PLACE(?obj-can ?l-loc ?rp-pose ?g-grasp)
pre robotAt(?rp) $\wedge$ inManip(?obj ?g)
$\wedge$ isGraspPose(?obj ?rp ?l)
mid $\emptyset$
post $\neg$ inMaip(?obj ?g) $\wedge$ objAt(?obj ?l)

4 TASK PLAN OPTIMIZATION

A common operation in task and motion planning is *plan refinement*. This is the process of converted a partially specified abstract plan into a fully specified trajectory. We focus on a special case of plan refinement where all discrete variables are fixed by the task plan. This is a common type of abstract plan that is used in, e.g., (Toussaint 2015),(Lozano-Pérez and Kaelbling 2014), (Chitnis et al. 2016), (Lagriffoul et al. 2014).

First, we describe how our formulation of task and motion planning encodes a joint trajectory optimization over intermediate states and plan parameters. Then, we discuss our trajectory initialization and reuse schemes. These are important in light of the size and non-convexity of the trajectory optimization problems we consider. We show how the movement primitives of (Dragan et al. 2015) can be used to leverage previous solutions to guide initialization. Finally, we give an algorithm for early detection of infeasibility. This is crucial for task and motion planning, because it is important to fail fast if no motion planning solution exists.

Abstract Plans Encode Trajectory Optimizations

Abstract plans in our formulation encode trajectory optimizations. While we are not the first to apply this idea, our approach has a precise and explicit connection between a task plan and its corresponding trajectory optimization.

Before describing the optimization formulation in general, we go through an example from the CL-2D-NAMO domain.

Example: Trajectory Optimization for a Pick-Place Consider an abstract task plan for the CL-2D-NAMO domain.

- MOVE(rp$_{init}$ gp$_1$)
- PICK(can$_1$ c1$_{init}$ gp$_1$ g$_1$)
- MOVEWITHOBJ(gp$_1$ pdp$_1$ can$_1$ g$_1$)
- PLACE(can$_1$ c1$_{goal}$ pdp$_1$ g$_1$)

This plan moves to a grasping pose for can$_1$, picks up can$_1$, moves to a goal location, and then places the object at the goal. The parameters plan refinement determines are the continuous action parameters: the grasping pose, gp$_1$; the grasp to use, g$_1$; and the putdown pose, pdp$_1$.

Setting the values for these parameters defines the intermediate states in the plan, so these variables are directly constrained by the pre-conditions and post-conditions of actions in the plan.

Next, we need to find trajectories through the state space that connect these intermediate states. The variables in the trajectory optimization will be a sequence of world states. We fully determine the world state by setting a value for each primitive predicate, so we optimize over the continuous parameters for a sequence of primitive predicates, subject to the mid-conditions from the high-level action and dynamics constraints. This results in the following trajectory optimization:

$$\min_{gp_1, g_1, pdp_1, \tau^0, \tau^2} \quad \sum ||\tau_t^0 - \tau_{t+1}^0||^2 + \sum ||\tau_t^2 - \tau_{t+1}^2||^2.$$

subject to

$$\tau_0^0 = rp_{init}, \tau_T^0 = gp_1$$
$$\tau_0^2 = gp_1, \tau_T^2 = pdp_1$$
$$|\tau_t^0 - \tau_{t+1}^0| \leq \delta$$
$$|\tau_t^2 - \tau_{t+1}^2| \leq \delta$$
$$\forall o \in O \; SD(\tau_t^0, o) \geq d_{safe}$$
$$\forall o \in O \; SD(\tau_t^2, o) \geq d_{safe}$$
$$isGraspPose(can_1, c1_{init}, gp_1)$$
$$isGraspPose(can_1, c1_{goal}, pdp_1)$$
$$inManip(can_1, g_1)$$

The constraints on the start and end of the trajectories come from the robotAt preconditions. The final inManip constraint holds for every state in τ^2. Each constraint defined above is either linear or a signed distance constraint. This means that the problem is suitable for the sequential quadratic programming approach described in Section 2.

Converting a General Abstract Plan to a Trajectory Optimization To translate a general high-level action A(p_1, p_2, ...) we apply the following sequence of steps. First, determine the parameters in the high-level action that are not set. Second, determine the variables for a trajectory for this action. In our formulation, these are defined by the set of primitive predicates. In the CL-2D-NAMO domain, this adds variables for robot poses and object locations.

Now that we have a set of variables, we can add in constraints. We iterate through A's pre-conditions. We add them as constraints on the parameters of the action and the first state in the trajectory. We repeat that process with the post-conditions and the last state in the trajectory. Finally, we add A's mid-conditions as constraints on each intermediate step of the trajectory. Algorithm 2 shows pseudocode to set up and refine this trajectory optimization.

The sequential quadratic programming approach that we use is a local improvement algorithm, so good initialization leads to faster convergence. In trajectory optimization bad initializations often fail to converge, even when a solution exists. This is a difficult challenge in regular trajectory optimization and trajectories considered here are substantially longer than those considered in typical motion planning.

To deal with this challenge, we use the structure of our formulation to help guide search. We define a distribution over continuous values for each parameter type, called a generator (Kaelbling and Lozano-Pérez 2011). Our first step in initialization uses these generators to obtain initial values for each parameter. After, we need to initialize trajectories and make sure the the parameters are self-consistent. We do this with an optimization that considers the trajectory costs but only includes constraints at end states. Finally, we add in all constraints and optimize the full problem.

Algorithm 2 Refining an Abstract Task Plan

```
Define: PLANOPT(π)
Input: partially specified abstract plan π.
/* iterate through high-level actions in the plan */
for a ∈ π.ops do
    params = GetVariables(a)
    for p ∈ a.preconditions do
        p.AddConstraint(params, τ_1^a)
    end for
    for p ∈ a.postconditions do
        p.AddConstraint(params, τ_T^a)
    end for
    for p ∈ a.midconditions do
        for t = 2, ..., T − 1 do
            p.AddConstraint(params, τ_t^a)
        end for
    end for
end for
/* call SQP to optimize all the τ^a */
```

Trajectory Reuse

Often, the first attempt at refinement fails to converge. Figure 3 (a) shows an example of one such trajectory. The initial grasp pose was sampled on the wrong side of the object, so it is unreachable. At this point, we want to use a randomized restart to try to find a solution. However, completely starting over from scratch as in Figure 3 (b) is undesirable because we through away a lot of information. In particular, the previous trajectory has figured out that it should go around the corner, not through it. The optimization can figure this out again, but it will require a lot of collision checks and will increase the total time. This problem gets much worse with very long plans (e.g., 20 different move actions). If a single action has no feasible trajectory, we shouldn't restart motion planning all of them from scratch.

What we would prefer to do is only re-initialize the variables in violated constraints. This often fails because the rest of the plan has too much 'inertia:' it has already settled into a local optimum and so the first step of the optimization simple moves the re-initialized variables back to their previous (infeasible) values.

A solution to this fixes the parameters to their new sampled initializations and then minimizes the norm of the difference between the new trajectory and the previous trajectory. This projects the previous trajectory into the set of trajectories consistent with the newly sampled parameters and propagates these changes to the rest of the plan.

This is on the right track, but it is important to choose the projection correctly. Figure 3 (c) shows what happens if this projection is performed under an ℓ_2-norm. Although some of the trajectory moves to account for the new parameters, enough of it is stuck behind the object that the optimization is still stuck in the same basin of attraction.

(Dragan et al. 2015) formulate movement primitives as projections under different norms in a Hilbert space of trajectories. We adopt their approach and use a *minimum velocity* norm to project old trajectories onto new initializations. This is shown in Figure 3 (d). We can see that the new trajectory maintains the qualitative structure of the previous solution (and so avoids collisions) and naturally moves to the

new pick pose.

Early Detection of Unsatisfiability

With long task plans, it is important that the optimization fail fast. Very often an optimization quickly determines that a constraint is infeasible and converges for that constraint. However, the rest of the plan may still be very far from a local optimum. Thus, a vanilla implementation of the convergence check may spend a large number of extra QP minimizations and collision checks optimizing a plan that we know to already be infeasible!

In SQP, one convergence test checks that approximate improvement in the objective value is above a threshold. This is the improvement we make during a QP solve, but measured with respect to the convex approximation. If the approximate improvement is small we know that there is very little room to improve on our current solution with respect to the real objective.

Our approach is to check this convergence constraint independently for each constraint. We terminate the optimization early if the following conditions are met: 1) there is a constraint that is currently unsatisfied; 2) the approximate improvement on the constraint's infeasibility is below a threshold; 3) any constraints that share variables with this constraint are satisfied or have a low approximate improvement. The first two conditions extend the standard convergence criterion to a per-constraint criterion. The final condition is there because sometimes we fail to make progress on a constraint, not because it is infeasible, but because the optimization chose to allocate its effort to a different (coupled) constraint.

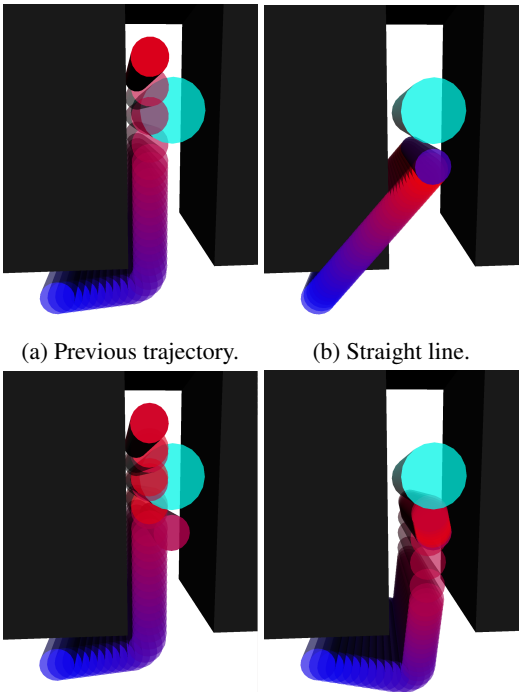
5 EXPERIMENTS

Methodology

We evaluate our approach in the NAMO domain with two distinct experimental setups: the *swap* task and the *putaway* task. In the swap task, there are two objects inside the closet. The robot must reverse the positions of both objects. This requires reasoning about obstructions and proper plan ordering. In the putaway task, two target objects are located among several obstructions in the room. The robot must retrieve the two objects and place them both anywhere inside the closet. An important aspect of this task is that once one object is placed inside the closet, the robot cannot navigate behind it to the place the other. We run experiments for this task with 0, 3, and 5 obstructing objects.

We compare performance with the backtracking baseline established in (Chitnis et al. 2016), which performs exhaustive backtracking search over plan parameters. We implement the motion planning for this method by applying SQP to each action independently.

Manipulated Variables. We perform two experiments. Experiment 1 compares the performance of four systems: the backtracking baseline (B), standard SQP (S), SQP with our early convergence criteria (E), and standard SQP initialized using the solution found by backtracking (T). There are two manipulated variables in this experiment: which of these systems is run, and which experimental scenario we test on (swap or putaway with 0, 3, or 5 obstructions).



(a) Previous trajectory. (b) Straight line.

(c) ℓ_2-norm. (d) Minimum velocity.

Figure 3: The plotted robot trajectory starts blue and transitions to red. Since the trajectory (a) has collisions, the robot end pose is resampled. (b) initializes with a straight line trajectory, and needs to rediscover the path around the wall. (c) initializes with a trajectory that minimizes ℓ_2-norm to the previous trajectory. This preserves the previous solution but doesn't change the trajectory enough to get to a new basin of attraction. The minimum-velocity trajectory (d) adapts to the new endpoint but reuses information from the previous trajectory.

Experiment 2 considers the effects of different types of trajectory reuse on each of our novel systems, S and E. There are two manipulated variables in this experiment: which system is run (S or E), and which trajectory reuse strategy we use. We consider three such strategies: 1) straight-line initialization (i.e., ignore previous trajectories), 2) ℓ_2-norm minimization (i.e., stay as close as possible to previous trajectories), and 3) minimum-velocity ℓ_2-norm minimization (i.e., stay close to a linear transformation of the trajectory). Our experiments reveal that minimum-velocity ℓ_2-norm minimization worked best, so Experiment 1 uses this technique.

Dependent Measures. We measure success rate, the sum of squared velocities on the trajectories, planning time, and number of new task plans generated.

Problem Distributions. Experiment 1 is evaluated on fixed test sets of 50 randomly generated environments. Environments for the putaway task are generated by randomly spawning N objects within the room and designating two as the targets. Experiment 2 is evaluated on a smaller test set of

Condition	% Solved	Traj Cost	Time (s)	# Replans
Swap, B	100	42.4	37.7	5.0
Swap, S	100	10.2	267.2	6.0
Swap, E	100	10.4	217.8	14.4
Swap, T	100	10.9	115.1	5.0
P(0), B	100	12.2	16.8	3.2
P(0), S	100	7.7	21.1	1.8
P(0), E	100	7.7	23.9	2.3
P(0), T	100	7.8	20.7	3.2
P(3), B	98	16.9	58.8	4.9
P(3), S	96	8.9	109.4	3.9
P(3), E	98	9.1	101.1	4.3
P(3), T	98	9.1	76.5	5.1
P(5), B	86	21.3	91.4	8.4
P(5), S	83	9.7	154.4	5.4
P(5), E	88	9.7	160.3	6.8
P(5), T	94	11.0	135.0	7.3

Table 1: Success rate, average trajectory cost, average total time, and average number of calls to task planner for each system in each experimental scenario. P indicates a putaway task. The number in parentheses is the number of obstructions. B: backtracking baseline. S: standard SQP. E: SQP with early convergence criteria. T: SQP with initialization from B. Results are obtained based on performance on fixed test sets of 50 randomly generated environments. All failures were due to timeout: we gave 1200 seconds for each swap task problem and 600 seconds for each putaway task problem.

30 environments for the swap task.

Our experiments are conducted in Python 2.7 using the OpenRave simulator (Diankov and Kuffner 2008). Our task planner is Fast-Forward (Hoffmann 2001). Experiments were carried out in series on an Intel Core i7-4770K machine with 16GB RAM. The time limit was set to 1200 seconds for the swap task and 600 seconds for the putaway task. Table 1 summarizes results for Experiment 1, and Table 2 summarizes results for Experiment 2.

Discussion

As our intuition suggests, Experiment 2 shows that trajectory reuse with minimum-velocity projection outperforms standard ℓ_2 projection and straight-line initialization. ℓ_2 projection performs quite poorly because it gets trapped in bad local optima frequently.

Experiment 1 shows that full joint optimization (systems S and E) over plan parameters leads to significant improvements in overall trajectory cost versus backtracking. Although this comes at the expense of increased running time. System T, which first performs backtracking, then initializes SQP using the solution, shows that the running time can be cut down significantly by combining these approaches.

Condition	% Solved	Traj Cost	Time (s)	# Replans
SL, S	100	10.7	338.6	9.4
SL, E	100	10.8	217.6	9.4
ℓ_2-norm, S	63	10.4	336.1	9.5
ℓ_2-norm, E	67	11.0	181.0	13.6
Min-V, S	100	10.0	247.3	5.7
Min-V, E	100	10.4	200.7	13.1

Table 2: Success rate, average trajectory cost, average total time, and average number of calls to task planner for several systems. SL indicates straight-line initialization; ℓ_2-norm and Min-V use an ℓ_2 or minimum velocity projection to initialize; S denotes standard SQ; E denotes SQP with early convergence criteria. Results are obtained based on performance on fixed test sets of 30 environments. All failures were due to timing out the 1200 second limit.

6 RELATED WORK

Related work for this paper largely comes from plan-skeleton approaches to task and motion planning. These are approaches that search over a purely discrete representation of the problem and then attempt to refine the task plans they obtain.

Toussaint (Toussaint 2015) also considers joint trajectory optimization to refine an abstract plan. In his formulation, the symbolic state from a task plan defines constraints on a trajectory optimization. The system optimizes jointly over parameters and use an initialization scheme similar to ours. However, the problems he considers are difficult because the intermediate states are complicated structures. In our experiments, difficulty largely stems instead from motion planning infeasibility. This leads us to focus on trajectory re-use with movement primitives and early convergence detection.

Lozano-Pérez and Kaelbling (Lozano-Pérez and Kaelbling 2014) consider a similar approach. They enumerate plans that could possibly achieve a goal. For each such abstract plan, they discretize the parameters in the plan and formulate a discrete constraint satisfaction problem. They use an off-the-shelf CSP solver to find a trajectory consistent with the constraints imposed by the abstract plan. Our approach to refinement draws on this perspective, but we do not discretize the plan parameters; instead, we use continuous optimization to set them.

Lagriffoul and Andres (Lagriffoul et al. 2014) define the fluents in their task planning formulation in a similar way to ours. They use these constraint definitions to solve a linear program over the plan parameters. They then use this LP to reduce the effort of a backtracking search for plan refinement. This is similar to the first initialization step that we and (Toussaint 2015) use, in that it only considers the intermediate states.

References

Chitnis, R.; Hadfield-Menell, D.; Gupta, A.; Srivastava, S.; Groshev, E.; Lin, C.; and Abbeel, P. 2016. Guided search for task and motions plans using learning heuristics. In *IEEE Conference on Robotics and Automation (ICRA)*.

Cohen, B. J.; Chitta, S.; and Likhachev, M. 2010. Search-based planning for manipulation with motion primitives. In *International Conference on Robotics and Automation (ICRA)*, 2902–2908. IEEE.

Diankov, R., and Kuffner, J. 2008. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA.

Dragan, A. D.; Muelling, K.; Bagnell, J. A.; and Srinivasa, S. S. 2015. Movement primitives via optimization. In *International Conference on Robotics and Automation (ICRA)*, 2339–2346. IEEE.

Garrett, C. R.; Lozano-Perez, T.; and Kaelbling, L. P. 2015. Backward-forward search for manipulation planning. In *International Conference on Intelligent Robots and Systems (IROS)*, 6366–6373. IEEE.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI Magazine* 22:57–62.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation (ICRA)*.

Kavraki, L.; Svestka, P.; Latombe, J.; and Overmars, M. 1994. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical report, Stanford, CA, USA.

Lagriffoul, F.; Dimitrov, D.; Bidot, J.; Saffiotti, A.; and Karlsson, L. 2014. Efficiently combining task and motion planning using geometric constraints. In *International Conference on Robotics and Automation (ICRA)*.

Lavalle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Technical report.

Lozano-Pérez, T., and Kaelbling, L. P. 2014. A constraint-based method for solving sequential manipulation planning problems. In *International Conference on Intelligent Robots and Systems (IROS)*.

Nocedal, J., and Wright, S. 2006. *Numerical optimization*. Springer Science & Business Media.

Ratliff, N.; Zucker, M.; Bagnell, J. A.; and Srinivasa, S. 2009. Chomp: Gradient optimization techniques for efficient motion planning. In *International Conference on Robotics and Automation (ICRA)*.

Schulman, J. D.; Ho, J.; Lee, A.; Awwal, I.; Bradlow, H.; and Abbeel, P. 2013. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Proceedings of Robotics: Science and Systems (RSS)*.

Stilman, M., and Kuffner, J. 2008. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research* 27(11-12):1295–1307.

Toussaint, M. 2015. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Discovering Domain Axioms Using Relational Reinforcement Learning and Declarative Programming

Mohan Sridharan
Electrical and Computer Engineering
The University of Auckland, NZ
m.sridharan@auckland.ac.nz

Prashanth Devarakonda
Electrical and Computer Engineering
The University of Auckland, NZ
vdev818@auckland.ac.nz

Rashmica Gupta
Electrical and Computer Engineering
The University of Auckland, NZ
rashmicy@gmail.com

Abstract

This paper presents an architecture that integrates declarative programming and relational reinforcement learning to support incremental and interactive discovery of previously unknown axioms governing domain dynamics. Specifically, Answer Set Prolog (ASP), a declarative programming paradigm, is used to represent and reason with incomplete commonsense domain knowledge. For any given goal, any unexplained failure of plans created by inference in the ASP program is taken to indicate the existence of unknown domain axioms. The task of discovering these axioms is formulated as a Reinforcement Learning problem, and decision-tree regression with a relational representation is used to incrementally generalize from specific axioms identified over time. These new axioms are added to the ASP program for subsequent inference. We demonstrate and evaluate the capabilities of our architecture in two simulated domains: *Blocks World* and *Simple Mario*.

1 Introduction

Robots[1] assisting humans in complex domains such as health care and disaster rescue, frequently find it difficult to operate without considerable domain knowledge. At the same time, humans interacting with the robots may not have the expertise or time to provide elaborate and accurate domain knowledge. Robots are likely to receive some commonsense domain knowledge, including *default* knowledge that holds in all but a few exceptional situations, e.g., "books are typically in the library, but cookbooks are in the kitchen". Robots also receive information by processing sensor inputs, and the reliability of this information may be expressed probabilistically, e.g., "I am 90% sure the robotics book is in the library". Furthermore, not all the axioms governing domain dynamics may be known, and the known axioms may need to be revised over time. For instance, if the floor of a room has just been polished, and the robot does not have an accurate model of the effects of executing movement actions on this surface, plan execution may not produce the desired outcome. To truly assist humans in such domains, robots thus need to represent, reason with, and learn from, different descriptions of knowledge and uncertainty at both the cognitive level and the sensorimotor level.

[1]We use terms "robot", "agent" and "learner" interchangeably.

Our prior work designed architectures that combined the non-monotonic logical reasoning capabilities of declarative programming with the uncertainty modeling capabilities of probabilistic graphical models, to address the planning and diagnostics challenges summarized above (Colaco and Sridharan 2015; Zhang et al. 2014; Zhang, Sridharan, and Wyatt 2015). The architecture described in this paper builds on prior work (Sridharan and Rainge 2014) to support incremental and interactive discovery of previously unknown domain axioms (Sridharan, Devarakonda, and Gupta 2016). Similar to our prior work, an action language is used to describe the known causal laws, state constraints and executability conditions. This description and initial state defaults are translated to an Answer Set Prolog (ASP) program that is solved for planning and diagnostics. Focusing on the ability to discover axioms, we abstract away the uncertainty in perception, and make the following contributions:

- For any given goal, unexplained plan failures are taken to indicate the existence of unknown domain axioms. Incremental and interactive discovery of these axioms is formulated as a reinforcement learning problem.
- A relational representation is used to improve computational efficiency, and to generalize from axioms identified through reinforcement learning. These newly discovered axioms are used for subsequent ASP-based reasoning.

These capabilities are evaluated experimentally in two domains, *Blocks World* and *Simple Mario*, to demonstrate reliable and efficient discovery of domain axioms. Section 2 discusses prior work and some background material, followed by a description of the problem and the architecture in Section 3. The experimental results are discussed in Section 4, followed by conclusions in Section 5.

2 Related Work

In this section, we motivate the proposed approach by reviewing some related work. We also provide some background information about ASP, reinforcement learning and relational representations, and their use on robots.

Probabilistic graphical models are used widely to formulate planning, sensing, navigation, and interaction, on robots (Bai, Hsu, and Lee 2014; Hoey et al. 2010), but these formulations, by themselves, make it difficult to reason with commonsense knowledge. Research in planning has provided many algorithms for knowledge repre-

sentation and reasoning on robots (Galindo et al. 2008; Varadarajan and Vincze 2011), but these algorithms require a lot of prior knowledge about the domain. Many of these algorithms are based on first-order logic, and do not support non-monotonic logical reasoning, default reasoning, and the ability to merge new, unreliable information with the current beliefs. Other logic-based formalisms address some of these limitations, e.g., Answer Set Prolog (ASP), a declarative language designed for representing and reasoning with commonsense knowledge (Gelfond and Kahl 2014), has been used by an international research community for cognitive robotics applications (Balduccini, Regli, and Nguyen 2014; Erdem and Patoglu 2012). However, ASP does not inherently support probabilistic models of uncertainty, or incremental and interactive learning of domain knowledge.

Combining logical and probabilistic reasoning capabilities is a fundamental problem in robotics and AI. Architectures have been developed to support hierarchical representation of knowledge in first-order logic, and probabilistic processing of perceptual information (Laird 2008; Talamadupula et al. 2010). Existing approaches have combined deterministic and probabilistic algorithms for task and motion planning (Kaelbling and Lozano-Perez 2013; Saribatur, Erdem, and Patoglu 2014), switched between probabilistic reasoning and first-order logic based on degrees of belief to use semantic maps and commonsense knowledge in a probabilistic relational representation (Hanheide et al. 2011), and used a three-layered organization of knowledge with first-order logic and probabilistic reasoning for open world planning (Hanheide et al. 2015). Other approaches for combining logical and probabilistic reasoning include Markov logic networks (Richardson and Domingos 2006), Bayesian Logic (Milch et al. 2006), probabilistic first-order logic (Halpern 2003), first-order relational POMDPs (Sanner and Kersting 2010), and probabilistic extensions to ASP (Baral, Gelfond, and Rushton 2009; Lee and Wang 2015). Many of these algorithms are based on first-order logic, and have the corresponding limitations, e.g., it is not always possible to express degrees of belief and uncertainty quantitatively. Other algorithms based on logic programming do not support all desired capabilities such as reasoning with large probabilistic components; dynamic addition of variables with different ranges to represent open worlds; and incremental and interactive learning of previously unknown domain knowledge.

Many tasks that require the agent to learn from repeated interactions with the environment have been posed as Reinforcement Learning (RL) problems (Sutton and Barto 1998) and modeled as Markov Decision Processes (MDPs). It is challenging to design algorithms that scale to large, complex domains, and allow a transfer of learned knowledge between related domains. Relational reinforcement Learning (RRL) combines relational representations of states and actions with relational regression for Q-function generalization (Dzeroski, Raedt, and Driessens 2001). The RRL formulation enables the use of structural similarities, and the reuse of experience in a subset of the state-action space when operating in related regions of the state-action space (Tadepalli, Givan, and Driessens 2004). These existing approaches, however, use RRL for planning, and generalization, e.g., using function approximation (Driessens and Ramon 2003; Gartner, Driessens, and Ramon 2003) or explanation-based RL (Boutilier, Reiter, and Price 2001), is limited to a single MDP corresponding to a specific planning task. Furthermore, these approaches do not fully support the desired commonsense reasoning capabilities for robots.

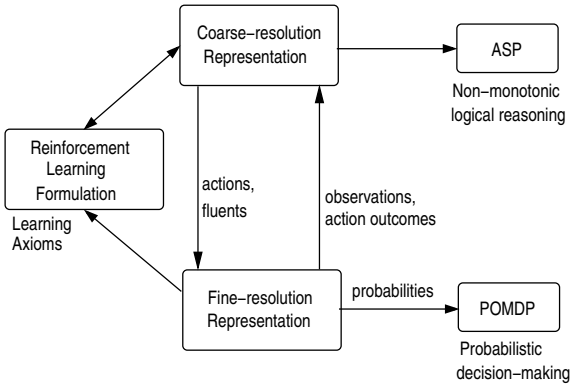


Figure 1: Architecture integrates the complementary strengths of declarative programming, probabilistic graphical models, and reinforcement learning, for knowledge representation, reasoning, and learning.

We have designed architectures that combine the complementary strengths of declarative programming and probabilistic graphical models for planning and diagnosis in robotics (Colaco and Sridharan 2015; Zhang et al. 2014; Zhang, Sridharan, and Wyatt 2015). In this paper, we abstract away the unreliability of perception, and combine declarative programming with RRL for incrementally and interactively discovering axioms, and generalizing across individual axiom instances. Unlike prior work that used inductive logic and ASP to monotonically learn causal rules (Otero 2003), or integrated ASP with RL for discovering domain axioms (Sridharan and Rainge 2014), we use relational representation for generalization. Unlike existing work in RRL, our approach uses the relational representation for discovering previously unknown domain axioms, and generalizes across different MDPs, i.e., different decision making tasks in the domain.

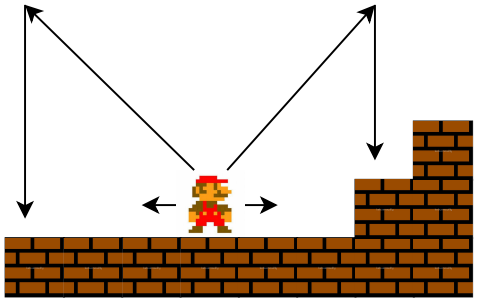
3 Proposed Architecture

The overall architecture is shown in Figure 1. In this paper, we abstract away the uncertainty in perception for simplicity, and do not discuss probabilistic planning. The focus is on ASP-based reasoning with commonsense knowledge for planning and diagnostics (Section 3.1), and RRL-based interactive discovery of domain axioms (Section 3.2)[2].

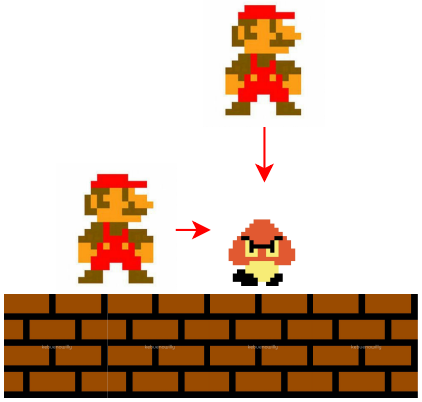
We illustrate the capabilities of this architecture using two simulated domains:

[2]Although our overall architecture shown in Figure 1 represents the domain at two different resolutions, in this paper we consider the domain representation at a single resolution for simplicity.

Figure 2: Blocks world scenario with four blocks.



(a) Actions



(b) Monster collision

Figure 3: Figure demonstrating safe and unsafe actions in the Simple Mario domain.

1. **Blocks World (BW):** a tabletop domain where the objective is to stack blocks of different colors, shapes, and sizes, in specific configurations. Figure 2 illustrates a scenario with four blocks, which corresponds to ≈ 70 states under a standard RL/MDP formulation (Dzeroski, Raedt, and Driessens 2001). The robot may not know, for instance, that a block should not be placed on a prism-shaped block, and thus the corresponding action should not be attempted.
2. **Simple Mario (SM):** a simplified version of the Mario game, where the agent (*mario*) has to navigate between locations while avoiding obstacles and hazards. The domain has ≈ 80 states and 4 actions in a standard MDP formulation. Safe actions are moving or jumping left or right (Figure 3(a)), while unsafe actions include collision with a monster (Figure 3(b)), landing on spikes (Figure 4(a)), and landing on empty space (Figure 4(b)).

3.1 Knowledge Representation

The transition diagram of our illustrative domain is described in an *action language* AL (Gelfond and Kahl 2014). Action languages are formal models of parts of natural language used for describing transition diagrams. AL has a sorted signature containing three *sorts*: *statics*, *fluents* and *actions*. Statics are domain properties whose truth values cannot be changed by actions, while fluents are properties whose truth values are changed by actions. Actions are defined as a set of elementary actions that can be executed in parallel. A domain property p or its negation $\neg p$ is a domain literal. AL allows three types of statements:

a **causes** l_{in} **if** $p_0, \ldots, p_m$ (Causal law)
l **if** $p_0, \ldots, p_m$ (State constraint)
impossible $a_0, \ldots, a_k$ **if** $p_0, \ldots, p_m$ (Executability condition)

where a is an action, l is a literal, l_{in} is an inertial fluent literal, and $p_0, \ldots, p_m$ are domain literals. A collection of statements of AL forms a system description.

The domain representation consists of a system description $\mathscr{D}$ and history $\mathscr{H}$. $\mathscr{D}$ has a sorted signature and axioms used to describe the transition diagram τ. The sorted signature is a tuple that defines the names of objects, functions, and predicates available for use in the domain. The sorts of the BW domain include elements such as *block*, *place*, *color*, *shape*, *size*, and *robot*, whereas the sorts of the SM domain include elements such as *location*, *block*, *material*, *size*, *direction* and *thing*—when some sorts are subsorts of other sorts, e.g., *agent* (i.e., *mario*) and *monster* may be subsorts of *thing*, they can be arranged hierarchically.

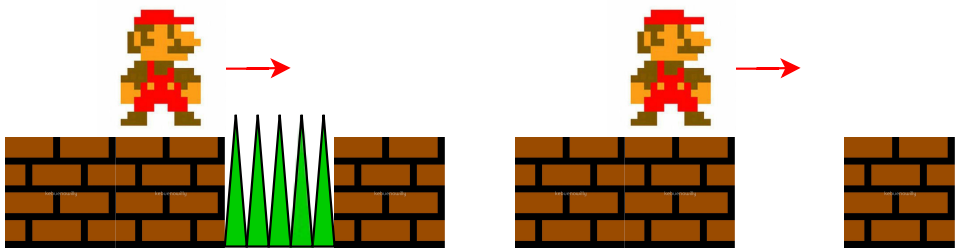
We describe the fluents and actions of the domain in terms of the sorts of their arguments. The BW domain's fluent $on(block, place)$ describes the place location of each block—this is an inertial fluent that obeys the laws of inertia. There are some statics for block attributes $has\_color(block, color)$, $has\_shape(block, shape)$ and $has\_size(block, size)$. The action $move(block, place)$ moves a block to a specific place (*table* or another block). In the SM domain, the fluents are the location of *mario* and the *monster* (assuming there is only one monster)—we reason about the former and assume the latter is a defined fluent known at all times, i.e., the inertial fluent is $loc(agent, block)$. Mario can move to the left or the right by one position, or jump to the left or right by up to three positions, which are represented as actions $move(mario, dir)$ and $jump(mario, dir, numpos)$ with direction $(left, right)$ and number of positions $(1, 2, 3)$ as arguments. We also introduce relations for block attributes, e.g., $has\_material(block, material)$, and location attributes, e.g., $right\_of(block, block)$.

For the BW domain, the dynamics are defined in terms of causal laws such as:

$$move(b_1, loc_1) \textbf{ causes } on(b_1, loc_1)$$

state constraints such as:

$$\neg on(b_1, loc_1) \textbf{ if } on(b_1, loc_2),\ loc_1 \neq loc_2$$



(a) Spike block (b) Air block

Figure 4: Snapshot of basic causes of death (i.e., episode termination) in the Simple Mario domain.

and executability conditions such as:

$$\textbf{impossible } move(b_1, loc_1) \textbf{ if } on(b_2, loc_1),\ b_2 \neq b_1$$

The SM domain's dynamics are defined using causal laws such as:

$$move(mario, right) \textbf{ causes } loc(mario, b_2),\ right\_of(b_2, b_1),\ loc(mario, b_1)$$

state constraints such as:

$$\neg loc(mario, b_2) \textbf{ if } loc(mario, b_1),\ b_1 \neq b_2$$

and executability conditions such as:

$$\textbf{impossible } move(mario, right) \textbf{ if } loc(mario, b_1),\ right\_of(b_2, b_1),\ has\_material(b_2, spike)$$

The recorded history of a dynamic domain is usually a record of (a) fluents observed to be true at a time step $obs(fluent, boolean, step)$, and (b) the occurrence of an action at a time step $hpd(action, step)$. Our architecture expanded on this view by allowing histories to contain (prioritized) defaults describing the values of fluents in their initial states (Zhang et al. 2014; Sridharan et al. 2015). For instance, we can represent a default statement of the form "blocks are usually on the table or on another block that is on the table" and elegantly encode exceptions to such default statements.

The domain representation is translated into a CR-Prolog[3] program $\Pi(\mathscr{D}, \mathscr{H})$, i.e., a collection of statements describing domain objects and relations between them. This program incorporates consistency restoring (CR) rules in ASP (Gelfond and Kahl 2014). ASP is based on stable model semantics and non-monotonic logics, and includes *default negation* and *epistemic disjunction*, e.g., unlike $\neg a$ that states *a is believed to be false*, *not a* only implies *a is not believed to be true*, and unlike "$p \vee \neg p$" in propositional logic, "*p or* $\neg p$" is not a tautology. ASP can represent recursive definitions, defaults, causal relations, and constructs that are difficult to express in classical logic formalisms. The ground literals in an *answer set* obtained by solving Π represent beliefs of an agent associated with Π. Algorithms for computing the entailment, and for planning and diagnostics, reduce these tasks to computing answer sets of CR-Prolog programs. Π consists of causal laws of $\mathscr{D}$, inertia axioms, closed world assumption for defined fluents, reality checks, and records of observations, actions, and defaults, from $\mathscr{H}$. Every default is turned into an ASP rule and a CR rule that allows the robot to assume, under exceptional circumstances, that the default's conclusion is false, so as to restore program consistency—see (Zhang et al. 2014; Sridharan et al. 2015) for details. Although not discussed here, the program representing the current beliefs of the robot also supports capabilities such as explaining unexpected action outcomes and partial descriptions extracted from sensor inputs—see (Colaco and Sridharan 2015; Zhang et al. 2014).

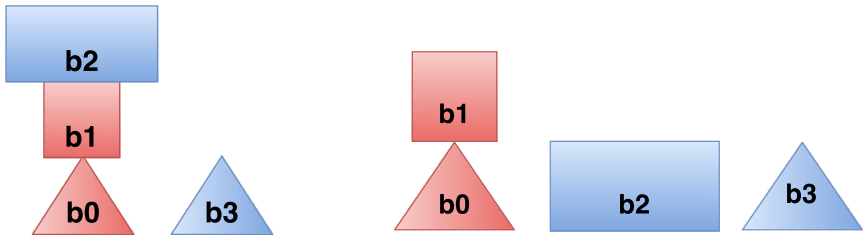
It is difficult to provide complete knowledge about any complex domain. This is especially true of domain axioms that may be unknown or may change over time. The plans created using this incomplete knowledge may result in unintended consequences. Consider a scenario in the BW domain in which the goal is to stack three of four blocks placed on the table. Figure 5(a) shows a possible goal configuration that could be generated based on the available domain knowledge. The corresponding plan (starting with all four blocks on the table) has two steps: $move(b_1, b_0)$ followed by $move(b_2, b_1)$. The robot expects to be able to use this plan to stack the blocks as desired. However, unknown to the robot, no block can be stacked on top of a prism-shaped block in this domain. As a result, execution of this plan results in failure that cannot be explained—specifically, action $move(b_1, b_0)$ does not result in the expected configuration shown in Figure 5(b). In this paper, we focus on discovering previously unknown executability conditions, which can prevent such actions from being included in a plan for any given goal.

3.2 Relational RL for Discovering Axioms

Our approach for incremental and interactive discovery of previously unknown domain axioms differs from previous work by us and other researchers. The proposed approach:

- Explores the existence of previously unknown axioms only when unexpected action outcomes cannot be explained by reasoning about exogenous actions.

[3] We use the terms "ASP" and "CR-Prolog" interchangeably.



(a) Planned goal state (b) Failure in plan execution

Figure 5: Illustrative example of (a) a planned goal state; and (b) failure during a specific step in plan execution.

- Uses RRL and decision tree regression for improving computational efficiency of identifying candidate axioms, and for generalizing from specific axioms.
- Focuses on discovery of unknown axioms by generalizing across multiple MDPs, instead of using RRL for planning, which limits generalization to a specific MDP,

Generalization and computational efficiency are key considerations for incremental and interactive learning. For instance, in the BW domain, discovery of the axiom "a red cube may not be placed on a blue prism", does not help with a *red prism* and *blue cube*, unless the agent realizes, over time, that this axiom is a specific instance of the general axiom "no object should be placed on a prism".

A sequence of steps is used to identify and generalize from candidate axioms. First, when a specific plan step fails, the corresponding state is considered the goal state in an RL problem, with the objective of finding all state-action pairs that lead to this error state. The RL problem uses an MDP formulation and the tuple $\langle S, A, T, R \rangle$, where:

- S: set of states.
- A: set of actions.
- $T : S \times A \times S' \rightarrow [0,1]$ is the state transition function.
- $R : S \times A \times S' \rightarrow \Re$ is the reward function

where T and R are not known in a RL problem. Each state, i.e., each element of S, is the assignment of specific (ground) values to the domain fluents. For instance, if we were to consider two blocks with known color and shape in the BW domain, each state would consider a possible configuration of the blocks—there would be three different states in this example. Similarly, each element of A is a valid action for the domain under consideration. Next, the known axioms in the ASP-based domain description are used to eliminate invalid combinations of states and actions in the domain. For instance, no two objects can be in the same location in either the BW domain or the SM domain. Also, it is not possible to move a block that is under another block in the BW domain, and it is not possible for *mario* to move forward if it is right next to a wall. Inconsistent state transitions are identified by constructing and computing answer sets of ASP programs with the specific state-action combinations and the axioms in $\mathscr{D}$. This "filtering" can significantly reduce the size of the problem because only valid state-action combinations are included as elements of T and R. Furthermore, the MDP is constructed automatically from the ASP system description, for any given goal state.

Popular RL algorithms such as Q-learning or SARSA, which estimate the Q-values of state-action pairs $Q(s,a)$, become computationally intractable as the state space increases in size and do not generalize to relationally equivalent states. The second step uses a relational representation to support generalization. After an episode (i.e., iteration) of Q-learning (with eligibility traces) for a specific goal state, all state-action pairs that have been visited, along with their Q-values, are used to construct a binary (i.e., logical) decision tree (BDT). The path from the root node to any leaf node corresponds to one state-action pair, and individual nodes correspond to specific fluents—the value at the leaf node is the average of the values of all training samples that are grouped under that node. The BDT created after one iteration is used to compute the policy (based on a soft-max function (Sutton and Barto 1998)) in the subsequent episode. When the learning is terminated after convergence of the Q-values or after a specific number of episodes, the BDT relationally represents the experiences of the robot. Figure 6 illustrates a subset of a BDT constructed for the BW domain.

The method described above only considers generalization within a specific MDP. To identify general domain axioms, the third step of our approach simulates similar errors (to the one actually encountered due to plan step execution failure) and considers the corresponding MDPs as well. The Q-value of a state-action pair is now the the weighted average of the values across different MDPs. The weight used is inversely proportional to the shortest distance between the state-action pair and the goal state based on the optimal policy for that MDP, i.e., $w_i = (1/d_i)/\{\sum_{j=0}^{N} 1/d_j\}$, where w_i is the weight of the state-action pair of MDP_i, d_i is the distance from the state-action pair to the goal state of MDP_i, and N is the number of MDPs considered. These similar MDPs are currently chosen randomly—future work will use the information encoded in the ASP program to direct attention to objects and attributes more relevant to the observed failure.

The fourth step identifies candidate executability constraints. The head of such an axiom has a specific action, and the body contains attributes that influence (or are influenced by) the action. We construct training samples by considering each such action and the corresponding attributes

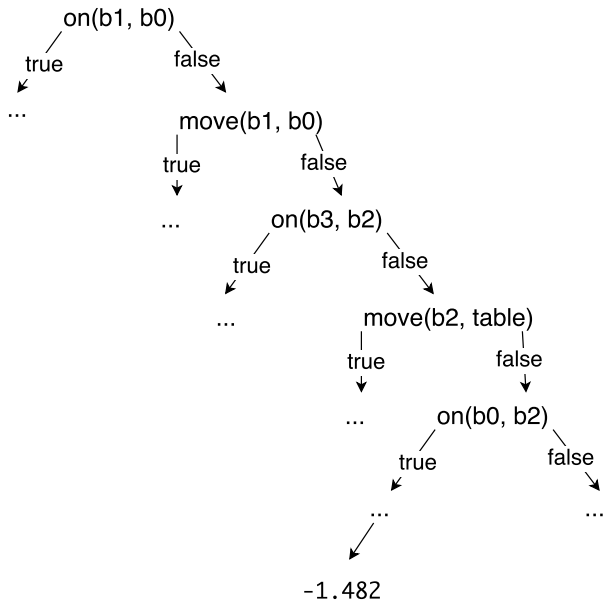


Figure 6: Subset of the binary decision tree for a specific scenario in the blocks world domain.

based on the BDT constructed above. These training samples are used to construct a decision tree whose root node corresponds to non-occurrence of the action, intermediate nodes correspond to attributes of object involved in the action, and the leaf nodes average the values of the training samples grouped under that node. Each path from the root node to a leaf is a candidate axiom with a corresponding value. Figure 7 illustrates a subset of such a tree for a specific action.

The final step considers all candidate axioms for different actions, and uses K-means algorithm to cluster these candidates based on their value. The axioms that fall within the cluster with the largest mean are considered to represent generalized axioms, and are added to the ASP program to be used in the subsequent steps.

4 Experimental Setup and Results

The proposed architecture and algorithms were grounded and experimentally evaluated in the Blocks World and Simple Mario domains. We describe the performance in illustrative execution scenarios drawn from these domains. We compare the rate of convergence of the proposed algorithm for discovering rules, henceforth referred to as "Q-RRL", with traditional Q-learning, using the average Q-value as the performance measure.

4.1 Blocks World

As stated in Section 3, the robot's objective in the BW domain was to stack the blocks in a specified configuration. Consider the experimental trials in which the robot did not know that it was impossible to move any block on top of a prism-shaped block. We considered different scenarios with

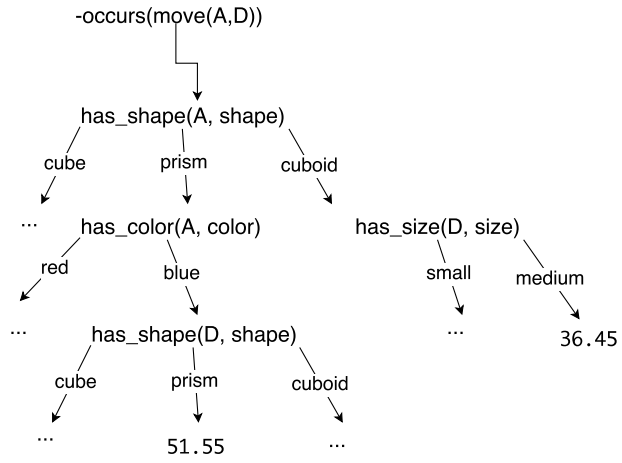


Figure 7: Decision tree representing candidate axioms related to an action in the blocks world domain.

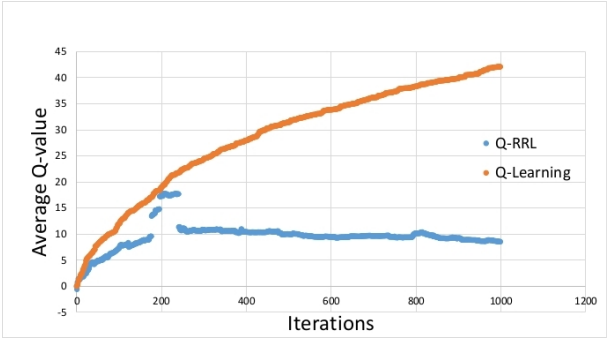


Figure 8: Comparing the rate of convergence of Q-RRL with that of Q-learning in a specific scenario in the BW domain—Q-RRL converges much faster.

blocks of different shapes and colors (but the same size). For instance, in one scenario, the robot was given four blocks: $b0$ (Red Prism); $b1$ (Red Cube); $b2$ (Blue Cuboid); and $b3$ (Blue Prism). All blocks were initially on the table, i.e., the initial state was:

$$on(b0,table),\ on(b1,table)$$
$$on(b2,table),\ on(b3,table)$$

We provided the goal state description as:

$$on(b0,table),\ on(b1,b0)$$
$$on(b2,b1),\ on(b3,table)$$

The plan obtained by solving the ASP program had actions $move(b1,b0)$ and $move(b2,b1)$. The action $move(b1,b0)$ fails. During Q-RRL and Q-Learning, the agent receives a reward of $+100$ when it reaches the goal state and a negative reward of -1.5 otherwise (i.e., for all other actions).

As stated earlier, RRL is triggered when executing the computed plan (to stack the blocks) results in an unexpected outcome than cannot be explained using the existing domain

knowledge. When such an error occurs, different related scenarios are simulated (e.g., different combinations of attributes of the blocks) to generate the training samples for generalization. Figure 8 shows the rate of convergence of the average Q-value obtained using Q-RRL and Q-learning. The Q-RRL algorithm has a much better rate of convergence, i.e., the optimal policy is computed in a much fewer number of iterations. Note we are primarily concerned about the rate of convergence in these experiment trials—it does not matter whether the actual average Q-values of Q-Learning are higher or lower than those of Q-RRL. The following are some axioms identified during the various iterations:

$$\begin{aligned}
-occurs(move(A,D),I) : -&has_shape(D,prism),\\
&has_shape(A,cuboid),\\
&has_color(D,blue)\\
-occurs(move(A,D),I) : -&has_shape(D,prism),\\
&has_shape(A,cube),\\
&has_color(D,red)\\
-occurs(move(A,D),I) : -&has_shape(D,prism),\\
&has_shape(A,prism),\\
&has_color(A,red),\\
&has_color(D,blue)\\
-occurs(move(A,D),I) : -&has_shape(D,prism),\\
&has_shape(A,cube),\\
&has_color(D,blue),\\
&has_color(A,red)
\end{aligned}$$

As the robot explores different scenarios, there are fewer and fewer errors because actions that are impossible are no longer included in the plans that are generated. Furthermore, the robot is able to incrementally generalize from the different specific axioms to finally add the following axiom to the CR-Prolog program:

% Action language description

impossible $move(A,D)$ **if** $has\_shape(D,prism)$

% CR-Prolog statement

$-occurs(move(A,D),I) : -has\_shape(D,prism)$

The proposed architecture resulted in a similar performance in other experimental trials in the BW domain, successfully discovering the corresponding (unknown) domain axioms.

4.2 Simple Mario

In the SM domain, the agent "mario" has to travel to a specific destination, from a starting position. As described at the beginning of Section 3, *move* actions move mario one position away from the current position, while *jump* actions attempt to jump between one to three positions from the current position. If an obstacle is present that prevents *mario* from moving to a certain position, then it lands on the closest (open) position available. Collision with any angry monster in the domain terminates the episode. Furthermore, blocks in the domain are made of different materials—*brick* blocks are harmless, whereas materials such as *spike* or *air* will result in episode termination. The objective is to pick actions

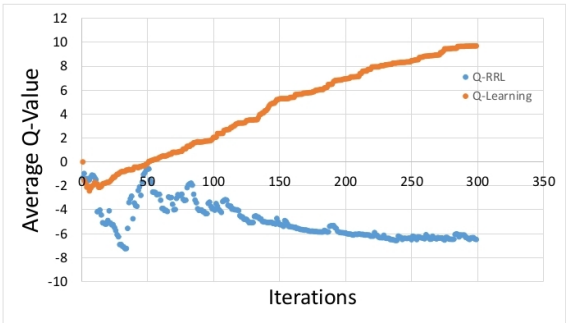


Figure 9: Comparing the rate of convergence of Q-RRL with Q-learning in a specific scenario in the SM domain—Q-RRL converges much faster.

that will not result in episode termination—any such unexpected termination triggers RRL for discovering axioms.

To evaluate the ability to efficiently discover generic domain axioms in the SM domain, we use an approach similar to that used in the BW domain. ASP-based inference is used to compute plan(s) to achieve the desired goal state. If any plan step results in an unexpected failure that cannot be explained using the existing knowledge, scenarios similar to the one causing the failure, e.g., with different block attributes and different locations for *mario* and monsters, are simulated automatically (also see Section 3.2). These simulated scenarios provide the training samples necessary for generalizing from the specific axioms discovered. For instance, in one scenario, three positions that would result in episode termination were used to trigger RRL. The first and second positions involve movement to blocks with *spike* material, while the third position involves collision with an angry monster. Figure 9 compares the rate of convergence of Q-RRL and Q-learning as a function of the number of episodes. Similar to the results obtained in the BW domain, Q-RRL converges a lot faster than Q-learning. Over some episodes, the following are some generalized axioms discovered:

$$\begin{aligned}
-occurs(move(mario,left),I) : -&loc(mario,b_1),\\
&left_of(b_2,b_1),\\
&has_material(b_2,spike)\\
-occurs(move(mario,right),I) : -&loc(mario,b_1),\\
&right_of(b_2,b_1),\\
&has_material(b_2,spike)\\
-occurs(jump(mario,left,1),I) : -&loc(mario,b_1),\\
&left_neighbor(b_2,b_1),\\
&has_material(b_2,spike)\\
-occurs(jump(mario,right,1),I) : -&loc(mario,b_1),\\
&right_neighbor(b_2,b_1),\\
&has_material(b_2,spike)
\end{aligned}$$

The rules generated specify that *mario* cannot execute a move or jump action if this action will lead it to a block with *spike* material. Similar performance was observed in

other scenarios that resulted in the failure of the corresponding plans, with the successful discovery of the corresponding (previously unknown) domain axioms.

5 Conclusion

Robots assisting humans in complex domains frequently need to represent, reason with, and learn from, different descriptions of incomplete domain knowledge and uncertainty. The architecture described in this paper combines the complementary strengths of declarative programming and relational reinforcement learning to discover previously unknown axioms governing domain dynamics. We illustrated the architecture's capabilities in the context of discovering previously unknown executability conditions in two simulated domains (Blocks World and Simple Mario), with promising results. Future work will explore the ability to discover other kinds of axioms in more complex domains. In addition, we will investigate the introduction of probabilistic models of the uncertainty in robot perception, which will transform the underlying decision-making problem from an MDP to a partially observable Markov decision process (POMDP)—this will also enable us to fully utilize the probabilistic planning component of the existing architecture. Furthermore, we will conduct experimental trials on a mobile robot assisting humans in indoor domains.

Acknowledgments

This work was supported in part by the US Office of Naval Research Science of Autonomy award N00014-13-1-0766. All opinions and conclusions expressed in this paper are those of the authors.

References

Bai, H.; Hsu, D.; and Lee, W. S. 2014. Integrated Perception and Planning in the Continuous Space: A POMDP Approach. *International Journal of Robotics Research* 33(8).

Balduccini, M.; Regli, W. C.; and Nguyen, D. N. 2014. An ASP-Based Architecture for Autonomous UAVs in Dynamic Environments: Progress Report. In *International Workshop on Non-Monotonic Reasoning (NMR)*.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1):57–144.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic Dynamic Programming for First-Order MDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 690–700.

Colaco, Z., and Sridharan, M. 2015. What Happened and Why? A Mixed Architecture for Planning and Explanation Generation in Robotics. In *Australasian Conference on Robotics and Automation (ACRA)*.

Driessens, K., and Ramon, J. 2003. Relational Instance-Based Regression for Relational Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 123–130. AAAI Press.

Dzeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational Reinforcement Learning. *Machine Learning* 43:7–52.

Erdem, E., and Patoglu, V. 2012. Applications of Action Languages to Cognitive Robotics. In *Correct Reasoning*. Springer-Verlag.

Galindo, C.; Fernandez-Madrigal, J.-A.; Gonzalez, J.; and Saffioti, A. 2008. Robot Task Planning using Semantic Maps. *Robotics and Autonomous Systems* 56(11):955–966.

Gartner, T.; Driessens, K.; and Ramon, J. 2003. Graph Kernels and Gaussian Processes for Relational Reinforcement Learning. In *International Conference on Inductive Logic Programming (ILP)*, 140–163. Springer.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Halpern, J. Y. 2003. *Reasoning about Uncertainty*. MIT Press.

Hanheide, M.; Gretton, C.; Dearden, R.; Hawes, N.; Wyatt, J.; Pronobis, A.; Aydemir, A.; Gobelbecker, M.; and Zender, H. 2011. Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Hanheide, M.; Gobelbecker, M.; Horn, G.; Pronobis, A.; Sjoo, K.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; Zender, H.; Kruijff, G.-J.; Hawes, N.; and Wyatt, J. 2015. Robot Task Planning and Explanation in Open and Uncertain Worlds. *Artificial Intelligence*.

Hoey, J.; Poupart, P.; Bertoldi, A.; Craig, T.; Boutilier, C.; and Mihailidis, A. 2010. Automated Handwashing Assistance For Persons With Dementia Using Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding* 114(5):503–519.

Kaelbling, L., and Lozano-Perez, T. 2013. Integrated Task and Motion Planning in Belief Space. *International Journal of Robotics Research* 32(9-10):1194–1227.

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. In *International Conference on Artificial General Intelligence*.

Lee, J., and Wang, Y. 2015. A Probabilistic Extension of the Stable Model Semantics. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2006. BLOG: Probabilistic Models with Unknown Objects. In *Statistical Relational Learning*. MIT Press.

Otero, R. P. 2003. Induction of the Effects of Actions by Monotonic Methods. In *International Conference on Inductive Logic Programming*, 299–310.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine Learning* 62(1-2):107–136.

Sanner, S., and Kersting, K. 2010. Symbolic Dynamic Programming for First-order POMDPs. In *AAAI Conference on Artificial Intelligence*, 1140–1146.

Saribatur, Z.; Erdem, E.; and Patoglu, V. 2014. Cognitive Factories with Multiple Teams of Heterogeneous Robots: Hybrid Reasoning for Optimal Feasible Global Plans. In *International Conference on Intelligent Robots and Systems*, 2923–2930.

Sridharan, M., and Rainge, S. 2014. Integrating Reinforcement Learning and Declarative Programming to Learn Causal Laws in Dynamic Domains. In *International Conference on Social Robotics (ICSR)*.

Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. 2015. A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. Technical report, Unrefereed CoRR abstract: `http://arxiv.org/abs/1508.03891`.

Sridharan, M.; Devarakonda, P.; and Gupta, R. 2016. Can I Do That? Discovering Domain Axioms Using Declarative Programming and Relational Reinforcement Learning. In *AAMAS Workshop on Autonomous Robots and Multiagent Systems (ARMS)*.

Sutton, R. L., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational Reinforcement Learning: An Overview. In *Relational Reinforcement Learning Workshop at the International Conference on Machine Learning*.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for Human-Robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology* 1(2):14:1–14:24.

Varadarajan, K. M., and Vincze, M. 2011. Ontological Knowledge Management Framework for Grasping and Manipulation. In *IROS-2011 Workshop on Knowledge Representation for Autonomous Robots*.

Zhang, S.; Sridharan, M.; Gelfond, M.; and Wyatt, J. 2014. Towards An Architecture for Knowledge Representation and Reasoning in Robotics. In *International Conference on Social Robotics (ICSR)*, 400–410.

Zhang, S.; Sridharan, M.; and Wyatt, J. 2015. Mixed Logical Inference and Probabilistic Planning for Robots in Unreliable Worlds. *IEEE Transactions on Robotics* 31(3):699–713.

Preliminary Deployment of a Risk-aware Goal-directed Executive on Autonomous Underwater Glider

Eric Timmons[1,3] and **Tiago Vaquero**[1,2] and **Brian Williams**[1] and **Richard Camilli**[3]
[1] Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, MA 02139
[2] Department of Control and Dynamical Systems
California Institute of Technology, Pasadena, CA 91125
[3] Deep Submergence Laboratory
Woods Hole Oceanographic Institution, Woods Hole, MA 02543
{etimmons,tvaquero,williams}@mit.edu, rcamilli@whoi.edu

Abstract

In this paper we describe a three-layered architecture for resilient control of autonomous systems and present the risk-aware goal-directed executive residing in the top layer of the architecture. The executive combines (1) a hierarchical activity planner that performs goal selection; (2) a generative planner for activities with probabilistic durations; (3) a kino-dynamic path planner that allows a vehicle to traverse an environment with bounded risk of obstacle collision; and (4) an execution monitor. A preliminary, simplified version of the executive has been used to plan for an autonomous underwater glider in the Timor Sea. In the process, the executive managed temporal constraints, the AUV's dynamics, and the lagoon's topology. We conclude with lessons learned that will be of interest to the PlanRob community and a plan forward for the deployment of the full exeuctive.

Introduction

The future of space missions is to visit ever more distant, exotic, and dangerous locations. For example, the recent planetary science decadal survey (Space Studies Board 2012) calls for missions such as a Trojan Asteroid Tour and Rendezvous and the Venus In Situ Explorer. Due to the time criticality of these missions (the high speed involved in orbital maneuvering and Venus' extremely corrosive atmosphere) and the light delay in communicating with Earth, the spacecraft that support these missions will need to be "resilient," capable of reasoning about their own state and the state of the environment in order to predict and avoid hazardous conditions, to recover from internal failures, and to ultimately meet critical science objectives in the presence of substantial uncertainties.

While there have been efforts to achieve resilience using state of the art planning techniques on real-world space missions, the deployed techniques either controlled the spacecraft for a short, well controlled experiment (Remote Agent on Deep Space One (Muscettola et al. 1998)), or have been focused on providing resilience to a single aspect of the mission (e.g., Cassini's fault-tolerant orbit insertion burn (Gray and Brown 1998)). Additionally, no deployed system has attempted to explicitly reason about risk in the course of its automated planning. Instead, most deployed systems rely on preprogrammed "reflexes," large technical margins, and failing safe if something unexpected happens.

In order to support these ambitious future missions, a team of researchers from Massachusetts Institute of Technology (MIT), California Institute of Technology, Jet Propulsion Laboratory (JPL), and Woods Hole Oceanographic Institution (WHOI) are collaborating on the "Resilient Spacecraft Executive" project (RSE). A goal of this team is to develop a resilient, risk-aware software architecture for the next generation of space missions and demonstrate it on Earth analogue missions. The proposed architecture utilizes goal-directed and risk-aware execution and decision making, correct-by-construction control policy synthesis, and model-based systems engineering processes for developing the underlying models. The RSE team has demonstrated the architecture on underwater gliders and planetary exploration rovers.

This paper focuses on the efforts of the MIT and WHOI contingent of the RSE team, and the work on developing and deploying a risk-aware execution and decision making executive on an autonomous underwater vehicle (AUV), an Earth-bound analogue to potential missions such as exploring Europa's or Titan's seas. The AUV domain has several features that make it interesting from a planning perspective. First, there are typically few chances for communicating with the vehicle and those opportunities tend to have limited bandwidth. Second, there is high uncertainty in vehicle position. Third, there are meaningful science objectives. Last, the science objectives are typically oversubscribed, meaning the planning system has to make tradeoffs in selecting a subset of the objectives to meet.

The Resilient Spacecraft Executive

The RSE architecture consists of three primary layers: deliberative, habitual, and reflexive. The *deliberative layer* is responsible for managing the overall achievement of the mission-level goals, by elaborating and scheduling these goals into sequences of control goals that nominally achieve the specified mission goals, dispatching these goals to the habitual layer, and adjusting the sequence of control goals in response to an onboard assessment of risk. The set of components and algorithms that compose the deliberative layer is being developed by the MIT team.

The *habitual layer* is being developed by the Caltech

team. This layer is responsible for achievement of the control goals, by executing actions determined by a set of precompiled robust control policies that are computed online and loaded onboard the spacecraft. The *reflexive layer* is based on existing technology used for missions by JPL. It is responsible for the low-level control of hardware.

This three tiered architecture is similar to other systems, such as the Remote Agent, Earth Observing One's Autonomous Spacecraft Experiment (Chien et al. 2005), and the Goal-Oriented Autonomous Controller (Ceballos et al. 2011). A key distinguishing feature of RSE is its focus on reasoning about risk. For a more detailed description of the RSE architecture in its entirety, see (McGhan et al. 2015). The remainder of this paper focuses on the deliberative layer.

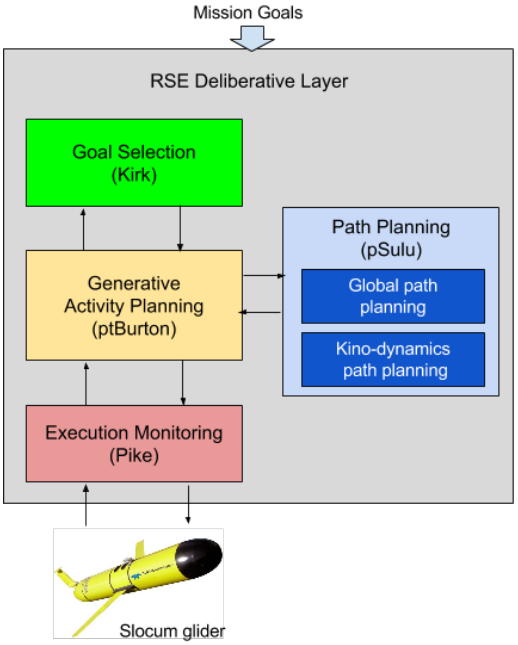


Figure 1: RSE deliberative layer interacting with Slocum glider.

The deliberative layer can be seen as another three-tiered architecture in Figure 1. In the grander scheme of the RSE, the deliberative layer operates on a high-level, abstract model of the system and produces control goals in the form of a plan of high-level actions to execute. The lower levels are responsible for confirming that actions are possible (using more detailed models), selecting between different implementations of actions within the risk and temporal constraints placed upon them by the deliberative layer, and actually executing the low-level actions.

The first component of the deliberative layer is a hierarchical planner called Kirk (Kim, Williams, and Abramson 2001) that is responsible for selecting a subset of goals to achieve, based on a reward function.

The goals are then provided to an activity planning component called ptBurton (a work-in-progress extension of the tBurton generative planner (Wang and Williams 2015)). Given a model of the actions available to it (with probabilistic durations), ptBurton generates temporal plans that satisfy the chosen goals and temporal constraints subject to a user specified chance-constraint. The temporal reasoning component is based on (Fang, Yu, and Williams 2014). If ptBurton is unable to find a plan, it communicates the set of incompatible goals to Kirk so that it can choose another subset of the goals.

Like tBurton, ptBurton interacts with a standard PDDL planner to solve subproblems of the larger planning problem. It can also interact with more domain specific planners, such as the pSulu path planner (Ono, Williams, and Blackmore 2013). pSulu consists of a global path planner and a kino-dynamic path planner. It computes a path and risk allocation between the waypoints such that a vehicle has a bounded risk of obstacle collision. ptBurton uses pSulu to compute both the path and duration of a transiting action.

Once ptBurton has found and elaborated a plan, it is passed to an execution monitor named Pike (Levine and Williams 2014). Pike is responsible for both dispatching the plan to next layer in the architecture and monitoring the estimate of the state of the world to ensure the plan is being executed correctly. If an issue or off-nominal situation is detected at run time, Pike alerts ptBurton of the issue and requests a new plan with the same goal subset. If the goal subset is no longer feasible, ptBurton elevates the request to Kirk and a new goal subset is chosen.

While the responsibilities of each layer in the deliberative layer are again similar to the standard three-tier architectures, we can start to expand on the distinguishing feature of explicitly reasoning about risk. The described version of the deliberative layer is capable of reasoning about temporal and positional uncertainty, two types of uncertainty commonly faced by exploration missions. While these uncertainties are specified by the model of the system, the mission goals provided to the deliberative layer contain chance constraints specifying how much risk the operator is willing to accept for violating constraints (e.g. missing a time-window or colliding with an object).

This ability to reason about uncertainty and chance constraints is in contrast to similar execution frameworks currently in use. This includes PDDL and generative planning-based frameworks such as ROSPlan (Cashmore et al. 2015a; 2015b) and constraint-based frameworks, such as T-REX (McGann et al. 2008).

Glider

In order to demonstrate the risk-aware executive, we are performing demonstrations on a Slocum glider, an autonomous underwater vehicle manufactured by Teledyne Webb Research. The Slocum glider is designed for long endurance missions so as to give a synoptic overview of some large area or phenomenon of interest.

The glider is propelled by a combination of a buoyancy engine, a movable weight, and wings. When at the surface a dive is initiated by moving the weight such that the glider sits angled downward in the water and decreasing the vehicle's buoyancy. As the glider falls, the angle of the glider in the water is maintained such that lift is produced by the wings that propel the glider horizontally forward. Once the

glider reaches its programmed maximum depth, the weight is shifted so the glider sits angled upward in the water and the buoyancy is increased. Again, lift produced by the wings propels the glider horizontally forward. The process repeats once the glider raises to its programmed minimum depth. In normal operations, each of these inflections is where the majority of the glider's energy is spent, meaning that if the glider can stay in deeper water it can move more efficiently.

The stock control system of the glider being used is script based. Before the start of a mission, a script must be uploaded to the glider that contains a list of waypoints to reach, the minimum and maximum depths to use, and how often to surface in order to obtain a GPS fix or communicate with the operators. These parameters can be updated during the mission, but only when the glider is surfaced and able to use its short-range radio or satellite phone. The stock control system has its own control policies to achieve goals and control hardware at the low-level. As such, the stock control system represents the habitual and reflexive layers of the RSE architecture.

Deployment

In order to obtain hands-on experience with the glider in a real-world scenario, a preliminary implementation of the risk-aware goal-directed executive was used during a technology validation cruise on board the *R/V Falkor* at the Scott Reef lagoon in the Timor Sea from March 24 to April 6, 2015. This expedition included AUVs of multiple types from multiple research institutions and had an overarching goal of understanding the issues involved with having multiple AUVs operating in close proximity.

In this deployment, a simplified version of the risk-aware goal-directed executive was used as a decision support system for a Slocum glider with an attached scanning sector sonar. The operators used the executive to plan a series of observations of target regions between surfacings for data communication. The executive then generated mission scripts that were directly executable by the glider. The simplified executive did not use a generative planner, instead the goals chosen by Kirk were already expanded into complete action sequences, and the pSulu path planner used a fixed, uniform risk allocation, provided by the glider operators.

Figure 3 illustrates the mission goals for the glider deployed during the expedition. Operators discretized a specific area of the lagoon in fifteen regions of interest, cells, to be visited by the glider. Each cell was assigned a priority and a path (dashed red line) for the glider to traverse. All AUVs on the deployment (five others) shared the cells, but each had unique goals in each cell. In order to avoid collisions, a constraint was placed on the AUVs that no more than one AUV could occupy a cell at a time. These constraints were presented to the executive as temporal constraints on when regions were available (in this case the other vehicles continued to use manually programmed scripts and their plans were available while planning the glider's mission). The glider's goals in each region were chosen based on the location of interesting features of the ocean floor that would be visible to the glider's sonar.

The executive received as input the missions goals depicted in Figure 3, along with temporal constraints, the glider's dynamics, and the lagoon's bathymetry. Kirk's task was to select and schedule a sequence of cell visitations around the schedule of the other AUVs while avoiding collision and maximizing science return. When planning paths in each cell there were two primary concerns. First, the planned paths should avoid obstacles, using user-specified buffers around the obstacles. Second, the paths should be minimum energy. While shortest paths in the reef were easy to find (there was a straight line path between most points), the shortest paths typically required the glider to pass over obstacles at a shallow depth. Due to the glider's method of propulsion, these shortest distance paths would require more inflections than taking a longer, yet deeper path. Figure 2 provides an example of an efficient path computed by the path planner.

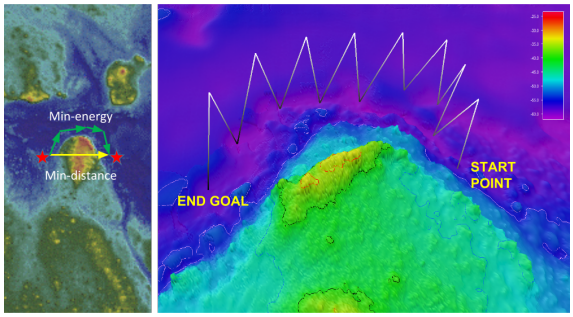


Figure 2: Example of an efficient glider trajectory generated by the path planner. A top view (left) and a perspective view (view) of the path taken from start point to end point in the lagoon.

At the beginning of the deployment, the pSulu path planner was used to plan transits for nine days in initial testings. The activity and path planner prototypes were then used in conjunction to successfully plan for two days of eight hour operations for the glider. The activity planner efficiently 1) selected subset of science goals with highest return based on science preference, and 2) ordered and scheduled visitation to respect the aforementioned constraints. Ocean currents in Scott Reef changed frequently and posed a challenge for the AUVs deployed during the expedition. The path planning component successfully planned safe routes around the reef. Moreover, we demonstrated the executive's capability to support re-planning after each glider surface activity. To the best of our knowledge, a Slocum glider has never before been used inside a reef before, due to the challenges present in that environment.

Lessons Learned

During this deployment, a number of lessons were learned that we will incorporate into the models used by the executive during the next deployment. Additionally, we feel the lessons learned may be of interest to the PlanRob community by providing real-world examples and motivations.

First, the uncertainty introduced by the currents was more

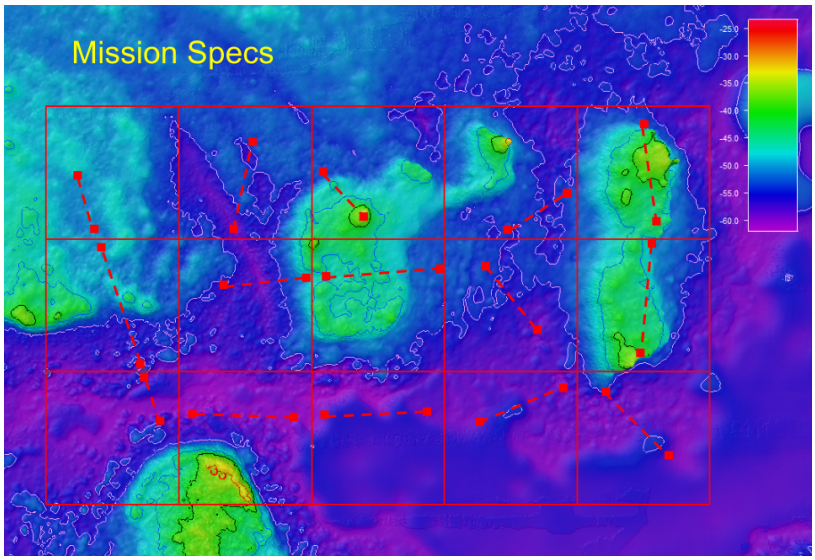


Figure 3: Mission specification provided during cruise expedition and used as input to the activity planner.

complex than anticipated. The currents at depth were mostly driven by the tides and varied both temporally and, due to the complex features of the reef, spatially. Meanwhile, the surface currents were largely wind driven and more constant spatially. Initially, we believed that changing glider parameters could be done easily and planned paths over short horizons, however an update often required the glider sit at the surface in excess of two minutes while it reinitialized the mission. On days with strong surface currents, this caused the glider to lose all forward progress made since the last surfacing. Additionally, there is not always a good estimate of the currents for a given location. This motivates the need for routines to adaptively estimate currents in the current location when transit durations start diverging significantly from modeled durations.

Second, we also learned that instrument configuration was a non-trivial matter, especially for future missions outside the reef. In order to increase maximum deployment time, instruments should be placed into a standby state when not in use. However, the uncertainty associated with transit times means it is non-obvious to a human operator when instruments should be programmed to turn on in order to observe regions of interest. The inclusion of the ptBurton generative planner should handle this in the future.

Third, we learned that there are scenarios where it is not sufficient to simply pass through a point or along a line while collecting data. For instance, on an upcoming deployment the glider will need to pass through points (preferably within 10-20 meters) while rising toward the surface. This is so that the glider can rinse off surface contaminants in deeper water and not have them contaminate the readings taken by the mass spectrometer that will be included on this deployment. This poses enormous challenges to pSulu as it must account for the uncertain currents, while minimizing surfacing to obtain GPS locks (so as to drift less and meet temporal constraints). It is highly doubtful that all but the most skilled human operators would be unable to plan such a path with the desired accuracy.

Last, we learned that there is much opportunity for the planning community to affect multi-AUV operations in the future. On the Scott Reef deployment, we learned that there is currently no effective solution used by AUV operators for planning day-long operations for multiple AUVs of varying capabilities in close proximity. This become obvious when four near collisions and one collision/entanglement happened during the two week long cruise. Luckily, the vehicles involved in the most serious near-misses had acoustic modems and were given override commands to avoid the collision, but it would be preferable to plan such that the possibility of collisions is vastly reduced.

Conclusions and Future Work

We have described the Resilient Spacecraft Executive initiative and described the authors' work-in-progress contribution of a risk-aware goal-directed executive for the deliberative layer. Additionally, we have described the deployment of a simplified version of the executive on a vehicle and lessons learned from that experience.

In the year since the Scott Reef deployment, work has continued on the design and integration of the non-simplified executive. Another deployment of the glider with the full executive is planned for July 2016 off the coast of Santa Barbara, CA. This deployment will use a similar input to the planner (areas of interest along with preferences and temporal constraints), but will additionally feature the risk allocation features of pSulu and use ptBurton as a generative planner.

Acknowledgments

This work was funded in part by the Keck Institute for Space Studies. The Scott Reef deployment was funded by the Schmidt Ocean Institute.

References

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015a. Rosplan: Planning in the robot operating system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; and Ridder, B. 2015b. Artificial intelligence planning for auv mission control. *IFAC-PapersOnLine* 48(2):262–267.

Ceballos, A.; Bensalem, S.; Cesta, A.; De Silva, L.; Fratini, S.; Ingrand, F.; Ocon, J.; Orlandini, A.; Py, F.; Rajan, K.; et al. 2011. A goal-oriented autonomous controller for space exploration. *ASTRA* 11.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davis, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication* 2(4):196–216.

Fang, C.; Yu, P.; and Williams, B. C. 2014. Chance-constrained probabilistic simple temporal problems. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Gray, D. L., and Brown, G. M. 1998. Fault-tolerant guidance algorithms for cassini's saturn orbit insertion burn. In *American Control Conference, 1998. Proceedings of the 1998*, volume 2, 905–908. IEEE.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 487–493.

Levine, S. J., and Williams, B. C. 2014. Concurrent plan recognition and execution for human-robot teams. In *ICAPS-14*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for auv control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 1049–1054. IEEE.

McGhan, C. L. R.; Murray, R. M.; Serra, R.; Ingham, M. D.; Ono, M.; Estlin, T.; and Williams, B. C. 2015. A risk-aware architecture for resilient spacecraft operations. In *Aerospace Conference, 2015 IEEE*, 1–15.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before,. *Artificial Intelligence* 103(1-2):5–48.

Ono, M.; Williams, B. C.; and Blackmore, L. 2013. Probabilistic planning for continuous dynamic systems under bounded risk. *Journal of Artificial Intelligence Research* 46:511–577.

Space Studies Board, N. R. C. 2012. *Vision and Voyages for Planetary Science in the Decade 2013-2022*. National Academies Press.

Wang, D., and Williams, B. C. 2015. tBurton: a divide and conquer temporal planner. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*.