Planning for a Small Team of Heterogeneous Robots: from Collaborative Exploration to Collaborative Localization

Jonathan Michael Butzke

CMU-RI-TR-18-02

08 NOV 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Maxim Likhachev, Robotics Institute, CMU (chair)
Sebastian Scherer, Robotics Institute
Nathan Michael, Robotics Institute
Dan Lee, GRASP LAB, University of Pennsylvania

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Multi-robot, Air-Ground Team, Planning, Exploration, Collaborative Localization

For Mom, Dad, Chris, Stacy, and Chelsea.

Abstract

Robots have become increasingly adept at performing a wide variety of tasks in the world. However, many of these tasks can benefit tremendously from having more than a single robot simultaneously working on the problem. Multiple robots can aid in a search and rescue mission each scouting a subsection of the entire area in order to cover it quicker than a single robot can. Alternatively, robots with different abilities can collaborate in order to achieve goals that individually would be more difficult, if not impossible, to achieve. In these cases, multi-robot collaboration can provide benefits in terms of shortening search times, providing a larger mix of sensing, computing, and manipulation capabilities, or providing redundancy to the system for communications or mission accomplishment. One principle drawback of multi-robot systems is how to efficiently and effectively generate plans that use each of the team members to their fullest extent, particularly with a heterogeneous mix of capabilities.

Towards this goal, I have developed a series of planning algorithms that incorporate this collaboration into the planning process. Starting with systems that use collaboration in an exploration task I show teams of homogeneous ground robots planning to efficiently explore an initially unknown space. These robots share map information and in a centralized fashion determine the best goal location for each taking into account the information gained by other robots as they move. This work is followed up with a similar exploration scheme but this time expanded to a heterogeneous air-ground robot team operating in a full 3-dimensional environment. The extra dimension adds the requirement for the robots to reason about what portions of the environment they can sense during the planning process. With an air-ground team, there are portions of the environment that can only be sensed by one of the two robots and that information informs the algorithm during the planning process. Finally, I extend the air-ground robot team to moving beyond merely collaboratively constructing the map to actually using the other robots to provide pose information for the sensor and computationally limited team members. By explicitly reasoning about when and where the robots must collaborate during the planning process, this approach can generate trajectories that are not feasible to execute if planning occurred on an individual robot basis.

An additional contribution of this thesis is the development of the State Lattice Planning with Controller-based Motion Primitives (SLC) framework. While SLC was developed to support the collaborative localization of multiple robots, it can also be used by a single robot to provide a more robust means of planning. For example, using the SLC algorithm to plan using a combination of vision-based and metric-based motion primitives allows a robot to traverse a GPS-denied region.

Acknowledgments

Getting a PhD is a strange accomplishment for someone that used to work as a residential electrician and joined the Navy out of college. I never really pictured myself spending so much more time in school when I was cruising beneath the sea in a submarine.

Reaching this point took a lot of assistance from other people along the way, starting with the first person I ever knew, my mother. She, among many other things, gave me an insatiable curiosity and an unending drive to understand everything about anything. My father, being the second person I ever met, was equally influential. He has always taken the time to show me how things work, whether it was helping him change the oil in the family car, or showing me his new toy in 1983 - an IBM PC. You both gave me the foundation that the rest of my life has been built on and I thank both of you for all that you have done.

Next, I would like to thank my brother, who was always there to be my friend. No matter where we moved to, or who else was around, he was constantly by my side as we grew up. I learned a lot from him along the way and remember all of the silly things we did. I'm equally proud of what he has done and wish him the best.

I would be remiss if I did not also thank my advisor, Max Likhavchev, for all of the great guidance over the years. I loved being at University of Pennsylvania and would not have switched schools for a lesser advisor. Along with Max, I would also like to thank the guys in the lab that taught me so much: Michael Phillips (possibly the smartest person I know), Ben Cohen (who got an old man like me to play ice hockey), Andrew Dornbush (who's coding skills kept many of our projects on track), and Brian MacAllister (who's assistance with the UAV's is the only reason those machines every successfully took to the skies), along with everyone else in the Search-Based Planning Lab.

Finally, I would like to thank my wife Stacy and daughter Chelsea. Your support and love were what kept me going during the tough times. I am glad to finally be done and hope to spend more time with you two.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hiding underground, Knowing we'd be found
Fearing for our lives, Reaped by robot's scythes
JUDAS PRIEST *Metal Gods*

We are in an era of amazing advances in electronics. Computers are simultaneously getting smaller and more powerful. Sensors are becoming more capable, using less power, and are dropping in price. As these supporting technologies continue to improve, they in turn lead to ever more capable robots becoming more affordable and more common. As robots proliferate and their abilities improve, they will be turned to for an increasing number and variety of scenarios, ultimately resulting in situations where more than one robot is available to perform a shared task. It will be beneficial for the robots to collaborate in a deliberate, principled manner in order to provide: computational or sensing resources for each other; information about the environment; communication relays; transportation; or other mutual support. Multi-robot collaboration can provide benefits in terms of shortening task completion times, providing a larger mix of sensing, computing, and manipulation capabilities, or providing redundancy to the system for communications or mission accomplishment. One principal challenge in employing multi-robot systems is how to efficiently and effectively generate plans that use each of the team members to its fullest extent, particularly with a heterogeneous mix of capabilities. This thesis develops approaches to planning in two areas of multi-robot collaboration: Exploration and Localization.

Multi-robot Exploration

When multiple robots are operating in an initially unknown environment, the most natural form of cooperation is multi-robot exploration. Exploration of unknown environments is a cornerstone of robotic systems attempting to operate in the real-world without requiring constant human operator input. The ability to seek out locations to gain information about the environment is a foundation for the coverage problem and plays a significant role in tasks such as search and rescue, infrastructure inspections, and any other tasks requiring a sensor to be repeatedly positioned and re-positioned so as to evaluate all possible locations in the environment. The planning system must be capable of quickly assigning goals to the team-members in order to efficiently cover the environment.

The first of the collaborative planners developed in this thesis uses a frontier-based approach to generate goal assignments to a homogeneous team of ground robots in order to explore an unknown environment with high-level guidance from an operator. As an improvement upon existing frontier-based approaches, this algorithm allows an expert to give broad objectives to the team as well as specific constraints such as minimum and maximum distance between robots. The exploration algorithm can then use these objectives and constraints when determining the trajectories of all available robots in order to efficiently search the environment.

For more complex real-world environments containing overhangs and non-convex obstacles, the planning algorithm may require modeling the environment in all three dimensions as simple planar assumptions can lead to incomplete exploration. This in turn may require that we reason about (and plan for) the robot's position, not just in the three spatial dimensions, but also accounting for the orientation of the robots as well. As a concrete example, consider a small wheeled robot in a typical office environment. While traveling on the ground it has difficulty seeing any items on top of a table. On the other hand, an aerial vehicle could easily see the tops of the tables, but may have difficulty navigating to a position under a desk.

Expanding our exploration to such heterogeneous air-ground teams in this case clearly results in improved performance. A flying robot can get a camera into some positions better than the ground robot can. However, since aerial robots have to support all of their weight through the expenditure of energy, they are often limited to small payloads and short mission durations. Ground robots typically have greater power reserves and are frequently capable of long duration missions while carrying significant amounts of payload, allowing for a wider range of possible sensors. These differences in sensing and locomotion within the team can be significant which can not only make the team more capable than the robots are individually, but can also place additional requirements on the planner. Looking again at the search-and-rescue scenario we can see that both the high endurance of the unmanned ground vehicle and the capability to traverse debris strewn environments typical of unmanned aerial vehicles are important. In order to adequately explore a large, obstacle strewn environment, the two robots will have to intelligently determine when is the best time and where is the best position to employ each of them. For the environments where an aerial vehicle is required to reach certain sections, a possible compromise is to search the accessible areas with a ground vehicle and reserve the aerial vehicle for just those areas that require the higher vantage point or are otherwise unviewable by the ground robot. The key is that the planner needs to be capable of making that determination at runtime.

To address these concerns, the earlier 2-D exploration planner is expanded to allow a heterogeneous air-ground team of robots to explore a full 3-dimensional environment to find an object of interest. In addition, this algorithm allows for a sensor system that is independent from the localization system. This permits the use of prior maps or other localization methods if they are available while still enforcing complete coverage of the environment by the designated sensor system. Similar to the 2-D planner, this algorithm offers the ability to tune the exploration through the use of high-level user input at runtime.

Collaborative Localization

The second area of multi-robot collaboration that we explore is the ability of robots operating as part of multi-robot teams to use other team members to help themselves localize their positions within the environment. While an ideal robot would always know its position with perfect certainty, this is often not the case for real-world robots. During operations as part of a search-and-rescue scenario it is conceivable that the environmental conditions will not be ideal and may even be adverse to some sensing modalities. For example, damaged buildings may lack power; therefore, localization may have to occur in the dark posing difficulties to vision-based sensors. Similarly, GPS is rarely reliable in any indoor environment limiting potential localization techniques. With these limitations on their capability to independently determine where they are, a robot team may have little ability to effectively determine where to go in the environment. However, the same sensors that are used to gather information from the environment can also be used to allow the robots to collaboratively localize using each other. As an example, consider two robots, *A* and *B*. Assume *A* has a robust localization capability while *B* is equipped with just a single camera. It is easy to see that *A* can maintain an accurate estimate of its position using its own localization abilities. However, if *B* can "see" and "hear" *A*, then it too could maintain an accurate estimate of its position. If *A* is in *B*'s field-of-view, then *B* can use standard image processing techniques to determine its position relative to *A*. Similarly, if the two robots can communicate, *A* can provide *B* with its current global position. From *A*'s global position and the relative pose between the two, *B* can accurately determine its global position as well. Using this collaborative localization approach, both robots could successfully navigate through an environment even though only one of them has the ability to determine its position independently. Again, the difficulty shifts to the planner in order to: reason about these special abilities; generate a trajectory for both robots that allows them to collaboratively localize when necessary; but, leaves them free to explore independently otherwise.

Our final collaborative planning algorithm allows for different capabilities in the team members through the combination of controller-based motion primitives with a state lattice planner (SLC). By using controller-based motion primitives we are able to incorporate a wide variety of sensing and control systems into the planner including controllers based on the interaction between multiple robots (e.g., collaborative actions such as collaborative localization). This algorithm also allows us to interrupt a controller when we receive a perceptual trigger allowing for more complex trajectories than from a pure controller based approach.

Summary of Contributions

In this thesis, I present a set of navigation planning algorithms that reason explicitly about collaboration in order to allow multi-robot teams to effectively operate in complex environments. The first set of these algorithms allows for collaborative exploration between teams of robots, while the second provides a method of conducting collaborative localization using State Lattice Planning with Controller-based Motion Primitives (SLC) and Planning with Adaptive Dimensionality (PAD) to handle otherwise unsolvable domains.

In summary, the contribution of this thesis includes:

- 2-dimensional exploration algorithm for a homogeneous team of ground robots
  - Frontier-based exploration
  - Allows high-level user input during execution
- 3-dimensional exploration algorithm using multiple heterogeneous robots
  - Allows exploration of full 3-D environment
  - Specifically designed for Air-Ground team of robots
  - Maintains capability for user input during execution
- Collaborative localization framework
  - Uses State Lattice Planning with Controller-based Motion Primitives and Planning with Adaptive Dimensionality to allow an air-ground robot team to effectively localize themselves
  - Permits robots to operate independently as necessary
- State Lattice Planning with Controller-based Motion Primitives framework
  - Allows reasoning explicitly about controllers and perceptual capabilities of a robot
  - Enables robust operation in varied environments without requiring a global metric space
  - Allows termination or switching of controllers by perceptual triggers

Outline

This document is organized as follows:

- Chapter 2: A brief overview of the related work regarding multi-robot collaboration including centralized versus decentralized planning approaches.
- Chapter 3: A detailed explanation of search-based planning algorithms such as A* and its related algorithms.
- Chapter 4: A planning algorithm for multi-robot exploration using multi-objective utility functions. This work is primarily from [Butzke and Likhachev, 2011] and [Butzke et al., 2015].

- Chapter 5: Applications of the algorithms developed in Chapter 4 to a homogeneous team of ground robots and a heterogeneous air-ground team.
- Chapter 6: A planning framework using controller-based motion primitives with perceptual triggers. This work is primarily from [Butzke et al., 2014].
- Chapter 7: Expanding the SLC framework to collaborative localization for air-ground teams. This work is primarily from [Butzke et al., 2016].
- Chapter 8: Discussion on the contributions contained in this thesis and future work in expanding these algorithms.

Chapter 2

Related Work

If you had the time to lose
An open mind and time to choose
Would you care to take a look
Or can you read me like a book
IRON MAIDEN *Caught Somewhere in Time*

Planning for multiple robots involves many trade-offs in order to remain tractable. Planners can be decentralized where each robot plans for itself using the only information it can sense directly. this approach allows for lower computational burdens, improved planning times, and lower inter-robot communication requirements. At the other end of the spectrum, a single, globally knowledgeable planner can plan for all of the team members simultaneously in a single joint state space. While this allows for globally optimal plans, it comes at a cost of higher communication and computational requirements[Clark et al., 2003].

The task being planned also has a significant impact on what method of planning is used. A large swarm of ground robots operating in a large environment is not tractable as a centralized joint planner[Li et al., 2017; Wang et al., 2009]. On the other hand, three robot arms working together in a tight environment need to ensure that the plans are coordinated to prevent mutual collisions [Cohen et al., 2015]. Some example tasks that can be solved with multi-robot planners are collaborative manipulation, exploration, collision avoidance, task scheduling, etc. [Arai et al., 2002].

In this thesis, we will develop algorithms specifically addressing collaborative exploration and collaborative localization.

Collaborative multi-robot planning falls into two primary areas: centralized and decentralized. Decentralized planners seek to allow each individual robot to determine its own trajectory by making assumptions concerning the other robots. Planning in this fashion provides a high degree of redundancy allowing for successful mission completion even in the face of multiple robot failures [Zelenka and Kasanicky, 2014]. However, in some environments, robots must move in locally non-optimal directions in order to allow for other robots to reach their goal state (Fig. 2.1).

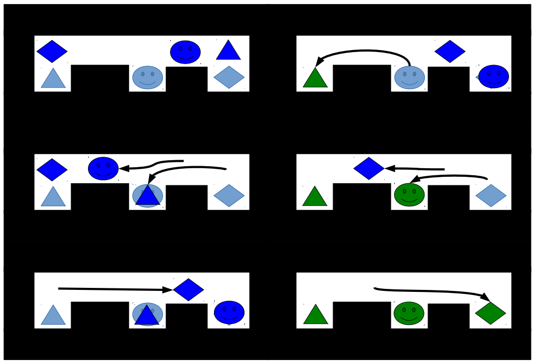


Figure 2.1: The robots (dark blue) are attempting to get to their assigned goals (light blue). If the circular blue robot goes straight to its goal location, the diamond and triangle robots will be unable to get to theirs due to the narrow passage. One solution is for the triangle to take the middle goal position (middle, left image), let the other two robots pass to the right (bottom, left image), before it moves to its goal (green, top, right image). The circle and diamond can repeat a similar maneuver to get them each in their goal locations (green, bottom right).

Centralized planners on the other hand assume that there is a single computational element that is performing the planning for all team members. In this case, each robot's position must be passed to the centralized planner and in return the planner provides each team member with a trajectory. This scheme can face difficulties occurring from lost or delayed communications, and the large amounts of data to be transferred.

While centralized approaches perform planning on a single computational element, there are two methods commonly employed [Latombe, 2012]. The first is planning in a combined joint state space. This can result in a very high dimensional planning problem. Due to this, the joint states space planning is typically only used for small teams of low-dimensional robots. The benefit of joint state space planning schemes, other than the deadlock prevention alluded to above, is that all of the individual trajectories can be fully synchronized eliminating collision hazards as well as being capable of producing globally optimal paths.

The second common centralized planning approach is the distributed state space approach. In this case, the planner plans for each robot individually, but with global knowledge of the environment and at least some knowledge of the other robot's trajectories. The approach is feasible for larger teams of robots, since the planning problem dimensionality remains significantly smaller than in the joint state space approach. This comes at a loss of global optimality and the need to develop a method to deconflict deadlock conditions.

The algorithms presented here are centralized planners and as such require a single node to perform at least a portion of the planning effort for all robots. The exploration planners plan in the individual state space of each robot sequentially allowing for the robots to generate their own trajectories. The planner only determines the goal locations. The collaborative localization planner plans in the joint state space of all involved robots. Due to this higher dimensional state space, we use the Planning with Adaptive Dimensionality framework to keep planning tractable. All or our work does assume that communications bandwidth and reliability are essentially perfect. While high-bandwidth wireless communications are reasonably robust in normal situations we have not tested their performance in realistic disaster scenarios. We attempt to minimize the amount of data transferred between team members and defer to the vast field of communications link-layer research to mitigate any remaining issues.

2.1 Collaborative Exploration Planning

Before a group of robots can perform a task, they must first gain knowledge about their local environment. This may come in the form of an a priori model of the environment or it may have to be discovered in an online fashion. The exploration task requires that a collection of sensor robots systematically traverse an environment in order to locate static and dynamic obstacles. Trade-offs between the rate of exploration versus the thoroughness of exploration must be made in order to minimize the overall search time. In addition, the algorithm must be capable of working in any environment with a minimum set of assumptions on robot behavior and sensing capabilities. For single robots there are a variety of approaches that can be used to guide the robot towards unexplored areas.

One of the most common approaches is using the concept of the boundary between the known areas of the map and the unexplored regions. The non-obstacle portions of this boundary are collectively known as the *frontier* [Yamauchi, 1997] and are used as candidate goal points. Frontier-based exploration directs robots to this frontier to gain information via their sensors of the unknown areas. This approach has been expanded to cover multi-robot teams [Yamauchi, 1998; Simmons et al., 2000; Zlot et al., 2002; Burgard et al., 2005; Visser and Slamet, 2008; Nieto-Granda et al., 2014; Colares and Chaimowicz, 2016] and outdoor environments [Tao et al., 2007] with great success. This work is an extension of our earlier work in this category [Butzke and Likhachev, 2011; Butzke et al., 2012, 2015]. However, most of these approaches, including some of our previous work, only deal with 2-dimensional (planar) exploration. Even the approaches that use 3-dimensional motions tend to treat the environment more as 2.5-D (elevation- or height-map) rather than full 3-D [Sujit and Beard, 2008; Sawhney et al., 2009; Yang et al., 2013]. In particular, few of these approaches consider the search target to be on the underside of obstacles or require movement under obstacles in order to get into a position to see the target.

One impressive approach that does consider flying under or through obstacles is the potential field/harmonic function approach used in [Rasche et al., 2011]. Like our approach they use an octree to represent the environment, and a camera to explore, but where they differ is that all of their simulated UAV's were identical. It is unclear how easy it would be to incorporate a heterogeneous team of robots (including ground robots) into their scenario. Additionally, they only consider exploration of the top surface of obstacles and have no method for finding a search target located on the underside of an obstacle.

Another work to explore in 3-D looks at underwater vehicles examining a cliff [Rathnam and Birk, 2015]. Like our approach they allow for sensing of the environment from above, below, or beside elements in the environment. In their case they explicitly change the pose of the robot to provide the best sensor view possible as the robots traverse the environment. However, their team of underwater robots is largely heterogeneous and they do not provide any method of taking into account high-level user input unlike the approach discussed here.

There is also areas of research on methods to improve the speed of determining the frontier through the use of better designed algorithms that work with individual laser scans rather than parsing the entire map [Keidar and Kaminka, 2014]. Improvements such as this can be incorporated into our approach to improve overall planning times.

When evaluating potential target points for the next goal location, most exploration algorithms use a combination of the information gain from sensing while traveling to the target and the cost of moving to that point. The information gain can be specified in several ways however, a typical method is to set the information gain for a state equal to a function of the change in knowledge about all states visible from that state after the robot moves into it. The information gain for a trajectory is equal to the sum of the information gain of all the states along the trajectory. It must be recognized that as each state is entered it may change the amount of certainty in surrounding states and thus lower the information gain for subsequent states along the trajectory. Due to this non-Markov property, optimizing the information gain over trajectories is computationally expensive, and many algorithms estimate the information gain of a trajectory by using the information gain of the final point [Zlot et al., 2002; Gonzalez Banos and Latombe, 2002; Visser and Slamet, 2008] while a few have attempted to explicitly include the expected information gain along the entire trajectory [Simmons et al., 2000; Sim and Roy, 2005]. Others have combined the information gain with additional features such as communications constraints [Burgard et al., 2005] or improved physical models such as the likelihood of specular reflection from nearby obstacles [Grabowski et al., 2003]. When combined with other features the information gain is generically referred to as the utility of the state. Similarly, the cost can be viewed as a function of time, energy, or distance between the goal and the start state.

The combination of the states utility and its cost directly affect the order in which states are selected as potential goals. This selection process can be centralized or distributed and can be performed through the use of previous high-level information [Oßwald et al., 2016], a market architecture [Zlot et al., 2002] or through a high-level task allocation scheme using Petri Net Plans [Calisi et al., 2007]. However the most straightforward method is a greedy assignment that selects the best robot-goal pair, assigns that to the robot, removes that robot from further consideration adjusting the utility of all states to reflect the assignment, and then repeats until there are no unassigned robots [Burgard et al., 2000, 2005] . This is the method we have adopted in our implementation.

Some recent work has explored the exploration problem in the context of hazardous environments, such as post-disaster areas [Schwager et al., 2017]. While our approach assumes an environment that, while potentially hazardous to humans, is benign for the robots over the time scales in question. It may be possible to include this type of analysis into an automatically generated logical region for our approach as an area of future work.

The planning algorithm frequently has to take additional information into account, such as communication range limitations, sensor effectiveness and range, and terrain traversal costs. Our approach uses a cooperative multi-robot exploration planner that utilizes a variant of a frontier based approach with the ability to specify specific hard and soft constraints on the robot through a central planning algorithm. While a few approaches allow for a generalized multi-feature utility function [Amigoni and Gallo, 2005] ours differs in that we also allow for the creation of logical areas to influence the exploration priorities. We define these logical areas or regions as a set of states assigned to one or more robots (either specifically or ordinally, e.g., to the "first and second to arrive") and an associated weight matrix. This concept can allow a higher level planning system or expert input to influence the direction and areas of primary exploration particularly for teams of heterogeneous robots.

2.2 Collaborative Localization as Part of Planning

Localization can be handled in different ways by the planner. The planner can explicitly model the uncertainty it has in its position or it can ignore uncertainty and assume that there is some method of localization available at all times. Our approach takes a middle ground. For areas of the environment that we know have poor localization characteristics (e.g., dimly lit or featureless areas when using visual localization schemes or GPS-denied regions when operating with GPS) we can permit the team members to collaborate. In our case, the ground and aerial vehicles have different sensing capabilities, so it may be possible for one of them to localize well and then assist the other with its localization.

We develop our state lattice planner with controller-based motion primitives (SLC) which, while it falls in the category of search-based planning, differs from all of these works presented in this chapter in that we allow explicit collaborative localization actions as part of the planning process. In order to integrate that ability directly into the planning stage, we use the planning with adaptive dimensionality framework (PAD), discussed in Section 7.2. The PAD planner allows us to expand beyond a single robot and generate plans for a team of robots operating in a high dimensional space while keeping planning times reasonable.

Looking at the other options for localization uncertainty, there have been approaches that considered the uncertainty of the robot as part of the planning problem. Of these, POMDP based approaches, even with modern approximate solvers, are slower than A* and do not scale as well to large environments [Kurniawati et al., 2008]. An alternative to the full POMDP method of handling uncertainty is by augmenting the state space with an uncertainty metric [Gonzalez and Stentz, 2007]. This approach uses detected landmarks to reduce uncertainty as a part of the planning process. In this way the planner can prune actions that raise the uncertainty above a threshold without incurring the overhead of solving a POMDP. Another example is the coastal navigation algorithm [Roy and Thrun, 1999] which models the positional probability using a Gaussian. None of our approaches explicitly model the uncertainty in localization unlike these methods making them less general but significantly faster due to not increasing the problem dimensionality.

Localization of aerial vehicles poses its own unique set of challenges, primarily due to the lack of reliable odometry measurements. Coupled with the recent increase in the availability of small, low cost aerial vehicles, in particular easy to use quadcopters, a substantial amount of research effort has been directed at teams of air-ground robots [Lacroix and Le Besnerais, 2011] including work that also includes elements of exploration [Burgard et al., 2005], and collaborative localization between the team members [Rekleitis et al., 2001]. Communications in a variety of forms has been the focus of several works in this area [Viguria et al., 2010; Vaughan et al., 2000], although frequently these include high-quality localization of all robots, including the use of GPS on both the ground and aerial vehicles [Grocholsky et al., 2006]. However, some approaches rely purely on well-localized ground vehicles [Li et al., 2011; Reardon and Fink, 2016], forcing the aerial vehicle to update its position estimate only by visually extracting the pose of the ground vehicle. In the case of Reardon and Fink [2016], collaborative localization is used to allow a simple aerial vehicle to maintain an accurate position relative to the ground vehicle. However, no independent motion of the air vehicle is allowed. Our SLC-based work, while similar to this last approach, differs from all of these by incorporating the collaborative localization element into a larger planning framework and allowing independent operation of the two robots when collaborative localization is not required.

Some work has taken a more indirect approach to collaborative localization. For example, having one robot construct an elevation map of the environment for the others to localize with [Käslin et al., 2016]. These approaches, while useful in generating informative maps for robots to use during navigation, does not allow the robots to act in parallel and calling for localization assistance only when needed.

Detection and pose estimation as part of collaborative localization has been a goal of robotics research for many years [Fox et al., 1999]. A lot of this work has been directed at making the detection of the other robots of a team more reliable and accurate [De Silva et al., 2012] even for chains of robots where the farthest ones have no direct knowledge or sensor measurements regarding any known landmarks and instead must rely entirely on their neighboring robots [Wanasinghe et al., 2014]. Other work focuses on the challenges of constructing the map with multiple robots and using collaborative localization to tie map segments together [Schuster et al., 2015]. Still other approaches have focused on the sensor integration from the data fusion side ensuring that the data is used more effectively [Song et al., 2008]. Our SLC-based approach keeps the localization scheme simple, we use only fiducial markers and a simple camera to determine the estimated pose of the ground vehicle from the aerial vehicle and then, accurately knowing the ground vehicles position, we can estimate the position of the aerial vehicle with reasonable precision. While we do not use these other advanced techniques in this work, our algorithms are capable of incorporating this improved data into the planning framework.

The key features that distinguish our SLC-based planners from the prior work in the field is that we include the collaborative localization element directly in our planning process. This allows the robots to go on separate trajectories and only meet up when required rather than travel in a fixed formation or conversely, operate completely independently. In addition, our planner provides guarantees on path quality and resolution completeness.

2.3 Planning for Air-Ground Teams

Within the multi-robot planning field, there have been numerous works relating specifically to multi-robot cooperation between aerial and ground vehicles [Vidal et al., 2002; Viguria et al., 2010; Tanner, 2007; Grocholsky et al., 2006]. Our work is based on the general principles found in these - minimize the amount of information that has to be shared between platforms, maximize the amount of computing that can be performed in a decentralized manner, and reduce the load on the operator. Our systems are not decentralized, however; they all require a globally knowledgeable planner and a centrally maintained map of the environment. While many of these only look at general collaboration between air and ground robots, our approach looks specifically at collaborative planning in cluttered 3-D spaces.

In Fankhauser et al. [2016], an air-ground team navigates through a pseudo-disaster environment. In this work, the two robots share map information and are able to navigate without the need for an external localization source (e.g., GPS). Their scheme of using globally identifiable landmarks could be used in conjunction with the exploration scheme presented in Chapter 4 to make our algorithm more robust. However, in contrast with our work, the planning algorithm presented does not coordinate motions between the flying and the ground robots. There is no method for the ground robot to summon the flying robot to map an area of importance or to provide landmarks in the vicinity of the ground robot for it to localize with.

Another air-ground exploration scheme aimed at disaster response is the work in [Michael et al., 2014, 2012]. This work uses an air vehicle initially mounted on a ground vehicle to explore earthquake damaged buildings. While this work demonstrated the viability of conducting exploration with a heterogeneous team of robots, the robots used in this case were teleoperated, unlike our work where the robots are fully autonomous.

Our work is based on using heuristic graph search algorithms. Path planning approaches based on heuristic graph searches such as A* have been popular for navigation but have not been as widely used for planning aerial robot trajectories due to the relatively high dimensionality of the planning problem. An early method for generating collision free trajectories in cluttered environments was to segment the free space into intersecting spheres, then plan through this network of spheres using A* [Vandapel et al., 2005]. This method had the advantage of not requiring any additional collision checking during the planning stage, and due to the reduced size of the search graph was capable of quick planning times. The drawback to this approach was that the complete map was needed beforehand in order to perform the segmentation, and by requiring a spherical footprint, valid trajectories were pruned from the search graph.

Chapter 3

Background

Lean not upon your own understanding
Ignorance is well and truly blessed
Trust in perfect love, and perfect planning
Everything will turn out for the best

RUSH *Clockwork Angels*

Our algorithms are based on the A* family of search algorithms as adapted for planning navigation tasks. This chapter will present the definitions and conventions used throughout this work as well as an overview of A*and ARA*.

3.1 Definitions, Conventions, and Abbreviations

This section contains the definitions and typeface conventions used throughout this report. Additional details will be presented in later chapters.

Graph Variables

ϕ, θ, ψ (Euler angles) Roll, Pitch, Yaw, respectively, of a robot.

s (State) A tuple containing all of the variable dimensions. For example, $\langle x, y, \psi \rangle$ for the x, y position and the orientation ψ of a single robot.

$\mathcal{X}$ (Set of states) The set of all valid configurations.

$e(x, x')$ (Edge) a single directed edge between states x and x'. Edge e is valid if $e \in \mathcal{E}$.

$\mathcal{E}$ (Set of edges) The set of all allowable transitions between states.

$\mathcal{G}$ (Graph) A graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ for motion planning is constructed from the set of states $\mathcal{X}$ and the set of valid directed edges $\mathcal{E}$.

$\mathcal{X}^{HD}$ (Set of high-dimensional states) States consisting of full-dimensional states.

$\mathcal{X}^{LD}$ (Set of low-dimensional states) States consisting of states of less than full dimensionality.

$\mathcal{X}^{AD}$ (Set of adaptive-dimensional states) A set of both high and low dimensional states.

$\mathcal{E}^{HD}$ (Set of high-dimensional edges) Edges between full-dimensional states.

$\mathcal{E}^{LD}$ (Set of low-dimensional edges) Edges between lower-dimensional states.

$\mathcal{E}^{AD}$ (Set of adaptive-dimensional edges) a subset of high and low dimensional edges, $\mathcal{E}^{HD}$ and $\mathcal{E}^{LD}$, as well as edges connecting states of different dimensionalities.

$\mathcal{G}^{HD}$ (High-dimensional graph) A graph constructed using the full-dimensional states, $\mathcal{X}^{HD}$, and edges, $\mathcal{E}^{HD}$.

$\mathcal{G}^{LD}$ (Low-dimensional graph) A graph constructed using the low-dimensional states, $\mathcal{X}^{LD}$, and edges $\mathcal{E}^{LD}$.

$\mathcal{G}^{AD}$ (Adaptive-dimensional graph) A graph constructed using adaptive dimensionality states, $\mathcal{X}^{AD}$, and edges, $\mathcal{E}^{AD}$, see Section 7.2.

c (Controller) a single controller that takes local sensor readings and outputs a motion command

$\mathscr{C}$ (A subset of Controllers) a set of controllers

$\mathcal{C}$ (Set of All Valid Controllers) All valid controllers for a robot

τ (Trigger) a perceptual input that can halt execution of a controller

$\mathscr{T}$ (A subset of Triggers) a set of triggers

$\mathcal{T}$ (Set of All Valid Triggers) All valid triggers for a robot

$\mathcal{P}(\cdot)$ (Powerset) the set of all possible subsets of $(\cdot)$.

General Abbreviations

UAV (Unmanned Aerial Vehicle) Any small autonomous aerial vehicle, also

	referred to as Micro-Aerial Vehicle (MAV).
UGV	(Unmanned Ground Vehicle) Any autonomous ground vehicle.
Additional Definitions	
cell	a state defined only by spatial dimensions, e.g., $\langle x, y \rangle$ or $\langle x, y, z \rangle$
free	a cell with no obstacles
Symbols	
$[\cdot]$	a discretized value
$\langle a, b, \ldots, c \rangle$	a tuple
$\mathscr{A}\mathscr{B}\mathscr{C}\mathscr{D}$	script font is used to denote a set
abcd	bold roman font is used to denote a multi-dimensional vector
$\mathcal{ABCD}$	calligraphy font is used to denote a space

3.2 The A* Family of Algorithms

Fundamentally, our approach is a direct extension of the A* algorithm [Hart et al., 1968]. A* is a graph search algorithm that finds a provably optimal route between a start and goal state on a graph. From the start state, it selects the next best state, s, to expand using the known cost from the start state to s plus an estimated cost from s to the goal - essentially running a Dijkstra's Search [Dijkstra, 1959] with the added heuristic estimate of "cost to go". When a state is expanded, all potential successor states are placed into an ordered list (the OPEN list) from which the next state to be expanded is selected.

3.2.1 The A* Graph

A* operates on a directed graph, $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ consisting of the states, $\mathcal{X}$, and valid (i.e., collision free) directed edges, $\mathcal{E}$. The states of the search graph are tuples representing individual points in configuration space (C-space) that include all of the independent variables that the search algorithm can search over. The search graph can have any number of spatial dimensions and can include dimensions representing curvature, velocity, or other state variables as well. For navigation a typical choice is $\langle x, y, yaw \rangle$ or $\langle x, y, yaw, vel \rangle$ [Likhachev and Ferguson, 2009] for ground vehicles and $\langle x, y, z, \phi, \theta, \psi \rangle$ for aerial vehicles [Anderson, 2010]. The environment is discretized into a regular grid for the translational dimensions and into a discrete set of values for other dimensions. The translational dimensions may be discretized in 1 m, 10 cm, or any other increment depending on the environment. Variables representing angles are typically discretized into a finite set of allowable angles such as $\{0°, 45°, \ldots, 315°\}$. This same procedure can be applied to any other variables used by the search - velocity, energy, uncertainty - by discretizing the potential values into a finite set. For all dimensions, smaller discretizations result in a higher fidelity model of the actual environment at the cost of slower search times and increased memory usage.

The edges of the search graph represent allowable transitions between states. A simple implementation is to connect each state with its immediately adjacent neighbors, e.g., $\langle x, y, z\rangle \to \langle x, y+\delta, z\rangle$ (where δ is the discretization size). Another alternative is to form edges based on the kinodynamically feasible trajectories the robot can perform. These kinodynamically feasible trajectories are referred to as *motion primitives* and the resulting graph as a *state lattice*. The edges of the state lattice are constructed by applying a set of pre-computed motion primitives to each state $s \in \mathcal{X}$ and then adding a directed edge from s to the state $s' \in \mathcal{X}$ that the motion primitive ends at. The motion primitives can be generated in a number of different ways. For example, they can be generated through an offline optimization process [Kelly and Nagy, 2003] or by applying a feasible control signal for a short period of time [Pivtoraiko and Kelly, 2005]. The trajectory resulting from this control input is the motion primitive. Typically, motion primitives are picked to span the space well and are symmetric in the space of controls. For example, for a car-like robot, there are usually motion primitives for both left and right hand turns, and for forward and reverse directions (see Fig. 3.1). They also need to be generated in a way that their start and end points land on the center of cells. Typically, they are generated for every possible orientation of a vehicle and then during planning they only need to be translated along the linear dimensions (e.g., the $\langle x, y, z\rangle$ coordinates) to the state for which the planner is determining successors. Since the individual motion primitives are feasible trajectories between states, the composition of multiple motion primitives between a series of states creates a feasible path for the robot. State lattice-based planning has been used in navigation for aerial vehicles [Thakur et al., 2013], automobiles [Likhachev and Ferguson, 2009], boats [Svec et al., 2013], and all-terrain vehicles [Pivtoraiko and Kelly, 2005], among others. This last work also contains an in-depth discussion on motion primitive construction.

It should be noted that in most implementations of A* the search graph is never explicitly

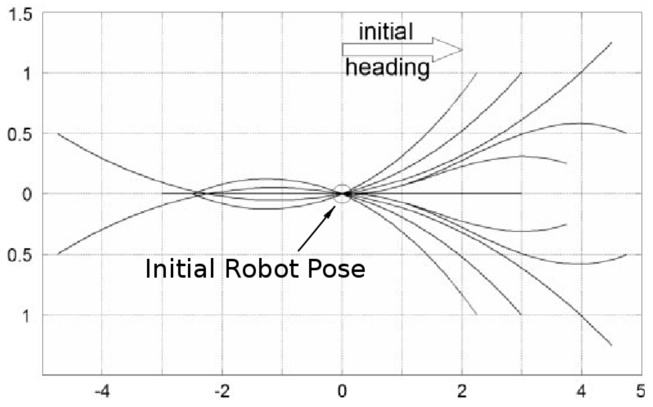


Figure 3.1: An example of motion primitives for a car-like robot

constructed completely, but instead each state and edge is constructed on-the-fly at runtime only as needed.

3.2.2 Functions and Data Structures Used by A*

A* relies on four sub-functions Successors($\cdot$), Predecessors($\cdot$), Cost($\cdot, \cdot$), and Heuristic($\cdot$). The Successors(x) function returns all of the states, x', for which the edge e from x to x' is in $\mathcal{E}$, i.e., $\{x' | \forall e(x, x') \in \mathcal{E}\}$. The Predecessors($x'$) function is identical, except that it returns the set of states x for which x' is their neighbor, i.e., $x' \in$ Successors(x). This function is not strictly required as there are alternative methods for regenerating the trajectory such as by maintaining back pointers during the search as long as there is a method to determine the preceding state of a given state on the trajectory π, i.e., $\{x | x, x' \in \pi \wedge e(x, x') \in \mathcal{E}\}$. Cost($x, x'$)[1] returns the cost of the edge between x and x' and requires that the edge exists in $\mathcal{E}$, returning ∞, otherwise. The heuristic function (frequently $h(\cdot)$) provides an underestimate of the "cost-to-go" and, in order to maintain the theoretical properties of A*, must be *consistent* and *admissible*. To be *admissible* the heuristic must be an underestimate of the cost from the state to the goal (which in turn implies that Heuristic(*goal*) = 0). *Consistency* governs the rate of change of the heuristic between states. For all states x and x' the triangle inequality must hold true:

$$|\textsc{Heuristic}(x') - \textsc{Heuristic}(x)| \leq \textsc{Cost}(x, x')$$

[1] We will also use the annotation $\textsc{Cost}_m(x, x')$ for the cost of using motion primitive m from x to x'.

For each state in the graph, A* maintains two scalar values, f and g. The g-value of a state x is the cost of traversing the graph from the start state to x along the best path discovered so far. For a given path π between start and x, $g = \sum_{\pi} \textsc{Cost}(i, i+1), \forall i \in \pi$. As the search continues, it is possible that a better path will be found to a state x' prior to it being expanded. In this case, the new, lower g-value will reach the top of the priority queue before the higher g-value, and thus, the lower one will be the one used (since the h-value is the same in both cases, the order of the f-values only depends on the g-value). The f-value is the sum of the states g-value and the value of the Heuristic function evaluated at that state. It is this f-value that is the key for the priority queue.

There are two lists maintained by the A* algorithm during execution. First is the ordered list OPEN that contains all of the potential states for expansion and is ordered by f-value. The element with the lowest f-value is the one chosen at each iteration for expansion. There is also the CLOSED list containing all states that have been previously expanded. With an admissible and consistent heuristic it is guaranteed that once a state is placed in the CLOSED list you will not find a better (lower cost) path to that state in the future.

3.2.3 A* Algorithm

The algorithm for A* is presented in Algorithm 1. Initially, the CLOSED list is empty while the OPEN list is initialized with the start state after it has itself been properly initialized (lines 1-4).

The algorithm continues to evaluate states (line 5) until there are either no more potential states in the OPEN list, in which case there is no solution, or the goal is expanded (line 7). As the algorithm expands each state (lines 6-19) it removes itself from the OPEN list (line 10), adds itself to the CLOSED list (line 11), and then adds all successor states to the OPEN list if they are not already in the CLOSED list (lines 12-18). Note: depending on the implementation, it may be more efficient to update the existing entry in the OPEN list if one exists rather than add additional entries. Before a state is added to the OPEN list the f- and g- values are updated (lines 14-15).

When the while loop exits with the goal as the currently expanded state, the path is reconstructed by traversing backwards from the goal state to the start, greedily taking the neighbors with the lowest g-value (line 24). For some implementations, including our approach, this Predecessors function can be implemented by storing a back pointer with

Algorithm 1 path = A*(start, goal)

```
CLOSED = $\emptyset$
start.$g$ = 0
start.$f$ = Heuristic(start)
OPEN = start
while OPEN $\neq\ \emptyset$ do
    CurrentState = OPEN.pop
    if CurrentState = goal then
        break
    else
        OPEN = OPEN $\setminus$ {CurrentState}
        CLOSED = CLOSED $\cup$ CurrentState
        for all $s \in$ Successors(CurrentState) do
            if $s \notin$ CLOSED then
                $s.g$ = CurrentState.$g$ + Cost(CurrentState, $s$)
                $s.f$ = $s.g$ + Heuristic($s$)
                OPEN = OPEN $\cup$ $s$
            end if
        end for
    end if
end while
if CurrentState = goal then
    while CurrentState $\neq$ start do
        path = {CurrentState; path }
        CurrentState = min$_g$(Predecessors(CurrentState))
    end while
    return path
else
    return NULL
end if
```

each state added to the OPEN list.

3.2.4 ARA⋆: Anytime Version of A⋆

An improvement to A⋆ that allows for faster searches at the cost of sub-optimal paths is ARA⋆ [Likhachev et al., 2003]. By adding a weight, $\epsilon > 1$ to the heuristic component,

$$f(x) = g(x) + \epsilon \cdot \textsc{Heuristic}(x)$$

the search will preferably expand states along the direction of the heuristic. This allows fewer expansions to find a feasible path to the goal state, with a cost bounded by the ϵ value. For the optimal cost, $\textsc{Cost}^*(start, goal)$ the actual cost, $\textsc{Cost}(start, goal)$, is guaranteed to be less than ϵ times higher.

$$\textsc{Cost}(start, goal) \leq \epsilon \cdot \textsc{Cost}^*(start, goal)$$

In addition, ARA⋆ is capable of stopping after a given period of time, saving the search information, and returning the best path found so far. If the search is later restarted, it can continue to decrement ϵ as it continues to refine the search, and, if given sufficient time, will reach the optimal solution when $\epsilon = 1$.

Chapter 4

Planning for Multi-Robot Exploration With Multiple Objective Utility Functions

Acting like a robot, its metal brain corrodes
You try to take its pulse, before the head explodes
MEGADETH *Symphony of Destruction*

Collaborative exploration involves the systematic traversal of a group of robots through an environment typically with the purpose of determining the extent of the environment or of finding a particular element in the environment. Tasks relying on exploration are mapping, search-and-rescue, and infrastructure inspections. When multiple robots are involved, a key element of collaborative exploration is to determine, in a principled manner, the goal location assignments for each team member. There have been numerous methods proposed as discussed in Chapter 2 and Chapter 3. Our approach is an extension to the "frontier" exploration scheme [Yamauchi, 1997]. Generally, the frontier is defined as the set of states that are both *free* and *known* but have an *unknown* neighbor. For example, in a 2-D scenario, the frontier could be the 2-dimensional points ($\langle x, y \rangle$) that is *free* and *known* but with at least one *unknown* state in an 4-connected neighborhood.

To guide the exploration process towards "better" portions of the environment we use the concept of information gain (IG) to score the usefulness of states based on how much new information they can provide if a robot moves to that state. All states initially start out as unknown, and as we gain sensor readings on that state, it becomes known - independent of whether it is known to be empty (free space) or known to be an obstacle. We combine this information gain with other factors representing exploration priority, distance the robot must travel, etc. to determine an overall utility score for that state. One key question is whether it is more important to have complete coverage in the shortest amount of time, or to maximize the rate of coverage. By adjusting the weightings of the different factors, the user can exert this kind of high-level control on how the exploration is taking place.

Overall, the multiple objective utility function exploration algorithm operates by:

1. Finding the frontier of the unknown region
2. Determining the information gain for the states
3. Scoring states based on information gain, distance, and other user definable preferences
4. Assigning "best" state to the "best" robot
5. Repeating 1-4 until all robots have assignment

The benefits of this approach is that some of the work is easily parallelizable allowing for different threads or even different robots to do the bulk of the computational work. Only the final selection requires the scores from each robot (step 4). The other benefit is this scheme allows for a rich set of operator input into the exploration strategy without compromising any of the guarantees about performance. The operator is capable of selecting preferred regions, assigning priorities to different areas, or to leave the system alone and allow the robots to determine on their own where to explore.

4.1 2-D Exploration - Homogeneous Multi-Robot Teams

The first version of this algorithm is focused on exploration of a 2-dimensional environment by a team of homogeneous ground robots. The algorithm assumes that all of the robots are similar in terms of sensing, computation, and locomotive capabilities. In addition, the algorithm assumes that planar ray-casting is sufficient to determine which states are visible from a given state. While this second assumption is not required by the algorithm, by enforcing this requirement planning times can be reduced.

The algorithm follows the general form laid out in Section 4 and can be divided into three primary stages: pre-processing of the input map, approximate ranking of the frontier states based on a heuristic, and forward simulation of the most promising goal states to determine the best assignment for each robot, as seen in Algorithm 2.

The pre-processing stage generates the coverage, cost, distance, and information gain (IG) graphs for use in the later stages as shown in lines 2-4 and line 8 of Algorithm 2. These graphs are used as the basis for conducting the search for a candidate trajectory.[1] To increase the likelihood of choosing the optimal goal state each frontier state is placed in a sorted list ranked by the estimated utility score divided by the cost to reach that state as part of the second stage of the algorithm, lines 8-10. In the third stage, lines 11-22, the paths to these states are forward simulated in rank order and the best goal state is assigned to the corresponding robot. Since the information gain is dependent on the trajectory taken, finding a globally optimal path is intractable for online real-time applications; instead we settle for its approximation.

The estimated utility score is composed of the information gain per unit traversal cost along with any bias terms related to the additional features and their associated weights. By using the information gain per unit traversal cost to select the next path, the algorithm may bypass isolated states while moving towards larger unexplored areas of the map. By applying a weighting factor to the traversal cost, the ratio can be skewed to make information gain more or less important relative to the distance. In this way the thoroughness of exploration can be adjusted to meet the exploration requirements at execution time. The bias term has been introduced to control robot behavior such as minimum and maximum separation distances, and region assignments.

While our example implementation uses a two-dimensional 8-connected grid map as our base graph for the environment, the algorithm is capable of operating on any graph based map.

4.1.1 Stage 1: Map Processing

In the first stage of the planner, the input map is pre-processed to gather the necessary statistics for adequate ranking. The input map is an $m \times n$ grid with values proportional to the log probability of each state being occupied or free. The output is a set of array's.

The cost map is the first array and it represents the environment as an $m \times n$ grid with

[1] In our approach, the graph is 8-connected, but we only use 4-connected frontier states to ensure sensor visibility.

Algorithm 2 Exploration Algorithm

```
// pre-process maps
[costMap, coverageMap] = Process(map m)
costMap = Inflate(costMap)
distMap = Dijkstras(costMap, coverageMap)
RR = remainingRobots()
while RR.notEmpty do
    // calculate IG and rank states
    IGMap = InfoGainEstimate(coverageMap)
    bias = CalcBias(m)
    frontierHeap ← CalcScore(RR, IGMap, distMap, bias)
    while t < planTime do
        // forward simulate each state
        pt ← frontierHeap.pop
        $\xi$ = PlanPath(pt)
        score = EvaluateTrajectory($\xi$);
        if bestScore < score then
            bestScore ← score
            bestTraj ← $\xi$
            bestRobot ← pt.robot
            robotGoals[bestRobot] ← pt
        end if
    end while
    coverageMap ← coverageMap ∪ visible($\xi$)
    RR.pop(bestRobot);
end while
return robotGoals
```

the value of each state proportional to the cost of traversing the corresponding state of the input map, line 2. This cost value can be augmented to include other factors such as time or energy to traverse the state. In our example implementation, the cost is proportional to the distance moved through the environment. This cost map is augmented by a buffer applied to each non-traversable obstacle, line 3. This artificially increases the cost associated with traversing near obstacles; in effect forcing the robots to take slightly longer paths but providing additional margin to collision. These parameters can be specified and updated at any time by the higher level systems to adjust the behavior of the robot.

The $m \times n \times r$ distance map is the second array generated, line 4. The m and n dimensions are identical to those used for the cost map while r is the number of robots currently exploring. Each $m \times n$ layer, r_{id}, has values corresponding to the distance in meters from a given state to robot *id* as calculated using a standard Dijkstra's search algorithm on an 8-connected grid. For our implementation we did not allow traversal of unknown areas by setting the cost for all unknown states to $+\infty$ prior to running the Dijkstra search, however any non-negative weight can be given to unknown areas depending on the desired traversal characteristics.

The third and final array is the $m \times n$ coverage map which provides an estimate of the information gain possible from each cell. The information gain is derived from the occupancy grid provided as input, line 8. As the absolute value of the information about a state grows higher (the state is more confidently free or more confidently an obstacle), the level of knowledge about that state increases. Each state k is assigned a value q_k that is inversely related to the level of knowledge of that state; in a state for which no doubt existed $q = 0$, whereas for a completely unknown state $q = 1$. As the robot learns more about a state k through sensing, the q_k value decreases towards zero. Since the value of q_k is a measure of the confidence of the sensor reading for state k, it can be converted directly from the log probability of occupancy (if using an occupancy grid) or derived from the sensor model. The information gain is calculated for a state j from the coverage map by (4.1).

$$ig_j = \sum_{k \in \textsc{vis}(j)} q_k \tag{4.1}$$

The function $\textsc{vis}(j)$ returns the set of states visible from a state x_j. For our implementation $\textsc{vis}(j)$ assumes a horizontally mounted laser scanner capable of sensing along rays emanating from the robot and extending out to the first obstacle taller than (and thus intersecting with) the plane of the sensor. This can also be a probabilistic model of the sensor suite, to account for less accurate sensors.

If a state k is visible from two states i and j, then the information gain of i and j become dependent on the order the two are visited. Once the robot enters state i, it will gain knowledge of state k lowering its q value resulting in state j having a lower information gain. Due to this interdependence between all of the available states, the information gain for a given cell at a given time can only be estimated. Initially, we set $IG_{j,estimate}$ to the value for the terminal cell, ig_j, assuming that the robot does not gain any information prior to reaching state x_j. This assumption leads to an over-estimation of the information gain of the final state, but typically is an underestimate of the information gain for the route.[2]

4.1.2 Stage 2: Ranking Frontier States

The second stage of the algorithm is the determination of an ordered list of frontier states. A frontier state is a state that is known ($q = 0$) and that has at least one of its 4-connected neighbors with $q \neq 0$. Each frontier state, i is ranked based on the score calculated element-wise as in (4.2). The frontier list itself is simply a max-heap based on the score of each entry, line 10.

$$\textsc{Score}[i] = \frac{(IG_{i,estimate})^{\Xi}}{(\textsc{Distance}[i])^{(1-\Xi)}} \cdot bias_r \cdot bias_d \cdot bias_\gamma \tag{4.2}$$

The $IG_{estimate}$ and the $\textsc{Distance}[\cdot]$ values are taken directly from the information gain and distance maps, respectively, computed in the first stage. The Ξ-value is a user selected parameter that adjusts the relative importance of distance relative to information gain, $0 \leq \Xi \leq 1$. As Ξ approaches 1 the algorithm prefers higher information gain goals, as it approaches 0 it prefers lower cost (shorter) paths, and when $\Xi = 0.5$ it is neutral. The final term, $bias_{r \cdot d \cdot \gamma}$ is a robot specific term that is the product of three additional elements to control the goal selection process; region bias, distance bias, and repetition reward. Since

[2] In free-space, this would be guaranteed to not be an overestimate of the information gain for the route, however, since the unknown states beyond the frontier may have an obstacle just out of sensor range, the information gain may be substantially less than predicted. In a typical office environment this lower information gain is typically of the same magnitude as the information gained during the traversal of the route resulting in the free space gain at the goal position being a relatively accurate estimate of the information gain for a path terminating at that same goal state. Different sensor and environmental characteristics may require a different estimation scheme.

this and the DISTANCE[·] term are robot specific, a given frontier cell will be in the frontier list as many times as there are robots exploring, line 9. Therefore, each individual entry on the frontier list specifies a robot-goal state pair.

During exploration with heterogeneous robots, it may be desirable to have a specific robot explore a specific subset of the environment. The algorithm handles this by allowing two distinct types of region assignment. The first is to assign a robot *id* to a given region, the other is to specify the number of robots allowed into a given region without specifying discrete id's.

The region bias term encodes this region information into the score of each frontier cell. In the first case, the region has a user specified set $\mathcal{R}$ of robots explicitly tasked to explore the area and a multiplier m specifying the level of attraction. For a robot numbered *id* the region bias is calculated by (4.3)

$$bias_r = \begin{cases} m & id \in \mathcal{R} \\ \frac{1}{m} & \text{otherwise} \end{cases} \tag{4.3}$$

The second type of region has no specified set of robots assigned to explore it. For this type of region the bias is calculated by (4.4). This function allows for up to n robots to be in an area without penalty.

$$bias_r = \begin{cases} m & id \text{ is among the first } n \text{ robot(s) in the region} \\ \frac{1}{m} & \text{otherwise} \end{cases} \tag{4.4}$$

Both types of regions reward robot-goal state pairs that place the robot either in their assigned regions or into a region without too many other robots.

The distance bias only applies to robots in the same region (including the default, "None" region). For these robots, a penalty is applied to any goal state evaluation that results in the robot being either too far or too close to another robot's assigned goal state. The distance bias is given by (4.5).

$$bias_d = \begin{cases} 1 & \text{if only robot in region} \\ 1 & D_{min} \leq d_{nearest} \leq D_{max} \\ \frac{d_{nearest}}{D_{min}} \cdot penalty & d_{nearest} < D_{min} \\ \frac{D_{max}}{d_{nearest}} \cdot penalty & d_{nearest} > D_{max} \end{cases} \tag{4.5}$$

D_{max} and D_{min} are the maximum and minimum ranges desired between a robot and its nearest neighbor. For our implementation, D_{max} is based on wireless coverage range, to prevent loss of network connectivity, while D_{min} is based on maximum sensor range, to minimize unnecessary overlap. The *penalty* term is used to tune the relative importance of the distance bias term.

The final bias term provides $bias_\gamma$ to the robot for selecting the same goal as during the previous iteration. This term effectively requires a new goal state to be at least γ times better than the previous goal before the robot would rank it higher and thus change goal states. With this implementation adding penalties or rewards to the utility function based on additional features was made simple since all bias terms were multiplicative.

At the end of this stage of the algorithm, all of the frontier states are in a sorted list, once for each robot.

4.1.3 Stage 3: Forward Simulation and Goal Assignment

The third stage of the algorithm is to forward simulate the paths to potential goal states in order to determine a better estimate for the information gain for for a given goal-robot pair. During this stage the highest ranked frontier state-robot pair is selected from the frontier list, line 13 and the trajectory is forward simulated to determine which states can be observed based on the current obstacle map, line 14. This information gain is then used in (4.2) in place of $IG_{estimate}$ to determine the actual score for the state-robot pair, line 15. As processing time allows, additional frontier states are removed from the frontier list and forward simulated, with the best robot-goal state pair being saved, lines 16-21. At the conclusion of the allowed processing time, the state with the highest score is paired with its associated robot, and that robot is removed from the list of eligible robots, line 24. The information gain array is recalculated based on the predicted movements of the newly assigned robot, line 23, and stage two and three are repeated for the remaining robots. This process repeats until the last robot receives an assignment. In this way each robot has received the best overall assignment that it could have from the states that were forward simulated.

4.1.4 Example

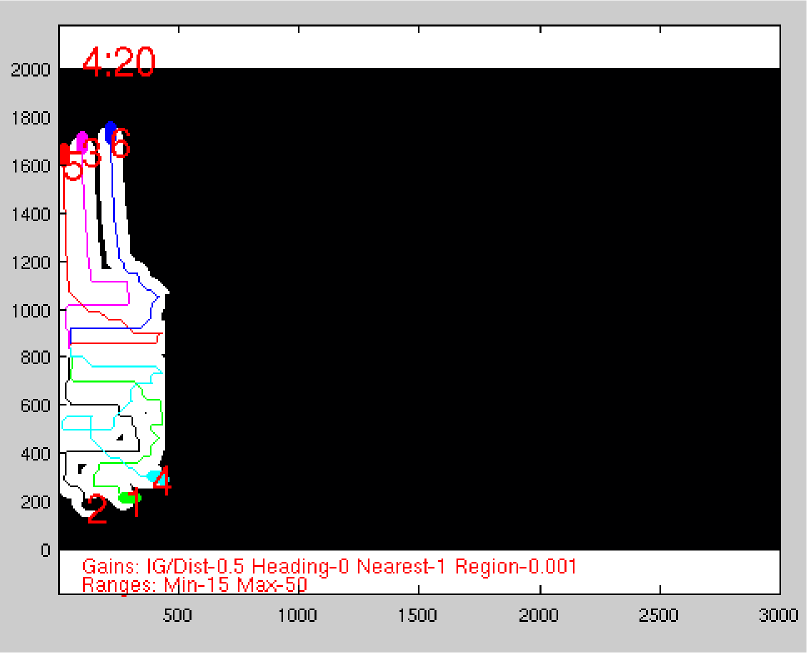


Figure 4.1: Six robots beginning an exploration simulation. The black area to the right is unknown, white is free space and the thick colored lines are the exploration trajectories for the next time step. The thin line represents the trajectory history of each robot.

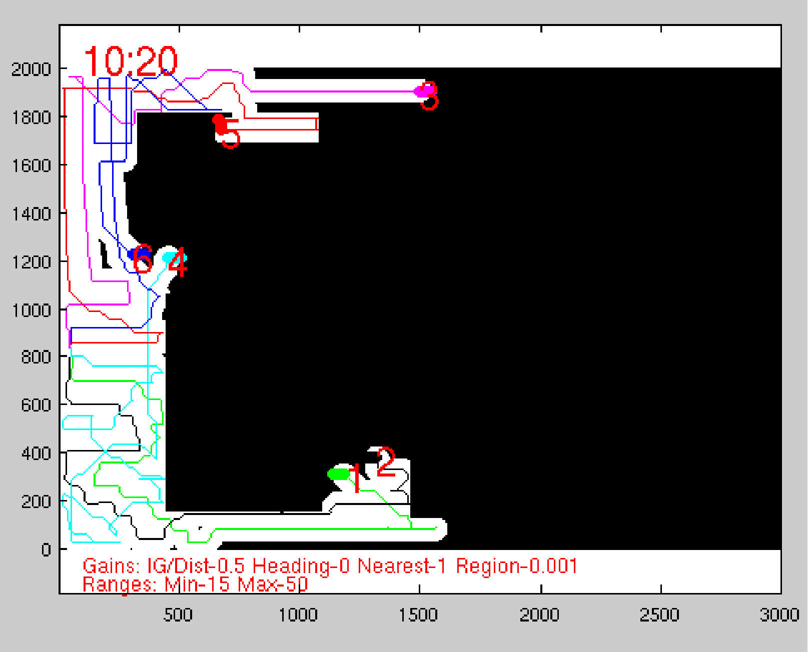


Figure 4.2: After ten minutes of exploration, robot 5 has entered a building area in the upper left corner.

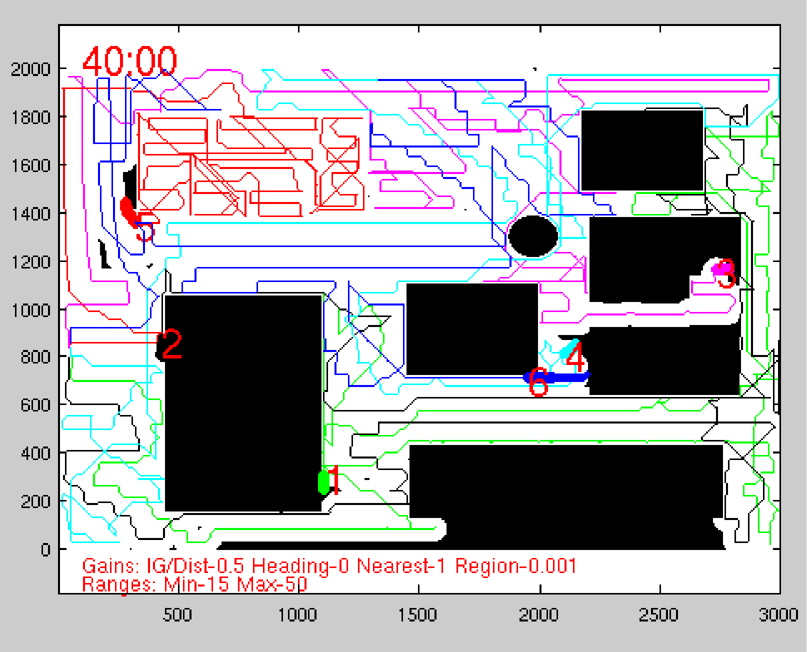


Figure 4.3: After 40 minutes of exploration. Robot 3 has now entered a building in the center right while robot 5 has finished the upper left building and resumed exploration outside. The medium thickness lines are exploration trajectories longer than one time step; the robot will complete the heavy line prior to the next trajectory evaluation.

As an illustrative example, Fig. 4.1-Fig. 4.3 depict various snapshots of the exploration process of a 200 m by 300 m area. The environment has two buildings that can be internally explored and several other obstacles. For this sequence, six robots were used with an effective sensor range of 7 m. The Ξ-value was 0.5 (balancing information gain with distance), multiplier *m* was 1000, *penalty* was 1 and D_{max} and D_{min} were 50 m and 15 m, respectively. In Fig. 4.1, the robots have just begun their exploration routines. The robots have separated into two groups with the spacing within each group at the lower end of the allowable window. After 10 minutes of exploration (Fig. 4.2) robot 5 has found one of the general exploration regions around an explorable building in the upper left corner of the image and has begun the exploration of the interior. After forty minutes of exploration, the map is largely complete with robot 3 surveying the other accessible building, and the other robots cleaning up islands of unknown regions as shown in Fig. 4.3.

4.2 3-D Exploration for Non-Homogeneous Teams

In Section 4.1 we introduced a framework for multi-robot exploration that allowed the operator to provide general guidance to the team of ground robots while they explored an unknown area. While the previous algorithm will work in higher dimensions, the visibility calculation performed while forward simulating the robots motion can be computationally expensive. In this section we expand the previous algorithm to account for the added complexity of performing a search in a full 3-dimensional environment using an air-ground team, as well as allow for an exploration sensing system that is independent from the localization sensors.

The 3-D exploration algorithm provides goal locations for the robots to survey in an effort to minimize search time and overlap between robots. One key change from the 2-D algorithm is that the frontier points themselves are not chosen as goal states, but rather the algorithm selects from all states that can sense the frontier. By selecting goals in this way, it is possible to gain information about multiple portions of the frontier simultaneously. As an example, given a 3-D environment with a frontier both above and below a desktop, it would be inefficient to direct a robot to one or the other. Instead, the robot should be directed to a location a few meters away and facing the desktop so that it can simultaneously sense the frontier both above and below the desktop. By evaluating all states rather than just the frontier states, the algorithm can reason about the locations providing the maximal information gain.

A second difference is that individual robot trajectories are not set by the planner. The navigation computations are left to the individual robots to determine so only the goal selection is performed in a centralized manner. In practice, one of the robots of the team performs the exploration computations alleviating any need for a separate ground station.

The algorithm follows the same general steps outlined in Section 4.

1. Generate distance to accessible states,
2. Determine frontier cells from map,
3. Accumulate the total number of frontier cells that are capable of being sensed from each state,
4. Score the states based on the counts, distance, and any user-defined inputs,
5. Select the best state and assign it to the applicable robot,

6. Repeat until each robot has an assignment.

While our example implementation only uses a two-robot pair, the algorithm is extensible to n independent robots and has been used in past work with up to eight robots.

4.2.1 Exploration Algorithm for Air-Ground Teams

The planner relies on a combined map produced from an independent map merger module. This combined map identifies each cell as *clear, obstacle* or *free* (or some probabilistic combination, e.g., an occupancy grid) as well as *seen/unseen*. This last label informs the planner whether the cell has been sensed by the exploration sensor and forms the basis for generating the frontier cells.[3] The algorithm is shown in Algorithm 3 and is a modification of the algorithm presented in Section 4.1. Of note, the subscript following variable names indicates the dimensionality of the data where 3 indicates the three spatial dimensions, $\langle x, y, z\rangle$ and 4 indicates the three spatial dimensions plus a heading dimension, $\langle x, y, z, \psi\rangle$.

Algorithm 3 $G_4[\cdot]$ = GetGoal(poses $p_4[\cdot]$, robots $r[\cdot]$)

```
Global: $map_3, NumRobots$
for all $i$ in $NumRobots$ do
    $CountMap_4[\cdot] = 0$
    $InflatedMap_4[\cdot] = \textsc{InflateMap}(map_3, r[i])$
    $CostMap_4[\cdot] = \textsc{Dijkstra}(InflatedMap_4, p[i])$
    $FrontierPts_3[\cdot] = \textsc{FindFrontier}(map_3, r[i])$
    for all $f_3$ in $FrontierPts_3[\cdot]$ do
        $ViewPts_4[\cdot] = \textsc{Views}(FrontierPts_3[f], r[i])$
        for all $v_4$ in $ViewPts_4[\cdot]$ do
            $CountMap_4[v_4]$++
        end for
    end for
    for all $pt_4$ in $CostMap_4[\cdot]$ do
        $score_4[p] = \textsc{ScorePt}(CountMap_3[pt_4], CostMap_4[pt], G)$
    end for
    $G_4[i] = argmax_{p \in CostMap_4}(score_4)$
end for
return $G_4[\cdot]$
```

[3] For our implementation, we use a LIDAR type sensor for localization and map building and a vision system to explore the environment

During each planning iteration the exploration planner will generate new goals to all robots (that are ready to accept new goals) based on the current map. In practice, we transmit a goal to each robot as soon as the goal is determined and repeat the loop continuously. The inputs to the GetGoal function are the $\langle x, y, z, \psi\rangle$ poses p of all of the robots and a parameter list r providing the footprint, nominal altitude (= 0 for a ground vehicle), sensor field of view, and sensor position and orientation relative to the robot body frame, for each robot.

The planner first inflates the obstacles in the combined map to account for the footprint of the robots (line 4). Because of having possibly non-circular robots, the inflated map has the robot heading ψ as a dimension. The inflation is followed by a Dijkstra search starting from the current location of the robot to determine the cost to each accessible state in the environment (line 5). Once these two initial processing steps are completed, the map is parsed to determine all of the frontier cells. In the 2-D algorithm we defined "frontier" as a *known, free* cell directly adjacent to an *unknown* cell. For the 3-D algorithm we modify that definition to be an *unknown* cell directly adjacent to a *known, free* cell. For our system we define "directly adjacent" to mean cells that differ along only a single dimension by one unit, i.e., $\langle x, y, z\rangle$ and $\langle x, y + \delta, z\rangle$ for a discretization size δ, are adjacent but $\langle x, y, z\rangle$ and $\langle x + \delta, y + \delta, z\rangle$ are not. Note: frontier cells are not defined by any heading information. We also modify the definition of frontier in that we consider a visually cleared cell as known, and all others, even if we have laser data on that cell, as unknown (line 6). Using these definitions, we generate a second category of cells, the *view-points*. These view-point cells are any cell that can sense a frontier cell and then from the set of all view-points, we select our goal cells. By using this approach, we can guarantee that the algorithm will fully explore the environment, given sufficient battery and permissible obstacle configuration,[4] as it will maneuver to all view-points and thus eventually sense all *unknown* but not blocked cells within the environment.

Since our goal is to determine information about the map, selecting goal points with higher information gain is beneficial. Similar to the 2-D algorithm, we approximate the total information gain accrued by traversing to a given 4-dimensional state by the number of frontier cells visible from that terminal state. To accomplish this, for each frontier cell we determine the 4-dimensional set of states that the robot could be at in order to successfully sense the frontier cell. This requires knowledge of the robot sensing model including the

[4] The obstacles are considered in a permissible configuration if the robot can maneuver to aim the exploration sensor at all desired locations.

field of view, mounting location, and maximum effective range (lines 8-11). The sensing model is used to ray-trace within the field of view of the sensor from the robot out to the nearest obstacle and to subsequently annotate the intervening states as obstacle-free and seen.

By only reasoning about the terminal state, it is possible that the robots could generate a trajectory to that state that if modified slightly would result in a significantly higher information gain enroute. Computing the total information gain for all possible paths to all possible points in the environment is intractable. In most closed environments (indoors, for example), the few directions that have sightlines greater than the sensor range are also the directions of possible exploration. The result is that the robot typically travels in a direction normal to the visible frontier viewpoints. Thus, reasoning about the terminal points in these situations results in identical behavior. Different methods of determining the information gain may be warranted for environments that do not follow this characteristic.

Finally, each potential state in the reachable space receives a score based on how many frontier cells are visible from that state, the distance the potential goal state is from the current state, and a penalty term. This is analogous to the score calculated in (4.2) with count replacing information gain and cost representing a more general form of distance. The distance and count terms are weighted by a user-defined parameter, $0 \leq \Xi \leq 1$ that adjusts the propensity to move farther to get a higher information gain or to select a nearby but not very lucrative state (4.6). For $\Xi = 1$ the planner will select the state with the highest information gain, ignoring the distance term. Conversely, setting $\Xi = 0$ will result in the planner selecting the lowest cost state without regard to the information to be gained.

$$score[i] = \frac{count[i]^{\Xi}}{cost[i]^{(1-\Xi)}} \cdot bias[i] \tag{4.6}$$

The bias term can incorporate a wide range of user preferred behavior. For our system, this term was constructed to downgrade states that are in close proximity to any other robots goal state. In addition, this term also penalized very short range motions that are harder to execute (4.7). The threshold$_L$ and threshold$_D$ values are set by the user. The Length threshold$_L$ duplicates to some extent the Ξ parameter. The difference being that threshold$_L$ is an absolute value - the bias is applied independent of the information gain - while the Ξ parameter only changes the relative importance of cost vs. information gain.

$$bias[i] = \textsc{Length}(i) \cdot \min_{j, i \neq j} \Big(\textsc{Proximity}(i, G(j)) \Big) \tag{4.7}$$

$$\textsc{Length}(a) = \begin{cases} 1 & \textsc{CostMap}(a) \geq \text{threshold}_L \\ \frac{\textsc{CostMap}(a)}{\text{threshold}_L} & \text{otherwise} \end{cases}$$

$$\textsc{Proximity}(a, b) = \begin{cases} 1 & \text{dist}(a, b) \geq \text{threshold}_D \\ \frac{\text{dist}(a,b)}{\text{threshold}_D} & \text{otherwise} \end{cases}$$

Due to the UAV's limited flight time, we provide a further enhancement to maximize the UAV's value. When determining the frontier points on line 6, the exploration planner first considers only those points that are not visible to the UGV. In this way, the planner will first send the UAV to cover portions of the environment the UGV cannot sense. Once it has exhausted the UAV-only points without finding any candidates, it will reevaluate based on all frontier points. This process may transition multiple times between UAV-only and all frontier points as the environment is explored and new obstacles discovered.

In order to determine which states are or are not visible to the UGV, we ray-cast from the potential target state to the set of states that the UGV sensor could be located in (taking into account orientation and obstacles). If all rays encounter obstacles then the state is not visible. In all other cases there exists at least one viable configuration of the UGV that will allow the target state to be sensed.

For the non-homogeneous version we use the same greedy assignment scheme detailed in Section 4.1.3. However, this scheme may introduce sub-optimal behavior when used with heterogeneous robots with significantly different capabilities. For example, given two robots, one with a sensing range of 1 m and the other with a sensing range of 100 m, and an environment with a large number of overlooks[5], the second robot will need to visit many locations, independent of whether the first robot has visited them or not. In this case, it would be better if the first robot went to the areas where it could clear on its own (i.e., , narrow hallways). This non-optimal behavior is not typical, however, and in practice the robot assignments were in line with what an outside observer would assign.

[5] We define "overlook" in this context to be positions from which the robot can sense regions of the environment, but is unable to move closer to some portions.

4.3 Exploration Summary

We have presented two algorithms to perform multi-robot collaborative exploration in complex environments. By incorporating the idea of the multiple objective utility functions, we allow for high-level user input during exploration to modify the priorities of the robots. Our algorithm can perform the bulk of the computational effort in a decentralized manner, only requiring the final goal assignment to be done in a centralized fashion.

Chapter 5

Applications of the Multi-robot Exploration Planner with Multiple Objective Utility Function

A call from the heart to the god of love,
Send us an angel, a sign from above.
All of the damned and a silent scream,
Save us and damn the machine

GAMMA RAY *Damn the Machine*

In this chapter we will present some applications of the algorithms developed in Chapter 4.

5.1 Exploration with a Homogeneous Robotic Ground Team

Our first application is of the 2-D exploration algorithm developed in Section 4.1 and implemented with a team of homogeneous ground robots both in simulation and in real-world testing.

5.1.1 Simulation of 2-D Homogeneous Exploration Planner

The algorithm was run on a series of randomly generated 2-dimensional grid maps similar to Fig. 5.1. For each test, 5 robots were run on maps of $1000 \times 1000, 2000 \times 2000$, and

3000×3000 cells at 0.1 m per cell. During testing the minimum distance was set to the sensor range we saw in practice while the maximum distance was based on achievable communications ranges. A range of values was tried for all other user adjustable values and validated during subsequent real-world testing. The final values used for all of the tests were Ξ set to 0.5, and $bias_r$ to 1000 and 0.001 for the first and subsequent robots, respectively. The distance bias penalty was set to 1 and $D_{min} = 20$ m and $D_{max} = 50$ m. The robot was given a sensing radius of 10 m and was allowed up to 0.3 s of planning time during the forward simulation stage, resulting in an overall system replan every 10 s. The robots were also assumed to have a top speed of 1 m/s allowing 10 m maximum between replans. The results can be seen in Fig. 5.2 displayed as the fraction of the accessible space explored versus number of time steps. In addition, a combined indoor/outdoor map was generated to show the effect of logical area assignments and a hard inter-robot distance constraint, as shown in Fig. 5.3 and Fig. 5.4.

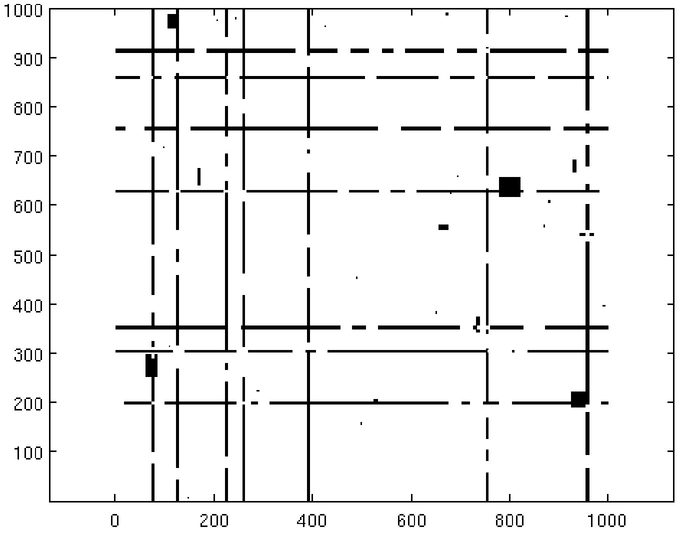


Figure 5.1: Example of the randomly generated map used to test the algorithm

5.1.2 Real Robot Results: MAGIC2010 Challenge

This algorithm was put to use on the University of Pennsylvania's team for the Multi-robot Autonomous Ground International Challenge 2010 (MAGIC2010) held in Adelaide,

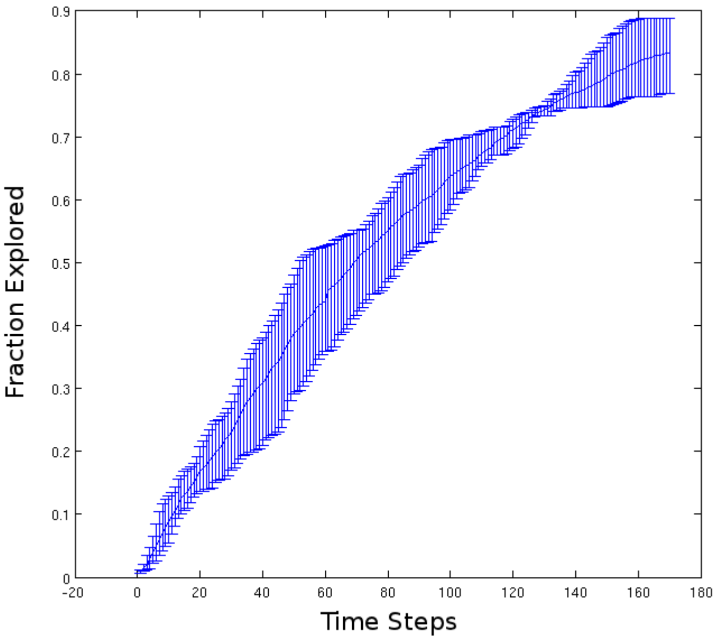


Figure 5.2: Exploration versus Time Step (10 second increments) for a 2000 × 2000 cell map at 0.1 m resolution and comprised of ~ 11% obstacle cells. Solid line represents the average fraction of free cells that have been detected while the error bars represent the maximum and minimum over the five random maps. Similar results were obtained for the 1000 × 1000 and 3000 × 3000 maps with a corresponding change in number of time steps.

Australia in November 2010. For the main competition, a team of robots were required to enter an area comprised of indoor and outdoor environments, explore it, identify five different types of Objects of Interest (OOI's) and provide a detailed map with the objects locations pinpointed. In addition, for two types of objects, a specific sequence of actions had to be performed before a robot could move past the OOI. Two of the OOI types were mobile, while the rest were stationary. The 3 areas used ranged from 20.000 m^2 to 30.000 m^2. A second competition using a similar set of rules, but without any mobile OOI's and conducted entirely indoors was also held (Fig. 5.5 and Fig. 5.6).

In the main competition, the team took 2^{nd} place. Unfortunately, unrelated mapping issues prevented the gathering of useful data regarding the exploration algorithm. However for the second competition, the exploration algorithm was able to completely map a space approximately 3600 m^2 with 5 robots in under 35 minutes, identifying 9 of 12 OOI's in the process and garnering a first place for the team. For this implementation the algorithm was

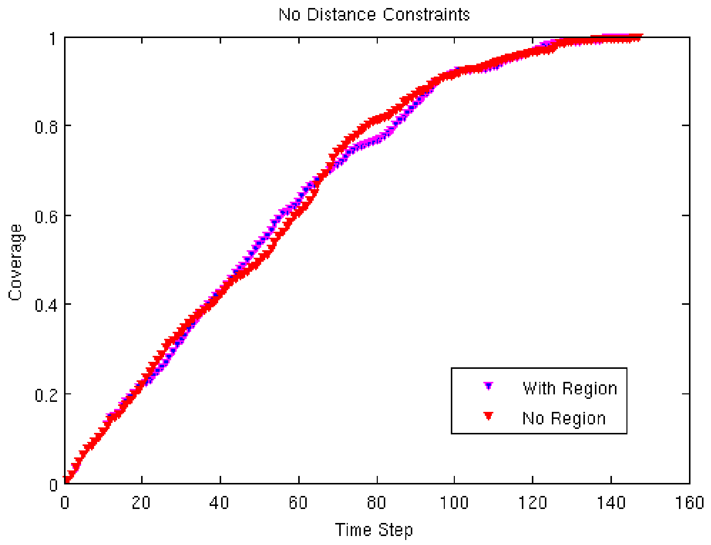


Figure 5.3: Comparison between coverage rate with and without regions for the case with no inter-robot distance constraints. When the robots are free to wander they perform about the same in both cases.

given five seconds total to process before having to provide an assignment for each of the robots. Computation occurred on a 2.80 GHz quad-core i7 running Ubuntu 10.04. Eight threads were spawned to evaluate the frontier cells and perform other tasks concurrently.

5.1.3 2-D Exploration Application Conclusions

In this section I have presented applications demonstrating our extension of the frontier-based approach to multi-robot exploration that allows for the incorporation of multiple objective utility functions. This extension enables the operator to adjust the exploration priorities both for the individual robots and the group as a whole. The algorithm was implemented on a team of five physical robots that took first place at the Old Ram Shed Challenge and second place at the MAGIC2010 main competition, both devoted to search and rescue operations.

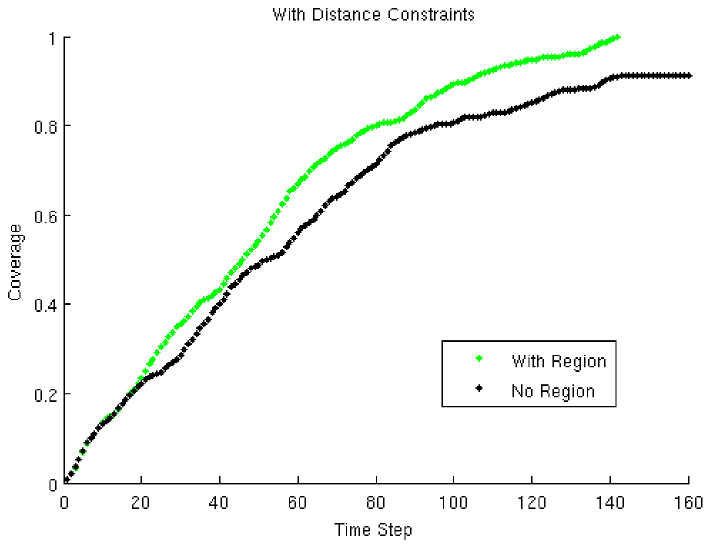


Figure 5.4: With distance constraints there is a clear advantage to specifying regions for exploration.

5.2 Air-Ground Robotic System for 3-D Exploration

In this next section I will demonstrate the 3-D variant of the exploration algorithm presented in Section 4.2.

5.2.1 Overview of Air-Ground Robotic System

We concentrate on the problem of autonomous exploration for the purpose of finding an object of interest (OOI) within an initially unknown environment. The robotic team will have minimal human input: the human operator will initiate the exploration, concur with launching the UAV, and concur with any OOI detections, but will not provide any other direction or guidance to the robots.[1] Therefore, all other navigation, sensing, and decision making must be done on-board. To achieve this, we developed several modules to guide the robots through the environment as shown in Fig. 5.8.

Our system uses two primary components: a team of robots executing their own software for localization, navigation, object detection, and other local processes; and a set of high-level software modules that combine the individual robot maps, select goal locations for

[1] As part of a separate line of research regarding the User Interface, we imposed the constraint that the human would have to confirm OOI detections. Because of this, we included a decoy object in the environment during testing.

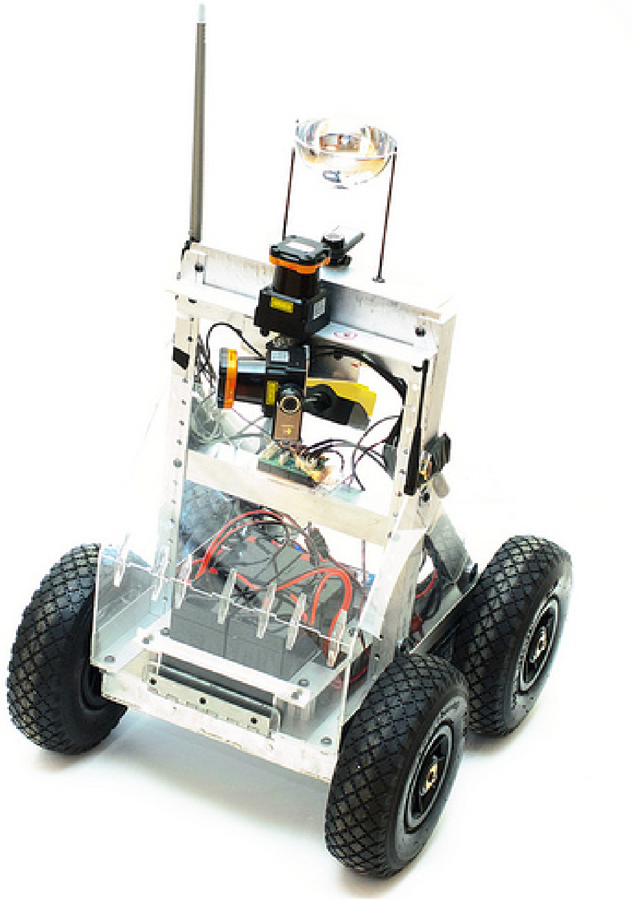


Figure 5.5: One of the University of Pennsylvania's MAGIC2010 robots. The robots used two laser range scanners, an omni-directional camera, and a panning camera to gather information about the environment. An on-board Mac Mini provided computational power, and a 802.11 antenna provided the data link with the ground control station (GCS). (Photo © 2010 Paul Vernaza)

each robot, and interface with the user.

The first part is our two robots: an Unmanned Aerial Vehicle (UAV) and an Unmanned Ground Vehicle (UGV) (Fig. 5.7). The UGV has a large battery capacity (sufficient for 3-4 hours of operation), substantial on-board computing power, and is very stable. On the other hand, the UAV has a limited flight time (10 minutes maximum) and computing power, but can traverse terrain that the UGV cannot. In addition, it can move vertically allowing it to get a better vantage point of the environment and "see" areas the UGV cannot. The robots are detailed in Section 5.2.2 and their on-board software in Section 5.2.3.

The second part of the system is the high-level modules which are responsible for

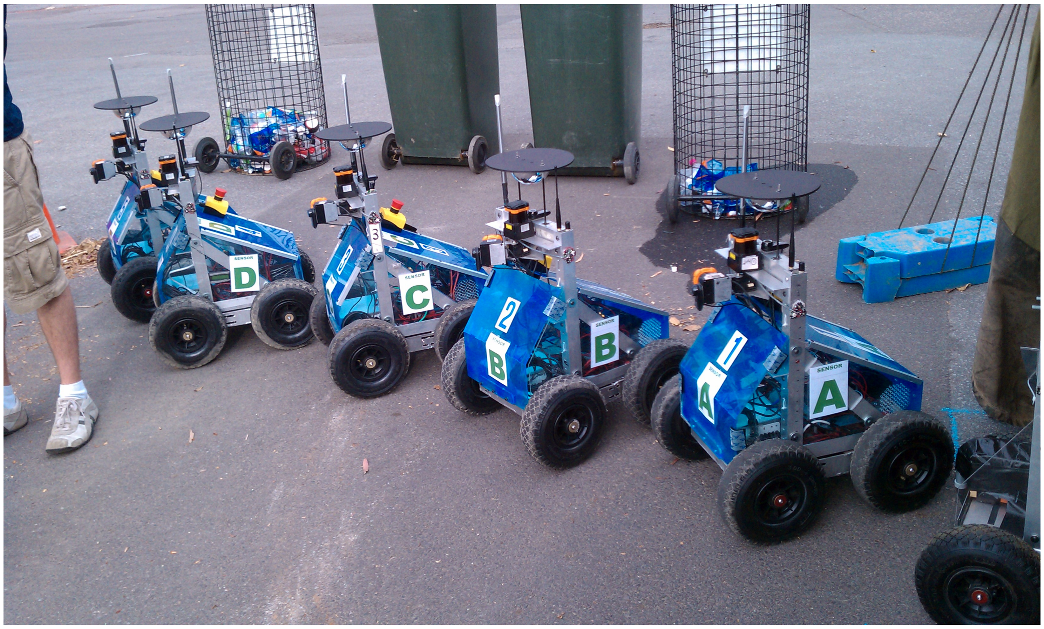


Figure 5.6: The team of exploration robots. The two neutralization robots are not shown.

coordinating activities between the two robots and can be executed from any available computing platform.[2] To perform their coordination function, the high-level executive has access to a map merge capability and the exploration planner. The map merger receives map updates from the two robots and forms a global map. Using this map, the exploration planner determines appropriate goals for each robot and provides them to the executive module for transmission to the robots. In addition, the high-level software incorporates a mechanism for providing feedback to, and input from, the human operator. The high-level software modules are discussed in Section 5.2.4.

5.2.2 Robot Platforms

Melvin the Segbot

The ground vehicle component of our system is Melvin the Segbot. Melvin is a custom designed robot built on a Segway RMP 200 base. Attached to this base are two computers, two 30 m Hokuyo scanning laser range finders, and a Logitech webcam (see Fig. 5.9a).

The computing power is split between two distinct hardware components. The first is

[2] In practice, the high-level systems were all executed on the UGV computers.

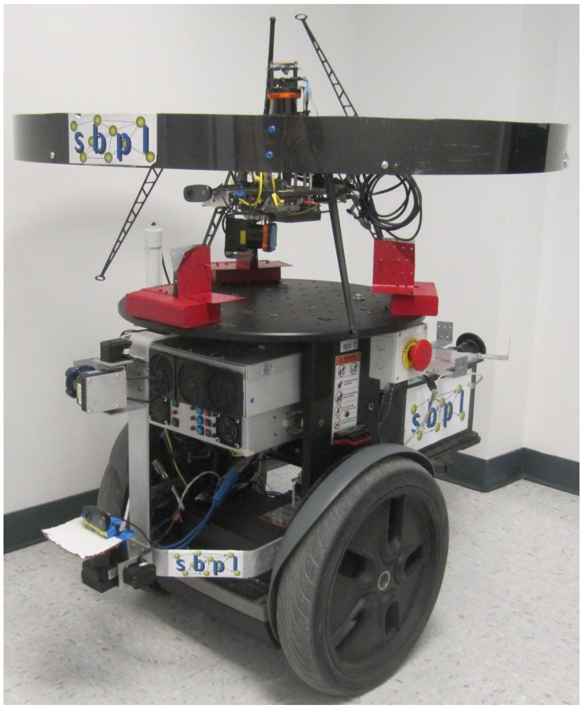


Figure 5.7: UAV mounted on UGV.

the controller computer. This machine is responsible for the direct planning and control of Melvin and is a 3.0 GHz i5 with 8 GB RAM. This machine handles all hardware interfaces including the motion interface to the base. The second computer is a dual quad-core Xeon server with 16 GB of RAM that executes all of the high level planning including the exploration planner and map merger nodes.

Hexacopter

The Hexacopter serves as the aerial component of the exploration team. Like Melvin, it is equipped with two 30 m Hokuyo scanning laser range finders, a Logitech webcam, and its own computer. The computer is an i7-2660 with 16 GB of RAM that runs all of the automation on-board. The body of the Hexacopter is a modified Mikrokopter Hexa XL frame with custom sensor and computing mounts, power distribution electronics, and blade guards (see Fig. 5.9b).

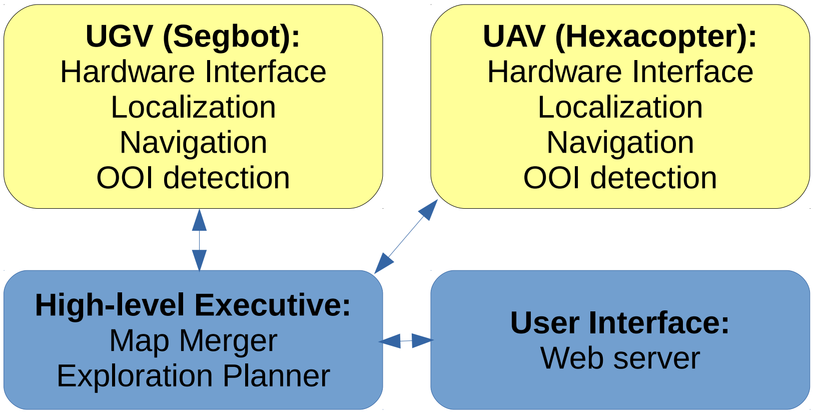


Figure 5.8: Overview of system. The Robots both have sufficient computing capability and sensors to move through the environment based on higher-level goals without further guidance.

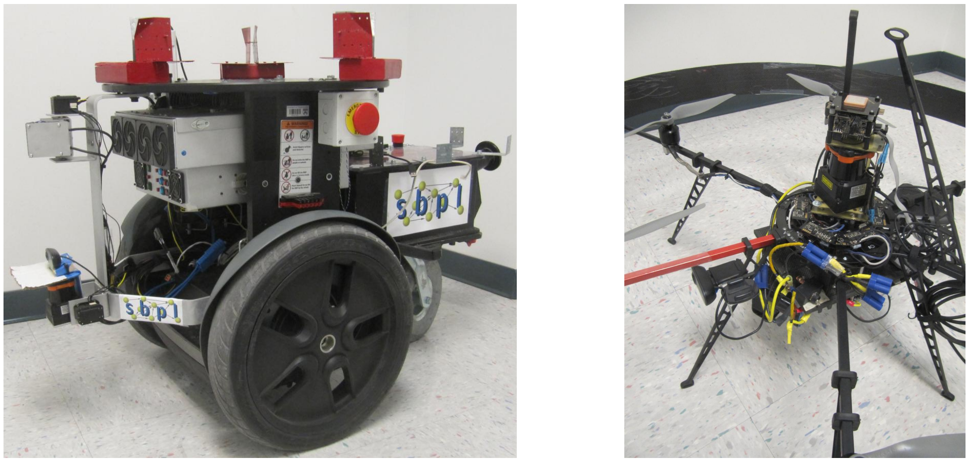
5.2.3 Local Software

General Software & Communication

All of the computers run Kubuntu 12.04 with ROS Groovy. Each robot has its own instantiation of a roscore with an additional roscore for the exploration planner and map merger, and one for the user interface. Both of these additional roscores are physically executed on the UGV server computing system. For the few messages that needed to be passed between systems, we used the ROCON software package to transfer standard ROS messages to multiple roscores. This setup was made to allow for movement of the high-level modules to any available computing system. By having its own roscore, all that was necessary was to update the ROCON links if we ran it on a different physical machine.

Localization

Both robots were fully capable of independent autonomous behavior and only used the executive for coordinating goal locations. To achieve this, each machine ran its own SLAM subsystem based on the Hector SLAM package [Kohlbrecher et al., 2014]. The SLAM system only maintains a 2-D map for determining the x, y, ψ position and heading of the robot. A 2-D SLAM system was used as it provided adequate positional accuracy with significantly lower computational load compared to a full 3-D system. For the UAV, there was an independent system that used the vertically oriented panning lidar to determine the ground plane. Since the UAV was initially mounted on top of the UGV, and during



(a) UGV

(b) UAV

Figure 5.9: Robots used for experiments. a) UGV - Large box on right is main battery pack. Upper gray box is server, lower gray box is controller computer. Hokuyos and camera are mounted on structure on left. b) UAV - closeup showing bottom panning vertical Lidar, upper fixed horizontal Lidar, and forward facing camera.

operation it was allowed to fly over obstacles, the height estimation system could not update the absolute height with every received lidar scan. On initialization, the height estimator would analyze several scans in order to determine its initial height. Then, while flying, it would filter scan points that deviated more than expected from the current estimate in order to maintain an accurate height and allow operation over obstacles. To backup this system, there was an emergency system that would automatically reset the height estimate if the perceived height exceeded a threshold even if the estimate did not. Knowing that we were operating indoors, this system ensured that we did not inadvertently collide with the ceiling.

Navigation

Each robot was responsible for its own navigation to the provided goal positions. We used a 3-D state lattice-based planner [MacAllister et al., 2013] running AD* to generate kinodynamically feasible trajectories for the UAV and a simple 2-D planner for the UGV. Upon receiving a goal, the on-board planner would perform a search on the local obstacle map to generate a trajectory from the current reported position (from the SLAM subsystem) to the goal state. In the event the robot was unable to generate a feasible trajectory (for example, if the goal was too close to an obstacle) the system would time-out and receive an updated goal from the exploration planner. The states used for planning are tuples consisting of a discretized translation in two or three dimensions, and a rotation in one dimension. The UGV planned using planar spatial coordinates and heading, $\langle x, y, \psi \rangle$, with 10 cm cells[3] and 16 discretized headings, while the UAV planned in $\langle x, y, z, \psi \rangle$ with 5 cm cells and 16 discretized headings. The UAV was provided with a user defined "nominal height" that it would preferentially fly at during transits, but would deviate from as necessary. In addition, the exploration planner was provided this same height and would preferentially select goal locations at this altitude. This setup allowed the planner to construct full 3-D trajectories going over, below, or around obstacles while maintaining a preferred height for the UAV to operate at.

Once the robot had a feasible trajectory, it would use the local controller to generate motor inputs to follow the trajectory. For the UGV this was performed using a trajectory roll-out scheme that estimated different motions based on a small finite set of short-time horizon control inputs and selected the input that provided an endpoint closest in position and orientation to the desired trajectory. The UAV controller used a PID control for position to generate its control inputs based on the measured error between its current location and the next way-point along the desired trajectory. Altitude and yaw were handled by separate PID controllers in a similar fashion.

Object Detection

Besides the motion control subsystems, each robot performed its own analysis of the video feed in an effort to detect the OOI. This subsystem was based around an existing vision detection system CMVision [Bruce et al., 2000]. This system was trained to detect a specific colored object; for our experiments it was a green tablet case (Fig. 5.10). Like many other color based object detection systems, recalibration was required for different lighting conditions. Color-only detection was selected in order to keep the processing requirements minimized. Even so, this sub-system required the largest percentage of computing power used. When a robot detects a possible OOI, it will transmit a still image to the user (Fig. 5.11, right image - OOI is bottom center), pause exploration, and hold position until it receives either confirmation or rejection of the reported OOI.

[3] In this chapter, we use the term "cell" to refer to 3-dimensional discretized locations defined by a center-point, $\langle x, y, z \rangle$. We use the term "state" to refer to a 3- or 4-dimensional discretized pose, $\langle x, y, (z), \psi \rangle$.

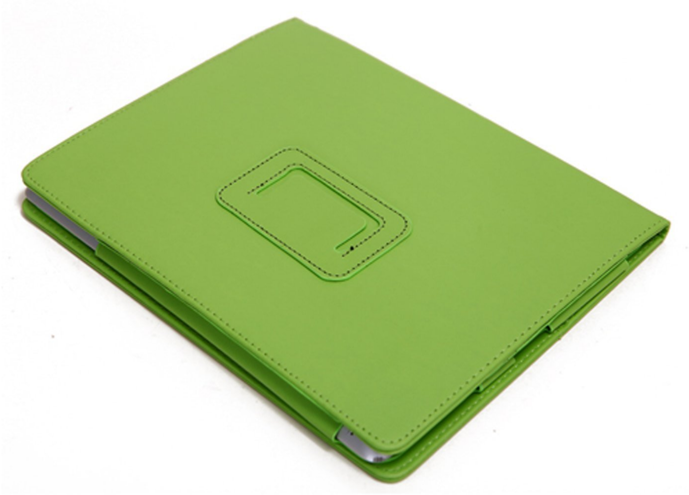


Figure 5.10: The OOI.

5.2.4 High-Level Software

High Level Executive

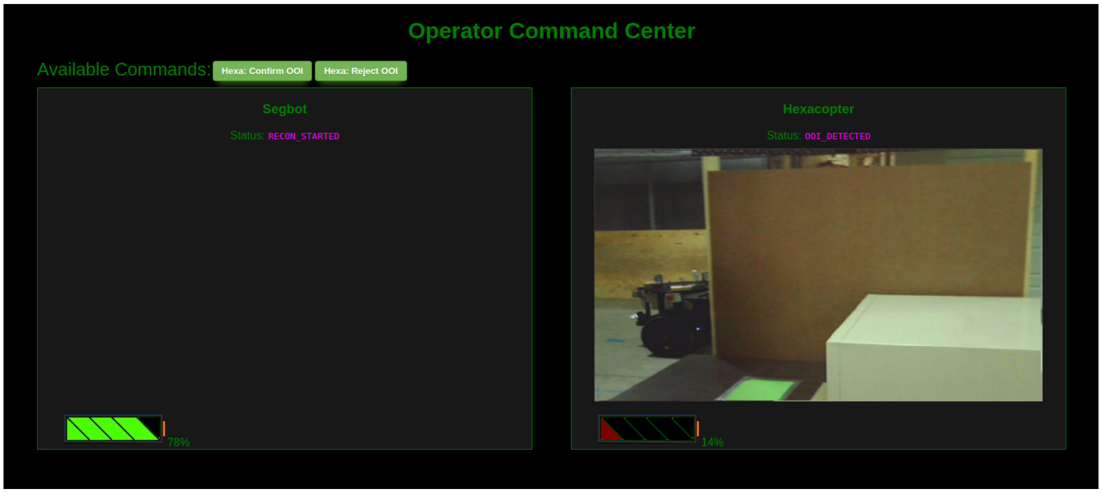
The high-level executive is responsible for coordinating the robots. As part of this function, it provides the interface between the map merger / exploration planner module, the user interface module, and the robots.

User Interface

The human user only has a a few inputs to the system during runtime. The two chief inputs from the user are to start and concur with the completion of exploration. The only other input is concurrence on launching the UAV which is included for safety considerations. All three of these inputs are handled via the User Interface module, Fig. 5.11. This module displays a dynamic web page to the user with zero to four buttons: accept OOI, reject OOI, and commence exploration, for two robots plus enter area for the UGV, of which a maximum of four are ever available at a given time. In addition, when one of the robots has a possible OOI detection, the best image of the OOI is forwarded to the user for confirmation.

If the user confirms the OOI, the executive directs both robots to cease exploration (and presumably return to a home location and land, as appropriate). If the user denies the OOI detection, whether it is due to a false positive or other reason, the executive will direct the detecting robot, who had paused, to resume exploration. This setup reduces the cognitive load of the human operator requiring them only to judge whether the provided image is indeed of the OOI.

The interface can be accessed with any HTML browser on any device with a WiFi connection. During our testing we verified that a tablet, an Android smart-phone, and a laptop were all able to provide the high-level commands and receive the images, satisfactorily.



Status: RECON_STARTED

Status: OOI_DETECTED

Figure 5.11: The user interface. On the right, the UAV has detected an OOI and sent the image to the operator for confirmation (green tablet bottom center). Two buttons are visible along the top in green.

Map Merging

The map merging algorithm seeks to generate a unified 3-dimensional map of the environment by taking into account the offset between the two robots starting position. The global map was tracked in three spatial dimensions, $\langle x, y, z \rangle$, with 10 cm cells and was stored as an occupancy grid [Elfes, 1987] using an Octomap [Hornung et al., 2013]. When the system initially starts, it attempts to align the two existing maps. It starts with a rough estimate of the offset between the two robots, and iterates through a finely discretized set of points around the initial estimate. This process checks for deviations in translation, $\langle x, y, z \rangle$, as well as angular deviations in heading, ψ (it assumes no roll or pitch errors, ϕ, θ). Once the initial transform between the two maps is determined, this value is no longer modified. This does have the potential to cause drift errors over longer runs as the on-board SLAM system accumulates errors, however, in our testing, both robots had insignificant deviations from the global map at the time they found the OOI. Approaches such as [Jessup et al., 2014] or [Schuster et al., 2015] may alleviate this accumulated SLAM error if it becomes substantial.

During runtime, the map merger module uses the fixed starting transform to place new data into the global map. This data is characterized as either obstacle or free space and is represented as an occupancy grid where we store the log likelihood that the cell is either free or occupied with the middle value indicating unknown. In addition we track which cells have been viewed by the camera. To accurately annotate which cells have been seen by the camera, we must first determine which ones are free. The panning scanning laser rangefinder generates information about a wedge shape projecting from the center-line of the robot in its direction of motion. By ray-casting out towards each laser scan echo, we can identify free space between the robot and the nearest obstacle. In conjunction, with the receipt of each image, we ray-cast from the location of the camera in the direction of each camera pixel up until we reach either an unknown or an obstacle cell, or we reach the maximum effective detection range of the OOI subsystem. The set of cells traversed by the camera ray are marked as visually cleared. In this way we can positively track which cells are guaranteed to not contain the OOI in 3-D space (see Fig. 5.12).

We assume that a valid goal configuration contains a state accessible from the starting state and from which the OOI may be sensed.

Exploration

The exploration module is responsible for analyzing the environment and determining where to send each of the robots next. This module is capable of adding or removing robots from the assignment list during each iteration, if necessary, to accommodate dynamic teams or, in our case, the launching and landing of a UAV. This module can assign goal locations at any point in 3-D space to position a robot to view a particular location taking all known obstacles into account. In addition, this module is capable of differentiating which cells are visible from ground versus aerial robots and preferentially assigning those cells to the respective robot types. Technical details of the exploration planner are found in Section 4.2.

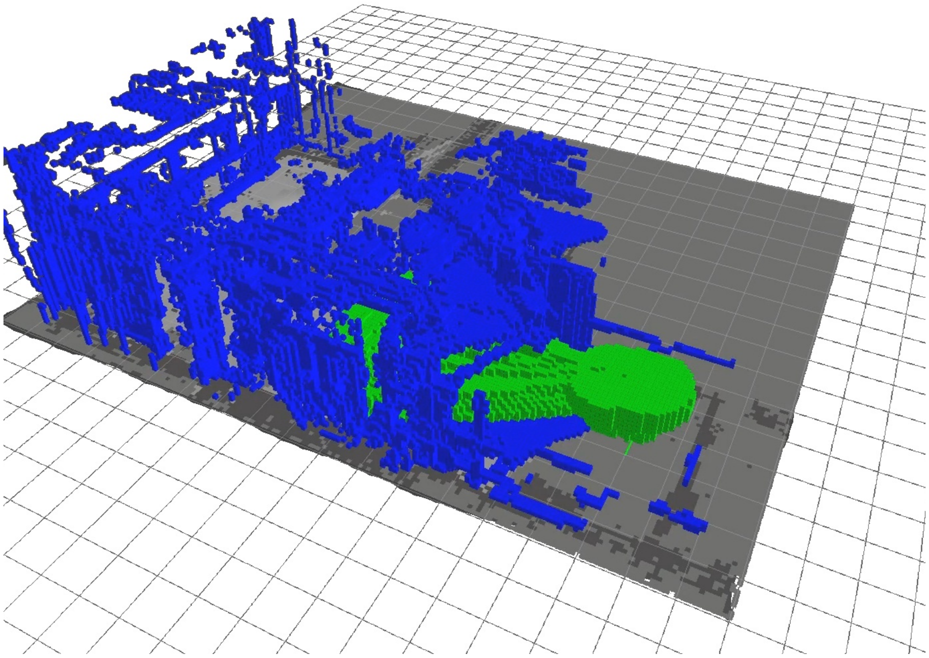


Figure 5.12: Combined map. Blue objects are obstacles, green objects are cells that have been visually cleared. Free and Unknown cells are not shown.

By using the ability to determine the existence of cells not capable of being sensed by the UGV the exploration algorithm could be used to determine when to launch the UAV. This determination could be based on a large number of factors including number of these un-sensable cells, the size or arrangement of these cells, or other parameters.

5.2.5 3-D Exploration Experiments

Setup

Our experiments aimed to validate our entire approach to exploration by having the robot team search a previously unknown area attempting to locate a particular tablet (our OOI) identified by its unique color. The operator stood outside the area and issued the allowed commands with no other interaction with the robots. We defined a successful run if either of the robots were able to identify the tablet before the UAV depleted its battery. Depending on the length of time spent airborne, and to a lesser extent the amount of processing load

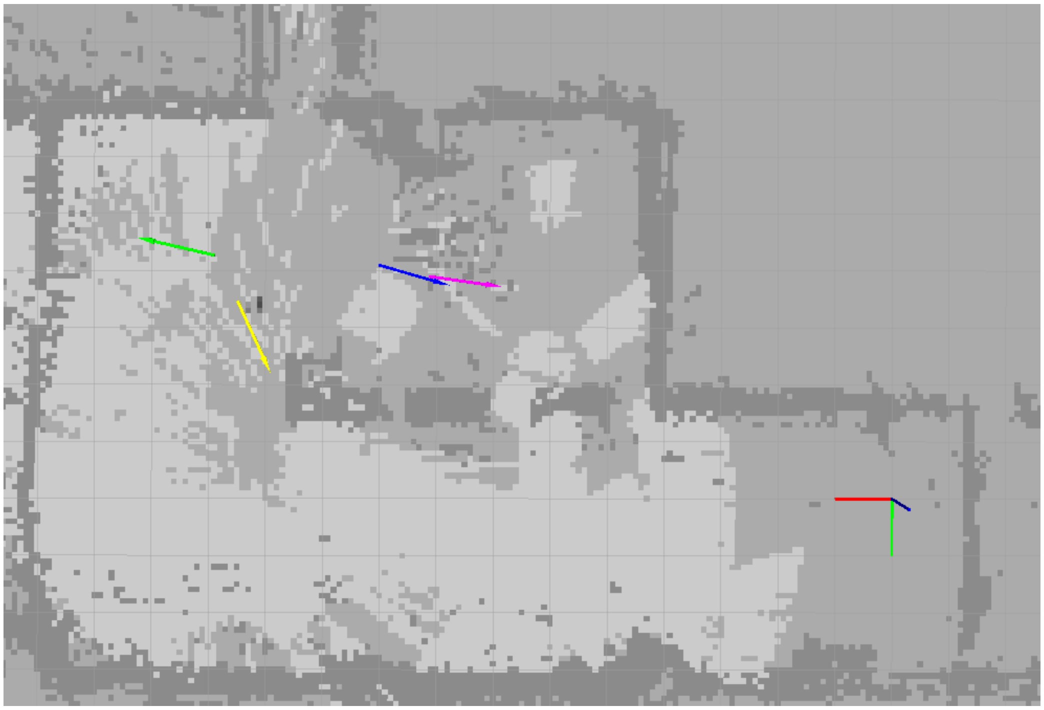


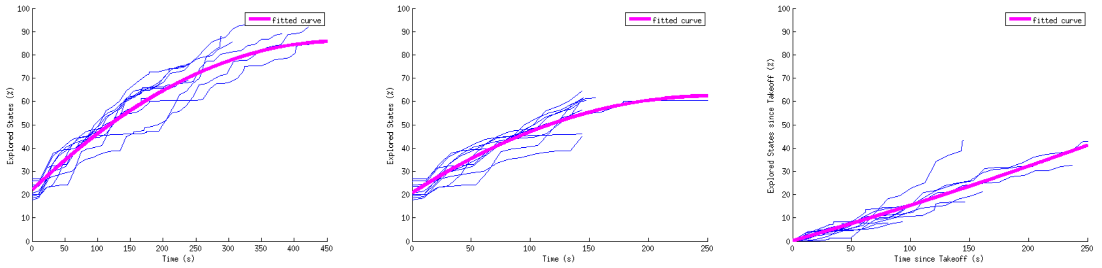
Figure 5.13: Overhead view of exploration planner goals. Dark Blue is current position of the UGV, Magenta is UGV goal, Green is UAV position, and Yellow is UAV goal. The UAV has been tasked to explore the top of the desk (dark object bottom center) while the UGV is exploring a corner that has not been explored yet. In this 2-D projection, the lighter areas indicate less uncertainty in the vertical column at that point.

and time spent operating while mounted to the UGV, the UAV is limited to between four and ten minutes of flight, compared to several hours of exploration time for the UGV. An unsuccessful run was one in which after conducting a search the robots were not able to find the tablet before a low battery forced the UAV to land. The land action was automatic and occurred when battery voltage under load averaged less than 13.8 V for 10 s. Furthermore, runs terminated due to a mechanical failure of a robot were not counted in the results.

We conducted our experiments in an enclosed indoor area measuring approximately 30 m × 10 m × 5 m of which the upper 2 m to 3 m were occupied with pipes and conduit. The area was partitioned into sections with movable walls and objects such as a desk and filing cabinet were placed inside the area. This resulted in approximately 400,000 cells capable of being detected and analyzed, depending on obstacle placement and room configuration. The robots started from the same location for all tests near the edge of the search area and had a predetermined first goal located 5 m into the exploration zone that the UGV moved to when commanded to "enter the area".

We had different people place the OOI during our experiments to rule out any bias in selecting locations that were particularly easy or hard for the system. The direction provided to the person placing the OOI was to place it so that the OOI would only be visible to the UAV once it was airborne. This was done by placing the OOI in a location not visible from the UAV prior to it taking off from the UGV and not visible to the UGV. Separate tests were performed to verify the UGV was capable of also detecting the OOI, and it was successful on all runs.

The test environment had one object that had an identical color as the OOI and served as a decoy for the detection system by causing a false positive. In the event the decoy was detected, the operator rejected the classification and resumed the exploration. Decoy detection was not deemed to be a failure but the time spent pausing and waiting for the operator to reject the OOI was counted towards the completion time.



(a) Full Run

(b) Prior to Takeoff

(c) After Takeoff

Figure 5.14: Percentage of cells explored over time (based on full volume of search area). Heavy magenta line indicates best fit.

Since the exploration algorithm has several user selectable parameters, we set them as follows: To achieve balance between distance traveled and information gain, we set $\Xi = 0.5$, to facilitate the robots spreading out, we set $\text{threshold}_D = 5$, and to discourage very short range goals, we set $\text{threshold}_L = 1.2$ for all runs. In addition, while the map merger process can determine when to launch the UAV based on detecting unreachable frontier states, for all of the test runs, we had the UAV launch at the first opportunity after 2 minutes 30 seconds of UGV only exploration.[4] The 2:30 takeoff time was based on prior experiments that

[4] There was one area of the environment that the overhead obstacles were too low to allow for the UAV to

indicated a reasonable percentage of the UGV accessible space was explored by that point on average (for an example, see Fig. 5.14b for the case where the UAV took off late - the UGV exploration progress plateaus at approximately 2:30).

The cameras used on both robots were identical Logitech C310 webcams that provided images at 1280 × 720 pixels at 25 Hz. The UGV camera was mounted approximately 20 cm from the ground along the vehicle center-line with the camera axis parallel to the ground. The UAV camera was mounted on the UAV forward axis and tilted downward by 10° from horizontal. With the UAV nominal flying height set to 1.65 m, the camera typically was at 1.6 m.

3-D Exploration Results

Table 5.1: Experimental Results for Multi-Robot Exploration.

Run	Detect OOI	Detect Decoy	% of Env. Explored	Time to Detect OOI	Time Since UAV Launch
1	✓	✘	55.6	7:08	2:23
2	✓	✓	63.3	5:51	3:21
3	✓	✓	57.9	4:49	2:19
4	✓	✘	50.5	4:01	1:31
5	✓	✘	45.6	3:48	1:08
6	✓	✘	49.3	3:50	1:21
7	✓	✘	45.3	3:56	1:26
8	✘	✘	60.5	-	-
9	✓	✘	58.5	6:20	3:50
10	✓	✘	56.2	5:02	2:32

We made 10 complete runs with the results as shown in Table 5.1. Only 1 run failed to find the OOI within the time allowed due to the UGV exploring the particular corner containing the OOI (but at too low an altitude) prior to the UAV launch. Since the area was already predominately explored, the UAV did not return to verify the small remaining unknown area during the available time. Two runs detected the decoy initially, but after the operator rejected the false positive, they both successfully found the actual OOI.

launch which caused the takeoff to be delayed on two occasions - once by 10 seconds (run 5) and once by 2:25 (run 1).

Using 400,000 as an approximation of the number of detectable cells[5] in the environment we can estimate the percentage of the environment covered prior to discovery of the OOI, Fig. 5.14. For all of the runs we cover 80-90% of the environment prior to detection of which 40-60% is discovered by the UGV prior to the UAV taking off. As can be seen in Fig. 5.14b, the UGV progress has already begun to plateau by this point. The UAV continues to make progress as it explores areas inaccessible to the UGV accounting for an additional 10-30% before detecting the OOI or landing, Fig. 5.14c. The values in Fig. 5.14c are a combination of both UAV and UGV information, however, since we know that there are significant portions of the map that are inaccessible to the UGV, and the pre-launch data is plateauing, we surmise that the majority of the gain post-launch is from the UAV.

Since the UAV preferentially visits locations that the UGV is incapable of exploring, it detected the OOI within the first few goals (1:30 of flight) almost half of the time.

Demonstration of Algorithm

The 3-D exploration algorithm was demonstrated to the US Marine Corps Warfare Development Laboratory and reporters from the CBS television program “60 Minutes” [Martin et al., 2017]. For this demonstration, the environment consisted of eight 3 m × 3 m × 3 m cubes with openings in the lower or upper portion of half of them (Fig. 5.15) in an area of approximately 500 m^2. The goal for the robots was to search around and inside the buildings, and around trees attempting to locate a specific person.[6]

5.2.6 3-D Exploration Application Conclusions

Our contribution in this section is two-fold: we have introduced an air-ground robotic system capable of operating autonomously and an algorithm for performing exploration in cluttered 3-D environments. With our experiments, we have shown how planning for heterogeneous teams of robots can be an effective method of exploring environments that either robot alone would be unable to explore. Our results show that the system is capable of entering, exploring, and detecting an OOI within the time constraints imposed by the limited battery life of a UAV.

[5] The remainder being internal to obstacles.

[6] Facial recognition system was run on our robots but was developed by a separate group for this project.

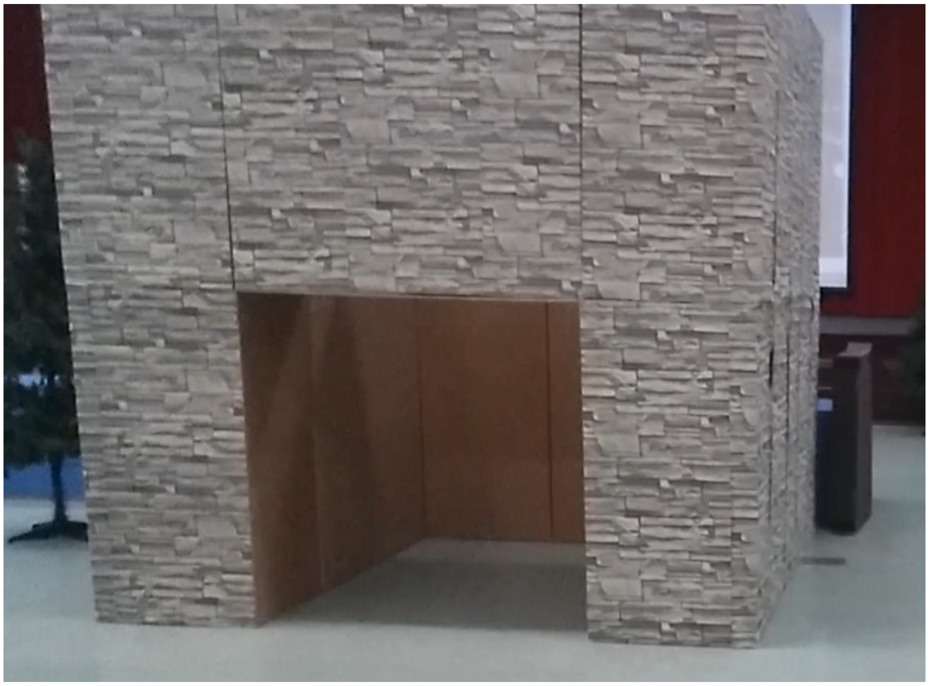


Figure 5.15: Example obstacle. The opening occurred either in the lower half, or the upper half of the building. A person was able to either sit or stand inside the building for the UGV and/or UAV to find.

The exploration algorithms introduced in this chapter and the preceeding one allow the exploration of large environments by teams of robots, allow for the user to provide high-level input to guide the exploration process, and can operate in full 3-dimensional environments.

One area that requires additional research is the relationship between discretization size on the individual robots and the central map. Due to stability issues, our UAV required a map discretized into 5 cm cells, while the UGV and central map both used 10 cm cells. We saw no ill effects from this slight difference in map resolution, but it is conceivable that different discretizations could result in paths that were not deemed feasible by the robot or assignments that were not given due to inaccurate path length assumptions by the planner. In particular this may become a significant issue if the environment has numerous narrow passages that are rendered untraversable at certain discretization levels.

Chapter 6

State Lattice Planning with Controller-based Motion Primitives (SLC)

Sail on till you reach the promised land,
We all drown in the fifth dimension
The ninth wave, I can feel it's coming, The ninth wave
BLIND GUARDIAN *The Ninth Wave*

In this chapter we present our algorithm: State Lattice Planning with Controller-based Motion Primitives (SLC). The SLC planner allows for robust navigation using a wide variety of sensors including in areas with no or limited high-quality localization information and is based on standard graph search algorithms such as A* [Hart et al., 1968] and ARA* [Likhachev et al., 2003]. It also allows the execution of controllers similar to the sequential composition of controller approaches [Burridge et al., 1999][Conner et al., 2011][Kallem et al., 2011] combined with the functionality of switching between controllers based on external perceptual triggers similar to the Linear Temporal Logics [Kress Gazit et al., 2007].

The SLC planning framework is a modification of a search-based planner implementing a state lattice graph. State lattices are typically constructed of simple motion primitives connecting one state to another. There are situations where these metric motions may not be available, such as in GPS-denied areas. In many of these cases, however, the robot may have some additional sensing capability that is not being fully utilized by the planner. For example, if the robot has a camera it may be able to use simple visual servoing techniques to navigate through a GPS-denied region. Likewise, a LIDAR may allow the robot to skirt along an environmental feature even if there is not enough information to generate an accurate pose estimate.

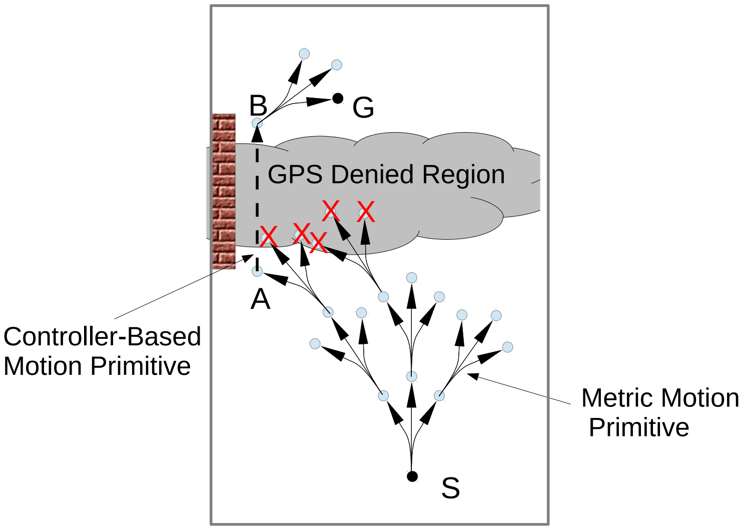


Figure 6.1: Example of controller based motion primitive. From start state, s, we construct the motion primitive lattice. Metric motions that result in states inside the denied region are considered invalid, as the robot would become hopelessly lost if it entered this area. However, state A is a valid state and is sufficiently close to the wall on the left to allow a controller-based motion primitive (using a wall-following motion) to generate state B on the far side of the denied region. From B the standard metric motion primitives can generate a path to the goal, G.

A state lattice-based planner uses a regular lattice constructed from motion primitives to form the search graph, $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ as shown in the lower half of Fig. 6.1. In a typical planner, the edges, $\mathcal{E}$, are formed by applying fixed motion primitives at each state, $x \in \mathcal{X}$. These metric motion primitives carry the implicit assumption that the robot has sufficient localization ability to be able to execute the motion and to determine the stopping point. In some environments, however, there may exist regions where the robot is unable to execute the typical motion primitives or they provide poor performance. For example, Fig. 6.1 depicts an environment with a large GPS-denied region between the start and the goal. If the robot is relying purely on GPS data in order to navigate, then there is not a valid path to the goal state as it would become hopelessly lost when attempting to transit the GPS-denied region. On the other hand, if, in addition to our normal suite of sensors, we had the ability to follow a wall the robot could then use that ability to successfully cross the GPS-denied region. Similarly, as shown in Fig. 6.2, while motion primitives are well suited for use in a parking lot, on the roads it is more common and practical to use lane-following controllers. By adding additional directed edges to the search graph based on forward simulating (either online or offline) different types of controllers, the planner is capable of finding trajectories through areas that were impassable using only metric-based motion primitives. These controller-based motions rely solely on the capabilities of the controller independent of the robots ability to localize. From the first example, the wall following controller does not know where in the environment it is with any degree of precision at any point during its trajectory along the wall, however, by executing this controller to its natural stopping point (i.e., the end of the wall) the robot ends up in a known (and repeatable) position.

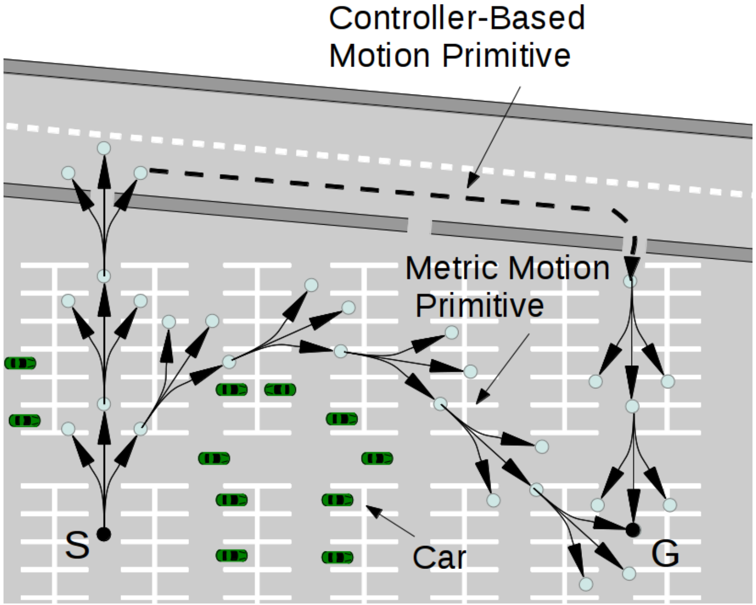


Figure 6.2: The state lattice is sufficient to navigate in the parking lot area, however, once on the roadway, the lane-following controller is used.

We present SLC operating on a single robot, to demonstrate a few basic controllers as well as validate the theoretical properties of the framework. The SLC framework is expanded in Chapter 7 to include a collaborative localization ability for an air-ground robot team, allowing them to traverse portions of an environment they could not have individually.

6.1 Navigation Planning Background

There are a wide variety of approaches to navigation; two common methods are the use of a planner to determine a trajectory through the environment coupled with a controller to subsequently guide the robot along the determined path, and controller only based approaches that react to the environmental stimulus as they move. The controller-based approach to navigation can directly encode sensing capabilities, such as the wall following assumption found in the simplistic, but effective, Bug2 and its related derivatives [Lumelsky and Stepanov, 1986], and the "go to goal" sensing ability inherent in potential field approaches [Khatib, 1985] and navigation functions [Connolly et al., 1990]. Navigation functions generate a gradient that directs the robot away from obstacles and towards the goal. By carefully constructing this gradient, local minima can be eliminated, ensuring convergence to a goal state. These however, are difficult to construct for arbitrary environments being traversed by arbitrary-shaped robots [Masoud, 2009; Arkin, 1989] and typically do not perform any explicit path cost minimization. For speed considerations these gradient approaches can be implemented as hierarchical planners [Scherer et al., 2007] with both a local and global planning component. However, since these methods do not evaluate the orientation of the robot while generating the gradient field for the global planner they are restricted to using the circumscribed circle of the vehicle as their footprint. This limitation means that environments requiring traversal of narrow passages cannot be completed. In addition, having two separate planners of different dimensionality can lead to inconsistencies between the two resulting in sub-optimal trajectories or failing to find a solution when the two disagree [Zhang et al., 2012].

One alternative to the navigation functions and potential field approaches is the sequential composition of controllers. By covering the valid states with a series of controllers that each moves the robot along to the next controller, this approach can alleviate the need to find a single globally attractive control law. Specific controllers are activated by a planner in order to approximate an arbitrary navigation function [Burridge et al., 1999; Conner et al., 2011; Kallem et al., 2011]. In one case, standard motion primitives were implemented with controllers and used with $A^\star$ to generate trajectories [Nagarajan, 2012]. While these functions are easier to construct than a single navigation function covering the entire domain, there is still some computational overhead. More importantly, they cannot react to arbitrary perceptual triggers. In other words, they typically cannot come up with plans that say "follow the wall, until a doorway is detected on the left". By using the controller-based motion primitives in combination with perceptual triggers, we are able to construct these types of strategies. It should be noted that the controllers developed in these other approaches can be used within the SLC framework to provide additional controller-based motion primitives.

Hybrid approaches seek to overcome the limitations that pure behavior-based systems exhibit. Sensor-based planning schemes using temporal logics to create an automaton or finite-state machine are capable of integrating sensor functionality into the planning scheme [Kress Gazit et al., 2007]. These planners also allow the robot to change its high-level behavior based on sensor inputs. However, these systems do not use any cost-minimization techniques to choose between controllers during the planning cycle. Instead they generate fixed controllers and select between them at runtime, typically shifting controllers when a higher level process detects a lack of progress towards the goal [Arkin and Balch, 1997], and so do not incorporate the controllers into the planning process. Our SLC-based planner can use the same library model and shares many of the same elements, but since we maintain an $A^\star$ based search, we can likewise retain guarantees on performance.

Partially observable Markov decision processes (POMDP) can be used for navigation planning. However, these approaches are significantly more computationally expensive than other approaches and do not scale well to large environments [Kurniawati et al., 2008]. One improvement that is similar to our SLC work is the use of macro-actions in POMDPs [He et al., 2010] or temporal abstraction in MDPs [Sutton et al., 1999]. Macro-actions, like our controller-based motion primitives, seek to connect more distant states to the start state but fundamentally are a concatenation of shorter existing motion primitives. On the other hand, the controller-based motion primitives constitute wholly new motions directly dependent on sensor feedback and basic controllers.

The second common method to navigation is the use of planners to determine the trajectory through the environment then use a controller to attempt to follow it. A variety of different planners can be used in this application from sampling-based planners such as Probabilistic Road-maps (PRM) [Kavraki et al., 1996] and Rapidly-exploring Random Trees (RRT) [Lavalle, 1998; Kuffner Jr. and LaValle, 2000; Şucan and Kavraki, 2012; Karaman and Frazzoli, 2010] to heuristic search-based approaches such as $A^\star$ [Hart et al., 1968], $ARA^\star$ [Likhachev et al., 2003], and Field $D^\star$ [Ferguson and Stentz, 2005]. Of this latter class, one approach is the use of lattice-based graphs. They are commonly used for robotic path planning for aerial vehicles [Thakur et al., 2013], automobiles [Likhachev and Ferguson, 2009], boats [Svec et al., 2013], all-terrain vehicles [Pivtoraiko and Kelly, 2005], etc.. These state lattices are constructed by applying a set of motion primitives to each state expanded during the search in order to generate valid successor states. By doing this, they generate edges in the search graph between the (possibly non-adjacent) discretized states which serve as the graph nodes. Any of the graph search algorithms, such as A*, can act on this lattice to generate a trajectory consisting of a sequence of motion primitives between the start and goal state. The advantage of using a motion primitive vice simple 4- or 8-connected grids is that the motion primitives can more accurately model the kinematic constraints of the robot. For example, a car-like robot is unable to move directly sideways, but it can curve to the left and right as well as travel forward and in reverse. In this case a small set of motion primitives encoding those maneuvers is typically sufficient for planning motions in a plane.

As an alternative to pre-defined motion primitives, we use available controllers to generate navigation trajectories. A typical robot is usually equipped with a suite of controllers that utilize on-board sensors, for example, common controllers include visual servoing towards a landmark using a camera [Agin, 1979], direct-to-goal navigation using a GPS sensor [O'Connor et al., 1995], and range maintaining actions using a radar [Choi and Hedrick, 1995]. Another example of planning integrated with control is the corner-localization scheme proposed by [Lewis and O'Kane, 2013]. This approach uses only a minimal set of poor accuracy sensors but can still provably reach a desired goal location under certain conditions. Controllers such as these operate well in real-world conditions due in part to their tight coupling of sensor information and actuation, specifically due to their ability to utilize the strengths of a given sensor system. For example, visual servoing works well in part due to the relatively large field-of-view and high angular accuracy of cameras. By using the SLC planner we can incorporate these controllers and thus add additional motion primitives into our planner augmenting the state lattice.

One approach that does use planning to choose between available controllers is [Mellinger and Kumar, 2010]. Their planner chooses between two controllers with two speeds available for each along the individual discretized path segments. By selecting the controller and speed best suited to the terrain, the planner can trade execution speed for safety. However, they require accurate localization information in order to execute the desired trajectory, relying on a VICON or LIDAR-based system to determine track error.

For our SLC planner, we use the extended definition of state lattice supporting all available controllers and search the resulting graph using the ARA* heuristic graph search algorithm. This allows us to provide guarantees on completeness and (sub-)optimality while constructing trajectories not available using other planners. In addition, we are able to use SLC to incorporate collaborative localization directly into our planner (the multi-robot expansion to SLC is discussed in more detail in Chapter 7).

The SLC planner has been utilized in other non-navigation domains as well. In [Kim and Likhachev, 2017], SLC is used to perform parts assembly with robot arms; using motion primitives derived from physics-based simulations to develop trajectories in the presence of uncertainty.

6.2 State Lattice Planning with Controller-based Motion Primitives

6.2.1 Overview

A state lattice planner uses predefined *motion primitives* to generate, as required, a graph, $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ spanning the environment. Motion primitives are short kinematically feasible motions that are designed to connect one state with another nearby state and form the set of edges, $\mathcal{E}$, of the search graph. This graph is traversed by a graph search algorithm such as A* or ARA* in order to find the minimal cost path from the start state x_{init} to the goal state x_{goal}, where $x_{init}, x_{goal} \in \mathcal{X}$. During the planning cycle, the graph generation and planning process are interleaved so that only those elements of the graph that are required for the search algorithm are explicitly constructed. The SLC planner [Butzke et al., 2014] modifies this construct by adding additional directed edges to $\mathcal{G}$ that correspond to executing a controller c from a given set of controllers C, each terminated by a perceptual trigger, τ, from a set of triggers $\mathcal{T}$, at a given start state. These new edges are formed by forward simulating the desired controller and trigger from a given state, x_i, in order to determine the end state, x_j, and thus forming a new edge, $e(x_i, x_j)$, which is added to the set of edges in the search graph, $\mathcal{E}' = \mathcal{E} \bigcup^n e_n$, then $\mathcal{G} = (\mathcal{X}, \mathcal{E}')$.

6.2.2 Definitions

Definition 6.1 (Configuration Space ($\mathcal{X}$)) *An n-dimensional space $\mathcal{X} \subset \mathbb{R}^n$ such that a robot is fully defined by a single point in $\mathcal{X}$. If the robot and environment are in a feasible,* *collision free pose, x, then $x \in \mathcal{X}_{free}$, otherwise $x \in \mathcal{X}_{obs}$. $\mathcal{X}_{free} \cup \mathcal{X}_{obs} = \mathcal{X}$ and $\mathcal{X}_{free} \cap \mathcal{X}_{obs} = \emptyset$ [Lozano-Perez, 1983].*

Assumption 6.1 *The initial state, $x_{init} \in \mathcal{X}_{free}$, of the robot is known without error.*

Definition 6.2 (Action (a)) *Comprised of a controller and trigger pair, $a = \langle \mathrm{c}, \tau \rangle$, $\mathrm{c} \in \mathcal{C}, \tau \in \mathcal{T}$. Executing an action, a_t, at a state x_t results in the robot moving to one (possibly identical [1]) of a set of states, $\Phi(x_t, a_t) :\sim \mathscr{X}_{t+1} \in \mathcal{P}(\mathcal{X})$.*

$$\Phi(x_t, a_t) : \mathcal{X} \times \mathcal{C} \times \mathcal{T} \rightarrow \mathcal{P}(\mathcal{X})$$

The Φ function can also provide the cost of executing the action.

$$\Phi(x_t, a_t) : \mathcal{X} \times \mathcal{C} \times \mathcal{T} \rightarrow \mathcal{P}(\mathcal{X}) \times \mathbb{R}$$

We assume the initial position of the robot is known without error, so uncertainty in position only grows from errors in motion and sensing. Therefore, given a start state x, an action $\langle \mathrm{c}, \tau \rangle$ produces a distribution of endstates, $\mathscr{X}$, rather than a single state.

When the action function is provided a set vice a single start state, the output is the union of the goal sets from the individual start states.

$$\Phi(\mathscr{X}, a) = \bigcup_i \Phi(x_i, a) \quad \forall x_i \in \mathscr{X}$$

Assumption 6.2 *Actions are deterministic based on their start state.*

Definition 6.3 (Domain ($\mathscr{D}_x^{\mathbf{r}}(\mathrm{c}, \tau)$)) *The set of points, $\mathscr{X}$, such that executing a given controller/trigger pair, $\langle \mathrm{c}, \tau \rangle$, results in all of the goal sets of $\Phi(x_i, \mathrm{c}, \tau), \forall x_i \in \mathscr{X}$ being located within some ball, $\mathscr{B}_x^{\mathbf{r}}$ of radius $\mathbf{r} \in \mathbb{R}^n$ (where n is the dimensionality of $\mathcal{X}$) and center $\mathscr{B}_x^{center}$.*

$$\mathscr{D}_x^{\mathbf{r}}(\mathrm{c}, \tau) : \mathcal{C} \times \mathcal{T} \times \mathcal{X} \times \mathbb{R}^n \rightarrow \mathcal{P}(\mathcal{X})$$

$$\mathscr{D}_x^{\mathbf{r}}(\mathrm{c}, \tau) = \forall x_i \in \mathcal{X}_{free} \;\mid\; \forall x_j \in \Phi(x_i, \mathrm{c}, \tau),\; \|x_j - \mathscr{B}_x^{center}\| \leq \mathbf{r}$$

$$\Phi(x_i, \mathrm{c}, \tau) \cup \Phi(x_j, \mathrm{c}, \tau) \cup \cdots \cup \Phi(x_n, \mathrm{c}, \tau) \subseteq \mathscr{B}_x^{\mathbf{r}} \iff x_i \cup x_j \cup \cdots \cup x_n \subseteq \mathscr{D}_x^{\mathbf{r}}(\mathrm{c}, \tau)$$

[1] For example, if the selected controller is of the type “Go to X”, and “X” is not currently in view of the robot, the robot will remain in the same state it was in initially. During planning, the same thing occurs - when the planner evaluates an action that has no meaning at the current state, the function returns the initial state.

Definition 6.4 (Path (ξ)) *A continuous set of points in $\mathcal{X}$ between configurations x_{init} and $\mathscr{X}_{goal}$, $x_{init} \in \mathcal{X}_{free}$, $\mathscr{X}_{goal} \subseteq \mathcal{X}_{free}$.*

$$\xi(x_{init}, \mathscr{X}_{goal}) : [0, T] \rightarrow \mathcal{P}(\mathcal{X})$$

such that $\xi_0 = x_{init}$ and $\xi_T \subseteq \mathscr{X}_{goal}$. A path is collision free if and only if $\xi_t \notin \mathcal{X}_{obs}\ \forall t, 0 \leq t \leq T$ (or conversely $\xi_t \in \mathcal{X}_{free}\ \forall t, 0 \leq t \leq T$). A path can also be specified by the sequence of actions taken to generate the path starting at x_{init} for a path length ℓ.

$$\xi(x_{init}, \mathscr{X}_{goal}) : (C \times \mathcal{T})^{\ell} \rightarrow \mathcal{P}(\mathcal{X})$$

From this $\xi(x_{init}, \mathscr{X}_{goal}) = \{\langle \mathrm{c}_0, \tau_0 \rangle, \langle \mathrm{c}_1, \tau_1 \rangle, \ldots, \langle \mathrm{c}_\ell, \tau_\ell \rangle\}$.

Definition 6.5 (Cost ($c(x, a)$ or $c(\mathscr{X}, a)$)) *A function that maps a state/action pair onto a real number.*

$$c(\mathscr{X}, a) : \mathcal{P}(\mathcal{X}) \times C \times \mathcal{T} \rightarrow \mathbb{R}$$

Typically, a function of the distance moved by the action, energy consumed, time elapsed, or some similar metric. For a set of states, the cost can be the average, maximum, minimum, weighted by start state probability, etc..

Definition 6.6 (Policy (π)) *A mapping specifying which action should be executed for a given state.*

$$\pi : \mathcal{X} \rightarrow C \times \mathcal{T}$$

By following policy π, a path from the initial state is generated. The cost of a policy is the accumulated cost of all of the actions taken between the initial state and the goal state.

$$c(\pi) = \sum_{i=init}^{goal} c(x_i, a_i)$$

Definition 6.7 (Optimal Policy (π^*)) *A policy, π^*, such that there are no other lower cost policies $\forall i, \pi_i \in \Pi$.*

$$c(\pi^*) = \min_{\Pi}\ c(\pi_i)$$

$$\min_{C,\mathcal{T}} \sum_{i=0}^{T} c(x_i, \mathrm{c}_i, \tau_i) \quad \text{s.t. } x(0) = x_{init},\ \Phi(x(T), \mathrm{c}(T), \tau(T)) \subseteq \mathscr{X}_{goal}$$

6.2.3 Controllers and Triggers

In order to incorporate controllers and triggers into the planning process, we introduce two new functions. The first function,

$$\mathrm{C}(\mathscr{X}) : \mathcal{P}(\mathcal{X}) \rightarrow \mathcal{P}(\mathcal{C})$$

provides us with a set of available controllers, $\mathscr{C}$, from the powerset of all controllers, $\mathcal{P}(\mathcal{C})$, available at a state $x \in \mathcal{X}$ where $\mathcal{C}$ is the set of all controllers available to the robot. In other words, for any given state x, $\mathrm{C}(x)$ returns all of the controllers which can be executed at that state. In the absence of motion and sensing errors, each controller would result in a single end state given a start state. However, when accounting for the inherent errors in execution, the result is a distribution of states as discussed in "Action" defined in Section 6.2.2. The controller function will only return a controller c as a valid controller if it is valid for all states within the input set, $\mathscr{X}_{in}$. In other words,

$$\mathrm{c}_j \in \mathrm{C}(\mathscr{X}) \Leftrightarrow \mathrm{c}_j \in \mathrm{C}(x) \quad \forall x \in \mathscr{X}$$

The successors relationship only holds if the goal set of the prior controller/trigger pair is a subset of the domain of the following controller/trigger pair.

$$\langle \mathrm{c}_i, \tau_i \rangle_x \prec \langle \mathrm{c}_j, \tau_j \rangle_y \Longleftrightarrow \Phi(x, \mathrm{c}_i, \tau_i) \subseteq \mathscr{D}^{\mathrm{r}}_y(\mathrm{c}_j, \tau_j)$$

Similar to the available controller function, the second function,

$$\mathrm{T}(c) : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{T})$$

provides a set of available triggers, $\mathscr{T}$, based on the given controller, $c \in \mathcal{C}$. $\mathcal{T}$ is the set of all triggers available to the robot, such as the ability to detect doors, intersections, etc.. Controller-based motions need to have a stopping condition as they are not necessarily fixed length or duration. To this end we introduce the notion of a perceptual trigger. A controller-trigger pair $\langle c, \tau \rangle$ represents the execution of a controller $c \in \mathcal{C}$ until either a perceptual trigger $\tau \in \mathcal{T}$ or an intrinsic trigger (explained below) is detected. Different controllers may have different triggers. For example, a controller for FOLLOWLEFTWALL may have a trigger OpeningOnLeft, whereas that trigger may not be valid for a controller performing VisualServoToLandmark.[2]

We classify triggers into two categories: *intrinsic* or *extrinsic*. Intrinsic triggers result from the natural completion of a controller. Intrinsic triggers are not selectable - they are always in effect. When following a wall, the robot must stop when the wall stops - there is no option to continue following a now non-existent wall. To account for these types of intrinsic triggers algorithmically, we do not individually include them in $\mathcal{T}$ and thus they are never returned as an element from the function T(c). Instead, a single universal trigger, Completion, is used to signify execution of a controller until its natural conclusion.[3] Compared to intrinsic triggers, extrinsic triggers are not directly related to the current controller, but instead result from some independent perceptual signal. When following a wall for example, an extrinsic trigger could be sighting a door on the left or an opening on the right.

6.2.4 Formal Problem Description

Shortest Path Problem

Given a robot $\mathcal{R}$, operating in a configuration space $\mathcal{X}$, equipped with a set of controllers C, and triggers $\mathcal{T}$, we want to determine the optimal policy from x_{init} to $\mathscr{X}_{goal}$ ($x_{init} \in \mathcal{X}_{free}$, $\mathscr{X} \subset \mathcal{X}_{free}$), if one exists or indicate no valid path if one does not (i.e., the *completeness* property). We assume the initial position of the robot is known without error, so uncertainty in position only grows from errors in motion and sensing.

Algorithm 4 outlines the general approach to solving this problem. Starting from the initial state (line 1), all possible controllers and triggers are evaluated (lines 8-12) and added to the queue (line 13). The queue is sorted such that the lowest cost element is always the one removed for evaluation (line 3, line 7, and line 16). When a subset of the goal set is retrieved from the queue (line 4) the full path is reconstructed (line 20). If no path is found before all possible points have been evaluated than NULL is returned (line 2 and line 22).

[2] We use relatively simple controllers in order to more clearly demonstrate the method of constructing the search graph. As such, the controllers and triggers listed are only a small subset of controllers and triggers that one can construct. For example, a wide collection of behavior based controllers are described in [Arkin, 1998].

[3] Some controllers have more than one intrinsic trigger.

Algorithm 4 path = FindPath(x_{init}, $\mathscr{X}_{goal}$)

```
vector(set, real) ENDPOINTS = $\langle x_{init}, 0 \rangle$
while ENDPOINTS $\neq \emptyset$ do
    $\langle$CurrentSet, $cost\rangle$ = ENDPOINTS.pop          //get least cost element
    if CurrentSet $\subseteq \mathscr{X}_{goal}$ then
        break
    else
        ENDPOINTS = ENDPOINTS $\setminus$ $\{\langle$CurrentSet, $cost\rangle\}$
        $\mathscr{C}$ = C(CurrentSet)
        for all $c$ in $\mathscr{C}$ do
            $\mathscr{T}$ = T($c$)
            for all $\tau$ in $\mathscr{T}$ do
                $\langle \mathscr{X}, cost_x \rangle = \Phi($CurrentSet, $c$, $\tau)$
                ENDPOINTS = ENDPOINTS $\cup$ $\{\langle \mathscr{X}, cost_x + cost \rangle\}$
            end for
        end for
        Sort(ENDPOINTS)                                  //sort on increasing cost
    end if
end while
if CurrentSet $\subseteq \mathscr{X}_{goal}$ then
    return ReconstructPath($x_{init}$, CurrentSet)
else
    return NULL
end if
```

Relaxation of the Problem

In order to generate feasible solutions in a reasonably short time for a real-world environment we make a few relaxations of the base problem.

Our first relaxation is to loosen the requirement for completeness. In the general problem, we want to find a path if *any* such path exists. To keep planning times tractable, we instead opt for *resolution completeness*. Resolution completeness is the property of finding a path on a discretized version of the environment if such a path exists rather than on the underlying continuous space. By controlling the discretization size, we can trade improved planning times for increased environmental fidelity.

This relaxation changes our definitions from integrals over continuous-valued fields to discrete summations. A path, $\xi(x_{init}, x_{goal}) : [0, T] \rightarrow \mathcal{X}$ is no longer defined on the

Algorithm 5 path = DiscreteFindPath(x_{init}, $\mathscr{X}_{goal}$)

```
vector(set, real) CLOSED = $\varnothing$
vector(set, real) OPEN = $\langle [x_{init}],\ 0 \rangle$
while OPEN $\neq\ \varnothing$ do
  $\langle$CurrentSet, $cost\rangle$ = OPEN.pop        //get least cost element
  if CurrentSet $\subseteq \mathscr{X}_{goal}$ then
    break
  else
    OPEN = OPEN $\setminus \{\langle$CurrentSet, $cost\rangle\}$
    CLOSED = CLOSED $\cup \{\langle$CurrentSet, $cost\rangle\}$
    $\mathscr{C}$ = C(CurrentSet)
    for all $c$ in $\mathscr{C}$ do
      $\mathscr{T}$ = T($c$)
      for all $\tau$ in $\mathscr{T}$ do
        $\langle \mathscr{X},\ cost_x \rangle = \Phi($CurrentSet, $c,\ \tau)$
        if $[\mathscr{X}] \notin$ CLOSED then
          OPEN = OPEN $\cup\ \langle [\mathscr{X}],\ cost_x + cost \rangle$
        end if
      end for
    end for
    Sort(OPEN)                               //sort on increasing cost
  end if
end while
if CurrentSet $\subseteq \mathscr{X}_{goal}$ then
  return ReconstructPath($[x_{init}]$, CurrentSet)
else
  return NULL
end if
```

interval $[0, T]$ but instead only over a discrete sequence of cells, $\{x_0, x_1, \ldots, x_n\}$. In turn, $\xi((x_{init}, x_{goal}) = \{a_0, a_1, \ldots, a_n\}$ remains valid given that the actions result in a discrete cell (or set of cells). Executing action $\langle \mathrm{c}, \tau \rangle$ results in a discretized x vice continuous-valued x. This last fact can be used to dramatically reduce computation requirements. An example implementation is shown in Algorithm 5.

By keeping track of the finite number of discretized states that have been evaluated (line 1 and line 9), if we receive an already evaluated endpoint (from line 14) it can be discarded if it has a higher cost, otherwise it is added to the queue (lines 15-17). By selecting from the OPEN queue in an intelligent manner, we can also guarantee that when a discretized state is evaluated, there will not be a lower cost path to that state in the future.[4] All states added to the OPEN list are discretized first (line 2 and line 16, where $[\cdot]$ indicates the discretized value) and the list is maintained sorted based on increasing cost (line 20).

We make one additional assumption when adopting this relaxation:

Assumption 6.3 *A discretized state,* $[x]$, *is only considered to be in the domain of a controller/trigger,* $\mathscr{D}_x^{\mathbf{r}}(\mathrm{c},\ \tau)$, *if the entire neighborhood of continuous-valued states, x, assigned to that discretized state are in the domain.*

$$[x] \in \mathscr{D}_x^{\mathbf{r}}(\mathrm{c},\ \tau) \iff x \in \mathscr{D}_x^{\mathbf{r}}(\mathrm{c},\ \tau) \quad \forall x \in [x]$$

By using this conservative approximation for inclusion in a domain we can guarantee if a controller results in an endpoint within a particular discretized state, we only need to track that discretized state to determine allowable follow-on controllers.

When determining the discretization size, it is helpful if the goal sets of the controllers are approximately the same size as the cells. Since the algorithm tracks which sets have been visited, keeping the set size low minimizes space requirements for the CLOSED list, and minimizes the number of elements analyzed from the OPEN list.

A second relaxation is applied to how uncertainty in motion and sensing are handled. One option is to ignore these uncertainties, assume deterministic motion and sensing, and only track the most-likely cell (Assumption 6.4). In this case, we make the explicit assumption that any error in motion or sensing is tightly bounded such that the end point of executing a motion is always within the domain of the following controller, without requiring any additional checks (i.e., $\Phi(\cdot, \cdot, \cdot)$ evaluates to a single cell). While this approach works in

[4] Algorithms such as A* operate in this manner to provide provably optimal paths.

practice with many controllers, it is straightforward to imagine situations where a bifurcation may occur in the environment such that small errors in either motion or sensing can result in significantly different paths taken.

Assumption 6.4 *The controllers, triggers, and/or the environment do not cause multi-modal goal-sets when executing actions. Therefore, the planner can plan for the most-likely position and the small errors in motion and sensing will remain within the domain of attraction for any follow-on controller.*

or

Assumption 6.5 *The controllers, triggers, and/or the environment may cause multi-modal goal-sets when executing actions. Therefore, the planner must explicitly evaluate whether the resulting positions from executing an action are members of the follow-on controllers domain before determining if that follow-on controller is a viable successor.*

A second option is to assume that executing a controller/trigger pair results in a relatively small grouping of end states a high percentage of the time (Assumption 6.5). For those actions that are multi-modal at a frequency higher than a user-defined value, the actions are disallowed. More formally, for an action a comprised of controller c and trigger τ, and an initial position x_0, if $x_0 \in \mathscr{D}_x^{\mathbf{r}}(\mathrm{c}, \tau)$, then $x_{\Delta t}$ is a valid transition if and only if $x_{\Delta t}$ is within the goal-set, $\Phi(x_0,\ \mathrm{c},\ \tau) \subset \mathcal{X}_{free}$, when τ is observed with probability $\mathrm{p} \geq \mathrm{p}_{thresh}$. p_{thresh} should be set high enough such that the overall probability of success of a trajectory remains satisfactory (e.g., if each action is successful with p = 0.998, then the overall probability of successfully completing 50 consecutive actions is ≥ 90%).

We can also replace line 14,

$$\langle \mathscr{X},\ cost_x \rangle = \Phi(\textsc{CurrentSet},\ c,\ \tau)$$

with

$$\langle \mathscr{X},\ cost_x \rangle = \widetilde{\Phi}(\textsc{CurrentSet},\ c,\ \tau)$$

where the function $\widetilde{\Phi}$ provides an estimate of the goal set for the current action being evaluated. This information could be an offline pre-calculated goal set, or generated by running multiple trials using empirically derived or model-based noise in the execution of motions and sensing. Assuming those noise values adequately portray real-world noise, the planner will converge to the true probability of completion of a controller given sufficient

Algorithm 6 path = SLCFindPath(x_{init}, $\mathscr{X}_{goal}$)

```
vector(set, real) CLOSED = $\varnothing$
vector(set, real) OPEN = $\langle [x_{init}],\ 0 \rangle$
while OPEN $\neq\ \varnothing$ do
   $\langle$CurrentSet, $cost\rangle$ = OPEN.pop        //get least cost element
   if CurrentSet $\subseteq \mathscr{X}_{goal}$ then
      break
   else
      OPEN = OPEN $\setminus \{\langle$CurrentSet, $cost\rangle\}$
      CLOSED = CLOSED $\cup\ \{\langle$CurrentSet, $cost\rangle\}$
      $\mathscr{C}$ = C(CurrentSet)
      for all $c$ in $\mathscr{C}$ do
         $\mathscr{T}$ = T($c$)
         for all $\tau$ in $\mathscr{T}$ do
            $\langle \mathscr{X},\ cost_x \rangle = \widetilde{\Phi}($CurrentSet$,\ c,\ \tau)$
            if $[[\mathscr{X}] \not\supseteq \mathscr{X}_i,\ \forall \mathscr{X}_i \in$ CLOSED then
               OPEN = OPEN $\cup\ \langle [\mathscr{X}],\ cost_x + cost \rangle$
            end if
         end for
      end for
      Sort(OPEN)                              //sort on increasing cost
   end if
end while
if CurrentSet $\subseteq \mathscr{X}_{goal}$ then
   return ReconstructPath([$x_{init}$], CurrentSet)
else
   return NULL
end if
```

samples.

Another modification would occur with line 15 changing from

$$\text{if } [\mathscr{X}] \notin \text{CLOSED then}$$

to

$$\text{if } [\mathscr{X}] \not\supseteq \mathscr{X}_i,\ \forall \mathscr{X}_i \in \text{CLOSED then}$$

Since the successor relationship must hold for all elements of the evaluated set, we can exclude any set that is a superset or equal to a previously evaluated set since the earlier evaluated set would have a lower cost.

While this option results in higher computational loads than the purely deterministic variant, it still allows for evaluation of the robustness of a path in the presence of typical errors in motion and sensing. In addition, running multiple trials in an online fashion remains computationally tractable compared to tracking the entire belief state across all possible configurations.

One limitation of this approach is the requirement that the goal-set from one controller/trigger must be small enough in size to fit entirely into the domain of a follow-on controller. Because of this, selecting controllers requires more effort from the user than selecting simple metric motion primitives to guarantee acceptable performance. A badly constructed controller will never be able to position the robot consistently enough to allow the planner to select a suitable follow-on controller. See [Nagarajan, 2012] for a more detailed discussion on chaining controllers.

A possible addition to alleviate this would be to construct plans without this requirement but with the capability of determining which branch was taken at a later point. For example, if a GoToGoal control was executed and the resulting goal state was not in the domain of a single FollowWall controller but rather in two different FollowWall controllers, a policy could be constructed that used subsequent sensor inputs to differentiate which of the two controller domains the robot was actually in. This approach would require significantly more processing particularly as the environment had more bifurcations like this. Whether this planner could be implemented and provide solutions within reasonable time constraints remains for future work.

6.2.5 Controller-Based Motion Primitives

With this description of the elements of the SLC planner, we can now modify how the state lattice is being constructed. The problem is formally a 6-tuple,

$$\mathcal{G} = \{\mathcal{X}, \mathcal{C}, \mathcal{T}, \mathrm{C}(\cdot), \mathrm{T}(\cdot), \Phi(\cdot, \cdot, \cdot)\}$$

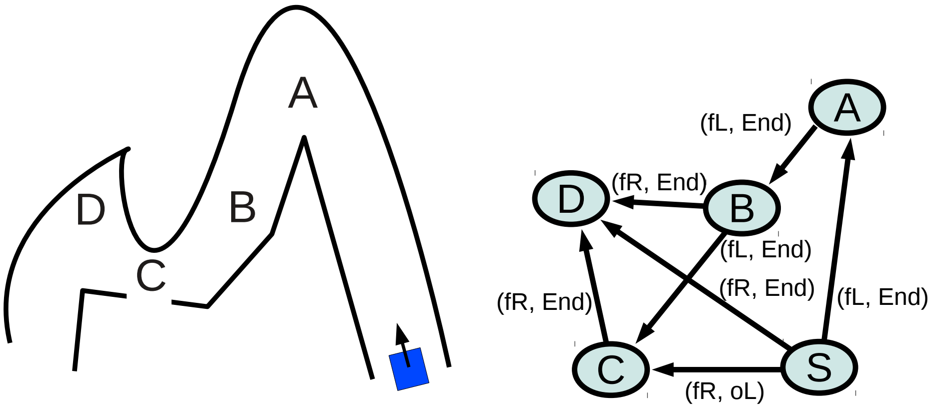
used to produce the graph, $\mathcal{G}$. Typically, when constructing a graph, there needs to be a function that returns all of the successors of any state x together with the corresponding edge costs. Let us call this function Successors(x), shown in Algorithm 7. This function incorporates lines 10-14 of Algorithm 6 into a separate function. During planning, a planner repeatedly calls this function to construct whatever portion of the graph it needs. Typically, the planner only calls this function for states that it expands. In this way, the full lattice is never explicitly constructed beforehand, but rather each edge and node is constructed as needed during the planning instance.

Algorithm 7 $succs$ = Successors(set $\mathscr{X}_{in}$)

```
vector($set, cost$) succs = $\emptyset$
$\mathscr{C}$ = C($\mathscr{X}_{in}$)
for all $c$ in $\mathscr{C}$ do
    $\mathscr{T}$ = T($c$)
    for all $\tau$ in $\mathscr{T}$ do
        $\langle \mathscr{X}_{succ},\ cost_x \rangle = \widetilde{\Phi}(\mathscr{X}_{in},\ c,\ \tau)$
        $succs = succs \cup \langle [\mathscr{X}_{succ}],\ cost_x \rangle$
    end for
end for
return $succs$
```

Besides the Successors(x) function (Algorithm 7), the test for the goal state (Algorithm 6:line 5), and the test for elements in the CLOSED queue (Algorithm 6:line 15), the remainder of the planning process remains identical to the typical search-based planning algorithm implementation such as in [Likhachev, 2012] or as described above in Algorithm 1 (Section 3.2).

To see an example of a lattice incorporating controller-based motion primitives, consider the environment shown in Fig. 6.3a. Suppose we are given a set of controllers $\mathcal{C}$ = {FollowLeftWall, FollowRightWall} with an intrinsic trigger of Completion correspond-



(a)

(b)

Figure 6.3: (a) Environment and (b) segment of graph $\mathcal{G}$ based on the SLC with $C = \{$FollowLeftWall(fL), FollowRightWall$(fR)\}$ and triggers $\mathcal{T} = \{$Completion(End), OpeningLeft(oL), OpeningRight$(oR)\}$.

ing to the end of the wall, and a set of extrinsic triggers $\mathcal{T} = \{$OpeningLeft, OpeningRight$\}$. Consider a state S, indicated by the square in the lower right corner and suppose both controllers are available at S. From state S there is an edge to A corresponding to the controller FollowLeftWall, fL, and trigger Completion, End, as shown in the portion of $\mathcal{G}$ shown in Fig. 6.3b. Likewise, with controller FollowRightWall, fR, and trigger End, the edge goes from S to D. However, if the trigger were OpeningLeft, oL, then the edge would have been from S to C. Note, unlike the typical implementation of A*, it is possible for multiple controller/trigger combinations to connect two nodes. For example, $B \rightarrow C$ is formed by the (fL, End) pair in the graph, however $B \rightarrow C$ is also connected by the pair (fR, oL) (which is not depicted). Also note that the relative costs of these controllers may vary independently across the environment so that from some states following a wall may be easier (and thus cheaper) than going to a landmark, while in other states it may be the opposite.

6.2.6 Cost Function

As with any graph search-based planning, the edges in graph $\mathcal{G}$ need to have costs associated with them. These costs can represent an arbitrary cost function that includes such factors as distance, risk, closeness to obstacles and other factors. For controller-based motion primitives the cost can also incorporate the reliability of different controllers and the risk of relying on the detection of perceptual triggers.

Since the State Lattice Planning with Controller-based Motion Primitivesis capable of working with non-metric maps or with mixed metric and non-metric motion primitives, extra care needs to be taken when developing the cost function. For example, when operating with a WallFollow controller and a conventional metric controller, the cost function must approximate the distance the robot will travel while performing the WallFollow lest the controller under-represent the true cost for long walls or over-represent the cost for short walls. This becomes more difficult for situations where there is no metric map. In these cases, it would be easy to under-represent the cost to follow a long wall compared to taking another action, if the cost function was meant to purely represent time or distance. The appropriate choice of cost function in these cases would be a cost based on the reliability or robustness of the different actions. With no metric information to base a plan on, the planner would instead produce the most robust path through a space.

6.2.7 Output of the SLC Planner

When a planner runs on a state lattice, the typical output is an ordered list of poses for the path-follower to execute. With the introduction of controller-based motion primitives the output of the planner becomes an ordered list of controllers together with the associated triggers.

6.2.8 Theoretical Properties

We evaluate the planner on feasibility, completeness, and optimality looking at both a fully deterministic planner without any errors in motion nor sensing, as well as a planner following Assumption 6.5 with small errors. In addition, where applicable, we compare the relaxation of using a discretized space with the underlying continuous state space.

Feasibility

The feasibility of trajectories produced by the planner reflects the ability of the robot to follow the generated paths successfully. We look at both feasibility in the absence of motion and sensing errors, and with typical errors found in real-world environments.

Theorem 6.1 *If a sequence ξ of controllers terminated by triggers is returned by the discretized or continuous planner, it is feasible for the robot to execute in the absence of any motion or sensing errors.*

By construction each individual action along a path is feasible for the robot to execute from the start state of that action.

The succession of individual actions is also feasible by construction. If we look at a path ξ and two sequential elements $\xi_i = \langle \mathrm{c}_i,\ \tau_i \rangle$ and $\xi_j = \left\langle \mathrm{c}_j,\ \tau_j \right\rangle$ on that path, we have:

$$\Phi(x_i, \xi_i) \subseteq \mathscr{D}^{\mathbf{r}}_{x_j}(\xi_j) \iff \mathrm{c}_j \in \mathscr{C} \wedge \tau_j \in \mathscr{T}$$

$$\mathscr{C} = \mathrm{C}(x_j)$$

$$\mathscr{T} = \mathrm{T}(\mathrm{c}_j)$$

for some user-specified radius, $\mathbf{r}$.[5]

The available controllers and triggers determined by the $\mathrm{C}(x)$ and $\mathrm{T}(c)$ functions (from line 10 and line 12, respectively, of Algorithm 6), take into account the domain and goal set of the applicable controllers. Therefore, we can guarantee that the transition between two controllers is feasible by construction.

Theorem 6.2 *If a sequence ξ of controllers terminated by triggers is returned by the discretized planner, it is feasible for the robot to execute with probability at least* $\mathrm{p}(\xi)$ *in the presence of motion and sensing errors.*

By construction, the successor actions $\left\langle \mathrm{c}_j,\ \tau_j \right\rangle$ are feasible if the predecessor action resulted in the robot ending up in the goal-set, $\Phi(x,\ \mathrm{c}_i,\ \tau_i) \subseteq \mathscr{D}^{\mathbf{r}}_{y}(\mathrm{c}_j, \tau_j)$. By definition, the robot will end up in that goal-set with probability p_i. If the robot is in the goal-set, the successor is guaranteed to be feasible. If the robot is not in the goal-set, there may be a non-zero probability of it remaining in the domain depending on domain configuration and controller failure mode. This results in an lower bound on feasibility of an individual transition of p_i. Overall, the probability of feasibility of the entire trajectory can be determined by the product of the transition probabilities:

$$\mathrm{p}(\xi) \geq \prod_i \mathrm{p}(\Phi(x_i, \xi_i) \subseteq \mathscr{D}^{\mathbf{r}}_{x_{i+1}}(\xi_{i+1}))$$

[5] The radius is user-specified, but the controller must be constructed to meet the specification.

Completeness

Completeness is the ability of a planner to discover a path through an environment if such a path exists. We evaluate completeness of the discretized planner with respect to the underlying continuous space, as well as the discretized planner with respect to the discretized space. For the latter, we look at both deterministic as well as stochastic environments (with noise in the execution of actions and sensing of the environment).

Theorem 6.3 *Even if there exists at least one sequence, ξ, of controllers terminated by triggers that move the robot from its starting position to the goal in the continuous environment, the discretized planner may not return a valid solution.*

If the discretized state is not fully in the domain of a required controller along the viable path, then the C(s) and T(s) functions (from line 2 and line 4 of Algorithm 7, will not return the required controller and trigger pair for evaluation, even if the majority of the discretized state is in the domain by Assumption 6.3 (Fig. 6.4).

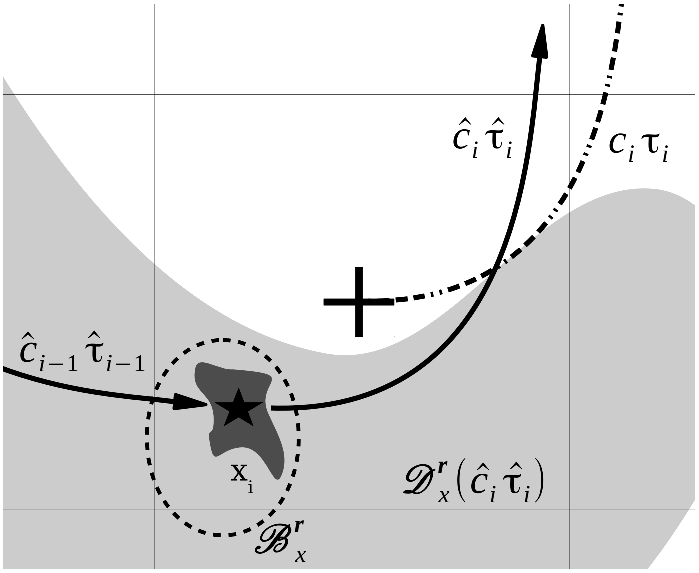

$\hat{c}_i \hat{\tau}_i$
$c_i \tau_i$
$\hat{c}_{i-1} \hat{\tau}_{i-1}$
x_i
$\mathscr{D}^{\mathbf{r}}_x(\hat{c}_i \hat{\tau}_i)$
$\mathscr{B}^{\mathbf{r}}_x$

Figure 6.4: Example of a path that is valid in the continuous space, but is not in the discretized space. At the cell coordinates, marked by the "+", the C(x) and T(c) functions do not return $\langle \hat{c}_i, \hat{\tau}_i \rangle$ since it is outside their domain shown in light gray (i.e., the dotted path from action $\langle c_i, \tau_i \rangle$ is not a viable successor). However, in the continuous domain, $\langle \hat{c}_i, \hat{\tau}_i \rangle$ is available at the end state, x_i, of the preceding action, annotated with the "$\star$" since the ball, $\mathscr{B}^{\mathbf{r}}_x$ is fully contained in the domain $\mathscr{D}^{\mathbf{r}}_x(\hat{c}_i,\ \hat{\tau}_i)$. The goal set, $\Phi(x_{i-1},\ \hat{c}_{i-1},\ \hat{\tau}_{i-1})$ is shown in dark gray.

Theorem 6.4 *If there exists at least one sequence, ξ, of controllers terminated by triggers that move the robot from its starting position to the goal in a discretized environment without uncertainty, then the* SLC *planner will return a valid solution.*

Assume ξ is a sequence of controllers terminated by triggers that moves the robot from start to the goal and that $\langle \mathrm{c}_i, \tau_i \rangle$ is the controller/trigger pair closest to the start of ξ that is not evaluated by the planner. Thus, the preceding action in ξ, $\langle \mathrm{c}_{i-1}, \tau_{i-1} \rangle$ is evaluated by the planner and results in an end state of x_i.

From state x_i, one of the following must be true:

1. $x_i \notin \mathscr{D}^{\mathbf{r}}_x(\mathrm{c}_i, \tau_i)$ If x_i is not in the domain of $\langle \mathrm{c}_i, \tau_i \rangle$, then $\langle \mathrm{c}_i, \tau_i \rangle$ is not a viable successor of $\langle \mathrm{c}_{i-1}, \tau_{i-1} \rangle$, contradicting the assumption that ξ was a viable path.
2. $x_i \in \mathscr{D}^{\mathbf{r}}_x(\mathrm{c}_i, \tau_i)$ If $\langle \mathrm{c}_i, \tau_i \rangle$ were viable successors, they would have been added to the OPEN list by Algorithm 6:line 16 or would have already been evaluated and would be in the CLOSED list by line 15.
   - If $x_i \in$ OPEN, then before the algorithm finishes, the state will be expanded and c_i and τ_i will be evaluated by line 3 unless a lower or equal cost path is found first in line 7.
   - If $x_i \in$ CLOSED, then state x_i has already been expanded by lines 8-9.

   In either case, state x_i will be expanded and c_i and τ_i will be evaluated or a lower cost path will be found prior to the algorithm completing without a path contrary to our assumption that this action was not evaluated.

Thus, by contradiction, if there is a valid sequence of controllers terminated by triggers in the discretized environment then the planner must return it or another sequence of equal or lower cost.

Theorem 6.5 *If a path ξ consisting of controllers terminated by triggers exists in the discretized state space with a probability of successfully transitioning from action ξ_i to ξ_j (i.e., $\Phi(x_i,\ \xi_i) \subseteq \mathscr{D}^{\mathbf{r}}_y(\mathrm{c}_j, \tau_j)$) of* p_i *then the planner will return a path with probability* $\overline{\mathrm{p}} \geq \prod_{\xi} \mathrm{p}_i$.

While we do not evaluate across a full belief state, we can track the probabilities of successfully entering a goal-set by modeling motion and sensor noise. Under these conditions, the planner trajectories are always feasible, but it becomes complete only probabilistically in the number of trails evaluated if the goal-sets are generated by sampling.

This forms a lower bound on the probability of generating a complete path, since while the controller may fail to reach the goal-set a percentage of the time, it is possible that it may still end up in the domain of the follow-on controller.

Optimality

The SLC is just a graph. Therefore, by running an optimal search such as A* on it we can guarantee optimality with respect to the discretization and given controllers:

Theorem 6.6 *If there exists a sequence of controllers* $\mathrm{c}_i \in C$ *terminated by triggers* $\tau_i \in \mathcal{T}$ *that move the robot from its starting position to the goal in the discretized environment without uncertainty, then the planner will return a solution and it will be an optimal sequence of controllers/triggers with respect to the cost function used.*

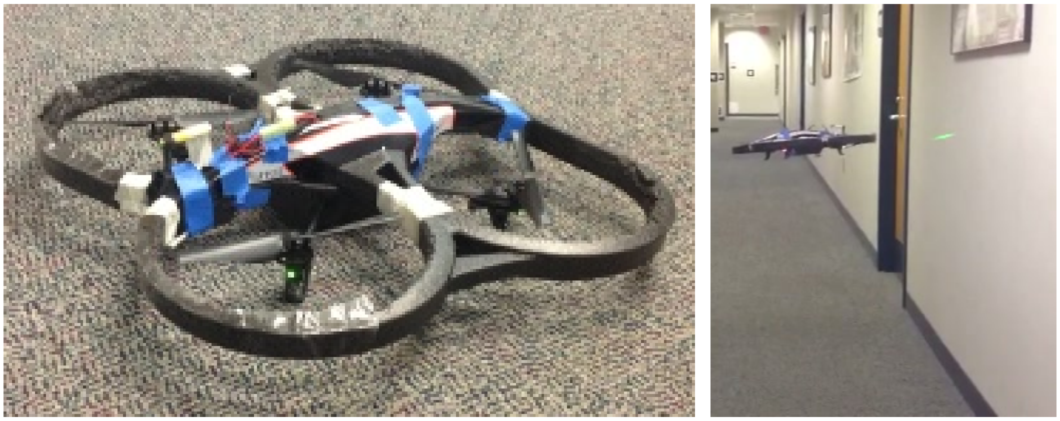
Since the algorithm is not complete when compared to the continuous version, there are no absolute optimality bounds. However, in environments with sufficiently small discretization, the discretized optimality bound is very close to the continuous optimality bound [Kim and Hespanha, 2003].

6.3 Vision-Based Micro Aerial Vehicle (MAV) Navigation Example

To demonstrate the SLC planner and develop controllers to use when the MAV is operating independently, we performed a series of flight tests using a low-cost AR.Drone 2.0 by Parrot. This quadcopter comes with 2 cameras, one forward and one downward looking, an ultrasonic and barometric altitude capability, a compass, on-board WiFi for interfacing and an ARM A8 micro-controller to perform low level controls. An off-board laptop running ROS-Fuerte performed the planning and control calculations as well as acting as the user interface to the robot. We added a 4.5 mW laser line projector to provide depth information using the forward looking camera (Fig. 6.5).

6.3.1 Controllers and Triggers for Single Robot SLC Planning

The MAV was equipped with a laser line generator and a camera as its sole means of gathering information about its surroundings. The laser line allowed the AR.Drone to calcu-



(a) (b)

Figure 6.5: The AR-Drone 2.0 with a laser line generator attached above its camera (a) and during flight (b).

late the distance to nearby objects within the field of view of the camera via triangulation (Fig. 6.6). Due to the drop-off in horizontal brightness associated with using a cylindrical lens, the distance information was only reliable over the central two-thirds of the image, or approximately 30° – 40°. Once calibrated, it proved fairly accurate and very consistent out to 1.5 m. In addition to processing the laser line from camera images, the vision pipeline was also trained to identify door jambs in the environment. Using color filters, and geometric reasoning, the door detection algorithm was able to provide the controller with the angular position, height, and width of any door jambs located in the image (see Fig. 6.7).

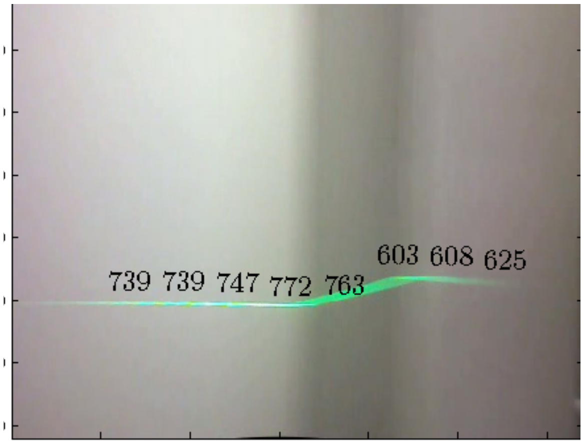
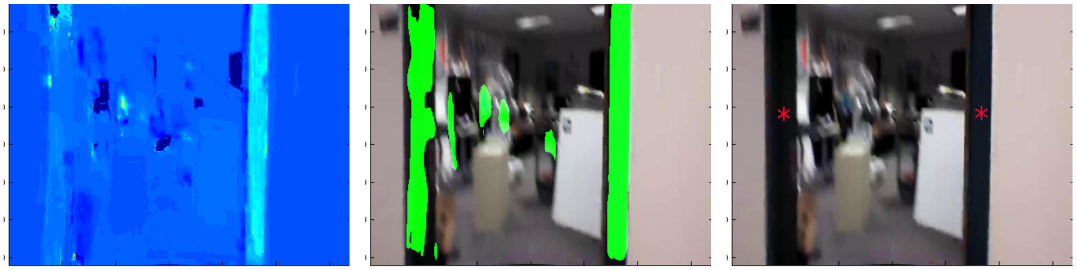


Figure 6.6: Example of Laser Line. Range in millimeters to wall with pillar on right hand side.



(a) Heatmap - lighter areas are more likely door jambs based on color.

(b) After thresholding and blurring. Highlighted ares are potential door jambs.

(c) Red star indicates jambs based on geometric constraints.

Figure 6.7: Door jamb detector in cluttered environment.

The platform was incapable of prolonged metric path following due to its highly imprecise localization system. As a result, translational metric motion primitives had to be disabled. Instead, using the laser line and the forward camera, we implemented both a FollowTheWall[Left|Right] controller and a GoToLandmark visual servo controller in addition to metric turn-in-place motion primitives. The only extrinsic trigger we used was DoorDetected which would halt execution of a controller when a new door was discovered in the camera image.

These four controllers worked for all tested door goal locations in our facility and their implementation details are presented here:

FollowTheWall[Left|Right] Follow the wall in the [Left|Right] direction (the robot attempted to remain pointed at the wall), see Fig. 6.8. Our implementation uses the laser line projector to optically determine the distance to the wall and to detect when there is an opening or door. The controller simply attempted to keep both ends of the calculated laser line distances the same, yawing as necessary to correct errors. The MAV kept a constant velocity in the direction of travel (to the left or right). The intrinsic trigger is activated when the sensor detects the end of the wall due to either a sharp increase in the range or loss of the laser from the image.

GoToLandmark(l) For this controller we implemented a detector to pick

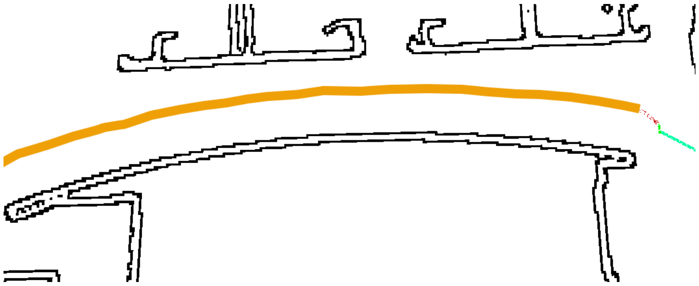


Figure 6.8: Example of a FollowTheWallLeft controller. The robot is coming from a start off the left side and proceeding to the right, facing the wall as it travels. The thick segment where it is executing the FollowTheWallLeft controller. It stops when the wall ends due to the intrinsic trigger.

out the door jambs from each image. If there was a visible and in-range door jamb *l*, the MAV would then move in a straight line towards the jamb, turning as appropriate along the way, see Fig. 6.9. In the event multiple jambs were identified, the system would use the range and angle to consistently proceed to the correct one. This controller has an intrinsic trigger when it reaches the landmark - in practice it stops just prior to the door jamb due to the anti-collision intrinsic trigger that is always present. It will additionally trigger if the landmark becomes occluded or lost enroute, however, in practice this was rare. We use door frames as our landmarks since they are visible both with the door open and shut, can be seen from either side, and are important for determining locations to transition between rooms.[6]

MetricTurn[CW|CCW] Performs a turn-in-place of 30° in either the clockwise or counter-clockwise direction. The robot uses the IMU for the intrinsic trigger when within 3° of the desired heading. Since the doors were indistinguishable from each other, the planner used this controller to re-orient itself to point at the next desired landmark/door when tran-

[6] As a convenient side effect, they were also how people gave oral directions on the floor.

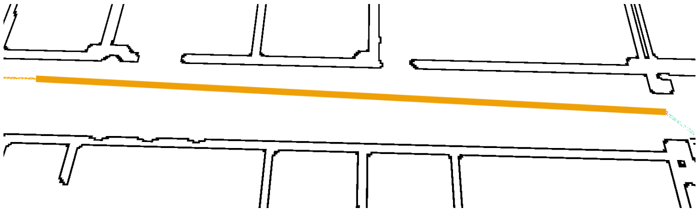


Figure 6.9: Example of the planner executing a generic GoToLandmark controller (used for any fiducial marker based landmark, not necessarily a door in this example). The start is off to the left and the goal to the bottom-right. The thick line goes directly to the landmark at the far right, terminating when reaching the landmark.

sitioning between wall following and landmark based navigation. This effectively allowed the controller to reliably transition to any adjacent door on the wall in front of it, or, by executing four turns (±120°), a door to the right or left on the opposite wall.

6.3.2 Single Robot Flight Test

In our experiments, we had the AR.Drone fly in a typical indoor environment consisting of a hallway with multiple doors along it. The MAV navigates from one area of the indoor environment to a specific door at the far end without running SLAM or generating any form of metric map. Fig. 6.10 shows an example of the plan encoding the sequence of controllers terminated by the perceptual triggers, namely the detected doors.

The cost function used for these experiments was distance traveled plus a value proportional to the time to perform each maneuver. By making the cost proportional to time, turn-in-place motions that do not have any translational component still had a positive cost.

There were 20 flights conducted which consisted of a series of GoToLandmark$_{door}$ controller phases intermixed with long FollowTheWall[Left|Right] controller execution phases, each terminated by the door detection trigger. Between these two types were an appropriate number of MetricTurn[CW|CCW] to align the MAV.

This experiment demonstrates the ability to use the SLC-based planner in situations where conventional planning is not feasible. Since the test MAV does not have the localization capability to translate reliably, typical search or sampling-based planners would not be able to construct a feasible trajectory. Pure controller-based approaches would be able to

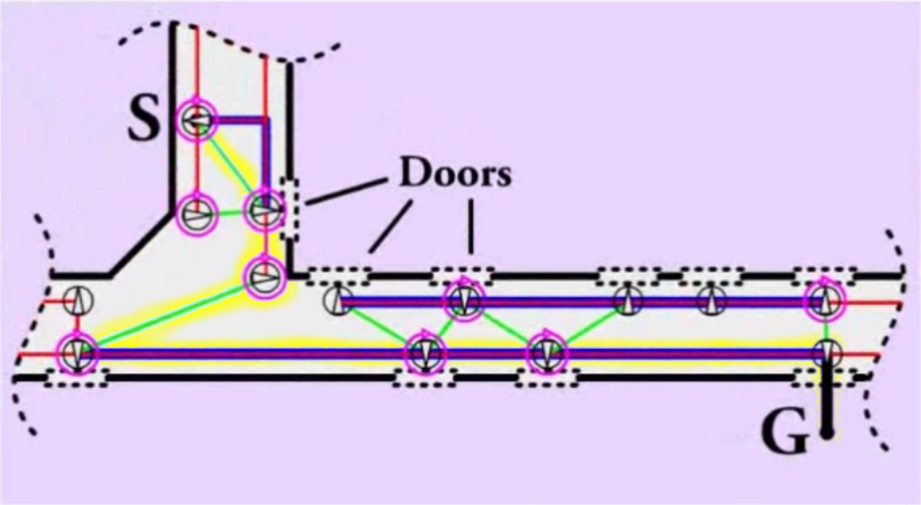


Figure 6.10: Floor plan of map traversed with start and goal positions and a portion of the search graph. Green lines are GOTOLANDMARK$_{door}$, Magenta circles are METRICTURN, Blue lines are FOLLOWTHEWALL controllers with an extrinsic trigger of DOORDETECTED, while the Red lines are the same controllers with only the intrinsic completion trigger. The final path from the planner was GOTOLANDMARK$_{door}$ → FOLLOWTHEWALLRIGHT → METRICTURNCW (×4) → GOTOLANDMARK$_{door}$ → FOLLOWTHEWALLLEFT until DOORDETECTED (×3) → at goal (yellow highlight).

use the same controllers that were used here, but would not be able to shift based on the perceptual triggers received to terminate the execution of a controller prior to reaching its goal-set. It also demonstrates the practical ability to navigate using a relatively simple set of controllers and triggers [Kerr, 2012].

6.4 Implementations of SLC

The SLC framework was used to find trajectories for a simulated K-Max helicopter navigating through a GPS-denied area. The simulation used the full vehicle dynamics to enforce constraints such as minimum turning radius, airspeed, etc.. The controllers available were:

- 13 metric motions - turn and straight combined with changes in altitude up and down.
- GOTOLAKE - similar to a GOTOLANDMARK type controller, if a lake was identified in the visual range of the helicopter, it could proceed directly towards it.
- FOLLOWROAD - when close to on top of a road, the helicopter could follow the road in either direction.

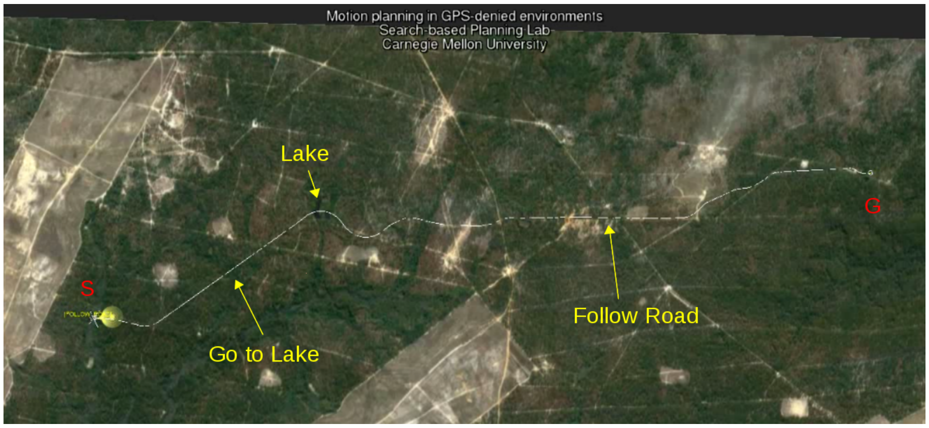

Motion planning in GPS-denied environments
Search-based Planning Lab
Carnegie Mellon University
Lake
G
S
Follow Road
Go to Lake

Figure 6.11: An example route showing GotoLake, MetricMotion, and FollowRoad controllers while operating in GPS-denied environment.

Coupled with these controllers, the helicopter had triggers for SeeNewLake and Completion. The Completion trigger when following a road was triggered at each intersection.

The overall environment was 10 km × 6 km and consisted of forests, cleared areas, and numerous roads. An example path is shown in Fig. 6.11.

The elements of the SLC framework have been implemented in two additional domains. The first is the Mitsubishi Humanoid Project entry from Carnegie Mellon University. The planner for this entry was developed by Andrew Dornbush and Karthik Vijayakumar [Dornbush et al., 2017]. The project goal was to maneuver a 4-legged humanoid robot through a complex 3-D environment to accomplish several tasks. The environment included low areas that would force the robot to crawl on all four limbs, ladders and stairs the robot had to climb, and other terrain. The planner used Bipedal, Quadrapedal, Crawl, and Climb controllers and only a distance and the intrinsic triggers.

Some of the controllers actions were pre-computed and stored, much like a typical metric motion primitive. Others were calculated during execution, for example, a inverse kinematic solver was used to transition between controllers at runtime. An example trajectory generated by the algorithm is shown in Fig. 6.12 showing a series of Bipedal, Distance controller/trigger pairs moving to the base of a ladder, followed by a Climb, Completion controller ascending the ladder.

The second implementation is in the landmark-based routing realm. This work was performed by Kalyan Vasudev Alwala, Margarita Safonova, and Maxim Likhachev [Alwala et al., 2017]. The goal of this work was to develop an efficient way of determining plans

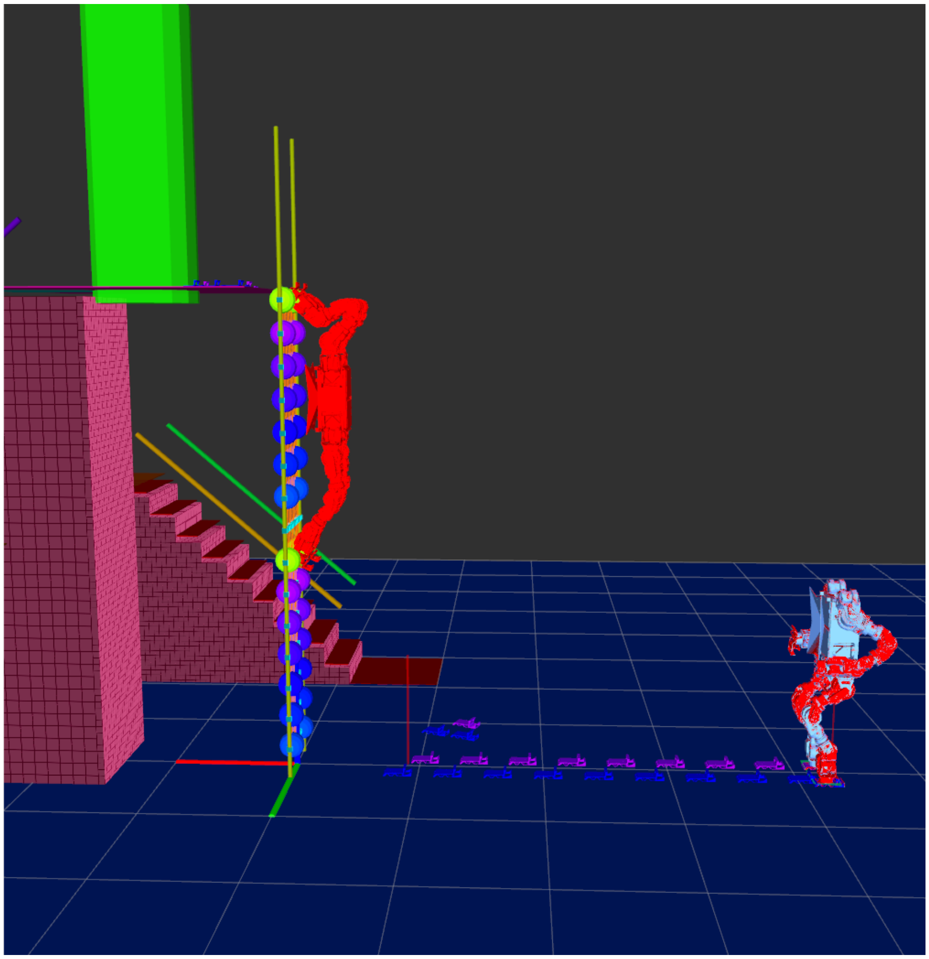
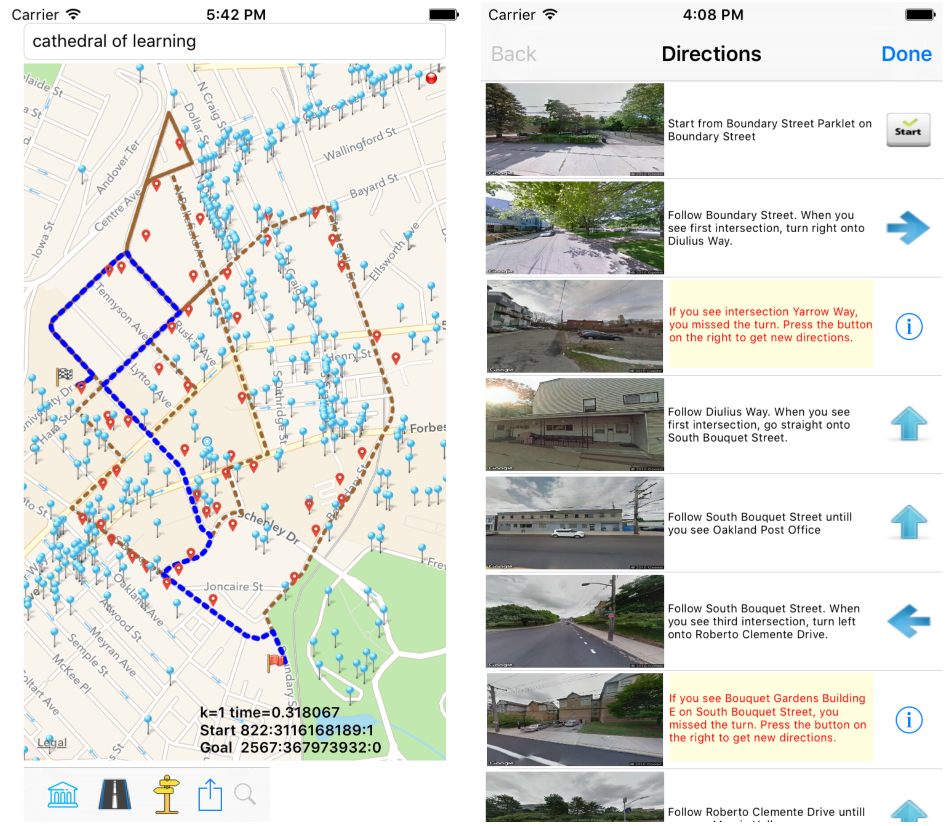


Figure 6.12: Planner expanding states using the BIPEDAL controller followed by the CLIMB controller to direct the robot up the ladder on left.

given the possibility of missing the sighting of a landmark. The navigation system consists of selecting straight, or one of the available left or right hand turns at a series of road intersections with a preference to select "Straight" unless the robot has just sensed a landmark.

This implementation allows plans of the form "Go straight until you see the library then turn left at the next intersection; if you miss the library turn left when you see the post office...". This extension accounts for the possibility of missing a landmark and the actions to take based on that occurrence. An example route and policy is shown in Fig. 6.13.



cathedral of learning

```
k=1 time=0.318067
Start 822:3116168189:1
Goal  2567:367973932:0
```

(a)

Start from Boundary Street Parklet on Boundary Street

Follow Boundary Street. When you see first intersection, turn right onto Diulius Way.

If you see intersection Yarrow Way, you missed the turn. Press the button on the right to get new directions.

Follow Diulius Way. When you see first intersection, go straight onto South Bouquet Street.

Follow South Bouquet Street untill you see Oakland Post Office

Follow South Bouquet Street. When you see third intersection, turn left onto Roberto Clemente Drive.

If you see Bouquet Gardens Building E on South Bouquet Street, you missed the turn. Press the button on the right to get new directions.

Follow Roberto Clemente Drive untill

(b)

Figure 6.13: (a) Policy and (b) instructions for landmark based routing with safety nets.

6.5 Experimental Analysis of Runtime and Quality of Single Robot SLC

6.5.1 Simulation Setup

While the first experiment showed that the SLC-based planner could successfully generate executable plans in domains where conventional planning could not, this series of tests was performed to demonstrate that for those domains where conventional methods do work SLC-based planning was also competitive in terms of planning time and path length. All tests were performed on an 2.6 GHz i7 processor with 8 GB RAM running ROS-Electric on 64-bit Kubuntu 11.04. The three planners we compared ours against were the lattice-based weighted-A* search from the SBPL package [Likhachev, 2012], and RRT-Connect and

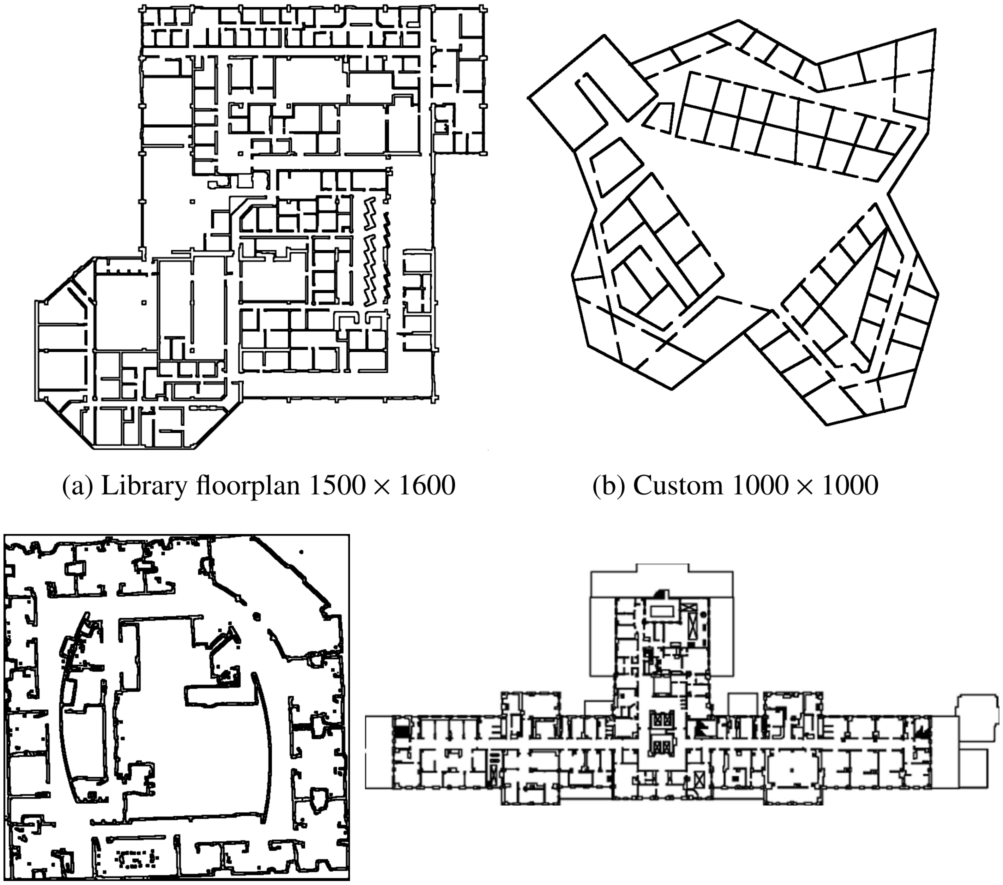
RRT* from the OMPL package [Şucan et al., 2012]. We ran the four planners on seven maps; four indoor and three outdoor. The indoor maps, shown in Fig. 6.14, were generated from building floor-plans,[†] converted laser scan data[†], and a custom drawn map. The outdoor maps, shown in Fig. 6.15 were a park[†], a set of buildings in a city, and a randomly generated map. None of the maps contained any regions that disallowed metric motions in order to not provide any advantage solely to the SLC. On each map, each planner was tasked with finding solutions between 100 randomly selected pairs of points (each planner was given the same set of (start, goal) pairs). The planners planned in $\mathbb{R}^3 = \{x, y, \psi\}$. In addition, each planner was given approximately 10 seconds of total processing time, including any post-generation smoothing (smoothing was performed on all trajectories generated by the RRT*, and RRT-Connect planners). Each pair of points was planned from scratch with the planner being restarted each time. While several planners can efficiently reuse data between plans, we only compared initial planning times since we did not allow for mid-execution replanning and we were interested in measuring the time before the robot could initially begin to execute a plan. For the SLC-based planning we used weighted-A* to search the graph constructed from the SLC. For the weighted-A* search running on both the normal lattice and the SLC the heuristic inflation (ϵ) weighting was fixed at 2.0. The heuristic was computed as the 2-D distance to the goal while accounting for obstacles. It was computed by running a single Dijkstra's search backwards from the goal, which was included in the planning times. The cost function used for these experiments was proportional to the time and distance traversed for each motion.

6.5.2 Controller-based Motion Primitives

For these experiments, in addition to WallFollowing and generic GoToLandmark controllers described in Section 6.3.1, we have also introduced a larger set of metric motion primitives and a GoToGoal controller, as well as a new perceptual trigger based on the distance traveled. The controller details are presented here:

ExecuteMotionPrimitive(m) $m \in \{forward, turnLeft, turnRight, reverse, reverseRight, reverseLeft\}$: Execute a motion primitive m, from a set containing: a left and right arcing motion and a straight motion,

[†] Some of the maps were obtained from the Robotics Data Set Repository (Radish) [Howard and Roy, 2003]. Thanks go to Ashley Tews, Richard Vaughan, and Dieter Fox for providing this data.



(a) Library floorplan 1500 × 1600

(b) Custom 1000 × 1000

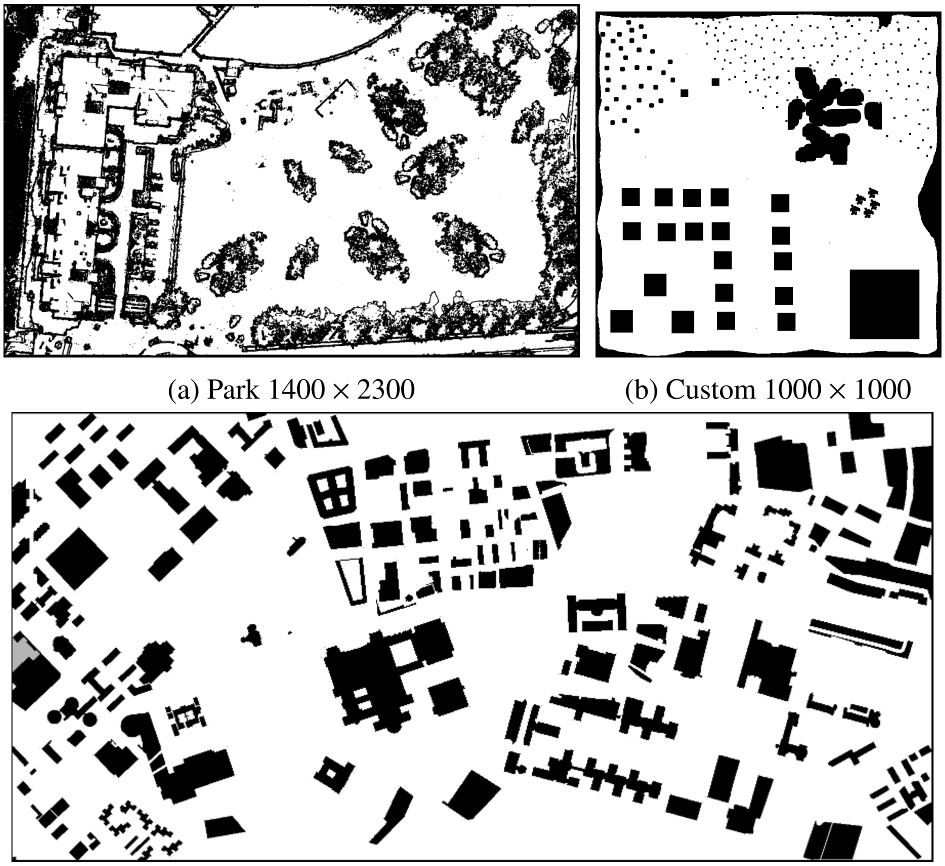
(c) Laser scan 600 × 600

(d) Hospital floorplan 500 × 1200

Figure 6.14: Indoor Maps

all in both forward and reverse direction, and a turn-in-place motion. This controller has an intrinsic trigger terminating control when the robot reaches the end of the motion primitive. Use of this controller requires that the robot have sufficient localization capability to follow the motion primitive and therefore should be penalized if localization is imperfect.

GoToGoal(g) g =goal coordinates. It incorporates the obstacle-avoidance and goal-seeking controllers to navigate towards the goal g while avoiding obstacles. The goal acts as an attractive force while obstacles supply a repulsive force for a limited range



(a) Park 1400 × 2300

(b) Custom 1000 × 1000

(c) City 900 × 1800

Figure 6.15: Outdoor Maps

similar to a modified potential fields approach. The force vectors are weighted and summed up to obtain a net vector for movement, Fig. 6.17. This controller is available for a GPS-equipped robot trying to reach a goal at a known GPS coordinate while using local sensors to remain clear of obstacles. This controller is terminated if the net motion vector is less than a given magnitude (at or near a local minima) or is directed away from the goal (too close to an obstacle). The GOTOGOAL controller was only used on the outdoor maps since it was implemented as an example of a GPS capable sensor.

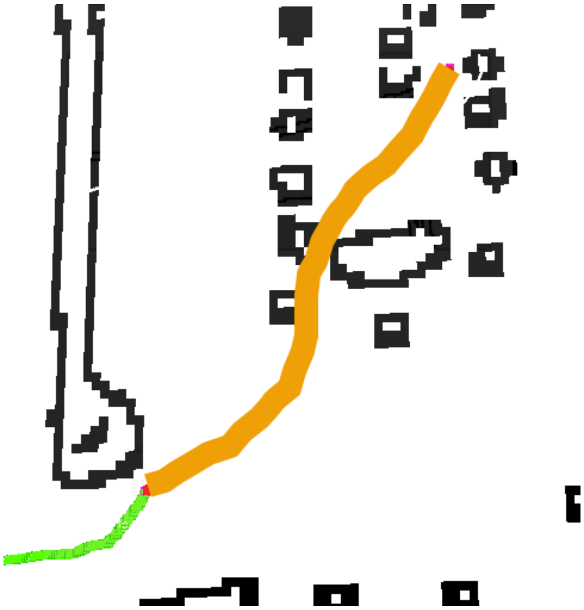


Figure 6.16: An example of the ExecuteMotionPrimitive controller moving down to the left. The thick sections in the center do not have an adequate wall or landmark near by, so the planner uses several of the standard motion primitives to avoid the obstacles. Note: the line is drawn thicker than the robot footprint for clarity; the robot trajectory remains collision free.

We also provided two selectable triggers available with all controllers:

None only stop due to an intrinsic trigger. This trigger allows the planner to execute a controller until its natural conclusion, when it will stop due to an intrinsic trigger. As part of the intrinsic trigger is the capability to halt a controller immediately before collision with an obstacle.

Distance(d) activates after a short fixed distance d is traveled. The distance trigger enables the planner to switch between controllers in the middle of their execution in order to find more optimal paths. This trigger also relies on measuring distance d. Therefore, the edge that corresponds to any controller terminating with this trigger should be penalized if measuring d accurately is difficult.

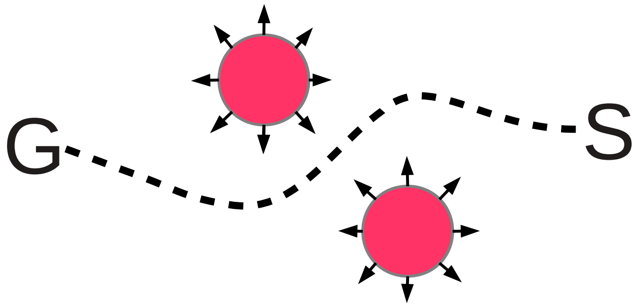


Figure 6.17: Example of the GoToGoal controller. The two obstacles repel the robot while the goal *G* attracts the robot resulting in the path shown by the dashed line.

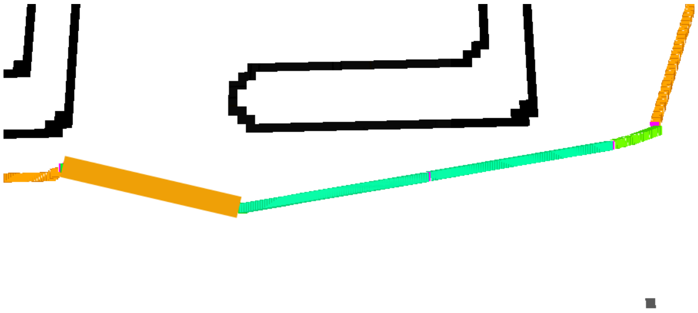


Figure 6.18: An example of a distance trigger. The "Go to Landmark" action (thick diagonal action on left) is going towards the landmark (dark gray dot in lower right), but is triggered after traveling 2m.

6.5.3 Simulation Results

The results from the 3500 planning attempts are shown in Table 6.1. Since the results across all of the indoor maps and all of the outdoor maps were similar, i.e., no one map was significantly better or worse than the others for any planner within each group, they were each combined into a single entry. Planning time was the wall time for the planner to run from when it is given the $(start, goal)$ pair until a solution is returned. The Average Planning Time Ratio takes the planning time for a single $(start, goal)$ pair and divides it by the time it takes the SLC-based planner for the same $(start, goal)$ pair, then averages it across all runs from that map. Average Path Length Ratio is calculated in the same way comparing the path length from each individual run to the path length returned by the SLC-based planner.

For some of the $(start, goal)$ pairs, not all planners were able to find a solution. This percentage is reported under the Plan Failure percentage column. The weighted A* row refers to running weighted A* on a normal lattice while SLC refers to the SLC-based planner.

Table 6.1: Experimental Results for SLC Simulation.

Map	Algorithm	Avg Plan Time (s)	Std Dev	Avg Plan Time Ratio	Std Dev	Avg Path Length Ratio	Std Dev	Planning Failure (%)
All Indoor	Weighted-A*	0.32	0.21	0.94	0.40	0.89	0.06	0
	RRT*	1.59	2.32	4.41	5.99	1.06	0.31	0
	RRT-Connect	0.56	0.72	1.69	1.23	1.09	0.29	0
	SLC	0.42	0.42	1	-	1	-	0
All Outdoor	Weighted-A*	1.00	0.40	0.74	0.26	0.92	0.05	0
	RRT*	10.25	0.34	8.73	4.22	0.94	0.09	15.6
	RRT-Connect	0.33	0.64	0.22	0.38	1.14	0.26	27
	SLC	1.50	0.85	1	-	1	-	0

As can be seen from this table, the planning time for the SLC planner is comparable to the other planners and produces paths that are similar in length. The small increase in average path length compared to the A* results is principally due to the SLC making wider turns around corners and both planners having $\epsilon > 1$. While the A* heuristic drives expansions right against the obstacles, the SLC's frequent use of the wall following and goal directed controllers resulted in many paths maintaining a larger standoff from the wall on one or both sides of a corner. The results of both tests indicate that incorporating controller-based motion primitives has only a minimal impact on planning time.

6.6 State Lattice Planning with Controller-based Motion Primitives Framework Conclusions

In this section we introduced our implementation of SLC-based planning. The SLC has the important feature that, by taking into account the specific sensing capabilities of the robot and any controllers already instantiated on the robot, it can use the corresponding additional motion primitives in instances when metric motion primitives are not available with only a minimal impact to planning times. Our primary contribution is this ability to seamlessly integrate controller-based motion primitives along with perceptual triggers into the existing state lattice planning framework due to our method of constructing the search graph.

Chapter 7

Planning for Air-Ground Collaborative Localization using SLC

> Unholy metal machine
> It's close to midnight and He's barking at the moon
> I'm a metal machine
>
> SABATON *Metal Machine*

7.1 Multi-Robot SLC Planner for Air-Ground Teams

One particularly challenging aspects of planning for multi-robots is determining at what points the robots should collaborate and when they should operate independently. We have extended our SLC planner to the multi-robot domain to demonstrate one method of solving this question. Typically, joint configuration space planning is only practical for small workspaces. However, when we combine the Planning with Adaptive Dimensionality (PAD)[Gochev et al., 2011] framework we can achieve tractable search times in high-dimensional spaces. We will provide some background on the PAD framework as well as our implementation in Section 7.2 followed by our experimental results in Section 7.3.

7.2 Planning with Adaptive Dimensionality

A key component to our collaborative planner is the use of the planning with adaptive dimensionality framework (PAD). This framework builds on the fact that many high-dimensional path planning problems have lower-dimensional projections that represent the problem very well in most areas. For example, path planning for a non-holonomic vehicle needs to consider the planar position of the vehicle $\langle x, y\rangle$, but also the heading angle, ψ, to ensure that system constraints, such as minimum turning radius, are obeyed. However, a two-dimensional representation of the problem, only considering the planar position of the vehicle $\langle x, y\rangle$, can work well in many areas of the state-space (Fig. 7.1).

With a free flying aerial robot the state space has six degrees of freedom, $\langle x, y, z, \phi, \theta, \psi\rangle$. When the aerial robot is combined with the ground robot the overall state space increases to 9 states - position and orientation of the ground vehicle, $\langle x, y, \psi\rangle$, plus the six aerial vehicle states. It is well known that this high dimensionality coupled with a large environment greatly increases planning times to the point that they become infeasible for on-the-fly computation. By planning with adaptive dimensionality, we can plan in only those dimensions that are critical at a given point. One example from our previous work, is to set the high-dimensional states to be the full planning state space $\left\langle (x, y, z)_{\text{uav}}, (x, y, \psi)_{\text{ugv}}\right\rangle$ while the low-dimensional states is just the UAV position, $\langle x, y, z\rangle_{\text{uav}}$.

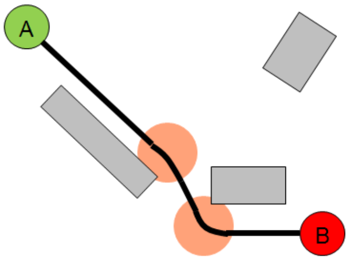


Figure 7.1: Example trajectory for a non-holonomic vehicle with minimum turning radius constraints. Planning for the heading of the vehicle is needed in areas that require turning in order to ensure constraints are satisfied (circles). Planning for the heading of the vehicle is unnecessary for areas that can be traversed in a straight line. A: start location; B: goal location; gray boxes: obstacles.

In this section we will provide a brief overview of the algorithm for Planning with Adaptive Dimensionality. For a more detailed explanation of the algorithm, we refer the reader to the original work on Planning with Adaptive Dimensionality [Gochev et al., 2011][Gochev et al., 2012]. We will use the notation $\xi_{\mathcal{G}}(x_i, x_j)$ to denote a path from state x_i to state x_j in a graph[1] $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ with a vertex set $\mathcal{X}$ and edge set $\mathcal{E}$. The cost of

[1] The SLC output is a graph, however, since the graph construction and planning are interleaved, $\mathcal{E}$ is generated online from $\{\mathcal{C}, \mathcal{T}, \mathrm{C}(\cdot), \mathrm{T}(\cdot), \Phi(\cdot,\cdot,\cdot)\}$ as defined in Section 6.2

an entire path ξ will be denoted by $\textsc{Cost}(\xi)$. We will use $\xi^*_{\mathcal{G}}(x_i, x_j)$ to denote a least-cost (optimal) path and $\xi^{\epsilon}_{\mathcal{G}}(x_i, x_j)$ for $\epsilon \geq 1$ to denote a path of bounded cost sub-optimality: $\textsc{Cost}(\xi^{\epsilon}_{\mathcal{G}}(x_i, x_j)) \leq \epsilon \cdot \textsc{Cost}(\xi^*_{\mathcal{G}}(x_i, x_j))$.

7.2.1 Graph Construction and Projection Function

The algorithm for Planning with Adaptive Dimensionality considers two graphs as defined by their corresponding state-spaces and transition sets—a high-dimensional graph $\mathcal{G}^{HD} = (\mathcal{X}^{HD}, \mathcal{E}^{HD})$ with dimensionality h, and a low-dimensional graph $\mathcal{G}^{LD} = (\mathcal{X}^{LD}, \mathcal{E}^{LD})$ with dimensionality l, where $\mathcal{X}^{LD}$ is a projection of $\mathcal{X}^{HD}$ onto a lower dimensional manifold $(h > l, |\mathcal{X}^{HD}| > |\mathcal{X}^{LD}|)$ through a projection function λ.

$$\lambda : \mathcal{X}^{HD} \to \mathcal{X}^{LD}$$

The projection function λ^{-1} maps low-dimensional states to their high-dimensional pre-images:

$$\lambda^{-1} : \mathcal{X}^{LD} \to \mathcal{P}(\mathcal{X}^{HD})$$

and is defined as

$$\lambda^{-1}(X^{LD}) = \{X \in \mathcal{X}^{HD} | \lambda(X) = X^{LD}\}$$

where $\mathcal{P}(\mathcal{X}^{HD})$ denotes the power set of $\mathcal{X}^{HD}$.

Each of the two state-spaces may have its own transition set. However, in order to provide path cost sub-optimality guarantees, the algorithm requires that the costs of the transitions be such that for every pair of states x_i and x_j in $\mathcal{X}^{HD}$,

$$\textsc{Cost}\left(\xi^*_{\mathcal{G}^{HD}}\left(x_i, x_j\right)\right) \geq \textsc{Cost}\left(\xi^*_{\mathcal{G}^{LD}}\left(\lambda(x_i), \lambda(x_j)\right)\right) \tag{7.1}$$

In other words, it is required that the costs of least-cost paths in the low-dimensional state-space always underestimate the costs of the least-cost paths between the corresponding states in the high-dimensional state-space.

7.2.2 PAD Algorithm

The algorithm for Planning with Adaptive Dimensionality iteratively constructs and searches a hybrid graph $\mathcal{G}^{AD} = (\mathcal{X}^{AD}, \mathcal{E}^{AD})$ consisting mainly of low-dimensional states and transitions. The algorithm only introduces regions of high-dimensional states and transitions into the hybrid graph where it is necessary in order to ensure the feasibility of the resulting path and maintain path cost sub-optimality guarantees. Each iteration of the algorithm consists of two phases: planning phase and tracking phase.

In the planning phase, the current instance of the hybrid graph $\mathcal{G}^{AD}$ is searched for a path $\pi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{AD}}(start, goal)$. Any graph search algorithm that provides a bound on path cost sub-optimality can be used to compute $\xi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{AD}}$. Similar to the original implementation of the algorithm, we used the weighted A* graph-search algorithm.

In the tracking phase, a high-dimensional tunnel $\mathbb{T}$ (a subgraph of $\mathcal{G}^{HD}$) is constructed around the path found in the planning phase. Then $\mathbb{T}$ is searched for a path $\xi_{\mathbb{T}}(start, goal)$ from start to goal. If $\textsc{Cost}(\xi_{\mathbb{T}}) \leq \epsilon_{\text{track}} \cdot \textsc{Cost}(\xi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{AD}})$, then $\xi_{\mathbb{T}}$ is returned as the path computed by the algorithm. If no path through $\mathbb{T}$ is found or $\textsc{Cost}(\xi_{\mathbb{T}})$ does not satisfy the above constraint, the algorithm identifies locations in $\mathcal{G}^{AD}$, where the search through $\mathbb{T}$ got stuck or where large cost discrepancies between $\xi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{AD}}$ and $\xi_{\mathbb{T}}$ are observed. The algorithm then introduces new high-dimensional regions in $\mathcal{G}^{AD}$ centered at the identified locations. For more details on how the locations of new high-dimensional regions are computed and how high-dimensional regions are introduced in $\mathcal{G}^{AD}$, please refer to [Gochev et al., 2011][Gochev et al., 2012]. The algorithm then proceeds to the next iteration.

7.2.3 Theoretical Properties

If the high-dimensional state-space $\mathcal{X}^{HD}$ is finite, the algorithm for Planning with Adaptive Dimensionality is complete with respect to the underlying graph $\mathcal{G}^{HD}$ encoding the search problem and is guaranteed to terminate. If a path ξ is found by the algorithm, then ξ satisfies

$$\textsc{Cost}(\xi) \leq \epsilon_{\text{plan}} \cdot \epsilon_{\text{track}} \cdot \xi^{*}_{\mathcal{G}^{HD}}(start, goal)$$

In other words, the cost of a path returned by the algorithm is bounded by $\epsilon_{\text{plan}} \cdot \epsilon_{\text{track}}$ times the cost of an optimal path from *start* to *goal* in the high-dimensional graph $\mathcal{G}^{HD}$, where ϵ_{plan} and ϵ_{track} are user-specified parameters. These theoretical properties are proven in [Gochev et al., 2011].

7.2.4 PAD Application to Multi-Robot State-Lattice Planning

In the particular application considered in this work, the task was to navigate a UAV with limited self-localization capabilities to a desired target location with the assistance of a UGV with good localization capability. The aerial vehicle is able to localize itself relative to the ground vehicle, when the ground vehicle is visible from the UAV's position. The full-dimensional system state was represented by 6-dimensional state-vectors: $\left\langle (x, y, z)_{\text{uav}}, (x, y, \psi)_{\text{ugv}} \right\rangle$ The transitions available for each state consisted of pre-computed motion primitives (metric motions) for both vehicles, and controller-based motion primitives for the aerial vehicle. The cost of each transition was proportional to the cumulative distance traveled by each vehicle during the transition and an additional modifier based on controller accuracy and robustness for the controller-based motion primitives on the UAV. The roll and pitch of the UAV, ϕ and θ, were derived variables from the desired velocity and error and were calculated by the controllers to meet the desired trajectory points. The heading (yaw), ψ, of the aerial vehicle was also not a free variable and was determined by the specific controller used in a transition, with the exception being the yaw controller explicitly controlling yaw when transitioning between other controller types. For example, when executing a WALLFOLLOWING transition, the heading is kept parallel to the direction of the wall, and for transitions using the ground vehicle for localization (LANDMARKNAV$_{\text{DYN}}$), the heading is kept facing the ground vehicle. We assume that in many areas of the environment the aerial vehicle is capable of autonomous navigation by using the independent controllers (following walls or going around corners, for example), and the localization assistance of the ground vehicle is needed only in rare occasions, when these independent controllers are unavailable or cost prohibitive to the UAV and metric motions need to be performed. Thus, the low-dimensional representation of the system used for Planning with Adaptive Dimensionality was a 3-dimensional state-vector $\langle x, y, z \rangle_{\text{uav}}$, only considering the position of the aerial vehicle. The costs of transitions in the low-dimensional space satisfies the requirements of PAD as shown in equation (7.1) as only the cost of moving the aerial vehicle is considered. High-dimensional regions are introduced in the hybrid graph only in areas of the environment where ground vehicle localization assistance is needed. The obvious downside to this approach is that the UGV sits idle if it is not needed by the UAV (in practice, it would move to its next localization position as soon as the UAV was finished with it, and since it moved slower than the UAV, it did not end up waiting significantly).

7.2.5 Output of the Planner

A state lattice with controller-based motion primitive planner generates trajectories that are defined as a series of controllers to execute. For this collaborative planner we expand that to include at each time step the appropriate controller for all robots in the team. When the robots are operating independently, the trajectory execution finite state machine of each robot independently tracks where it is in the plan. When a robot reaches a planner step that requires another robot to be at a specific location, the first robot will pause and hold position until the other one finishes its controller sequences preceding that point. When the two are back in sync, both will be allowed to continue executing their controllers. In practice, the UAV rarely has to wait for the UGV with the one common occurrence being using $\text{LANDMARKNAV}_{\text{DYN}}$ motion that went behind a pillar. The UAV would move as far as it could and still see the UGV, which would then make a quick motion to the side to allow the UAV to continue on.

7.3 SLC Planning for Air-Ground Teams

The experiment using SLC was to incorporate a collaborative localization controller into the planning process. Since this would require multiple robots, and the corresponding increase in the size of the joint state space, we turned to the PAD algorithm in order to keep planning times tractable for real-world robots. The system was implemented with an air-ground robot team operating in an environment where neither robot individually could accomplish the navigation tasks due to inaccessibility for the UGV and lack of sufficient localization capability for the UAV.

7.3.1 Air-Ground Team Details

For our testing we used two robots: a Segway-based ground robot named Melvin (Fig. 7.2a), and a Pixhawk/DJI-based aerial robot (Fig. 7.2b). All of the computers ran Kubuntu 12.04

and ROS Groovy with the exception of the computer on the aerial vehicle which runs Xubuntu 14.04 and ROS Indigo.

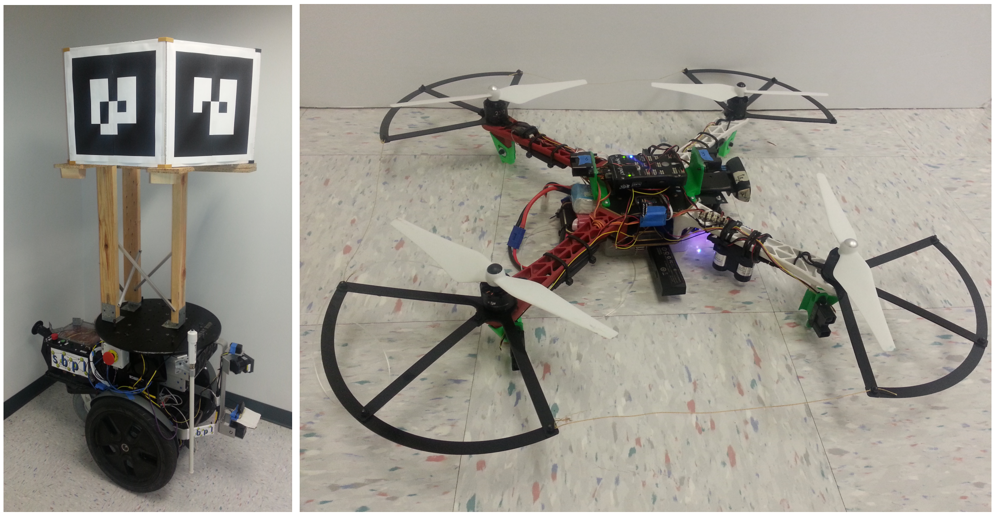
Melvin the UGV is a relatively large indoor robot with a significant payload capacity and high endurance. With a normal operating load, Melvin is capable of operating for 3+ hours running two independent computer systems, and carrying all required communications infrastructure. The first computer system is used as the low level controller and consists of an i3 3.4 GHz processor with 8 GB of RAM. This system is used for all navigation, sensing, and interfacing with the Segway base. The second system is a general purpose computer equipped with a dual processor Xeon with 8 physical cores and 16 GB of RAM. The planner and plan execution agent are both run on this computer. In addition, this computer is used to run the processor intensive tasks of the UAV such as AR marker detection/extraction and all of the mid-level controllers (the wall following controller, the metric motion controllers, and the landmark controllers). Melvin is also equipped with two Hokuyo scanning laser sensors mounted on tilt mounts for a full 3-dimensional scanning capability, and a web camera for visual sensing. To assist the UAV with collaborative localization, the UGV has six AR markers arranged in a horizontally aligned hexagon so that the UAV can detect and accurately determine orientation of the UGV even in the presence of some low obstacles.

Unlike the UGV, the UAV is relatively spartan in terms of sensing and computing power. The airframe itself is a DJI Flamewheel 450 with a Pixhawk flight control computer and an ODROID XU3 supplemental computer. A standard web camera is used for landmark detection, while 6 Sharp IR sensors with 1.5 m range are arranged around the perimeter to provide obstacle detection and wall following capabilities. The ODROID captures the images and transmits them to the UGV for processing, then receives the output from the mid-level controllers and translates them into the required format for the Pixhawk to execute.

7.3.2 Controllers and Triggers Implemented for Air-Ground Teams

In order to use the SLC planner, a set of available controllers and triggers were constructed. For the UAV, we implemented WALLFOLLOWING, LANDMARKNAV, METRICTURN, and METRICSTEP controllers. The UGV had high-quality localization data from its two scanning laser range finders and so was only given a METRICMOTION controller.

The WALLFOLLOWING controller on the UAV used two IR range sensors mounted on each side of the UAV in order to maintain a flight path parallel to, and a specified distance from,



(a) (b)

Figure 7.2: (a) Segway-based UGV. 2 scanning laser range finders, high gain antenna, webcam, general purpose server. (b) Pixhawk-based quadcopter UAV. Laser Altimeter, 6 IR range sensors, standard webcam.

any given wall in the environment. It was given two intrinsic triggers: the ability to trigger when the wall ended (Completion) and when an obstacle was within a certain distance of the front or back of the UAV (Obstacle).

For this experiment we used fiducial markers and a simple detector algorithm running on our vision processing pipeline. Due to this, our landmarks gave more than just the directional information we used in Section 6.3 and Section 6.5. We instead had a full 3-D pose of the UAV which we could then use to execute arbitrary trajectories. For our implementation we used six motion primitives based on the landmarks allowing motion in the $\pm x$, $\pm y$, and $\pm z$ directions while using the landmark. We also allowed two different instantiations of the LandmarkNav controller. The first (LandmarkNav$_{\text{Stat}}$) used static landmarks in the environment that the UAV could detect with its onboard camera system and knowing the position and orientation of the landmark, could determine its own position. The second controller (LandmarkNav$_{\text{Dyn}}$) used fiducial markers on the ground robot for the same purpose. However, this then requires that during the planning cycle the UAV and UGV positions are both considered simultaneously in order for the collaborative localization to occur (see Section 7.2.4). In other words, LandmarkNav$_{\text{Dyn}}$ could only be exercised while the UGV was close enough and within line-of-sight of the UAV.

Since the UAV does have an IMU and optical flow system there are locations within the environment that it is capable of generating short range metric motions. We used two such motions, an ability to yaw to a desired heading, and the ability to move a set distance forward. The two metric controllers were always used in groups to turn corners: MetricStep(0.5 m) → MetricTurn(±90°) → MetricStep(1.0 m). The other use of metric motions was when shifting between the Landmark and WallFollowing controllers, the metric yaw controller would orient the UAV. The accuracy of the IMU and optical flow system did not allow for continuous metric motion without receiving some external sensor information so the planner limited the allowable locations during planning time by imposing a high cost on these motions.

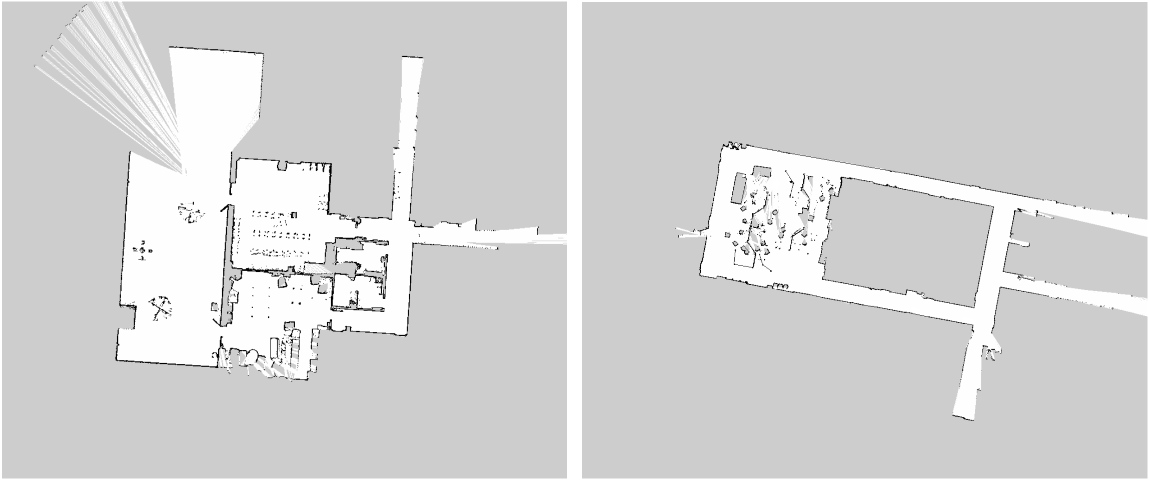
7.3.3 The Output of the Planner

A state lattice with controller-based motion primitive planner generates trajectories that are defined as a series of controllers to execute. For this collaborative planner we expand that to include at each time step the appropriate controller for all robots in the team. When the robots are operating independently, the trajectory execution finite state machine of each robot independently tracks where it is in the plan. When a robot reaches a planner step that requires another robot to be at a specific location, the first robot will pause and hold position until the other one finishes its controller sequences preceding that point. When the two are back in sync, both will be allowed to continue executing their controllers. In practice, the UAV rarely has to wait for the UGV with the one common occurrence being using LandmarkNav$_{\text{Dyn}}$ motion that went behind a pillar. The UAV would move as far as it could and still see the UGV, which would then make a quick motion to the side to allow the UAV to continue on.

7.3.4 Air-Ground Team Experiments

Environment

Our test environments are meant to replicate a standard indoor office environment (see Fig. 7.3). We used one area that consisted of two large conference rooms, an outdoor patio area, and a few hallways with small offices. The other test area was comprised of a cluster of cubicles, boxes, equipment, and office furniture in half the area, while the other half is a set of featureless hallways. For our experiments, we restricted the UGV to operate only in the room portions of the environments by placing obstacles at each hallway entrance. The UAV was free to operate throughout the map with different areas performing better with different controllers. For example, since the hallways had no features and the UGV was unable to enter them, the LandmarkNav controllers were not usable (for both static and dynamic landmarks). On the other hand, the crowded, erratically configured cubicle area did not feature any navigable straight walls.



(a) Conference Rooms and Patio

(b) Cubicles and Hallways

Figure 7.3: Maps of two testing environments.

Test Setup

To test our planners performance in real-world scenarios, we randomly selected start and goal points throughout the environment for the UAV and start points only for the UGV. This allowed us to construct plans where the two robots started near each other but allowed the UAV to operate independently if required. The planner would allow the UGV to move as necessary to support the UAV motion to get to the goal.

The cost function used for these experiments was proportional to the time and distance traversed for each motion.

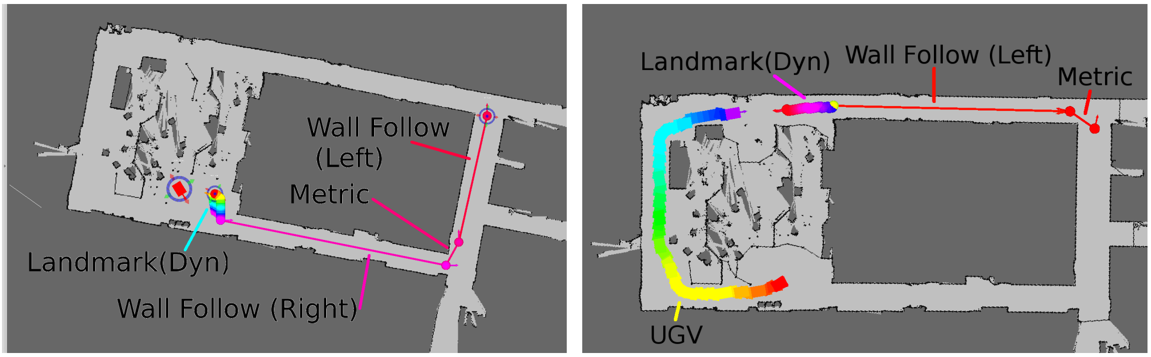
Table 7.1: Experimental Results for Multi-Robot SLC with Collaborative Localization.

Algorithm	Planning Time (s)		Num. Iter.	Num. Expans.	Path Cost	Final Eps.	Success Rate
	Avg.	Std. Dev.	Avg.	Avg.	Avg.	Avg.	(%) @10min
PAD MR SLC	13.87	20.04	1.41	2405	13754	1.25	100
Full-D ARA* MR SLC	30.19	76.84	n/a	5388	14069	1.35	67.5

Test Results

Overall the system was able to generate plans that would not be solvable without using the controller-based motion primitives due to the lack of an adequate localization capability of the UAV operating alone. In addition, the adaptive dimensionality planner played a key role in making these plans computationally feasible given the high dimensionality of the combined state space. Planning times for 40 randomly generated start-goal pairs on several indoor environments are shown in Table 7.1. The planning time is the time to the first solution with each planner initially starting at an $\epsilon = 1.35$. The performance of our collaborative localization algorithm (labeled PAD MR SLC, Planning with adaptive dimensionality using multi-robot state lattices with controller-based motion primitives) is compared against a full-dimensional ARA* algorithm running on the multi-robot SLC. The results shown in Table 7.1 are averaged over the 27 scenarios that both planners were able to solve successfully. The full-dimensional ARA* planner was unable to solve the 13 most difficult scenarios within 600 s, which was considered a planning failure, whereas the maximum time that our approach took to solve a scenario was 112.4 s. As can be seen, using adaptive dimensionality reduction we are able to produce significantly better paths quicker than using the full-dimensional ARA*.

An example of a generated plan is shown in Fig. 7.4. The plan initially has both robots near each other in the open portion of the operating area. The UAV is tasked to move to a location at the end of one of the hallways, then to return the the cubicles and land. During the return flight along the upper corridor, the UGV repositioned by going around the left side of the map to the top of the cubicle area. From here, it could provide localization to the UAV as it re-entered this region. Other testing used random start and goal locations on this and similar environments.



(a)

(b)

Figure 7.4: (a) Portion of plan showing the UAV starting at the lower left and using LandmarkNav$_{\text{Dyn}}$ motions to get into the hallway. Once in the hallway, the UAV uses WallFollowing and metric motions to reach the goal position in the upper right. The UGV is the magenta rectangle near the UAV start. The start and goal configurations have a blue circle around them. (b) The return flight to the cubicle region. The UGV repositions along the left side of the map to provide localization to the UAV at the end of its trajectory.

7.3.5 Air-Ground Team Results

State lattice-based planning using controller-based motion primitives combined with an adaptive dimensionality planner provides a method of solving complex, high-dimensional navigation problems that cannot be solved using existing methods. This is due to the computation requirements of planning in a high-dimensional space and the inherently limiting assumption that most planners use on the existence of sufficient localization ability of the target robots. We did discover that our existing controllers were insufficient to reliably pass through a standard doorway (which provides less than 20 cm clearance total around our UAV) and this remains an area for further development.

Chapter 8

Discussion and Future Work

And I will force you to be free,
Turn on the light so you can see
Your perception - just reflections on the wall
TURISAS *For your own Good*

8.1 Contribution

As we have discussed in the previous chapters, multi-robot collaboration is a key technology to allow robots to participate in real-world environments. The first step of this was to coordinate the exploration behavior of the participating robots. By generating trajectories encapsulating the aggregate knowledge of the search team, we were able to show that total exploration times were improved while simultaneously allowing for higher fidelity control of the exploration.

The second primary method of collaboration discussed is the ability to provide support for collaborative localization. Robots that are capable of using other robots in order to better position themselves can greatly expand the types of environments the team can successfully operate in. By not requiring each team member to have all of the sensor capabilities that may be required, the team can specialize, and since these capabilities operate over a distance, they can allow navigation even in areas that would not otherwise be navigable to an individual robot. It is this higher level of assistance and collaboration that we present here.

In the preceding chapters we have presented these collaborative planning algorithms and their contributions are summarized here:

The exploration planners developed allow:

- Exploration of teams of robots in large cluttered environments
- Ability to incorporate high-level user input at runtime such as:
    - Logical regions to guide exploration
    - Inter-robot distances to account for sensor and communication limits
- Exploration in 3-dimensions to include looking at the underside of obstacles if required

Our collaborative localization planner provides:

- Ability for a air-ground team of robots to localize if and when required while allowing independent operations otherwise
- Allowing operation in terrain otherwise non-traversable using conventional planning and localization schemes

In addition, we presented the State Lattice Planning with Controller-based Motion Primitives framework that can be used on teams or individual robots allowing for:

- Controller-based motion primitives that can use full sensing and locomotion capabilities of a robot
- Perceptual triggers allow for changing current active controller at any time as well as forming implicit counters
- The framework can operate without any explicit localization
- Can combine both metric motion controllers as well as funnel type controllers
- This framework has been put to use in other domains such as landmark based routing and humanoid planning

8.2 Future Work

While the algorithms presented in this work are capable in their current forms there are a few areas that exist for potential future work.

First, the exploration and localization algorithms are independent. An open question remains as to whether these two algorithms can be combined to allow for a team of robots to not only use each other for localization, but do so in an initially unknown environment. In order to expand our earlier exploration framework to include the ability for robot team members to collaboratively localize there are a few obstacles that must be overcome. The first of which is the high-dimensional joint state space when dealing with a team that includes aerial vehicles. The extra dimensionality makes standard trajectory planning approaches untenable for real-time, real-world operation. Through the use of PAD we can limit the dimensions that must be handled in the majority of environment if we can determine an appropriate low-dimensional manifold on which to conduct the bulk of the planning. In the work presented, we plan for just the aerial vehicle as the ground vehicle has no independent goal that it is attempting to reach. In a combined exploration mission, there is a trade-off to be made: help another robot at the cost of not performing any additional independent exploration plus the cost of traveling to and from the location of the team member. A possible approach to solving this problem is to modify the utility of a potential goal position by accounting for the loss of information gain accrued by the assisting robot.

The second major obstacle lies with the directed nature of most controllers. Starting at a position A and moving towards a landmark can consistently get you from a region of the environment to a single repeatable point in that environment. However, there is no clear inverse action. Starting at the landmark and backing away is unlikely to result in the robot ending up at position A. Without this inverse action capability, exploration using controllers may result in robots reaching regions of the environment that they are unable to extricate themselves without outside assistance. This assistance, like collaborative localization, is a benefit in that it prevents permanent loss of a robot, unfortunately, it still has a cost in terms of overall exploration efficiency. There will need to be some analysis of how long should a robot explore while "lost" before being rescued by another robot. In addition, what method the second robot uses to reach the first also needs to be determined (since the first robot is "lost" it does not have a global coordinate to pass to its rescuer).

Another area of future work is in the development of more advanced controllers for the SLC framework. As presented, only collaborative localization controllers have been developed but there exists a wide range of other collaborative efforts that may benefit from this approach.

Bibliography

Alphabetical order by first author.

G. J. Agin. *Real time control of a robot with a mobile camera*. SRI International, 1979. 6.1

K. V. Alwala, M. Safanova, and M. Likhachev. Planning with Safety Nets for Landmark-based Navigation and Routing. In *In Submission*, 2017. 6.4

F. Amigoni and A. Gallo. A Multi-Objective Exploration Strategy for Mobile Robots. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3850–3855, 2005. doi: 10.1109/ROBOT.2005.1570708. 2.1

J. D. Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill Education, 5th edition, 2010. 3.2.1

T. Arai, E. Pagello, and L. E. Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, 18(5):655–661, 2002. 2

R. Arkin. *Behavior-based robotics*. MIT press, 1998. 2

R. C. Arkin. Motor Schema — Based Mobile Robot Navigation. *The International Journal of Robotics Research*, 8(4):92–112, 1989. doi: 10.1177/027836498900800406. URL http://ijr.sagepub.com/content/8/4/92.abstract. 6.1

R. C. Arkin and T. Balch. AuRA: principles and practice in review. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):175–189, 1997. doi: 10.1080/095281397147068. 6.1

J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2061–2066vol.3, 2000. doi: 10.1109/IROS.2000.895274. 5.2.3

W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 476–481vol.1, 2000. doi: 10.1109/ROBOT.2000.844100. 2.1

W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3):376–386, June 2005. ISSN 1552-3098. doi:

10.1109/TRO.2004.839232. 2.1, 2.2

R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential Composition of Dynamically Dexterous Robot Behaviors. *IJRR*, 18(6):534–555, 1999. doi: 10.1177/02783649922066385. URL http://ijr.sagepub.com/content/18/6/534.abstract. 6, 6.1

J. Butzke and M. Likhachev. Planning for multi-robot exploration with multiple objective utility functions. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3254–3259. IEEE, 2011. 1, 2.1

J. Butzke, K. Daniilidis, A. Kushleyev, D. D. Lee, M. Likhachev, C. Phillips, and M. Phillips. The University of Pennsylvania MAGIC 2010 multi-robot unmanned vehicle system. *Journal of Field Robotics*, 29(5):745–761, 2012. ISSN 1556-4967. doi: 10.1002/rob.21437. 2.1

J. Butzke, K. Sapkota, K. Prasad, B. MacAllister, and M. Likhachev. State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 258–265, Sept. 2014. doi: 10.1109/IROS.2014.6942570. 1, 6.2.1

J. Butzke, A. Dornbush, and M. Likhachev. 3-D Exploration with an Air-Ground Robotic System. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept 28 - Oct 03, 2015, Congress Center Hamburg, Hamburg, Germany*, 2015. 1, 2.1

J. Butzke, K. Gochev, B. Holden, E.-J. Jung, and M. Likhachev. Planning for a Ground-Air Robotic System with Collaborative Localization. In *[Submitted] ICRA 2016*, 2016. 1

D. Calisi, A. Farinelli, L. Iocchi, and D. Nardi. Multi-objective exploration and search for autonomous rescue robots. *Journal of Field Robotics*, 24(8-9):763–777, 2007. ISSN 1556-4967. doi: 10.1002/rob.20216. 2.1

S.-B. Choi and J. Hedrick. Vehicle longitudinal control using an adaptive observer for automated highway systems. In *American Control Conference, Proceedings of the 1995*, volume 5, pages 3106–3110vol.5, June 1995. doi: 10.1109/ACC.1995.532088. 6.1

C. M. Clark, S. M. Rock, and J. C. Latombe. Motion planning for multiple mobile robots using dynamic networks. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 3, pages 4222–4227 vol.3, Sept 2003. doi: 10.1109/ROBOT.2003.1242252. 2

B. Cohen, M. Phillips, and M. Likhachev. Planning single-arm manipulations with n-arm robots. In *Eighth Annual Symposium on Combinatorial Search*, 2015. 2

R. G. Colares and L. Chaimowicz. The next frontier: Combining information gain and distance cost for decentralized multi-robot exploration. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 268–274, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3739-7. doi: 10.1145/2851613.2851706. URL http://doi.acm.org/10.1145/2851613.2851706. 2.1

D. C. Conner, H. Choset, and A. A. Rizzi. Integrating planning and control for single-bodied wheeled mobile robots. *Autonomous Robots*, 30(3):243–264, 2011. 6, 6.1

C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using Laplace's equation. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 2102–2106vol.3, May 1990. doi: 10.1109/ROBOT.1990.126315. 6.1

I. Şucan and L. E. Kavraki. A Sampling-Based Tree Planner for Systems With Complex Dynamics. *Robotics, IEEE Transactions on*, 28(1):116–131, Feb. 2012. ISSN 1552-3098. doi: 10.1109/TRO.2011.2160466. 6.1

I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 2012. URL http://ompl.kavrakilab.org. To appear. 6.5.1

O. De Silva, G. Mann, and R. Gosine. Development of a relative localization scheme for ground-aerial multi-robot systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 870–875, Oct. 2012. doi: 10.1109/IROS.2012.6386015. 2.2

E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (1):269–271, 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. 3.2

A. Dornbush, K. Vijaykumar, S. Bardapurkar, F. Islam, and M. Likhachev. A Single-Planner Approach to Multi-Modal Humanoid Mobility. In *In Submission*, 2017. 6.4

A. Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3):249–265, June 1987. ISSN 0882-4967. doi: 10.1109/JRA.1987.1087096. 5.2.4

P. Fankhauser, M. Bloesch, P. KrÃijsi, R. Diethelm, M. Wermelinger, T. Schneider, M. Dymczyk, M. Hutter, and R. Siegwart. Collaborative navigation for flying and walking robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2859–2866, Oct 2016. doi: 10.1109/IROS.2016.7759443. 2.3

D. Ferguson and A. T. Stentz. Field D*: An Interpolation-based Path Planner and Replanner. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, Oct. 2005. 6.1

D. Fox, W. Burgard, H. Kruppa, and S. Thrun. Collaborative Multi-Robot Localization. In W. FÃűrstner, J. Buhmann, A. Faber, and P. Faber, editors, *Mustererkennung 1999*, Informatik aktuell, pages 15–26. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66381-2. doi: 10.1007/978-3-642-60243-6_2. 2.2

K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev. Path Planning with Adaptive Dimensionality. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2011. 7.1, 7.2, 7.2.2, 7.2.3

K. Gochev, A. Safonova, and M. Likhachev. Planning with Adaptive Dimensionality for

Mobile Manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012. 7.2, 7.2.2

J. P. Gonzalez and A. T. Stentz. Planning with Uncertainty in Position Using High-Resolution Maps. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1015–1022, Apr. 2007. doi: 10.1109/ROBOT.2007.363118. 2.2

H. H. Gonzalez Banos and J.-C. Latombe. Navigation Strategies for Exploring Indoor Environments. *The International Journal of Robotics Research*, 21(10-11):829–848, 2002. doi: 10.1177/0278364902021010834. URL http://ijr.sagepub.com/content/21/10-11/829.abstract. 2.1

R. Grabowski, P. Khosla, and H. Choset. Autonomous exploration via regions of interest. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1691–1696vol.2, 2003. doi: 10.1109/IROS.2003.1248887. 2.1

B. Grocholsky, J. Keller, V. Kumar, and G. Pappas. Cooperative air and ground surveillance. *Robotics Automation Magazine, IEEE*, 13(3):16–25, Sept. 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.1678135. 2.2, 2.3

P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2): 100–107, July 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136. 3.2, 6, 6.1

R. He, E. Brunskill, and N. Roy. PUMA: Planning under Uncertainty with Macro-Actions. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, 2010. 6.1

A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3): 189–206, 2013. ISSN 0929-5593. doi: 10.1007/s10514-012-9321-0. 5.2.4

A. Howard and N. Roy. The Robotics Data Set Repository (Radish), 2003. URL http://radish.sourceforge.net/. †

J. Jessup, S. Givigi, and A. Beaulieu. Robust and efficient multi-robot 3D mapping with octree based occupancy grids. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 3996–4001, Oct. 2014. doi: 10.1109/SMC.2014.6974556. 5.2.4

V. Kallem, A. Komoroski, and V. Kumar. Sequential Composition for Navigating a Non-holonomic Cart in the Presence of Obstacles. *Robotics, IEEE Transactions on*, 27(6): 1152–1159, Dec. 2011. ISSN 1552-3098. doi: 10.1109/TRO.2011.2161159. 6, 6.1

S. Karaman and E. Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010. 6.1

R. Käslin, P. Fankhauser, E. Stumm, Z. Taylor, E. Mueggler, J. Delmerico, D. Scaramuzza,

R. Siegwart, and M. Hutter. Collaborative localization of aerial and ground robots through elevation maps. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 284–290, Oct 2016. doi: 10.1109/SSRR.2016.7784317. 2.2

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug. 1996. ISSN 1042-296X. doi: 10.1109/70.508439. 6.1

M. Keidar and G. A. Kaminka. Efficient frontier detection for robot exploration. *The International Journal of Robotics Research*, 33(2):215–236, 2014. doi: 10.1177/0278364913494911. URL http://dx.doi.org/10.1177/0278364913494911. 2.1

A. Kelly and B. Nagy. Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control. *The International Journal of Robotics Research*, 22(7 - 8):583–601, July 2003. 3.2.1

O. S. Kerr. A Theory of Law. *16 Green Bag 2D 111*, 2012. URL http://www.greenbag.org/v16n1/v16n1_ex_post_kerr.pdf. 6.3.2

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 500–505, Mar. 1985. doi: 10.1109/ROBOT.1985.1087247. 6.1

J. Kim and J. P. Hespanha. Discrete approximations to continuous shortest-path: application to minimum-risk path planning for groups of uavs. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, volume 2, pages 1734–1740 Vol.2, Dec 2003. doi: 10.1109/CDC.2003.1272863. 6.2.8

S. K. Kim and M. Likhachev. Parts assembly planning under uncertainty with simulation-aided physical reasoning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4074–4081, May 2017. doi: 10.1109/ICRA.2017.7989468. 6.1

S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. von Stryk. Hector open source modules for autonomous mapping and navigation with rescue robots. In *RoboCup 2013: Robot World Cup XVII*, pages 624–631. Springer, 2014. 5.2.3

H. Kress Gazit, G. E. Fainekos, and G. J. Pappas. Where's Waldo? Sensor-Based Temporal Logic Motion Planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116–3121, Apr. 2007. doi: 10.1109/ROBOT.2007.363946. 6, 6.1

J. J. Kuffner Jr. and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001vol.2, 2000. doi: 10.1109/ROBOT.2000.844730. 6.1

H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. 2008. 2.2, 6.1

S. Lacroix and G. Le Besnerais. Issues in Cooperative Air/Ground Robotic Systems. In M. Kaneko and Y. Nakamura, editors, *Robotics Research*, volume 66 of *Springer Tracts in Advanced Robotics*, pages 421–432. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-14742-5. doi: 10.1007/978-3-642-14743-2_35. 2.2

J.-C. Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012. 2

S. M. Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, 1998. 6.1

J. S. Lewis and J. M. O'Kane. Planning for provably reliable navigation using an unreliable, nearly sensorless robot. *International Journal of Robotics Research*, 32(11):1339–1354, Sept. 2013. 6.1

B. Li, Y. Zhang, Z. Shao, and N. Jia. Simultaneous versus joint computing: A case study of multi-vehicle parking motion planning. *Journal of Computational Science*, 20:30 – 40, 2017. ISSN 1877-7503. doi: https://doi.org/10.1016/j.jocs.2017.03.015. URL http://www.sciencedirect.com/science/article/pii/S187775031630254X. 2

W. Li, T. Zhang, and K. Kuhnlenz. A vision-guided autonomous quadrotor in an air-ground multi-robot system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2980–2985, May 2011. doi: 10.1109/ICRA.2011.5979579. 2.2

M. Likhachev. The Search-Based Planning Library. 2012. URL http://sbpl.net. 6.2.5, 6.5.1

M. Likhachev and D. Ferguson. Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009. doi: 10.1177/0278364909340445. URL http://ijr.sagepub.com/content/28/8/933.abstract. 3.2.1, 6.1

M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems (NIPS)*, 16, 2003. 3.2.4, 6, 6.1

T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, Feb 1983. ISSN 0018-9340. doi: 10.1109/TC.1983.1676196. 6.1

V. Lumelsky and A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *Automatic Control, IEEE Transactions on*, 31(11): 1058–1063, Nov. 1986. ISSN 0018-9286. doi: 10.1109/TAC.1986.1104175. 6.1

B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev. Path Planning for Non-Circular Micro Aerial Vehicles in Constrained Environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3933–3940. IEEE,

2013. 5.2.3

D. Martin, M. Walsh, and T. Lascari. New generation of drones set to revolutionize warfare. 60 Minutes, 2017. URL https://www.cbsnews.com/news/60-minutes-autonomous-drones-set-to-revolutionize-military-technology-2/. CBS television broadcast. 5.2.5

A. A. Masoud. Managing the Dynamics of a Harmonic Potential Field-Guided Robot in a Cluttered Environment. *Industrial Electronics, IEEE Transactions on*, 56(2):488–496, Feb. 2009. ISSN 0278-0046. doi: 10.1109/TIE.2008.2002720. 6.1

D. Mellinger and V. Kumar. Control and planning for vehicles with uncertainty in dynamics. In *2010 IEEE International Conference on Robotics and Automation*, pages 960–965, May 2010. doi: 10.1109/ROBOT.2010.5509794. 6.1

N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, 2012. ISSN 1556-4967. doi: 10.1002/rob.21436. URL http://dx.doi.org/10.1002/rob.21436. 2.3

N. Michael, S. Shen, K. Mohta, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro. *Collaborative Mapping of an Earthquake Damaged Building via Ground and Aerial Robots*, pages 33–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-40686-7. doi: 10.1007/978-3-642-40686-7_3. URL http://dx.doi.org/10.1007/978-3-642-40686-7_3. 2.3

U. Nagarajan. *Fast and Graceful Balancing Mobile Robots*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2012. 6.1, 6.2.4

C. Nieto-Granda, I. John G. Rogers, and H. I. Christensen. Coordination strategies for multi-robot exploration and mapping. *The International Journal of Robotics Research*, 33(4):519–533, 2014. doi: 10.1177/0278364913515309. URL http://dx.doi.org/10.1177/0278364913515309. 2.1

M. O'Connor, G. Elkaim, and B. Parkinson. Kinematic GPS for closed-loop control of farm and construction vehicles. In *PROCEEDINGS OF ION GPS*, volume 8, pages 1261–1268. Citeseer, 1995. 6.1

S. Oßwald, M. Bennewitz, W. Burgard, and C. Stachniss. Speeding-up robot exploration by exploiting background information. *IEEE Robotics and Automation Letters*, 1(2):716–723, July 2016. ISSN 2377-3766. doi: 10.1109/LRA.2016.2520560. 2.1

M. Pivtoraiko and A. Kelly. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2005. 3.2.1, 6.1

C. Rasche, C. Stern, L. Kleinjohann, and B. Kleinjohann. A distributed multi-UAV path planning approach for 3D environments. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 7–12, Dec. 2011. doi: 10.1109/ICARA.2011.6144847. 2.1

R. Rathnam and A. Birk. Multi-robot exploration with auvs on cliffs and other 3d structures with a predominant orientation. In *OCEANS 2015 - Genova*, pages 1–7, May 2015. doi: 10.1109/OCEANS-Genova.2015.7271563. 2.1

C. Reardon and J. Fink. Air-ground robot team surveillance of complex 3d environments. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 320–327, Oct 2016. doi: 10.1109/SSRR.2016.7784322. 2.2

I. Rekleitis, R. Sim, G. Dudek, and E. Milios. Collaborative exploration for the construction of visual maps. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 3, pages 1269–1274vol.3, 2001. doi: 10.1109/IROS.2001.977157. 2.2

N. Roy and S. T. Thrun. Coastal Navigation with Mobile Robots. In *In Advances in Neural Processing Systems 12*, pages 1043–1049, 1999. 2.2

R. Sawhney, K. M. Krishna, and K. Srinathan. On Fast Exploration in 2D and 3D Terrains with Multiple Robots. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 73–80, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-6-1. URL http://dl.acm.org/citation.cfm?id=1558013.1558022. 2.1

S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli. Flying Fast and Low Among Obstacles. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2023–2029, Apr. 2007. doi: 10.1109/ROBOT.2007.363619. 6.1

M. J. Schuster, C. Brand, H. Hirschmüller, M. Suppa, and M. Beetz. Multi-Robot 6D Graph SLAM Connecting Decoupled Local Reference Filters. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 2015. 2.2, 5.2.4

M. Schwager, P. Dames, D. Rus, and V. Kumar. *A Multi-robot Control Policy for Information Gathering in the Presence of Unknown Hazards*, pages 455–472. Springer International Publishing, Cham, 2017. ISBN 978-3-319-29363-9. doi: 10.1007/978-3-319-29363-9_26. URL https://doi.org/10.1007/978-3-319-29363-9_26. 2.1

R. Sim and N. Roy. Global A-Optimal Robot Exploration in SLAM. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 661–666, april 2005. doi: 10.1109/ROBOT.2005.1570193. 2.1

R. G. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. L. S. Younes. Coordination for Multi-Robot Exploration and Mapping. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on In-*

novative Applications of Artificial Intelligence, pages 852–858. AAAI Press, 2000. ISBN 0-262-51112-6. URL http://portal.acm.org/citation.cfm?id=647288.723404. 2.1

K.-T. Song, C.-Y. Tsai, and C.-H. C. Huang. Multi-robot cooperative sensing and localization. In *Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on*, pages 431–436, Sept. 2008. doi: 10.1109/ICAL.2008.4636190. 2.2

P. Sujit and R. Beard. Multiple UAV exploration of an unknown region. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):335–366, 2008. ISSN 1012-2443. doi: 10.1007/s10472-009-9128-7. 2.1

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999. 6.1

P. Svec, B. C. Shah, I. R. Bertaska, J. Alvarez, A. J. Sinisterra, K. von Ellenrieder, M. Dhanak, and S. K. Gupta. Dynamics-aware target following for an autonomous surface vehicle operating under COLREGs in civilian traffic. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3871–3878. IEEE, 2013. 3.2.1, 6.1

H. Tanner. Switched UAV-UGV Cooperation Scheme for Target Detection. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3457–3462, Apr. 2007. doi: 10.1109/ROBOT.2007.364007. 2.3

T. Tao, Y. Huang, F. Sun, and T. Wang. Motion Planning for SLAM Based on Frontier Exploration. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 2120–2125, aug. 2007. doi: 10.1109/ICMA.2007.4303879. 2.1

D. Thakur, M. Likhachev, J. Keller, V. Kumar, V. Dobrokhodov, K. Jones, J. Wurz, and I. Kaminer. Planning for opportunistic surveillance with multiple robots. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 5750–5757, Nov. 2013. doi: 10.1109/IROS.2013.6697189. 3.2.1, 6.1

N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-D Path Networks in Unstructured Environments. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4624–4629, Apr. 2005. doi: 10.1109/ROBOT.2005.1570833. 2.3

R. T. Vaughan, G. S. Sukhatme, F. J. Mesa Martinez, and J. F. Montgomery. Fly spy: Lightweight localization and target tracking for cooperating air and ground robots. In *Distributed autonomous robotic systems 4*, pages 315–324. Springer, 2000. 2.2

R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *Robotics and Automation, IEEE Transactions on*, 18(5):662–669, 2002. 2.3

A. Viguria, I. Maza, and A. Ollero. Distributed Service-Based Cooperation in Aerial/Ground

Robot Teams Applied to Fire Detection and Extinguishing Missions. *Advanced Robotics*, 24(1-2):1–23, 2010. doi: 10.1163/016918609X12585524300339. URL http://dx.doi.org/10.1163/016918609X12585524300339. 2.2, 2.3

A. Visser and B. Slamet. Balancing the Information Gain Against the Movement Cost for Multi-robot Frontier Exploration. In H. Bruyninckx, L. Preucil, and M. Kulich, editors, *European Robotics Symposium 2008*, volume 44 of *Springer Tracts in Advanced Robotics*, pages 43–52. Springer Berlin / Heidelberg, 2008. 2.1

T. Wanasinghe, G. Mann, and R. Gosine. Distributed collaborative localization for a heterogeneous multi-robot system. In *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on*, pages 1–6, May 2014. doi: 10.1109/CCECE.2014.6900998. 2.2

K.-H. C. Wang, A. Botea, et al. Tractable multi-agent path planning on grid maps. In *IJCAI*, volume 9, pages 1870–1875. Pasadena, California, 2009. 2

B. Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151, July 1997. doi: 10.1109/CIRA.1997.613851. 2.1, 4

B. Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, pages 47–53, New York, NY, USA, 1998. ACM. ISBN 0-89791-983-1. doi: 10.1145/280765.280773. 2.1

K. Yang, S. Keat Gan, and S. Sukkarieh. A Gaussian process-based RRT planner for the exploration of an unknown and cluttered environment with a UAV. *Advanced Robotics*, 27(6):431–443, 2013. doi: 10.1080/01691864.2013.756386. URL http://dx.doi.org/10.1080/01691864.2013.756386. 2.1

J. Zelenka and T. Kasanicky. Outdoor UAV control and coordination system supported by biological inspired method. In *Robotics in Alpe-Adria-Danube Region (RAAD), 2014 23rd International Conference on*, pages 1–7, Sept. 2014. doi: 10.1109/RAAD.2014.7002224. 2

H. Zhang, J. Butzke, and M. Likhachev. Combining global and local planning with guarantees on completeness. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4500–4506, May 2012. doi: 10.1109/ICRA.2012.6225382. 6.1

R. Zlot, A. Stentz, M. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Robotics and Automation, 2002. IEEE International Conference on*, volume 3, pages 3016–3023, 2002. doi: 10.1109/ROBOT.2002.1013690. 2.1