RICE UNIVERSITY

Motion Planning with Uncertain Information in Robotic Tasks

by

Devin Kieber Grady

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Dr. Lydia E. Kavraki, Chair
Noah Harding Professor
Computer Science

Dr. James McLurkin
Assistant Professor
Computer Science

Dr. Mark Moll
Adjunct Assistant Professor
Computer Science

Dr. Marcia K. O'Malley
Associate Professor
Mechanical Engineering

HOUSTON, TEXAS

MAY 2014

ABSTRACT

Motion Planning with Uncertain Information in Robotic Tasks

by

Devin Kieber Grady

In the real world, robots operate with imperfect sensors providing uncertain and incomplete information. We develop techniques to solve motion planning problems with imperfect information in order to accomplish a variety of robotic tasks including navigation, search-and-rescue, and exposure minimization.

This thesis focuses on the challenge of creating robust policies for robots with imperfect actions and sensing. These policies map input observations to output actions. The tools that exist to solve these problems are typically Partially-Observable Markov Decision Processes (POMDPs), and can only handle small problem instances. This thesis proposes several techniques to expand the size of the problem instance that can be considered. Because executing a policy is simple once the offline computation is done, even inexpensive, computationally constrained robots can use these policies and solve the tasks mentioned.

First we show that the solution of an abstracted action space can be used to bootstrap a complete solution for navigation. Generalizing this action space abstraction to both action and state spaces expands the set of problems that can be solved. Additionally, the concept of abstraction is applied to the workspace – we develop a method to compute local solutions to a noisy navigation problem, then stitch them together into a global solution. Our proposed

methods are run on large problem instances, and the output policies are compared against policies generated with existing techniques. Though these large tasks are often unsolvable with previous methods, abstraction allows us to find high quality policies.

Our findings show that these techniques significantly increase the size of tasks involving planning with uncertain information for which solutions can be found. The techniques presented generally offer significant speed increases and often solution quality improvements as well.

Additionally, this thesis includes work on two separate problems. First, we solve a task where several robots cooperate to quickly classify an observed object as one of several possible types using a camera. Then, we proceed to solve a task where a single robot navigates to a destination quickly, but the robot may need to allocate time towards obtaining information about a new object discovered along the way.

Acknowledgments

I would like to thank my adviser, Dr. Lydia Kavraki, for her guidance and dedication to helping me produce quality research. Dr. Mark Moll has also been instrumental in poking holes in my ideas and has reviewed many, many paper drafts. My thesis committee also deserves many thanks for spending their valuable time on my behalf.

Additionally, the entire Physical and Biological Computing Group at Rice deserves to be thanked for providing a comfortable, innovative atmosphere to work in, and for their valuable criticism.

Finally, I would like to thank my wife Becky, our combined family, and my friends for their support and understanding. Without their support, none of this would have been possible.

Of course, none of the research presented in this thesis is possible without funding. This work was funded in part by Rice University, NSF 1139011, NSF DUE 0920721, CCF 1018798 and the U. S. Army Research Laboratory and the U. S. Army Research Ofce under grant number W911NF-09-1-0383. D. Grady was additionally supported by an NSF Graduate Research Fellowship. Equipment was funded in part by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097 as well as by NSF EIA-0216467 and a partnership between Rice University, Sun Microsystems and Sigma Solutions Inc.

Contents

Illustrations

Chapter 1

Introduction

Robotic planning with uncertain information is required for all autonomous motion in the physical world. Uncertainty could be caused by some or all control inputs that, when executed, drive the robot from a current state to a distribution of possible future states. In some cases, robotic systems and their environments can be carefully designed so that mechanically, they exhibit negligible uncertainty. In particular, this is seen in industrial settings. In order for robotic systems to move out of strictly-controlled industrial settings and into "the wild" of homes, offices, and outdoors, robots have to contend with environments that were not meticulously engineered. At the same time, although it might be acceptable to spend hundreds of thousands of dollars on an industrial car welding robot, the consumer market has only shown a willingness to pay a few hundred dollars for robots. This drastic cost reduction will only happen with less reliance on precision engineering and massive scale. Software algorithms that can directly account for uncertainty in the world as well as in the engineering of the robot are an important aspect of developing robots that can co-work and co-habitate with humans.

Planning for both the uncertainty in control and sensor measurements is an important part of this robotic planning problem. Extreme examples would be attempting to navigate a walking robot through mud (actuation noise) and trying to account for sensor noise caused by radiation in a nuclear power plant. Most robotic tasks, have some intermediate level of both sensor and actuation noise, meaning that finding a set of control inputs requires

planning that accounts for the uncertainty in the information available to the robot. For example, if a robot is tasked with loading a dishwasher the location and type of dishes in a full sink may be difficult to observe without beginning the task and moving some dishes that were obstructing others.

Considering the uncertainty present is critical because, in many tasks, the motion constraints of the robot are not the fundamental challenge. If the primary difficulty lies in the interpretation of limited sensor data, then abstracting some of the motion constraints can help find a good solution faster. In these types of robotic tasks, the robot must consider the sensor input, the expected noise model, and how the world state should be reflected in the sensor data. Sensors tend to be limited, in both noise/quality and by their transformation of world state to sensor data – it may be difficult to invert this transformation and find the probability distribution of world states given particular sensor data. These difficulties are generally present in physical robot platforms, and even characterizing the uncertainty itself is difficult [1].

1.1 Problem Overview

In this thesis, we will use a very general definition of a robotic task as maximizing a reward function through a sequence of actions selected using only uncertain information. Although many robotic tasks can be reduced to planning for a path, the effect of uncertain information is that a solution must map all possible stochastic events to an action selection. This mapping is what we will refer to as a policy. The problem to solve, then, is to compute a policy that will maximize the total reward accumulated over a sequence of actions. The policy must take into account all sources of uncertainty present in the task definition.

The problem of maximizing reward over a sequence of actions, given stochastic actions and only partial observability of the current state, is exactly what finding an optimal policy for a Partially-Observable Markov Decision Process (POMDP) entails [2]. A formulation of uncertainty could be general in the extreme, but we will use the POMDP formulation (discussed in more detail in Chapter 2). Finding an optimal policy is referred to as solving the POMDP, and for each task, returns a policy that very simply tells the robot what action to execute given the current observation. Past observations are theoretically required, but can be implicitly stored as the current policy state. Once a policy has been computed, the time to look up the next action is negligible. This property means that even low-cost robots with little on-board computation can execute complex behaviors resulting from many hours of computation using a POMDP.

The notion that a POMDP policy, often represented as a decision tree, is useful for physical systems is supported by studies that have shown POMDPs to be effective for physical robots solving tasks in robot soccer [3], household robotics [4], coastal survey [5], and even nursing [6]. The wide variety of useful robotic implementations using these POMDP policies is a clear motivation for this thesis to develop techniques for performing planning for robotic systems under uncertain and imperfect information, and in particular, to increase the size and scope of the task that can be solved using a POMDP.

As an example, consider Figure 1.1. The task is navigation from start (purple) to goal (green); this can be encoded in a reward function as $+10$ at the goal, -10 at the region to avoid (red), and -1 (motion cost) elsewhere. Clearly, to maximize accumulated reward, the robot should get to the goal as quickly as possible and stay there forever. However, constructing a policy may be difficult if the actions of the robot can result in not just one state but any of a distribution of states. In the example shown, each action has up to three possible resulting states. Constructing a policy that guarantees the optimal action will be

taken in any state is difficult, because there exists a risk/reward tradeoff. Perhaps driving straight into the left wall over and over is a good policy to avoid the bad region? This could yield a very conservative policy that may avoid the bad region but still fail to maximize the total reward.

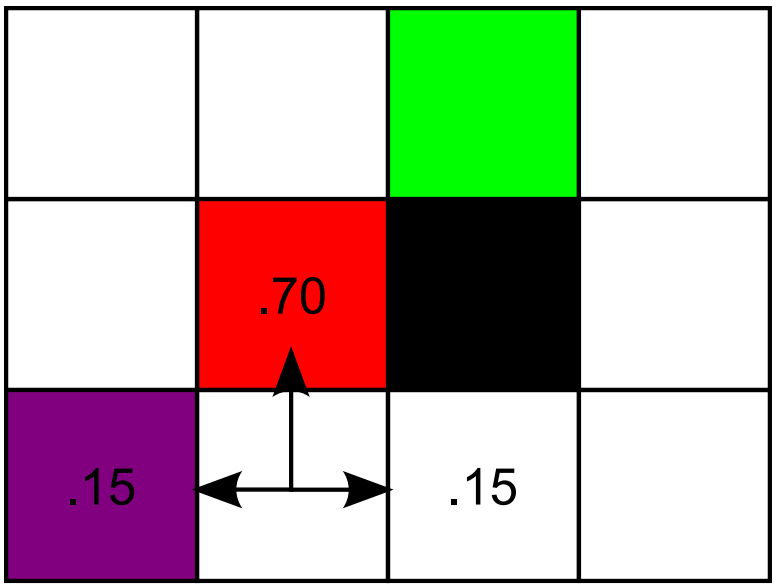


Figure 1.1 : A small example task in a grid world. The start state is the lower left purple region, the goal is the top green region and a high-penalty red state is in the middle next to an obstacle state in black. For each action, there is uncertainty, illustrated for one state and one action here – the robot actually transitions one grid cell up 70% of the time, and the other 30% is split between the two sides.

Further complexity is added because the robot does not know what state it is in, but can only make observations of some stochastic function of the current state (sensor data). Now the policy must be over all possible observations given a stochastic action function and this stochastic observation function. Computing a policy that can navigate through the propagation of distributions of states the robot may be in is far more complex than finding path to the goal for a single robot state without uncertainty. Finally, physical robots must contend with real-valued continuous states, not grid-worlds. We will be presenting techniques to improve solutions in both types of worlds.

Although the POMDP may be difficult to solve, the policy it produces is very easy to execute. This is a very important fact, as solving a POMDP is known to be a PSPACE-Complete problem (any PSPACE problem can be reduced to solving a POMDP) even to compute an approximately optimal solution [7], a complexity class vastly more difficult than the PTIME variant that is fully-observable, the Markov Decision Process (MDP). This complexity difference cannot be overstated: it is the difference between a fully-observable problem that is solvable in a fraction of a second and its partially-observable counterpart that is only approximately solvable in hours. An interesting note is that it has recently been proposed that PSPACE may be a tight upper bound on the computational potential of a biological system [8], so it is unsurprising that long-term optimal decision making with uncertainty is difficult for humans as well.

1.2 Problems Considered

1.2.1 Problems Solved using the POMDP Formulation

This thesis will present techniques to improve the solutions of three types of robotic tasks encoded as POMDPs:

1. navigation of a holonomic robot on a grid with noisy sensors and uncertain start state,
2. navigation of a car-like robot in a continuous space with noisy sensors, uncertain start state, and only partially-known map, and
3. navigation of a holonomic robot in a continuous space with noisy sensors and uncertain start state, in a particularly large environment where multiple tasks may need to be solved in the same workspace.

Additionally, the technique for item 2 is expanded into exposure minimization (get to the goal without being seen) and search-and-rescue, as evidence of the generality of the technique. As the tasks presented above are defined using the fairly general construction of POMDPs, it is reasonable to expect some level of generality of the techniques presented, but it is an open question to characterize exactly which problems the three POMDP abstraction techniques (presented below) can be applied to.

1.2.2 Problems Solved without using the POMDP Formulation

Finally, we will also look at two techniques to solve related tasks, each with a very focused definition of the uncertainty present that is specifically addressed. The tasks are less general than a POMDP, but are interesting applications of very specific instances of imperfect information.

1. Gather enough sensor data from an unknown target to verify a guessed class using a multi-robot team.

2. Opportunistic sensing of a secondary target while executing a primary navigation task.

1.3 Contributions

This thesis primarily focuses on improving our capability to solve tasks in a POMDP formulation. In this setting, only a very limited set of problems can be directly solved. This is due to the computational complexity that grows exponentially with the size of the problem in both space and planning time horizon (this time horizon is the maximum number of

actions to consider when computing the total reward). The core of this thesis is the proposal of three techniques to build abstractions of these problems, and demonstrating how a solution to these abstractions can be used to effectively address the input problem that could not be solved with previous techniques.

Chapters 3–7 will each focus on one specific technique to solve the problems described in Sections 1.2.1 and 1.2.2. First we will discuss the core of this thesis, the solutions to the POMDP tasks of Section 1.2.1:

1. The solution to the discrete navigation task is based on abstracting the action space. In Chapter 3, it is shown that the improvement in solution speed can allow us to solve larger problems than were possible before.

2. A more general technique for improving solution speed is developed in Chapter 4. The same general idea of abstraction is applied, but now to both the state and action spaces, and at a much coarser level. Not only can larger problem instances be solved effectively, but replanning is shown to be a useful technique to follow the coarse policy that is computed when the underlying continuous robot model is more constrained than the model in the POMDP.

3. The final POMDP technique layers a fully-observable MDP on top of many smaller POMDP instances. Compared to solving the single larger POMDP task, the two-layer approach can solve large problem instances of indoor navigation significantly faster. Additionally, the technique can reuse information to generate multi-query solutions for environments that have several different tasks. The direct POMDP solution of a single task is shown to take longer even for a single task than the multi-query method that solves all possible tasks at once.

Often, POMDPs are viewed as a theoretical construct with little applicability to real-world robotics due to the limited task size that can be solved. It is our hope that by building techniques to expand the size of the tasks that can be considered, this thesis represents a step towards making POMDPs practical for robotics. Because the policy is trivial to execute online, the potential applicability of POMDP solutions to low cost, ubiquitous robotics is worth exploring.

Overall, the solution techniques that will be presented represent a significant contribution to the problem size that can be considered within the POMDP framework. Not every robotic task is expected to benefit from the exact same abstraction technique. Our results support the general utility of abstraction, and we can at least say that the techniques presented are effective for the tasks presented as test cases. Solving larger POMDP instances is a necessary step towards expanding the type and complexity of task that can be considered in physical robots with POMDP policies.

The additional tasks we have presented in Section 1.2.2 also have their own unique solutions that do not use the POMDP formulation.

1. The first technique will utilize a previously-developed multi-robot coordination and replanning framework and show that it can be used to solve an image manifold target verification task for a specific sensing model. Previously, this framework operated wholly independently from sensing.

2. The second technique introduces a secondary sensing task in addition to a primary navigation objective. When the secondary sensing task is discovered, the robot automatically deviates from the primary objective (navigation), providing a framework for partially-known tasks that becomes well-specified during online execution.

These two additional solutions were effective at solving the task that was specified for them, however, these solutions are not expected to generalize to other tasks as the POMDP formulated techniques may.

Chapter 2

Background

This chapter will provide some necessary background material that will be used throughout the remainder of the thesis. First, the Markov Decision Process (MDP) will be formally introduced. The structure of the MDP will then be extended into a formal description and discussion of the Partially-Observable Markov Decision Process (POMDP) that is a fundamental building block of this thesis. Last, a brief description of robotic motion planning is provided, which will be used as a tool to run some of the online executions.

2.1 MDPs

If the current and all future states can be observed perfectly, then a construction known as the Markov Decision Process (MDP) can represent reward-maximization tasks with uncertain actuation [2]. An MDP is defined as $M = \langle S, A, T, R \rangle$, where S is the state space, A is the action space, T is the set of transition probability functions describing the probability of an action $a \in A$ updating the current state $s \in S$ to a new state $T_a(s, s')$, and R is the reward function, defined for each action over all possible state transitions as $R_a(s, s')$.

The MDP formulation is not an instance of path planning because the transitions are probabilistic. Following from the probabilistic transitions, a solution to a task defined as an MDP is necessarily a *policy*, not a path. An MDP policy is a mapping from a state to an action: $\pi(s) = a$. Any particular path is only a possible execution of a policy. The

solution of an MDP is the optimal policy, π^*, that maximizes the expected reward over all executions.

Solving an MDP to produce π^* has a known solution in polynomial time utilizing dynamic programming. Generally, value iteration [9] is performed to compute the value function V over all states, in the form

$$V(s) = \max_a \sum_{s'} T_a(s, s') \left[R_a(s, s') + \gamma V(s') \right] \tag{2.1}$$

where γ is a discount factor between 0 and 1 that causes the value of reward to be higher if it can be achieved sooner rather than later (if less than 1). This is referred to as future discounted reward, or just discounted reward.

2.2 POMDPs

In many robotic tasks, sensing may be more important than the action uncertainty, or both may be equally important. Considering the sensor noise means that the robot cannot perfectly observe the current state as we assumed in the MDP. Partial observability means that the agent does not know the current state, but instead gets an observation that is some (stochastic) function of the state. Let O be the set of possible observations, and $\Omega_a(o, s) = \Pr(o|s, a)$. To solve tasks under partial observability requires maintaining a posterior distribution over the possible states of the world, given all observation data accumulated so far and all actions performed. This posterior distribution is formally represented as a *belief* of the system, as opposed to a state. A belief (sometimes called a belief point) can be defined as a probability density function over S. If S is finite, then we can represent elements of B as vectors in $mathbbR^N$ where $N = |S|$ and each of the N elements has a

value in the unit interval $\mathbf{I} = [0 \dots 1]$. This representation is, intuitively, the probability of being in every state.

Clearly, there is a problem if the state of the system is itself in $\mathbb{R}^N$ where the cardinality of the state space is the continuum $2^{\aleph_0}$. In this case, the belief space is an infinite dimensional unit interval $\mathbf{I}^{2^{\aleph_0}}$, known in topology literature as a Tikhonov cube [10]. This prevents us from even representing a single point, unless there is some significant structure of the belief space that can be taken advantage of. However, if we assume that time, A, and O are all discrete and finite, it is easy to demonstrate that there are only finitely many *reachable* belief points. This follows from the fact that the current belief b_t can only depend on the information that the robot has at time t. Let $b_t = f(a_t, o_t, b_{t-1})$, that is, the current belief is some function of the current action, the current observation, and the previous belief. This is a valid definition because an ordering of the actions performed and the observations obtained are a complete representation of the information known to the robot. Given the finiteness and discreteness assumption on T, any belief after bounded time t can be written by performing the recursive expansion finitely many times. In other words, we can be assured that if we write a belief as $b_t = f(a_t, o_t, a_{t-1}, o_{t-1}, \cdots, a_0, o_0)$ there are only a finite number of terms. Further, if A and O are discrete and finite, than there are only a finite number of possible sequences of action and observation of length T. If we naturally call this sequence of actions and observations a *history*, then we can rephrase this result as the finiteness of the number of reachable belief points following from the bounded number of possible histories that can be encountered in finite time. In fact, many of the techniques to solve these problems make exactly these assumptions.

At this point, we now formally define the Partially-Observable Markov Decision Process, or POMDP [11]. A POMDP is defined as $P = \langle S, A, O, T, \Omega, R \rangle$. Each element is either defined in the MDP discussion, or has just been defined in the discussion of partial

observability. Solving a POMDP also means computing an optimal policy. Because the state is not known to the robot, unlike in the MDP discussion, a POMDP policy cannot be defined directly on the current state. Instead, the solution is $a_{t+1} = \pi^*(b_t)$, or, equivalently, $a_{t+1} = \pi^*(a_t, o_t, a_{t-1}, o_{t-1}, \cdots, a_0, o_0)$. The second definition is preferred because it helps make clear one reason why POMDPs are a useful tool. The policy only depends on the execution history, and, if the policy can maintain the current history up to time $t-1$, then the next action to execute only depends on the previous observation and the explicit computation of the current belief is actually unnecessary during execution of a policy. To see why this is the case, consider a policy as a discrete finite automaton (DFA), where the states of this DFA are the possible bounded-time histories. Every state also can be labeled with the action that should be taken if that history were to take place, computed before execution begins. Transitions between states are therefore exactly the set of possible observations that could take place given the action with which the current node is labeled.

Given that a POMDP is defined as the collection of generic sets described above, many tasks can be defined as maximizing the expected accumulated discounted total reward. That is, given a terminal history (either due to a terminal state or a finite time horizon) and a stochastic start state, find the optimal policy defined over P that selects actions to maximize the expected value over the distribution of start states. Again, this can be solved through value iteration [2]. If the value function has converged to the optimal value everywhere, then the optimal policy must be to follow the action with maximum value from each belief (recall that a belief b is a probability distribution over states).

$$V(b) = \max_a \ \mathrm{E}\left\langle \sum_{s'} T_a(s, s') \left[R_a(s, s') + \gamma V(s') \right] \right\rangle \tag{2.2}$$

This equation shows us two key characteristics of the value iteration computation. First, it is fundamentally the same as the MDP value iteration, except now the computation re-

quired for a single update is, for every belief, an expectation value over both current state and next state, rather than just the next state. Our first hint at just how bad the computational challenge is is that there are exponentially many more beliefs than states. The value function is defined over beliefs $B = [0 \dots 1]^{|S|}$, and the expectation is over $S \times S$ (current and next states) instead of just S now.

To further illustrate the computational complexity, consider that an MDP policy is defined over the entire state space, so any deviation from the expected path is acceptable. At each state, the value of reaching any particular state in the future can be used as a partial solution, so dynamic programming can be applied. However, in a POMDP, the current state is not known, therefore the policy now needs to be defined over all possible *distributions* of states (beliefs). The belief (analogous to the state in an MDP) will change with every observation and action. Because these beliefs are potentially (and highly likely) different for every possible (stochastic) action and (stochastic) observation, a partial solution for one belief does not inform the value of any future distribution. This illustration gives the flavor of why the complexity increases so much; the policy to solve a task specified as a POMDP must take into account the conditional probabilities for all future beliefs as they evolve through time, given any possible history of actions and observations.

This turns even a seemingly simple task into an extremely computationally intense challenge. The most recent, and very promising, avenue of research into constructing a solver that can handle these challenges has yielded the iterative, belief point-based POMDP solver. The simplification that is made to trade computation time for approximation by not considering the whole belief space. Rather, these iterative belief point based methods ([12, 13, 14, 15, 16, 17, 18, 19, 20, 21], among others), consider only a small set of belief points. At each iteration (starting from the initial belief), one belief is added to the set of current beliefs. The selection of this belief is a major factor in the performance of the

algorithm, and there are many heuristics to guide this selection. Intuitively, most of these heuristics revolve around selecting beliefs that are expected to improve the estimate of the value function through an upper and/or lower bound analysis. Furthermore, perhaps best explained in [18], the belief space may be huge, but the reachable set of beliefs is much smaller, and the reachable set of beliefs under optimal action may be smaller yet. This lends credence to the idea that a very small subset of belief points may be an acceptable approximation of the belief space for many POMDP tasks. Additionally, because the finite horizon, discrete action and observation space POMDP is known to have a piecewise-linear convex value function, it can be shown under fairly general conditions that the point-based iteration converges to optimal as the number of iterations goes to infinity [16].

Robotic planning has advanced considerably, but accounting for uncertainty during planning is still very challenging [22]. Even with these advances in POMDP solution techniques, many robotic tasks, such as grasping [23], navigation [24] and exploration [15] remain difficult for large problem instances. The point-based iterative methods are anytime, so even if an optimal solution is not found, a suboptimal "solution so far" is returned. We will focus not on tasks that can be solved optimally using these methods, but instead look at tasks that push the boundaries of what is solvable. The hypothesis is that, with the appropriate abstraction, useful robotic tasks can be solved even when the POMDP representing the unabstracted world in more detail is not even approximately solvable. Many robotic tasks require hours of computation to find an optimal policy even for relatively small discrete problems with just 100 discrete locations and less than 10 discrete actions that the robot can take.

The state of the art problems presented as benchmarks in various solvers varies considerably. Some tasks involve a reward function such that most solvers never visit most states

in constructing an optimal policy (e.g., RockSample [25, 26, 15, 14, 17, 27]), or degenerate into fully-observable problems at some point (e.g., Underwater [18, 20]). The actual solution time of POMDPs is in general extremely hard to predict. Many of the existing benchmark tasks seemed to be somewhat tuned to the particular solver at hand, and no results were presented for problems that were currently found to be unsolvable. Solution times for grid-based worlds even as small as 10×10 were found to take anywhere from seconds to hours to find even an optimal policy – depending on the sensing and action uncertainty present. Given the success mentioned earlier in using POMDPs for physical robots performing tasks from robot soccer [3] to nursing [6], it is important to carefully define the task in a way such that it will fit in the constraints of what a modern PODMP solver can handle.

Some problems can be addressed relatively quickly if the belief of the robot is guaranteed to always be Gaussian [28, 29, 30], however, this does not work well when belief needs to consider a categorical state (a discrete state that does not have an ordering). For example, if part of the state space is a selection index describing if a particular obstacle is present from a set of possible obstacles, then a Gaussian distribution around a particular state may simply not make sense – a high probability of obstacle 3 being present does not mean that there is more of a chance for obstacle 2 to be present than obstacle 7. A Gaussian belief enforces this "nearness" that only makes sense for some types of state variables. This thesis will use MCVI [20] to solve POMDPs, which does not make the assumption that any belief about the current state of the robot can be accurately represented as a Gaussian distribution.

A further consideration is what to do if the model does not capture all the belief dynamics and an unexpected observation is seen. Even in pure simulation, the policy may simply not include enough belief points and the same problem can occur. This problem has been considered in the literature through online POMDP methods (e.g., [31, 25, 27]),

online policy improvement [32], and Gaussian belief approximations [29]. One alternative is to use reinforcement learning [33, 34, 35, 36, 37] rather than building a policy offline. Although incorporating at least some of these online policy update methods can be useful for any realistic physical system, we will be focus instead on the effect of the abstraction that we are proposing in isolation from these policy update methods.

2.3 Motion Planning

In Chapters 5, 6, and 7, we will assume the existence of a general path planning system for online execution of plans. This path planning system can solve the path planning problem: find a valid trajectory that connects a start state and a goal state. This problem definition does not include any uncertainty in the state of the robot or the execution of the actions.

The definition of what constitutes a valid state is the fundamental building block for path planning systems, and can incorporate information about the dynamics of the robot, any possible self-collisions between parts of the robot, as well as any obstacles in the surrounding environment. This system has many instances in robotic motion planning literature. In general, they are interchangeable as far as the methods presented in this thesis are concerned, as they are usually used as a "black box," where only the input (start, goal, validity checker) and output (valid trajectory from start to goal) is important.

These motion planning methods must have some basic properties in order to fit the requirements of our particular black box, however.

- They must not produce a path that enters any state that is defined to be a terminal collision,

- they must be generalizable across several motion models, as we wish to avoid effects of changing planners when the motion model is changed,
- they must eventually exit local minimia, if given a cost model,
- and they must be relatively cheap computationally, as they are often used online.

In general, it is hard to find optimal solutions for motion planning problems [38, 39], but many practical approaches have been proposed. Most relax guarantees on optimality and provide weaker notions of completeness. With perfect knowledge, sampling-based algorithms [39] can provide probabilistic or resolution completeness but cannot verify that a solution does not exist. Since their introduction [40, 41], these algorithms have been successfully used for decentralized replanning for dynamic systems [42, 43, 44]. Such algorithms have also been combined with cost maps to find paths that minimize some cost function over paths [45]. In [46] it was shown that a rich set of precomputed maneuvers for a car combined with cost maps was an effective approach in the DARPA Urban Challenge. This thesis utilizes a sampling-based planning framework [47] in chapters 4, 6, and 7; it poses few constraints on the underlying dynamics and includes implementations that can effectively bias the search for dynamically feasible paths using a cost function [48], particularly relevant in Chapter 4.

There are many other techniques that could be applied. This thesis does not focus on their differences; for an overview of the techniques available, there are several robotic motion planning books published recently (two of which are [39, 49]) that discuss sampling-based algorithms as well as other methods e.g., navigation functions, discrete search, and compositions of controllers.

Fundamentally, these methods all share a common assumption of perfect knowledge. If the robot is in a state, a path to the goal is computed, and then the robot executes the

path. Although sampling-based planning is not natively set up to produce policies across the entire state space, it has been shown to be quite effective to re-plan online to account for unexpected events [50, 51, 52, 53].

Chapter 3

POMDP Action Space Abstractions

3.1 Motivation

As discussed in Chapter 2, solving POMDPs poses a significant computational challenge. The policies they produce, however, are extremely useful in the robotics domain. Specifically, they encode an optimal solution, and can be computed off-line. The on-line computation then is a simple sequence of calls to a look-up table. Therefore, there is significant interest in improving the maximum problem size that can be considered with these methods. This chapter presents one such method, first published in [54], called Automated Model Approximation, or AMA.

3.2 Problem Definition

Given a POMDP model M as input, a POMDP solver will compute a globally optimal policy with respect to total reward that is achieved. The POMDP solver will reason in the belief space B, which has dimension equal to the size of the state space. A point $b \in B$ encodes the probability of being in each state. Thus for even a 10×10 discrete 2D grid of states, the space that must be reasoned over is of dimension 100. AMA will work specifically with these discrete state and action spaces. State-of-the-art POMDP solvers (see Chapter 2) take hours to solve even these problems. Upon execution, AMA builds

a problem-specific approximation of the state and/or action spaces. This delays the curse of dimensionality, while attempting to maintain an approximation that can build a near-optimal policy. That is, using AMA, we generate an approximated model $\hat{M}$ that, when solved, will produce a policy that also solves, approximately, the original task definition M. The output of AMA is this problem-specific approximate model.

To reduce complexity of the discussion and implementation in this chapter, we assume that only the *action model*, defined as the state and action space product, will be modified by AMA. Other simplifications will be discussed in later chapters. The action model is defined this way because it represents the selection of actions allowed in any particular region of the state space. Therefore, action models will comprise the primary input and the output of AMA.

The user will define the true action space, a simplified version of the action space, and an operator to augment the simple action model with the true complexity in specific regions of the state space. These three items implicitly define three action models: the simplified action space everywhere, the true action space everywhere, and the output of AMA, which is a mix of the prior two. For example, a simple action model may be moving with speed 1 in any direction, and the true action model also allows moving with speed 2. Then, AMA will build an action model where speed 2 is allowed in useful directions in regions of the state space that are probably going to be entered, but in all other areas only allows a speed of 1. By reducing the number of options globally available, it decreases the computational burden, and by using all options where needed, we allow the construction of near optimal policies for a given problem instance.

3.3 Related Work

In this chapter, we utilize an existing point-based method to solve POMDPs, MCVI [20], to evaluate the action model produced using AMA.

AMA builds an approximate action model that is used to speed up POMDP computation. In doing so, it also builds an initial guess for a policy. This initial guess is used to bootstrap solving the AMA generated POMDP problem instance. Although this seems similar to applying a heuristic to the underlying POMDP solver, it only initializes the guess of the lower bound of expected reward better, and does not affect the sampling strategy of the POMDP solver. Point-based solvers generally compute upper and lower bound estimates to establish convergence and inform some types of heuristics. Many heuristics [14, 13, 55, 21, 56] have been successfully applied to various POMDP problem instances, however, we only utilize the extremely general sampling heuristic built in to MCVI. This avoids interaction between a selected sampling heuristic and AMA's action model computation. There is no reason to think that utilizing one of these sampling heuristics to improve solution speed would be problematic, but it would complicate the analysis of results. This is an open avenue for future research.

Hierarchical POMDP methods [57, 58, 6] are related to AMA in that they focus on building and solving multiple POMDP models that in turn solve a global problem definition. However, these methods are constructing decompositions of the action and state spaces. AMA, on the other hand, focuses on pruning and removing parts of the action cross state space to construct a useful approximation, rather than decomposing it. Thus, AMA could be incorporated with hierarchal methods.

Because the computational complexity of solving a POMDP problem instance grows exponentially with the size of the state space, some recent methods try to group similar

states and actions together to maintain high simulation fidelity where needed [30, 59]. This work is similar in concept to our proposed method, but tends to require a strong set of assumptions and/or user-defined mapping functions to create these groups. In contrast, AMA requires definition of action models and an action model refinement function as opposed to mapping functions or the existence of stabilizing feedback controllers. Also, AMA does not attempt to provide an optimal approximation over the whole space, but instead only focuses on regions found to be important for a specific problem instance, determined from an initial coarse abstraction. The ideas of state and observation grouping are used to extend a POMDP solver to continuous spaces [59]. This modification to the underlying POMDP solver is generally orthogonal to the thrust of AMA, where we alter the cardinality of the spaces considered, depending on the results of solving a low accuracy model. Integrating these ideas could provide AMA extra information for model building, based on the groups constructed in the low accuracy solution.

Probably the closest idea to AMA is found in [60], which builds a decomposition of the state space with varying resolution depending on the environment features. However, AMA uses knowledge from an approximate solution to the specific problem instance, rather than only the environment specification. This means that AMA will retain a simple action model even where the environment is complex if those regions are not useful in the current problem instance. This is in contrast to a variable resolution method based on building quadtrees over the entire space, where resolution depends on environment complexity only. Additionally, we focus on the action model, while [60] focused on the state space.

The work on policy transformation under changing models [61], called Point-Based Policy Transformation (PBPT) is extremely relevant although orthogonal to our approach. Specifically, AMA focuses on building an action model with the minimum complexity required, while PBPT focuses on making minimal modifications to a policy to fit a new

model. AMA does not use such a sophisticated policy transformation, instead bootstrapping the new approximate solution with information about how the original, simple instance was solved. This simpler method requires fewer assumptions on the differences between the simple action model, the approximation of the true action model, and the true action model. In particular, PBPT requires a bijection between the (discrete) state, action and observation spaces between the two models under consideration, and our approach can not meet this requirement because we are explicitly changing the cardinality of these spaces between the three action models.

3.4 AMA Framework

We will denote the true action model that defines the problem as M. Then let M^- be the simplified version of that model. The function $\hat{M} = \text{Refine}(M^-, R)$ is defined by the user and modifies the action model of M^- by adding new options in the product of state and action spaces, selected by analyzing the set of states R. That is, if an action model M^- and set of states R are passed into Refine, it shall return a new action model $\hat{M}$ with additional complexity added. The analysis of R determines what actions are made available in what states. For example, if the simple model M^- only uses the 4 cardinal directions, and R includes states that turn a corner, then diagonal actions to short-cut that corner can be added. In general, if the states are nodes in a graph connected by action edges, then more complex models can be built with actions that, when added, will create shortcut paths in this graph. If all possible refinements are made to the simple action model M^-, the true action model M should result. Thus $\hat{M}$ is an action model that has at least as much complexity as M^-, but no more complexity than M, and uses the information from the solution of the POMDP problem instance under M^- and the function Refine to increase action model complexity in only the relevant areas.

Algorithm 3.4.1 Automated model approximation algorithm

1: $\hat{M} \leftarrow M^{-}$
2: **for** $i = 1 \rightarrow iterations$ **do**
3: $\quad policy \leftarrow \text{Solve}(\hat{M})$
4: $\quad states \leftarrow \text{Execute}(\hat{M}, policy)$
5: $\quad \hat{M} \leftarrow \text{Refine}(\hat{M}, states)$
6: **end for**
7: **Return** $\hat{M}$

In Line 3 of Algorithm 3.4.1, Solve should call any POMDP solver, and return a policy that maximizes expected reward, as described in Section 3.2. On Line 4, *states* is the set of ordered lists of true world states that the robot entered while executing *policy*. This is a set of lists because the simulation must account for all possible true initial conditions with the correct probability distribution. To accomplish this, k samples are drawn from the initial belief and executed.

Our implementation of AMA then operates in an iterative fashion as described in Algorithm 3.4.1. We found this to be an intuitive way to pass information to Refine, although in general, any function of the current robot model and policy could work here. Then $\hat{M}$ is updated with the information that this sequence of states provides by the call to Refine on Line 5. The exact form of the Refine function will differ between various POMDP models. Our implementation can be found in Section 3.8.

To implement AMA, we chose to work with a state-of-the-art POMDP solver known as MCVI [20]. The authors of MCVI provide an implementation of their method for download, and new problems are defined by writing C++ code. This enabled AMA implementation by separating out our AMA code into the action model implementation, and required less modification to the underlying code than may be the case with other POMDP solvers. MCVI is an iterative method, so it constructs a series of policies. The final policy will have

the highest lower bound and lowest upper bound of expected reward. MCVI exits when upper bound − lower bound ≤ convergence criterion. This criterion is 1 for all simulations in this chapter.

In MCVI, the lower bound is found by particle filter simulation of the model applying a policy. Thus the actual reward of the policy may be lower than this lower bound, depending on the exact samples that get drawn in the particle filter. We will continue to use the term lower bound even though it has a stochastic nature and may not be a strict lower bound for all samplings. Similar arguments apply for the upper bound, although there is less variance here so it tends to have a smaller effect.

3.5 Problem Instances

In this section, we will define the specific tasks that are formulated as POMDPs and used by AMA and MCVI. The robot definition that we will concentrate on is a point robot living in a grid discretization of a 2D world. The robot state space thus is a set of square grid cells, and the action space is defined as noise free movement between cells. In this case, a clear choice for iterative refinement is over which diagonal edges are available. Our simple action model, M^-, can always move in the four cardinal directions, but not on diagonal edges. The true action model, M, is a model that can always move in the four cardinal directions as well as the four diagonal directions. Therefore, M^- should be able to solve all problems presented but the optimal path length could be larger by up to a factor of $\sqrt{2}$, as long as the environment does not include diagonal zero width corridors. We only present results in environments that fit this requirement. The Refine function can add diagonal actions where needed based on a set of execution traces (Algorithm 3.4.1), and is

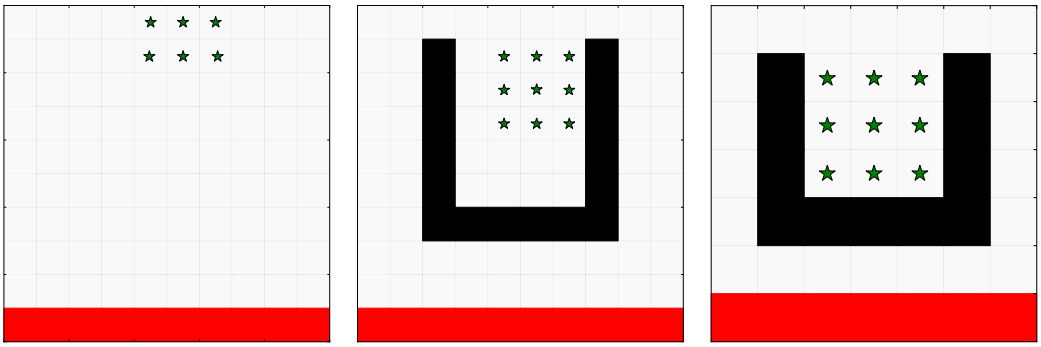
used in AMA to create $\hat{M}$ from M^{-}. The full complexity action model, M, is the baseline for comparison.

A POMDP model must provide costs or rewards for all actions possible. Moving in a cardinal direction and sensing always has cost 1, while moving diagonally has a cost $\sqrt{2}$. Sensing while in a goal region is the success condition for our task, so in this case the robot gets a large reward. MCVI uses a discount factor, and we leave it at the default value of $\gamma = .95$. The robot gets a null observation for every action except for the sense action. In this case, a two dimensional observation is provided that is sampled from a Gaussian around the current true state of the robot with variance three. The initial position of the robot is defined as a uniform sampling of a problem specified state and all adjacent non-obstacle states. This uncertainty in initial position drives the challenge of solving the problem, and the uncertainty in sensing prevents the system from collapsing belief to a single state. There is no penalty for attempting motion into an obstacle other than the lost time (discount of future reward) and the cost above.

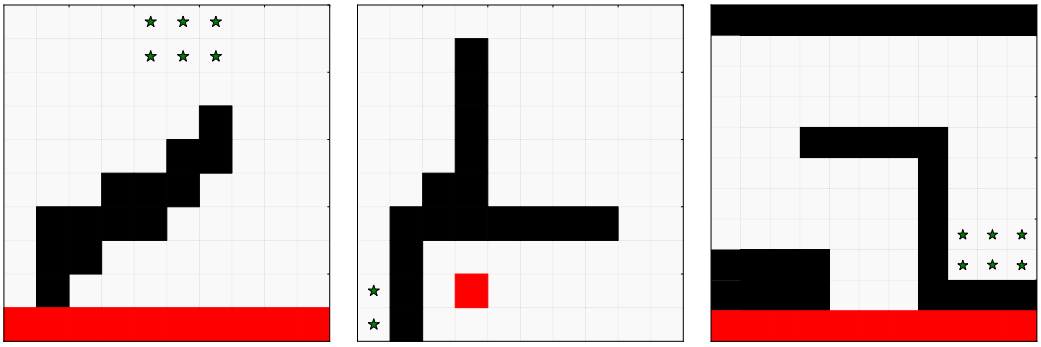
3.6 Environments

AMA is fairly general in definition, but to evaluate the performance of the approximate robot models it constructs, we will define several concrete instances of problems to solve, as depicted in Figure 3.1.

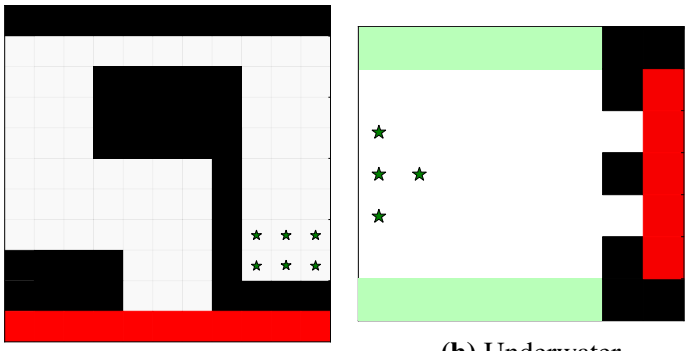
(a) A reasonable starting point is an empty environment, where the goal is simply to navigate from a point in the north to anywhere in the south. This is a useful example because it should not require diagonal actions to be optimal, it should run quickly, and the optimal policy should follow our intuition. Therefore we present it as a baseline



(a) Empty **(b)** U **(c)** Small U



(d) Diagonal **(e)** Spikes **(f)** Maze



(g) Constrained Maze **(h)** Underwater

Figure 3.1 : Environments used for evaluation, where the black regions are hard obstacles, stars indicate an element of initial belief, and the red area is the goal region. The final environment is for the underwater robot model, and has the goal region to the right while the green regions at top and bottom correspond to areas that allow localization.

for comparison. Solving $\hat{M}$ should be fast and not take much time beyond the time to solve M^-, because the policy from M^- should have been near optimal already. So this environment should provide the best possible advantage to our method.

(b,c) In the U environment, the robot starts inside a U shaped obstacle. In this case, some diagonal actions are needed for an optimal policy. We expect to see improvement with $\hat{M}$ as compared to M, although not as much as the empty environment. We also test a smaller, more constrained version of this environment.

(d) The Diagonal environment has a diagonal line cutting from the southwest up to just beyond the center. This is expected to be the hardest environment to solve because not only are there two very different navigation paths to follow, but the choice needs to be made soon without spending too much time sensing.

(e) The Spikes environment has three narrow passages. This environment should solve quickly, because the narrow passages actually allow the robot to reduce the uncertainty in position. In addition, the starting location is fairly constrained, so the initial belief does not have as much spread as the other environments. The spikes are along the X and Y axes, so optimal navigation requires many diagonal actions.

(f,g) The Maze environments require navigating four turns. The difference between the two mazes is that one is always as wide as the initial uncertainty, while the second has one side as a narrow passage that should reduce the uncertainty part way through execution.

(h) The final environment is for the underwater robot model in the MCVI package as found in [20]. In this model, the robot can only localize in the area on top and bottom (shaded lightly), while the goal region is to the right. A major difference in the model is that the true action space of M is smaller. Only five movement directions

are available: north, northeast, east, southeast, and south. Therefore, the difference in action model complexity is significantly less than in our navigation model.

3.7 Evaluation Strategy

To investigate the merit of this approximation, we will focus on the robotic navigation problem under state and sensing uncertainty. The initial state of the robot is unknown, but is instead a belief spread uniformly across a set of states. Sensing uncertainty prevents the robot from ever collapsing this belief to a single state except in highly-constrained environments. The robot navigation task we will focus on is to take a sensor measurement while in any one of several goal states. Sensing is defined as its own action and therefore takes time, so an optimal policy should avoid sensing except when necessary. Finding the optimal balance between motion and sensing is very computationally expensive in the presence of uncertainty. To evaluate if AMA was successful in generating a useful approximate model, we give all three action models (simple, true, AMA) to a POMDP solver with the same task, and evaluate the utility of the action models based on the policies that the solver constructs.

We evaluate the performance benefit of AMA and discuss the relative merits of the action model approximation constructed by AMA as compared to the true action model. The use of the action model provided by AMA is shown to generally, although not always, improve the POMDP solution time while not dramatically impacting total expected reward. Simulations are performed over a range of environments to verify the scope of our results. Finally, we apply our method to an example system presented in the software library we used, and our results show the same trend of faster solution without quality decrease.

3.8 Iterative Refinement

In this section we will describe our implementation of Refine(M^-, R) that is used in Algorithm 3.4.1 to construct $\hat{M}$. The first step is to find a solution policy with the simplest possible general action model, M^-. The simple action model, M^-, when passed to MCVI, should be able to solve any problem instance, although it need not be able to create an optimal policy with respect to the true model M. Because the policy generated from solving M^- is only used to initialize the next step, we define convergence as the upper and lower bounds being within 10 of each other, rather than the (arbitrary, user defined) true convergence criterion of 1 specified earlier, used in M and $\hat{M}$. Once this policy is computed, it is simulated k times (our implementation uses $k = 1000$), and each execution trace is recorded in order to call the Refine function as described in Algorithm 3.4.1. The value of k is arbitrary, and simply must be large enough to capture all likely execution traces of the policy that was computed using M^-. This is so that the approximate model $\hat{M}$ will be constructed with increased model complexity in all the relevant states. If too few executions are run, then the algorithm will not fail but simply not provide the best possible refinement. Because solving POMDPs is very slow and executing them is very fast, using a large overapproximation of the minimum k is recommended to avoid potentially missing states that should have been refined.

Furthermore, because the POMDP solution under the action model M^- tends to yield policies that are optimal up to short-cutting corners, each state in the trace is checked to see if a sequence of diagonal actions can connect to a future state in the trace. If so, then $\hat{M}$ will have that sequence of diagonal actions added to it. This is a problem-specific optimization, but is fairly general in concept – if a sequence of actions connects two states with the correct time ordering, then the entire sequence of actions should be added. As an

implementation detail, the center of the start distribution always has all 4 diagonal actions added to ensure that our initial belief is identical between $\hat{M}$ and M, because it is initialized based on adjacency of regions.

Adding actions based on short-cuts tends to add exactly the edges that might be useful in the next iteration, but does not add extraneous edges that will increase computation time. In addition, the transitions are combined to create an initialization map and a new initial policy. This initial policy follows whatever transition was most likely from a given state in the previous iteration. In this way, the work done in the simpler action model can be applied in the more complex action model easily, while searching for improvements using the new diagonal actions available in $\hat{M}$. After processing all execution traces to create $\hat{M}$, it is then passed into the MCVI framework as a new problem instance, and it is run until convergence or timeout, with identical parameters to M. Our hypothesis is that a problem-specific action model $\hat{M}$ will allow us to find a policy that is almost as good as the policy that the true action model provides, but in significantly less time. This iterative refinement could be applied multiple times, but in practice we found that AMA worked well with just one refinement step. Therefore there is no ambiguity in the use of M^{-}, $\hat{M}$, and M to describe the action models.

Each environment was run in MCVI using our iterative refinement model on a single core to avoid timing discrepancies with varying levels of parallelism. All experiments were repeated 60 times to obtain statistically significant means to compare. All error bars represent 95% confidence intervals. The time limit for solving M^{-} and $\hat{M}$ was 10,000 seconds, and the time limit for M was 20,000 seconds. Time taken to perform the refinement step is less than one second and is neglected here for clarity of plots. This time grows asymptotically linearly in the number of execution traces considered and the planning horizon, so it is not expected to ever be a significant influence on the total runtime.

All experimental results were executed on identical 12-core machines with 48 gigabytes of RAM. Although MCVI supports local parallel computation, this was disabled. Some models may have greater parallelism, so looking at the solve-time would not be representative of the actual CPU effort. In the implementation of MCVI, in fact, changing the number of actions generally does strongly correlate with changing parallelism.

3.9 Experimental Evaluation

MCVI was able to converge to a correct solution in our test of the empty environment (Figure 3.1(a)), as our performance data shows in Figure 3.2. As expected, M^- followed by $\hat{M}$ did very well, taking about 2500 seconds less than the time to solve M – more than 60 times faster! As should be expected, increasing the size of the action space when the actions are not needed makes a large difference in computational power required. We see that the difference in time to solve M^- and $\hat{M}$ is much smaller than the difference of time between M^- and M due to additional actions available in M.

When the U shaped obstacle (Figure 3.1(b)) was tested, M and the hybrid model $\hat{M}$ did not converge within the time limits, although the simpler M^- did. Therefore it is somewhat uninteresting to compare run-times, as they are simply pegged at timeout for everything except M^-. Thus we present a plot of the reward in Figure 3.3 of the best policy that the three cases were able to find. The results here are somewhat surprising – the solution using M did not return any policy that got a positive reward at all, while using the simpler M^- could, and then using $\hat{M}$ was able to improve on the expected reward. Therefore, we see a significant benefit for AMA. It is apparent that solving with $\hat{M}$ is able to provide partial answers quickly. In this case, using M failed to provide a reasonable solution at all. Because of the difficulty in solving this environment, we constructed a smaller version of it with about

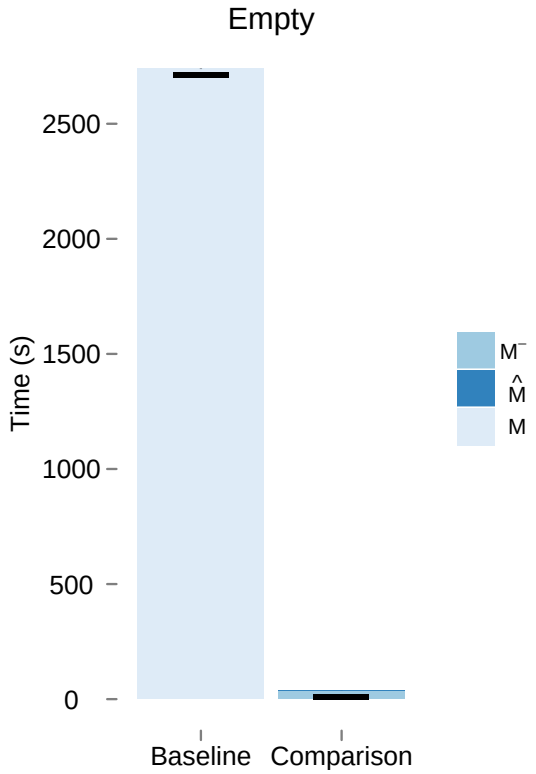


Figure 3.2 : In the empty environment, our approximate action model is more than an order of magnitude faster. The baseline is the true robot model M, while the comparison is the time to solve the model AMA built, $\hat{M}$, stacked on top of the time to solve the simple action model (M^{-}) that is used by AMA as a starting point. Note that the time for $\hat{M}$ is so small as to be invisible on this plot.

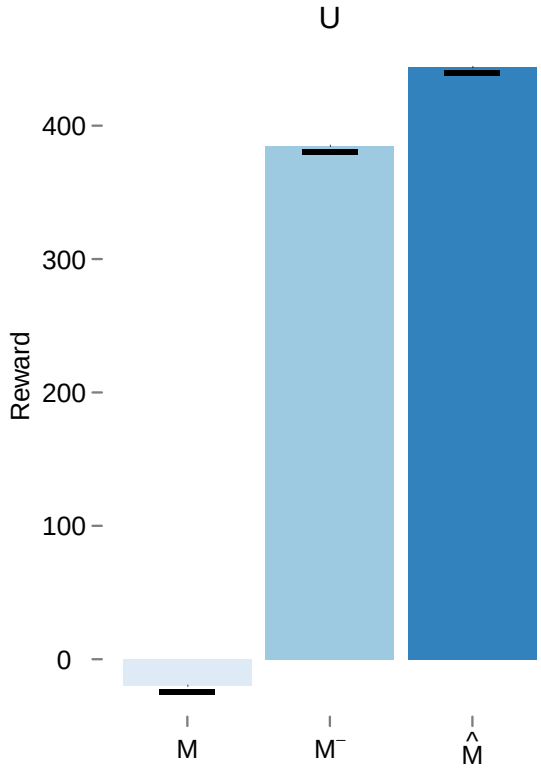


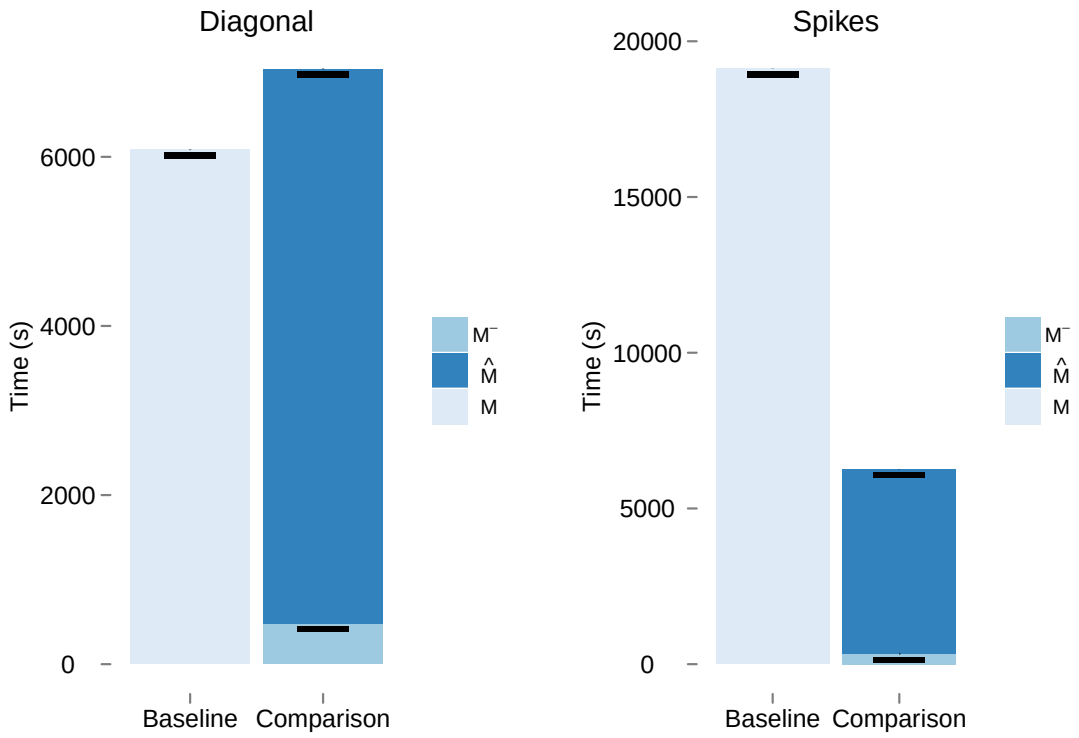
Figure 3.3 : In the U environment the use of our action model approximation method yielded an approximate solution, while the full complexity action model was not able to achieve any positive reward in 20,000 seconds.

half the number of discrete states. However, the results were qualitatively identical: the full complexity action model M still did not yield a policy that got positive reward within 20,000 seconds in any of the 60 runs. Although running MCVI longer may have produced a better solution, it seems clear that there will tend to exist problems that cannot be solved within some given timeframe that have good solutions in our abstraction. We note that this is presenting one of two fair comparisons – comparing the final expected reward given equal time was chosen over running M until equal expected reward was attained due to the expected extremely long runtimes required.

The environment with a diagonal line in it (see Figure 3.1(d)) was found to be able to converge in all cases. In this case, we see that the time of solving M actually beats the time to solve using M^- and $\hat{M}$ combined, although not by a large margin, as shown in Figure 3.4(a). This fits our expectations, because M^- cannot provide as much information to the AMA, and there are many diagonal actions needed for an optimal policy. If we also look at the reward of the three methods in Figure 3.6(a), we can see again that using $\hat{M}$ gets a value very close to when we use M, but clearly was missing out on some actions that were needed to be optimal. However, the absolute difference in reward is small.

The fourth environment with three spikes to navigate, depicted in Figure 3.1(e), also converged to a solution in all cases. The total time to solve, shown in Figure 3.4(b), is significantly less when using AMA. However, based on the number of diagonal actions required, it is reasonable to think that there will be degradation in the reward that $\hat{M}$ was able to achieve. The results presented in Figure 3.6(b) indicate that this is exactly what happens, a significant decrease in runtime is paid for by a smaller percentage decrease in reward.

The two maze environments of Figures 3.1(f), 3.1(g) had varying results. As before, we will compare the reward that their partial execution could achieve, noting that in the



(a) L environment

(b) Spikes environment

Figure 3.4 : The diagonal environment of Figure 3.1(d) caused our approximate models to use slightly more runtime, shown in (a). The spikes environment depicted in Figure 3.1(e) barely completed on time, and AMA provided a significant runtime advantage, shown in (b). Here, the time for solving M^{-} is hard to see because it is so small at the bottom of the comparison column.

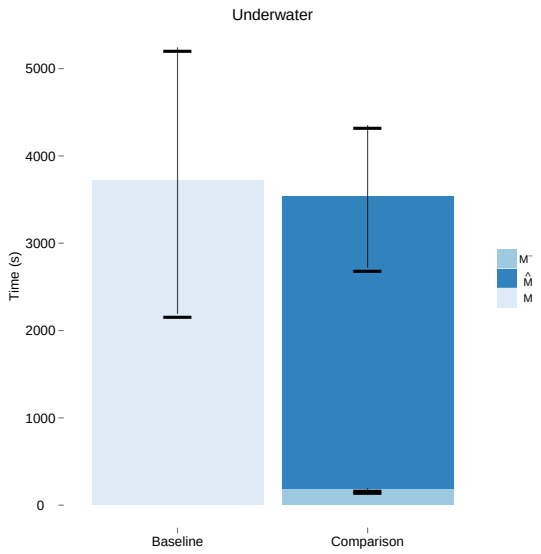
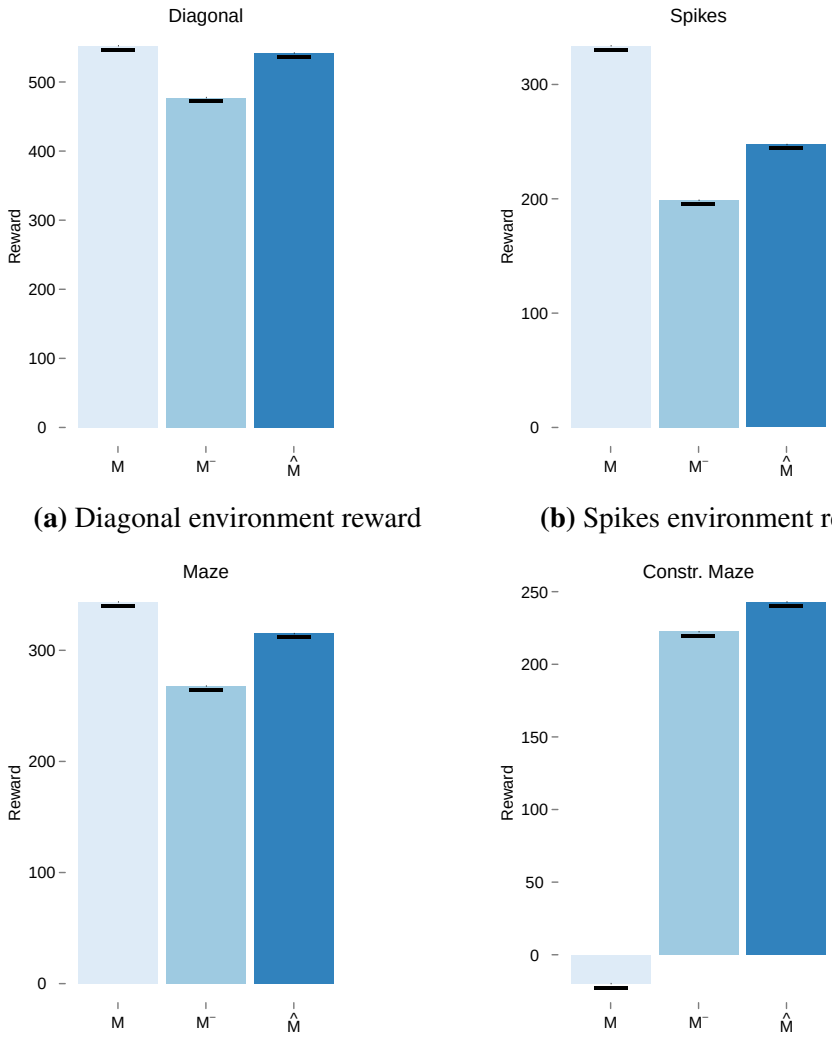


Figure 3.5 : In the underwater setting, the convergence time was highly variable, and the means are not statistically significantly different.

constrained version of the problem, solving with the simple M^- was fast, but all other runs in these environments hit timeouts. We believe that the unconstrained maze is able to find a good solution with all three models (see Figure 5(c)) because the uncertainty in the start state doesn't matter as much. This is because there is enough room to make progress in an open-loop fashion without even trying to disambiguate between the possibilities for the true start state. The true state only significantly affects the policy in the vicinity of the goal. The true model is favored here because each corner taken must use diagonal actions to achieve an optimal policy, and the reward in the unconstrained case favors the true model that is not bootstrapped with a straight-line policy which can take extra time to show to be suboptimal. However, the constrained maze results (see Figure 3.6) are believed to be so different because the robot needs to effectively collapse the uncertainty in its Y position to reliably enter the narrow passage. The simpler models are able to do this within the time limits and therefore can get to the goal on average, while the true model is not able to find a policy that achieves this effective localization in the time limits provided.



(a) Diagonal environment reward

(b) Spikes environment reward

(c) Maze environment reward

(d) Constr. Maze environment reward

Figure 3.6 : The diagonal environment (a) produced a policy that achieved slightly lower reward. In the spikes environment (b), the simpler models are able to run much faster, however, they miss some areas where additional complexity in the model was required to achieve optimality. In the maze environments (c,d), the constrained version of the problem could not be solved by M, but the approximate models could; when unconstrained, both M and $\hat{M}$ could be solved for approximately equal reward, noting that they all timeout so the total runtime for M and $\hat{M}$+ M^{-} are approximately equal at 20,000 seconds.

Finally, when we tested AMA using a noisy underwater action model provided in MCVI as the true M, we found very high variance in runtime (see Figure 3.5). The reward was found to be very consistent and positive across the polices computed in the three models.

All trials completed and found a good policy, however, some of them took much longer than others to converge to that result. This was the case for both the true action model M and our intermediate complexity action model $\hat{M}$. The simple model, M^{-}, was much more consistently fast. Overall, the mean time using AMA was found to be less than the mean time to solve the true action model, although the variance present in the data prevents a definite result.

3.10 Summary

We introduced AMA as a general method to compute an approximate action model that can be used to find fast approximate solutions to POMDPs. Our high-level AMA framework is action, observation, and transition model agnostic, and our specific POMDP implementation for simulation results is presented in Section 3.5. Given a robot model and environment that induce a belief space large enough to be computationally infeasible for existing POMDP solution methods, AMA probes the specific problem instance and automatically creates an action model that estimates where high fidelity is required to produce an optimal policy.

Abstracting the action space using AMA is clearly effective, allowing existing POMDP solvers to be applied to larger problem instances. The abstraction was taking a subset of actions, so the resulting policy was still valid on the true action model M. Following chapters will present techniques for more abstraction techniques that are not as restricted

by presenting a method to execute a policy on an action model fundamentally different from the one used to generate the policy.

Chapter 4

Coupling POMDP Sensing Policies with Replanning

4.1 Motivation

The goal of this chapter is, again, to extend the size of tasks that are solvable with current solvers. As in the previous chapter, we propose an abstraction of a POMDP, which is combined with online replanning to address the simplifications made in the abstraction. However, unlike in AMA, we do not use model refinement and only solve M^{-} because replanning allows us to use more drastic simplifications to the robot model than subsets of the action space. We demonstrate problem instances in which the POMDP model of a car-like robot can not be solved with modern techniques, but a solution to the abstraction can be found. Online replanning does not incorporate the observations from sensing in a globally optimal way as the POMDP policy does. Combining a POMDP abstraction with online planning yields a best of both worlds solution in our experiments. The computational burden has been well discussed in prior chapters and we will not reiterate the specifics.

Unlike in the previous chapter, we will now consider a task of navigating a car-like robot with a partially-known map and noisy sensors. This car-like robot has much more complex motion constraints and it acts in a continuous state space, as opposed to the holonomic point robot operating on a grid previously considered. We propose that the natural level of abstraction for this task is a holonomic robot operating in $\mathbb{R}^2$. This abstraction reduces the computational burden by ignoring the complex motion constraints. Intuitively, this

abstraction is effective because it attacks the computational complexity by reducing the state space, and the curse of history is eased by using simpler constraints. Although the problems we consider have a similarly sized state space as prior work, we show that added motion constraints can lead to unsolvable POMDPs. Our experimental results indicate that our POMDP abstraction with replanning is an effective technique. Executing a policy constructed in an abstract motion model is not trivial, and is discussed in the Methodology section. Details connecting the theoretical method with our evaluation platform appear in Experimental Setup. Our evaluation shows that, counter-intuitively, incorporating the true constraints takes may also generate hard to execute policies, reducing reward even when given time to converge to an optimal policy.

Motivating this technique is the recent success of sampling-based replanning for many robotic tasks (e.g., [62, 51, 50, 63] among many others). This strategy works well to handle challenging tasks with generic robot dynamics. Executing a policy can be modeled easily as a navigation task that changes at each time step, and the sampling-based planning at each time step can find plans under the kinematic or dynamic constraints of the robot even if they were not considered in the POMDP policy to improve computational speed.

4.2 Problem Definition

The task we set out to solve is an escape problem, where a robotic agent must avoid detection. This robot has a radio signal strength sensor that estimates the distance to a radio source emanating from an adversarial detection outpost. The problem is to determine a sensor-based policy that a car-like robot can follow without detection to a goal region. There are some existing, known obstacles in the workspace, ans the position of the outpost is unknown. The range to the unknown obstacle (the outpost) can be sensed, however the

sensor is noisy and noise increases with range. Additionally, only noisy position sensing is available, preventing the agent from ever fully localizing.

Our problem is similar in construction to RockSample, introduced on 7×7 grids [17], since expanded and modified [14, 64]. Similar to RockSample, the environment is defined as a grid, and there exists an exit region that is rewarding and terminal. Unlike RockSample, costly and terminal obstacles exist. RockSample, unlike our problem, has perfect localization. In addition, our variables for sensing the environment are not boolean values of rocks with known positions but, instead, are range measurements to a terminal obstacle with unknown position. Overall, these modifications make this new problem much more difficult.

The POMDP representing this task, P_0, is the input to our algorithm, with a discount factor of $\gamma = 0.95$ that penalizes future rewards to encourage short policies (see Equation 2.2). The output is a sequence of states (execution trace) that follow the propagation of a simulated car-like robot as computed using online replanning. An execution trace has a reward calculated using R and γ from above. We define success as when a trace ends in a terminal exit region, and failure in all other cases.

4.3 Related Work

The two-phase method we propose may appear similar to hierarchical POMDP methods [57, 58, 6]. However, these methods generate POMDP policies "all the way down" to the full system detail through decomposition of the state, action, and sensor spaces. Our proposal instead uses an abstracted POMDP, and then applies replanning to incorporate the constraints.

Point-Based Policy Transformation (PBPT) [61] and online POMDP methods [32, 25] propose a similar idea to this chapter, by limiting offline computation and then fixing the policy when necessary to decrease the required offline computation time. However, we are breaking out of the POMDP framework entirely and not changing the input policy. In our work, the input POMDP policy is for an abstracted model, therefore replanning is necessary to execute the policy. As previously discussed in Chapter 3, PBPT uses bijections $S \Leftrightarrow S'$ and $A \Leftrightarrow A'$. We will be operating in state spaces that differ in dimensionality, and our action spaces have very different constraints, precluding the applicability of PBPT. These changes in state and action spaces are precisely what drives the computational benefits of our proposed framework.

There has been significant interest recently in problems where a belief can be approximated by a Gaussian distribution [65, 29]. Although we use Gaussian distributions in our sensor model, the belief state of the world cannot effectively be collapsed into a unimodal distribution due to the uncertainty of the location of obstacles. Therefore, these extremely fast and effective techniques are not applicable to the problem instances we will be solving.

In [66], a method similar to MCVI is proposed for an explicit belief variance minimization and navigation task. Local policies are explicitly user-designed and given as input. These local policies are designed to decrease uncertainty in the current belief and drive the system to a goal state. This will allow the use of high-level abstract actions similar to our method, however, our technique builds per-action policies online instead of using user defined policies, and we use MCVI, which converges to optimal policies.

Recent efforts to improve POMDP solution times by decreasing the number of states [30, 59, 60] and actions [54] are also relevant to our goal of decreased computation time without sacrificing reward. In general, these methods will produce abstractions of navigation and/or

carve out "interesting" areas of the spaces to focus on. The papers mentioned here alter the discretizations, while the underlying motion constraints and state spaces are unchanged. However, in this chapter, we develop an algorithmic framework to utilize a POMDP abstraction of the underlying constraints and state spaces.

4.4 Overall Framework

P_0 is useful to describe the underlying continuous problem, however, continuous action spaces have only recently been addressed using techniques from sampling-based planning to find useful action samples [59]. We directly define a solvable POMDP abstraction as $P = \langle S, A, O, T, \Omega, R \rangle$ where A and O are discretized approximations of A_0 and O_0. T and Ω are then functions over these discretized spaces. Modern solvers, such as Monte-Carlo Value Iteration (MCVI) [20], can solve a POMDP in this form.

As we will show, computing even an approximate policy for P is intractable, despite the discretization of sensing and action. Therefore, this chapter proposes an abstracted POMDP model, $P' = \langle S', A', O, T', \Omega, R \rangle$. Although the motion model has changed (S', A', T'), the components of the task description that will not be addressed with replanning are unchanged (O, Ω, R). In P', $S' = \mathbb{R}^2$, $A' = \{\text{North}, \text{South}, \text{East}, \text{West}, \text{Stop}\}$, and T' is the same Gaussian, but centered on the state returned by this simpler motion model.

For comparison purposes, we solve both P and P' using the powerful solver MCVI, and the policies generated will be used in online execution. The task is complete, either in success or failure, when the policy update of the online execution indicates that the robot has entered a terminal state.

4.5 Algorithm

The proposed algorithm, Algorithm 4.5.1: POMDP+Online, operates in two phases. Line 1 is the entire offline solution phase, taking the input POMDP model and getting an optimal policy. Lines 2–4 set up the initial state of the replanning system. The use of policyNode assumes a decision tree style policy representation where each node is the optimal action and has outgoing edges for all possible observations. Line 5 initializes the output of the execution trace. Lines 6–12 perform the replanning loop until a terminal observation is sensed, appending to the execution trace after each action. The state referenced on Line 7 is the maximum likelihood estimate from the current observation — on a physical robot robustness could be improved using a history-aware localization algorithm, not implemented here. Finally, on line 13, the execution trace is returned as output. The policy is represented as a deterministic policy graph, where each node is the optimal action and edges are taken dependent on observations. The policy node update on line 11 thus depends on the observation (from line 10). This observation may be critical to the overall reward, as noisy sensing plays a pivotal role in the task. Therefore, the execution of each action is planned independently (line 7). This ensures that policy node updates are done correctly and the agent follows the reward-maximizing policy from line 1.

POMDP+Online is a high-level view, and wraps up a several important details in the functions called Solve(), Plan(), and SampleObs(). Many of these implementation details will be covered in the Methodology and Experimental Setup sections. Critical to our combination of an offline policy and online replanning is having the correct goal state for Plan(), and we will expand that function here.

A brief walkthrough of Plan() starts at the initialization on lines 1–2. The loop on lines 3–9 finds the best possible future trajectory over N simulations. At the end of this loop,

Algorithm 4.5.1 POMDP+Online

Require: ***P* is a POMDP model**

```
 1: policy ← Solve(P)                                          // Offline Planning
 2: state ← SampleInitState(P)                                 // Initialization
 3: policyNode ← policy.root(P)
 4: obs ← SampleObs(state, policyNode, P)
 5: trace ← {state}
 6: repeat                                                     // Online replanning loop
 7:    path ← Plan(obs.state(), policy, policyNode, P)
 8:    state ← state.IntegrateAlong(path)
 9:    trace.append(state)
10:    obs ← SampleObs(state, policyNode, P)
11:    policyNode ← policyNode.child(obs)
12: until TerminalObservation(obs)
13: Return trace
```

Algorithm 4.5.2 Plan() from line 7 of POMDP+Online

Require: ***start* is a valid continuous state for *planner***
Require: ***policyNode* is a node in *policy***
Require: ***P* is a POMDP model**

```
 1: maxReward ← P.minReward()
 2: bestFuture ← NULL
 3: for i = 1 ... N do
 4:    future, reward ← Simulate(policyNode, P)
 5:    if reward ≥ maxReward then
 6:       bestFuture ← future
 7:       maxReward ← reward
 8:    end if
 9: end for
10: goal ← GoalSelect(bestFuture)
11: path ← planner.plan(start, goal)
12: Return path
```

bestFuture is the set of states along this best possible future trajectory. This is used on line 10 to access *bestFuture*[1], the next state from the best future. Line 10 selects a goal – when P and *planner* have different state spaces this is non-trivial and will be discussed later. Finally, on line 11 the planner is actually called to solve the motion planning problem, and the path found is returned on line 12.

The call to *planner*.plan on line 10 of Algorithm 4.5.2 must generate collision-free trajectories that respect the system constraints. The online system is also computationally constrained for each cycle. During online execution, the same noise model is applied as in the offline stage. Due to the computation budget and noise, even simple problems and environments fail occasionally. This occurs when Plan() returns an empty path because the planner could not connect the start and goal states given the constraints of the system.

4.6 Experimental Setup

In this section, we will first give the details of the offline models, P and P'. We will proceed to provide the details of our implementation of online replanning. These details will give the algorithms presented earlier some concrete structure.

The stealth escape problem will be solved in two stages:

1. Offline POMDP policy generation, and
2. Online replanning agent policy execution.

To represent a car-like model, the state space S is the continuous space of rigid body poses in the plane, denoted as $SE(2)$. Action space A will be discretized for computational

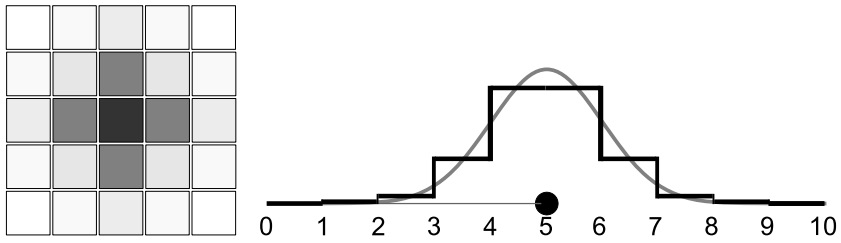


Figure 4.1 : Position (left) and range (right) sensing noise models are Gaussian distributions truncated to integer values.

efficiency into 5 actions as shown in Figure 4.2. Sensor observations of robot position and distance to adversarial outpost consist of $(x, y, range)$ tuples, discretized to integral values. T will be a Gaussian distribution centered around the expected outcome from the motion model. Ω will implement a similar Gaussian noise model for sensing. The current orientation, θ, is noise-free. We will define the reward function R to be independent of A and only depend on S. A discount factor, γ, encourages short policies and exploration of belief points that can be reached quickly.

Both P and P' share a common noise model for sensing, shown in Figure 4.1. Position sensing noise is applied by drawing a Gaussian sample around the true state, with $\sigma = 0.2$, and then only the integer part is returned. The range sensor observations follow the same general noise model, except that σ is multiplied by the current true range. Therefore, the range sensor samples from a discretized Gaussian centered around the true range to the unknown obstacle. Because our sensing model does not depend on orientation, P' still captures the essential characteristics of our task, but under fewer motion constraints.

For offline planning, MCVI is used. MCVI is a limited time horizon offline planner. We set the horizon to 100 actions. For our problems, the policies produced by MCVI always terminated significantly before this cutoff.

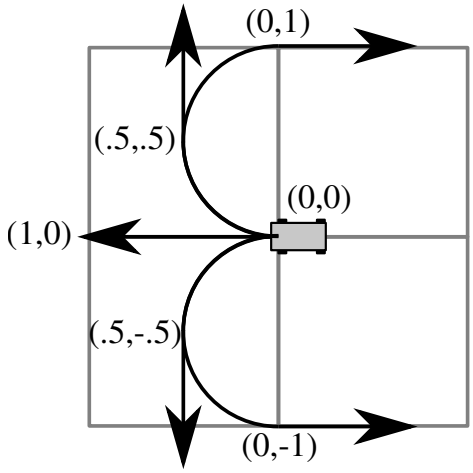


Figure 4.2 : The action model of P.

The action space A is shown in Figure 4.2. A is a discretized-action first-order robot with bounded curvature. This robot model is a discretized version of the online system that will be discussed later in this section. A does not allow the system to back up since, in the task we are solving, reverse movement is of little use at the POMDP level – turning around and continuing forward is generally enough without doubling the number of actions. This is enough because the workspaces we will consider do not have extremely narrow passages that would prevent turning around. A' consists of the 4 cardinal directions with unit length, plus staying still.

The reward function R is assumed constant over unit squares of $\mathbb{R}^2$. This is explicitly defined by the user as the input specification of the environment. Obstacles and exit regions are sets of terminal states. P or P' is solved with MCVI, and a policy mapping observations to actions is returned. This reward-maximizing policy will be followed one action at a time with an online system that does respect all the constraints of the system.

The online planner uses locally-optimal Dubins [67] and Reeds-Shepp [68] paths. These first-order, bounded-curvature paths are time-independent, kinematically valid paths for

many car and UAV models. Controllers can easily track these paths under the full system dynamics. The Reeds-Shepp model is almost identical to the Dubins model with the additional ability to go backward. Reeds-Shepp paths clearly are not valid for fixed-wing UAVs, but are more applicable to car-like UGVs. The difficulty of online planning is in satisfying the motion constraints while executing the input policy. Being able to reverse allows the Reeds-Shepp car to easily approach a target orientation and correct for position noise. The effect of this will be seen in the solution percentages.

Transition-based RRT [48] was chosen for online replanning because it plans in a space with an arbitrary reward function to produce low-cost paths. TRRT reward is based on repeated simulations of the policy, encouraging planning to follow the same set of states as the policy expects, even in a failure to reach the current goal of online replanning. Additionally, the POMDP reward function R is used by TRRT to infer known obstacle locations. Online replanning is implemented using the open-source OMPL package [47]. Dubins and Reeds-Shepp models with a minimum turning radius of 0.5 are instantiated. $N = 1000$ runs of the policy are executed from the current state, given all previous sensor measurements.

A benefit of our technique is that the same offline policy can be utilized for multiple online systems. In this chapter, we will take advantage of this abstraction and compute one policy offline, and then execute it under two different vehicle models, each with its own set of constraints. This represents the potential for massive computational savings, as it indicates that a policy does not need to be re-calculated for small differences in system constraints.

The noise model in the online stage is the same as in P and P'. As we are not testing the effectiveness of online localization algorithms, we have omitted implementing such algorithms. Omitting localization at the online stage is not a limitation of the method, but

will negatively affect success rates. Because it affects all experimental conditions equally, it changes the overall results by a uniform reduction in reward and success rates.

As discussed earlier, execution of the policy is done one POMDP action at a time, by computing an intermediate goal that accomplishes each action (Algorithm 4.5.2, line 10). Each intermediate goal is a state (x, y, θ), where θ is taken directly from the P policy, and is predicted with a 2-action lookahead when using the P' policy. This prediction considers the direction of the action that joins the next and the next-next state, and then assigns that direction as the goal orientation. If there is only 1 state in the future of either the P or P' policy, then it is assumed to be a final state and an escape goal is used that does not constrain orientation or position within the exit region.

4.7 Environments

The environments used are depicted in Figure 4.3. Most of the space has reward of -1 (gray), terminal escape regions have reward +900 (white), and the terminal obstacle regions have a reward of -1000 (black). The large grey circles are the possible locations of the unknown obstacle. The small green circle is an initial belief 95% confidence circle. No green circle appears in the slalom environment because this environment was run without position noise – the start state is the same as the mean state of the double stealth environment. Possible obstacle positions are placed so that, without sensing the unknown obstacle's location, reaching a state with positive reward is unlikely.

4.8 Evaluation Strategy

Metrics tested are total solution time, policy execution reward, and task success rate. Each metric is computed across 500 trials in each environment and the mean value is shown

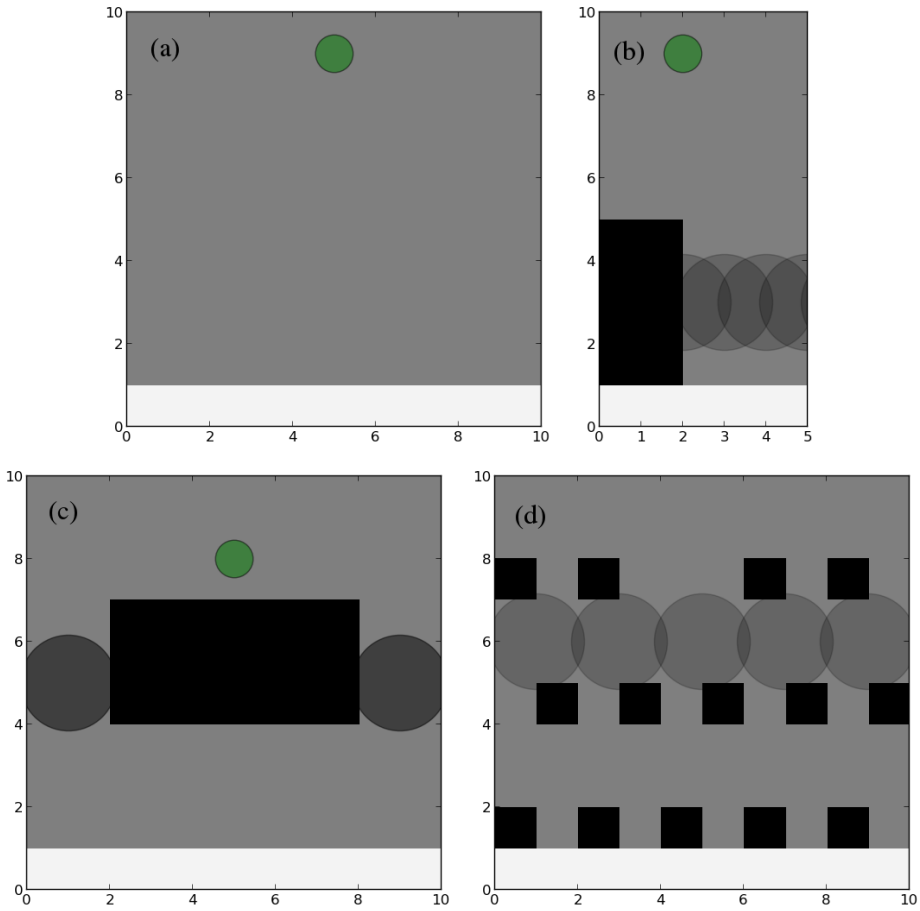


Figure 4.3 : Environments used in trials. They are: (a) empty, (b) single stealth, (c) double stealth, and (d) slalom.

with a 95% confidence interval in Figure 4.4. Although solution time is inclusive of both the offline POMDP solution and online replanning, the online time is negligible in all but the simplest tests. To compare between the policies generated from P and P', reward is considered at the online level. This is the reward the robot is actually getting under the true constraints. Each set of bars shows a comparison of the policy generated by solving P and the policy from solving P', when each are executed on the Dubins and Reeds-Shepp models.

4.9 Experimental Evaluation

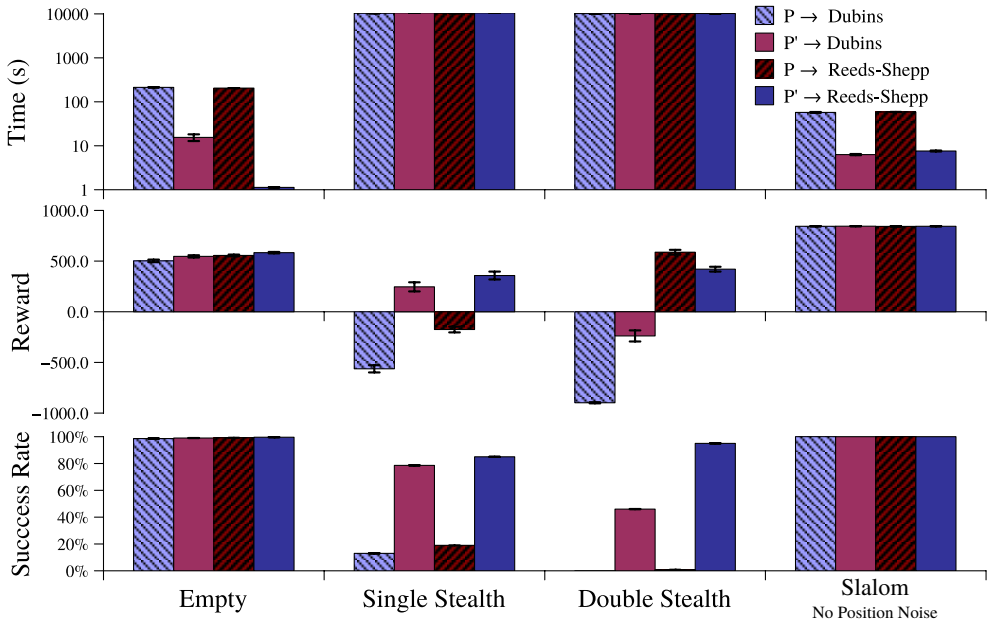


Figure 4.4 : Experimental results are presented in this summary figure. Striped bars used P, solid bars used P'. 500 trials per experimental condition.

Note that some of the policy generation tests time out at 10,000 seconds – we are specifically interested in tasks that are at the limit of the state-of-the-art in our ability to solve. Therefore we have specifically chosen a mix of environments that can and cannot

be solved optimally. Success rate is the percentage of trials that ended with a terminal observation indicating the destination had been reached, and is included in addition to reward to validate the expected behavior (higher reward is correlated with higher success rate).

The first test, Figure 4.3(a), was an empty environment, only limited by the bounds and kinematics of the system. In this test, we see that the total computation time for P is more than an order of magnitude greater than for P' (note the log scale). The success rate and reward shows that the policies for P and P' were equally optimal.

In the stealth single passage problem in Figure 4.3(b), neither policy is optimal within time limits, however, the policy for the simpler POMDP abstraction P' achieves significantly higher reward for both robot models. This test shows that the policy for the more complex POMDP was not just somewhat worse, but performed no better than chance. We conclude that all of the time was spent in determining a policy for only the navigational aspects of the problem, without being able to build a large enough policy to localize the unknown obstacle.

For the stealth double passage in Figure 4.3(c), making the wrong navigation decision based on the noisy sensing is catastrophic. This makes the problem harder than the single passage problem. If P' is used to generate a policy, the Dubins model gets negative reward. The same policy executed under the Reeds-Shepp car model achieves a very high success rate and a positive reward. From this we can conclude that the policy was good, but actually following it was difficult for the Dubins model. When simulations were visualized, it was apparent that turning corners was often problematic for the Dubins model, as it would often compute paths that were believed to be barely collision free, and, due to the position sensing noise, were actually in collision.

In the slalom environment in Figure 4.3(d), the noise model was too challenging for either P or P' to produce a policy that had a non-zero success rate. We expect this is due to the large number of turns that are made, a difficulty discussed above. Results from this environment were therefore obtained *without* position noise. In this case, MCVI can find an optimal policy for both P and P'. Note the system does not reduce to an MDP because the unknown obstacle is still only partially observable. Due to the increased clutter, the difficulty of achieving specific orientations is highlighted by the reward and success degradation when using the policy from P.

When MCVI converges for both models, P converges significantly faster and the reward of both policies are approximately equal. When MCVI times out, it is clear that the policy from P' is closer to optimal and yields significantly higher reward. The generality of online replanning is hinted at by using a single offline policy for two online models.

To evaluate if the two-stage framework can successfully solve a variety of robotic sensing andnavigation tasks, we have proposed two additional tasks to the one shown in prior work, namely exposure minimization and search-and-rescue (SAR). The specific instances of the exposure-minimization and SAR tasks will be described, and performance evaluation will be discussed. Each data point will be plotted as the mean and 95% confidence interval calculated over 500 runs of the online phase. This is necessary not only because the true world states are sampled, but also because TRRT is a randomized algorithm.

Our replanning and POMDP both have assumed perfect knowledge and update of the θ part of the state space. We made a brief evaluation of the ability of the POMDP under a noisy orientation in the empty environment. Adding noise to θ (uniform sampling within 0.5 radians of true) degraded performance to a success rate of zero when executing both the policy computed for P and P'. When the same noise was added to policy computation,

the situation was not improved because the resulting policy was not able to correct for the added action noise. This is expected because errors in θ can cause long-term changes in the execution of an action. Even when this noise was reduced to .25 radians of true orientation, success rate was still zero. Furthermore, assuming perfect knowledge of θ is reasonable because there exist inexpensive, accurate compass and inertial rotation sensors that have been widely deployed in robots. Therefore, subsequent experiments will assume a perfect knowledge of robot orientation.

4.10 Evidence of Generality

Using the same abstraction strategy for the exposure minimization and SAR tasks provides some evidence that this strategy may generalize to a variety of robotics tasks. As with most robotic tasks, both of these additional tasks require physical motion and provide only noisy sensor observations of the world.

4.10.1 Exposure Minimization Task

We evaluated the exposure minimization task in two distinct environments called L and Branched (see Figure 4.5). The POMDP P is much closer to the true robot model, a Reeds-Shepp car [68], than the abstracted P' holonomic model. In Figure 4.6, we see that because the online planner needs to respect the additional constraints in P, the online planning time is higher. The policy computed for P, however, fails to achieve positive reward. Looking at the abstracted POMDP, P', we see that it achieves greater reward, even though it incorporates fewer motion constraints. This is consistent with prior results utilizing this abstraction coupled with replanning for a different task.

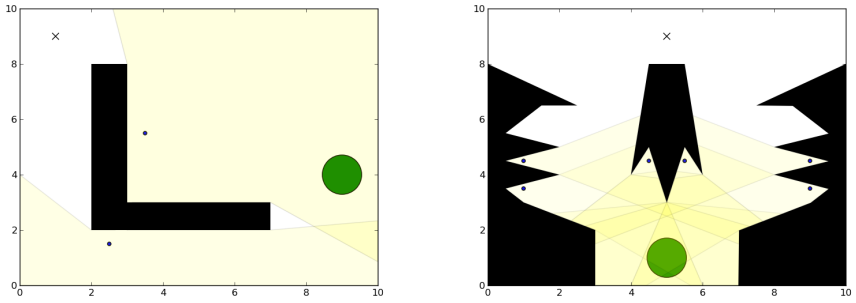


Figure 4.5 : Environments, denoted L (left) and Branched (right). In L, one observer location is selected out of only two, while in Branched, two are selected out of six. X denotes starting location, green circle denotes goal region. Possible observer locations are indicated by small black dots, and regions of observation are shaded yellow.

It is interesting to note that in the Branched environment, a lower percentage of time observed is not correlated with higher reward, as might be expected. This is because the dominant factor is simply if the robot reached the goal in the end. The observability plays a significant role, but if the policy cannot reach the goal a significant percentage of the time then observability becomes less important. This is task-dependent, based on the particular values given to the reward for the goal and the penalty for being observed. Similarly, in the L environment, although the time observed is approximately equal, the rewards achieved are very different. A factor to consider is exactly how the % time observed is calculated here – we have taken the average over all successful runs, with different path length solutions being equally weighted.

The initial evaluation of the L environment showed that the robot was able to quickly use sensing to determine which topologically distinct path to follow. The Branched environment was designed to increase the number of possibilities, as well as introducing harder sensing conditions due to selecting two of the six locations for observers. This environment therefore introduces symmetries as well as "masking" of the second, farther observer because the sensor model only observes the closest one.

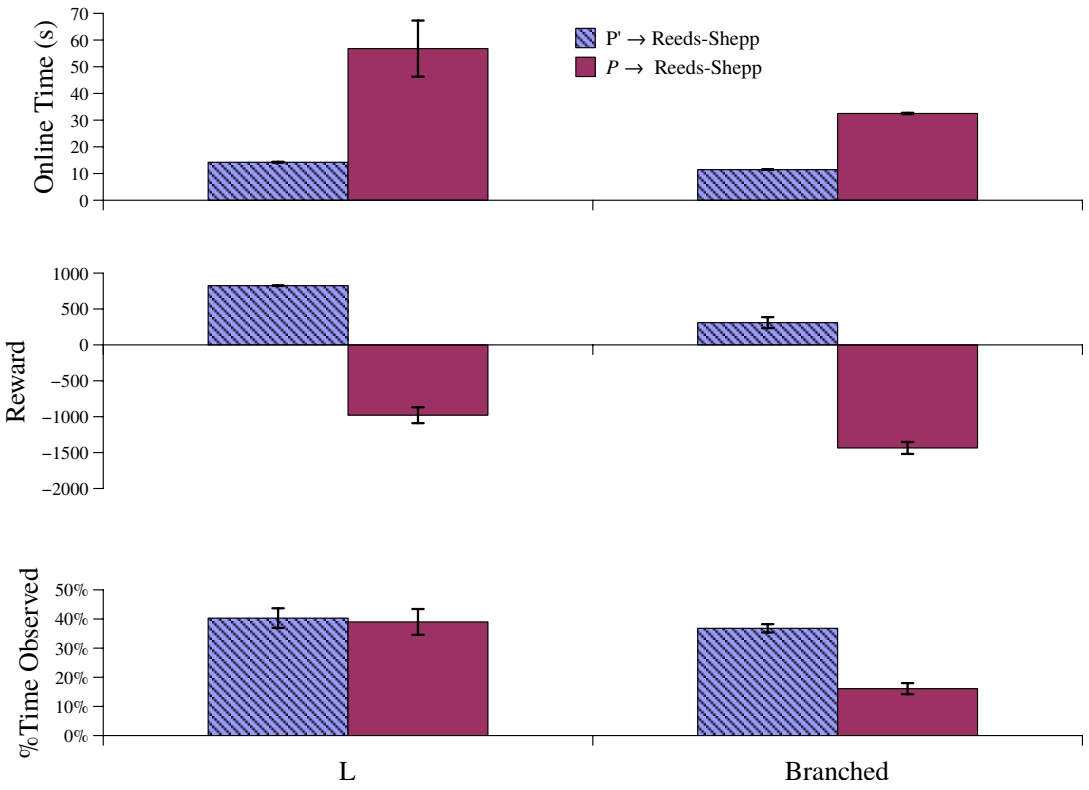


Figure 4.6 : Experimental results comparing P (solid) and P' (striped) in the exposure minimization task, with $\gamma = .99$ over 500 trials each.

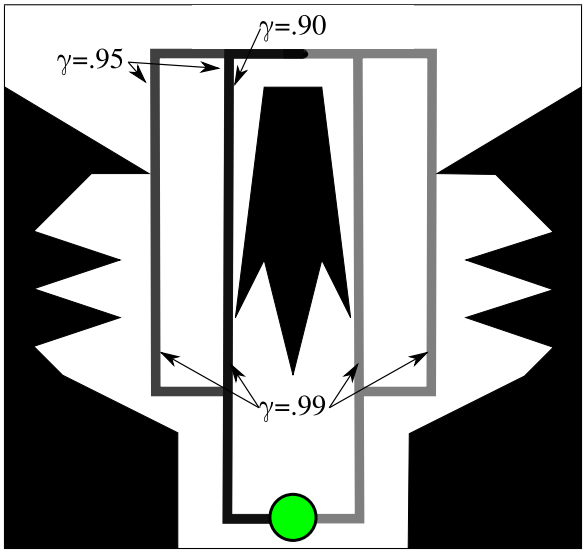


Figure 4.7 : Qualitative visualization of the different path classes executed by the policies generated in the Branched environment.

Interestingly, testing with the discount factor $\gamma = .90$ (faster computationally) showed that only one path was ever taken in the Branched environment. Further analysis showed that the time necessary to determine which of the two topologically distinct corridors to take was long enough to be suboptimal. Increasing γ to .95 still did not split into the two corridors, however, it did cause the action policy to choose a longer path in the right hand corridor, going out of the way slightly to avoid a costly double-exposure region. Finally increasing γ to .99 was able to compute the human-expected policy that incorporated sensing to choose between 4 distinct paths, the two slightly different ones found in the $\gamma = .95$ case described above and their mirror counterparts in the left-hand corridor. Each level of γ and the different path classes the policy executed in the Branched environment are shown in Figure 4.7.

This prompted a full set of experimental results for the three experimental conditions for γ, shown in Figure 4.8. In the L environment, only very minor changes are seen in the exact paths taken. Although the reward is useful to compare these different policies, it is

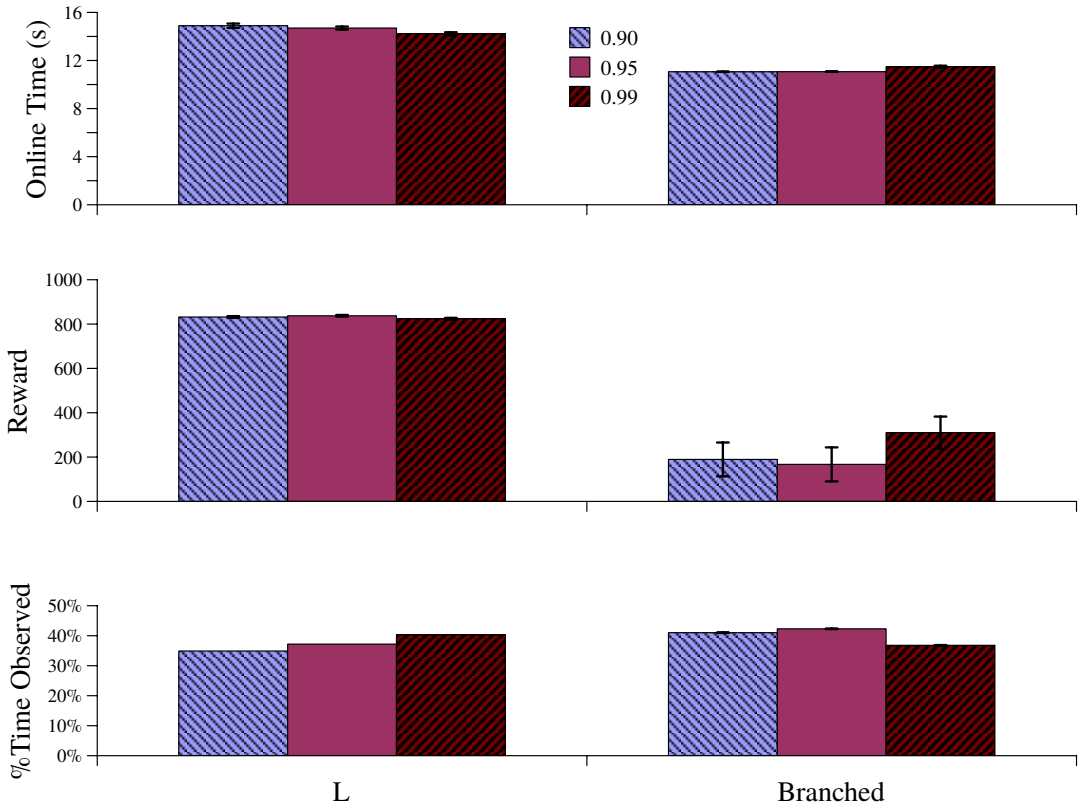


Figure 4.8 : Experimental results comparing three levels of γ, the discount factor for the exposure minimization task, over 500 trials each.

important to note that this reward evaluation is computed with $\gamma = .99$. Evaluation at other values would make the policy computed with that particular γ appear to be "more optimal" as expected.

As seen in our evaluation of γ, determining the correct parameters for a POMDP model is in itself difficult if there is an expected result to compare to. Careful analysis of the values involved can disambiguate between an error in the modeling of the task, or if the POMDP simply did find the optimal policy given the input parameters.

4.10.2 Seach-and-Rescue Task

The SAR task is harder to solve than exposure minimization because the goal is no longer a single location, but instead is selected from several possibilities. The selection is done from a uniform distribution, as are the obstacles. Three environments tested in this task are shown in Figure 4.9.

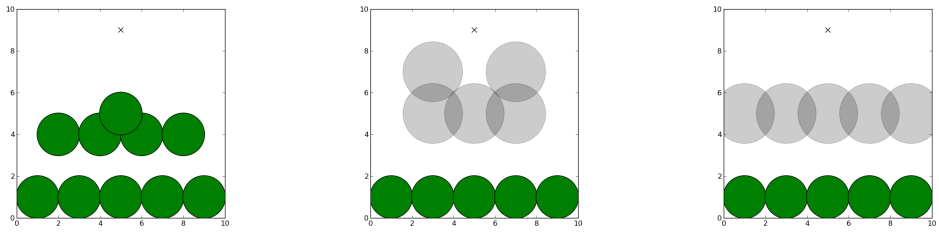


Figure 4.9 : Environments, left to right, are denoted Empty, Symmetric, and Line. In the Empty environment, one goal is selected from the ten possible locations, no obstacles are present. For the remaining two environments, one goal is selected from the five possible locations, and three out of the five obstacles are present in any given execution instance.

In our results (see Figure 4.10), it is clear from overall reward and success rates that the policy computed in 10,000 seconds did not fully capture a way to solve every case.

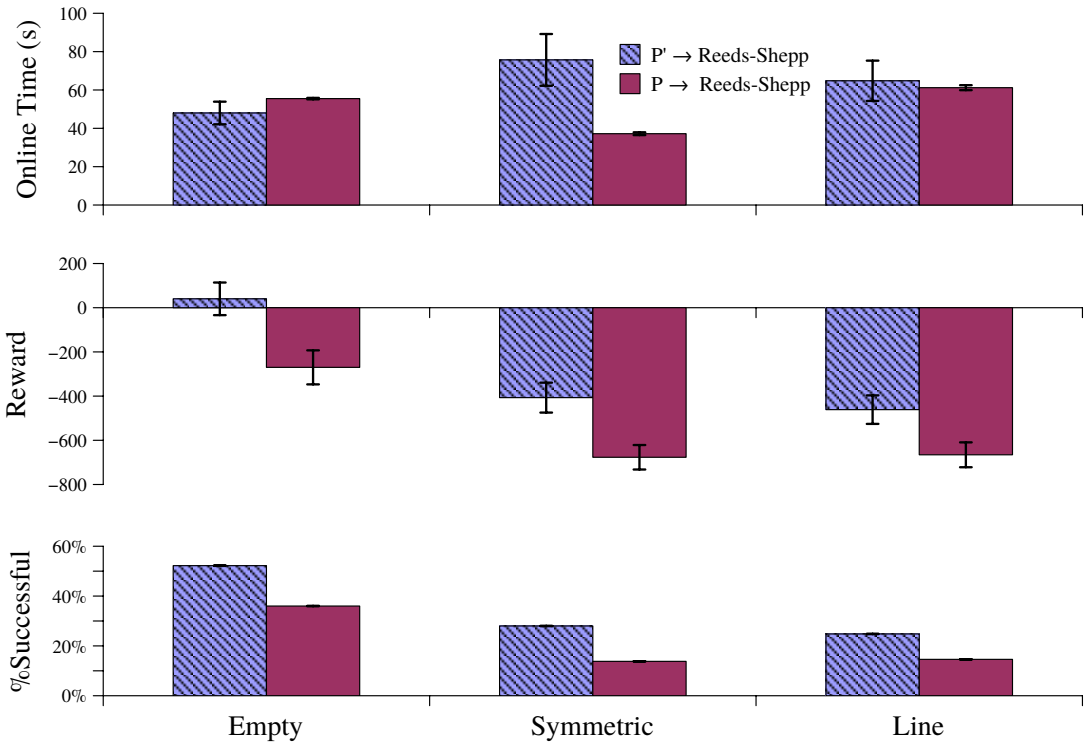


Figure 4.10 : Evaluation of the 2-stage framework applied to the SAR task in three environments over 500 trials each.

However, as in the prior example, using the abstracted POMDP P' found a policy with statistically significantly improved mean reward. Although the P' execution took more online replanning time in our results, this is because the policy from P only solved the simplest cases that are easy to navigate. The increased variance in time is due to some instances being fast to solve and others being much longer, but still solvable with the abstracted policy.

The Empty environment serves as a baseline for the maximum expected success rate. In this case, the policy had an expected success rate of 100% when simulated with the

abstracted motion model, however, in online replanning, significant failures are seen when the observations received did not appear in the policy. It is possible that increasing the time allowed and the number of particles used to approximate belief in MCVI will help. Improving the robustness of the online stage to avoid this mode of failure when possible is an important consideration for the future when considering this task and probably others like it. This robustness issue was not a significant problem previously.

The Symmetric and Line environments contain unknown obstacles, which pose an additional sensing challenge. As can be seen from their success rates and rewards, the policy often was inadequate to be successful in the task. That said, we note that the policy using our abstraction (P') was approximately twice as successful as the policy generated using P, and the reward was always significantly greater.

Due to the difficulty this task presented, all trials were also executed with sensing noise turned off. In these results, presented in Figure 4.11, it is clear that the overall trend where P always achieves significantly better reward is similar to the results with sensing noise, but overall success rates and reward are much higher. The policy computed with P', however, does not show significant improvement from the reduced sensor noise, except in the Empty environment. This indicates that even under the simpler belief dynamics, the POMDP solver was unable to capture much of the belief space for P'. As we are interested in the construction and execution of policies under noise, these policies are not solving the task at hand, however, they serve as a useful baseline of the most performance we can possibly expect in the time limits given.

One problem is that the sensors are extremely limited in the information they provide to help solve this task. This is because the sensors are range-only and nearest-only. In particular, nearest-only means that any information about obstacles farther away than the

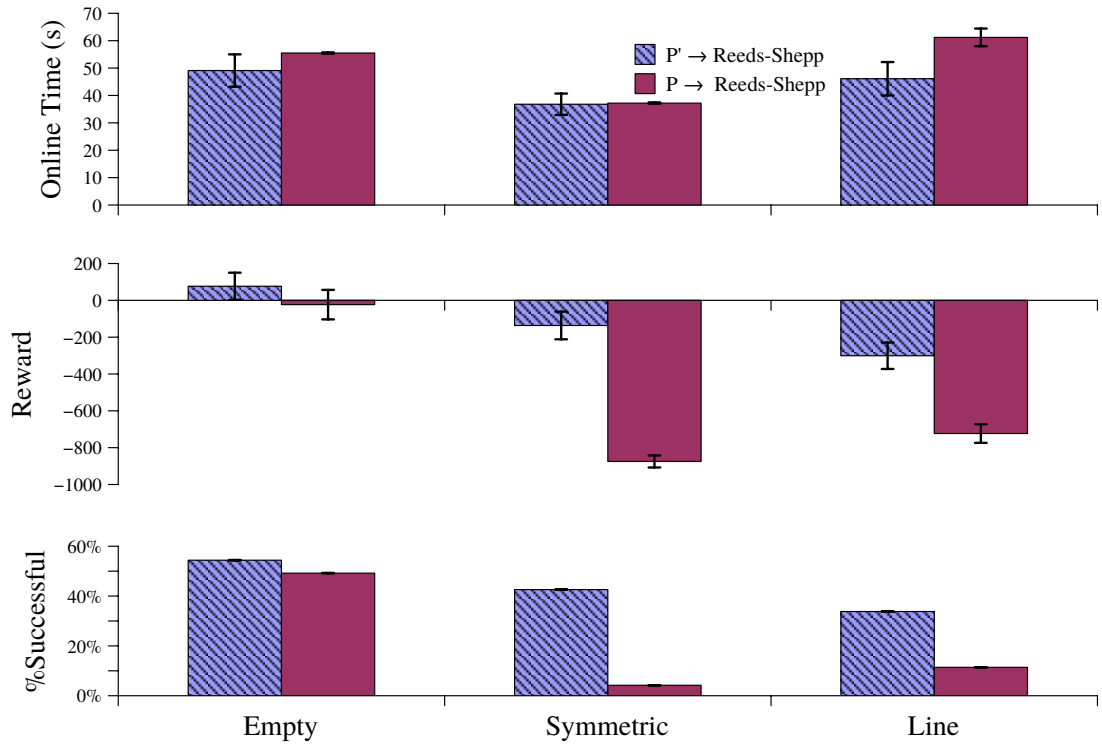


Figure 4.11 : Evaluation of the 2-stage framework applied to the SAR task in three environments with no sensing noise over 500 trials each.

closest one is completely masked. In order to gather enough information, a very long time horizon is necessary, to move in such a way that each potential obstacle could be the nearest to the robot. However, given the discounted reward and the maximum planning horizon, this is not a valid option. We confirm that this is the reason for even the abstract policies doing poorly by running the POMDP 10 times longer (27 hours) – the resulting policy has qualitatively identical expected discounted reward to the ones presented above.

4.11 Summary

To address sensing-based robotic tasks with a complex reward structure, we propose that the correct level of abstraction is to remove complex motion constraints from a POMDP. Removed constraints can be incorporated online with replanning. Our results show that, for the problem instances we have considered, our proposed algorithm is superior.

It is clear from our results that complex constraints may cause an explosion in the computation time such that the policy returned may have extremely poor success rates – 0% in some cases. Delaying the consideration of these constraints is shown to be an effective strategy to generate better policies and improve success rates.

For the future, it is clear that a localization algorithm will be required. Implementing this addition may make the results harder to analyze, but will help improve success rates.

Based on this abstraction, we see very good results in the exposure-minimization task. For this task, we also analyzed the effect of varying the discount factor, γ. This type of parameter sweep is rarely seen in POMDP literature, however, it is important to note that it can and did drastically change the overall behavior of the robot for the more complex task.

The results for the SAR task were less successful in absolute terms, though the abstracted model continued to provide significant relative improvements. Analysis of the failures pointed out the need for more robust policies, an interesting area of future work. One possibility may be to couple this two-stage framework with an online POMDP process that can patch the policy when it fails.

Chapter 5

A Fast Framework for Occasionally Markovian POMDPs

5.1 Motivation

This thesis has investigated several different tasks for planning with imperfect information. Once the task was cast as a POMDP, we could account for both action and sensing noise in a principled and near-optimal manner. However, the PSPACE-complete nature of the required computation severely limited the size of the state space our robot could operate in. Previous sections focused on ways to alleviate this limitation through state space and action space abstraction. It is generally accepted that the harder and more serious computational difficulty is imposed by the fact that a belief is constructed from all prior observations. This "curse of history" is what we will now attempt to address.

Consider a smart wheelchair application, where the user is in control but needs navigation assistance. In a smart but unactuated wheelchair, the robot may request an action and rely on the user to execute it without collision. This is in contrast to wheelchair applications [69, 70] where the user is not able to control the motion of the chair directly, relying on automatic collision avoidance and path planning. The use of a POMDP will produce a policy that is easy to execute while still considering noisy sensors and actuation. We expect that this may allow for the use of much cheaper systems with small batteries and that are very limited computationally. Furthermore, the navigation of a smart wheelchair is expected to be queried on the same set of regions many times. We will investigate the

applicability of our solution framework to a multi-query example, which are not generally considered in POMDP literature.

5.2 Problem Definition

Our proposed task is a navigation task with a long time horizon under uncertain motion and sensing. As in prior tasks, the problem formulation we use is that of a POMDP. The major difference in the task specification for this chapter is that we will add an extra sensor that can supply bounded observations in a small set of regions. These regions compose a natural decomposition of the problem in both space and time. The input to our novel solution framework is the set of subproblems (each itself a POMDP) in this decomposition, and the output will be a two-level policy, where the first selects a subproblem and the second is a policy that addresses the noisy navigation task in each subproblem approximately optimally.

The robot state will be a continuous (X, Y) position, so $|S| = \infty$. The actions in A include motion in 4 directions. Each of these motions also includes Gaussian noise in the final location. Because we wish to solve large problem instances, macro actions, which are composite actions that are built out of the atomic actions, (see [20] for more information on macro actions) were implemented as repetitions of motion in these 4 directions of motion until an obstacle is detected in the direction of motion or a region of bounded localization was detected. These 8 actions, plus an option to stay still, yield an overall $|A| = 9$.

Observations are taken from two different sources. The first is an obstacle sensor that checks if there exists an obstacle within a distance equal to one motion step plus noise in a given direction. With 4 directions and 2 possible results per direction (on/off), there are 16

possible observations from this sensor. The second sensor returns a region number if the robot is within a region of bounded localization, and a null value in the rest of the space. In addition, a special value is set if the robot is in an invalid state or if the robot has reached the goal. This second sensor provides the information needed to split the overall POMDP into smaller POMDPs joined by a fully-observable MDP. As discussed in Chapter 2, a fully-observable MDP does not suffer from the curse of history; it is the partial-observability of the POMDP that requires considering historical observations. This means the high-level policy is computationally trivial, as compared to the subproblems.

The observation model Ω defines noisy obstacle sensing as well as a perfect region sensor. Given the current state of the robot, s, $o = \Omega(s, a)$ returns a value described above, where the range of the obstacle sensor is sampled from $\mathcal{N}(stepsize, stepsize/10)$. Here, *stepsize* is the length of a single action from the action model. A standard deviation of $\frac{1}{10}$ of the length was selected as a value that provided significant complexity in observation, while being low enough to be useful. We have observed that when the standard deviation is too high, then the optimal policy will be to ignore all noisy observations.

The action model (T) also includes Gaussian noise. The final location of the robot is sampled as $x' = \mathcal{N}(x + stepsize \cdot \text{sign}(a_x), stepsize/10)$ and similarly for y'. That is, apply the motion of length *stepsize* in a direction given by a, where

$$\text{sign}(a_x) = \begin{cases} -1 & \text{if } a \text{ moves to the left} \\ 0 & \text{if } a \text{ does not change x} \\ 1 & \text{if } a \text{ moves to the right} \end{cases}$$

and the Gaussian is given a similar standard deviation to the sensing, $stepsize/10$. Intuitively, this is Gaussian noise around the nominal result of applying an action for one step.

The reward function R is 1 in the goal region and 0 elsewhere. Finally, a discount factor $\gamma = .99$ encourages short policies and also improves runtime.

This robot model is motivated by the example of a smart wheelchair application. The user may not exactly follow the robot commands, however, perhaps because of an unmodeled event (e.g., phone ringing in a distant room). In this case, the deviation from the expected observations could prevent the use of a POMDP policy. This is because POMDP policies define action based on all expected observation histories. A history that is not part of the policy has no information about which action to take at execution-time. In contrast, an MDP policy returns an action only based on current state (and is defined over all states), so can recover from a user taking an unmodeled transition. As cost is certainly an issue, we try to operate with minimal sensing – just the 4 obstacle sensors and a doorframe sensor (e.g., an upwards looking camera). Executing a policy is so computationally easy that it requires only minimal on-board computation, after offline policy generation is done with a blueprint of the environment. This low computation requirement is important for our vision of a low-cost embedded wheelchair system.

Thus, not only can we increase speed by combining MDPs with POMDPs, we can also improve robustness. The first example we will discuss in the Experimental Results section will focus on speed improvements, while the second will focus on robustness and multi-query performance. Additionally, although we do not directly investigate the physical sensors that can support our assumed sensor models without being expensive or requiring excessive power, we feel that basic infrared obstacle presence sensors and a simple vision-based doorframe sensor are both implementable and cheap. This intuition is from the low cost of modern smartphone technologies that have more than enough computation on board.

5.3 Related Work

Our proposed decomposition is similar to the idea of hierarchal POMDPs (e.g. [3, 57, 58]), with an important difference. The hierarchal POMDP, as its name would indicate, utilizes a POMDP at multiple levels. The hierarchy necessarily discards some information, or else the decomposition would not improve performance. Here, the proposal is also to discard information but explore the computational benefits of assuming the addition of a sensor that can detect bounded, disjoint regions, where the task requires navigating between these regions. Most of the space is not part of these regions, necessitating a POMDP at the low level, but this particular sensor means that it is reasonable to expect that perhaps a POMDP is not necessary at the top level and a simpler MDP solution can be used. Alternatively, one could formulate our framework as a two-level POMDP where the top level just happened to be fully observable, however, this would miss an important feature. The high level MDP policy in our proposed framework is history-independent, unlike a POMDP policy. If we are considering that a human user may take unexpected transitions, a POMDP would need to model every possible transition at every time step; this leads to exponential growth of the policy tree. By modeling the top layer as an MDP, if the user takes an unexpected transition, our probability of success will be incorrect and our policy may have been non-optimal, however, the policy can just pick up from the newly observed state without considering history due to the optimal substructure property of the MDP.

Although in many ways, the results of this framework could be reproduced using computationally cheaper localization methods (e.g., [71, 72, 73, 74]), as in the prior chapters, we propose the noisy navigation task as a starting point. Future work could investigate the applicability of the proposed abstraction framework to additional sensor-based tasks that are not well posed as localization problems. The POMDP framework we use will support

these extensions as demonstrated earlier, although they have not yet been implemented for this thesis. An advantage of the POMDP is that it also directly provides an action policy, while if the localization was solved with one of the methods cited above, an additional stochastic planning framework would still be needed.

The challenges in moving to long time horizon POMDP tasks are due to the requirement that the entire history must be considered to select an action. The deterministic policy that is generated is a large decision tree given a sequence of observations and assumes that the action performed was the one recommended by the policy. One method, completely different from what we propose here, is to instead encode a stochastic, limited memory policy using a neural network. It has been shown that incorporating what the authors call "Long Short-Term Memory" with a neural net, trained using reinforcement learning techniques with backpropagation, can utilize a set of output neurons corresponding to the actions available to the agent and select stochastically the actions relative to their activation potentials is a way to solve a POMDP with bounded history [75]. The method presented in [75] requires a policy gradient to be defined, and it is noted that local minima of this gradient can be a problem. Furthermore, although it is shown to be computationally competitive against other reinforcement learning techniques, it is unclear what the actual runtime of this algorithm is (reported in millions of iterations, so it is expected to be significant), and what a direct comparison to MCVI would yield. Nonetheless, this is an interesting alternative to the more well established techniques we have chosen to apply. Also worth noting is that this technique is thought to have some biological grounding in how the human brain makes action selection decisions for tasks represented as POMDPs through interactions of the basal ganglia and neocortex [76].

5.4 Two-level MDP+POMDP Approach

This chapter proposes a combination of both the POMDP and the MDP formulations in a novel two-level approach, referred to as MDP+POMDP. This combination will allow the consideration of sensor noise during robot execution, while directly attacking the curse of history by combining several POMDP policies in an optimal fashion through the use of an MDP at the high level. The POMDP used at a low level incorporates sensing and action noise. Given the sensor model discussed above, the robot is occasionally known to be within a particular region with absolute certainty, which we call bounded observability regions. This is to emphasize that, while some information is known with certainty, the true, continuous state of the robot is still unknown. This occasional information and associated certainty provides a natural set of boundary regions that separate the task into several smaller subtasks, illustrated by Figure 5.1.

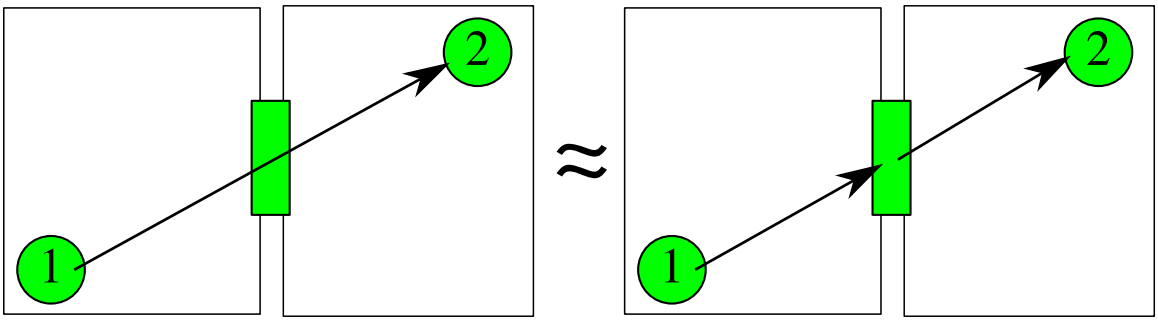


Figure 5.1 : A task that requires passing through well-observed boundary regions can be naturally decomposed into smaller subtasks. In this example, the task is to navigate from region 1 to region 2, passing through an intermediate region. The solution to the global task can be approximated through a subtask from region 1 to the intermediate region, followed by a subtask from the intermediate to region 2. The intermediate region is assumed to be a small region where additional sensing information is available.

Although we are assuming a powerful environmental aid in the added sensor that can provide occasional bounded observability localization measurements, note that the task

is still fundamentally a POMDP because it is never fully-observable. Localization within these regions is bounded but not exact. Furthermore, all tasks considered require navigating through space that is not part of these bounded-observability regions.

Given that the task is comprised of several normal regions joined by bounded localization regions, then we can decompose the task into natural subtasks. In doing so, we make the assumption that the global POMDP can be well approximated by a sequence of independent POMDPs, as shown in Figure 5.2. These subtasks are the input to the MDP+POMDP algorithm, fully described in Algorithm 5.4.1. The reason we assume that regions are joined by the bounded localization regions is so that at a high level, the task can be approximated as an MDP. This MDP is defined as $(S_{MDP}, A_{MDP}, T_{MDP}, R_{MDP}, \gamma_{MDP})$. S_{MDP} is the discrete set of bounded observation regions that our sensor model described above can perfectly observe. A_{MDP} consists of one action per state adjacency pair. That is, if two bounded observation regions are connected at the POMDP subtask level, the MDP will contain an action to traverse between the bounded observation regions. The instantiation of this action is a policy computed by a POMDP, defined on only the current regions of interest. The start region in this POMDP subtask is uniformly sampled, ignoring any history of previous actions or observations that could inform this distribution. The transition probabilities, T_{MDP}, are empirically derived from simulations of the policy. The reward structure is the same as in the POMDP formulation.

The assumption of bounded observation regions can decompose the problem naturally into sets of policies per room, given as a required input for Algorithm 5.4.1. Each element of the input $P_{i,j} \in \mathbf{P}$ is a POMDP that formulates the noisy navigation task between a particular pair of regions i, j. The set $\mathbf{P}$ must contain all such pairs of adjacent regions, where a region is considered adjacent to another region the second can be reached from the first without necessitating passage through a third region of bounded observability. In

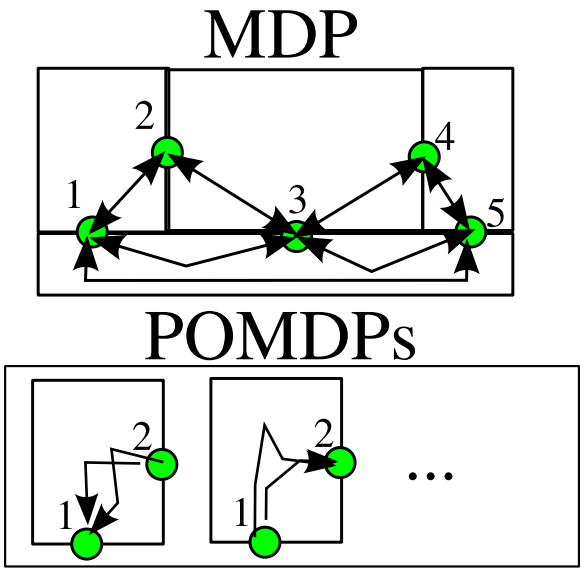


Figure 5.2 : Decomposition into a high-level MDP and a set of low-level POMDPs. Each action in the MDP is shown as an arrow connecting bounded observability regions at the task level in the diagram labeled MDP. Each action represents a separate POMDP that must be solved for varying start states sampled from the start region and accounting for the sensor noise – represented as multiple arrows showing several possible paths that could be taken between start and goal regions. One POMDP must be solved for each pair of adjacent regions, as defined by bordering the same room in the workspace. These POMDPs are directed — e.g., (1,2) is different from (2,1), depicted by different POMDPs.

lines 1–2, a loop is described where, for each pair of these adjacent regions, a policy must be computed by a POMDP solver. Now that the POMDP policies have all been computed, the MDP construction on Line 4 is straightforward. In this MDP, each state is one of the bounded observability regions of the POMDP model, and each action is a computed POMDP policy that can navigate between pairs of regions. The transition probability is empirically simulated by many executions of the policy $\pi_{P_{i,j}}$ for each pair i, j. The remaining probability is assigned to a transition that leads to an added dead state, that is simply a terminal MDP state with no transitions to other states. The MDP returns zero reward in all states except the goal, which has a reward of 1. Thus, when we generate a policy for this MDP on Line 5, it will attempt to maximize the probability of reaching the goal state (in

Algorithm 5.4.1 The MDP+POMDP algorithm

Require: P **are a set of POMDPs for navigation tasks, one per adjacent pair of discrete regions i,j**

1: **for all** $P_{i,j} \in$ **P do**
2: $\quad \pi_{P_{i,j}} = POMDP.Solve\left(P_{i,j}\right)$
3: **end for**
4: **Construct** M**, an MDP model with actions given by** $\pi_{P_{i,j}}$
5: $\pi_{MDP} = MDP.Solve\,(M)$
// Policies $\pi_{MDP}, \pi_{P_{i,j}}$ compose a global policy –
// if $\pi_{MDP}(i)$ is an action that attempts to get to region j
// execute POMDP policy $\pi_{P_{i,j}}$
6: **Return** $\pi_{MDP}, \pi_{P_{i,j}}$

the MDP) / goal region (in the POMDP).

The specific choice of *POMDP.Solve* and *MDP.Solve* are arbitrary. As in the previous chapters, MCVI [20] was used to solve POMDPs. A generic implementation of policy iteration from [9] was used for MDPs.

In theory, the POMDP policies could lead to another region of bounded observability than the expected one. However, in practice this behavior was not observed. This is a function of the noise model, the maximum time horizon, and the default policy set in MCVI. Certainly in a real system, either human-actuated or autonomous, a recovery mechanism of some sort should be added to return to a bounded observability region. Returning to a bounded observability region for recovery is required because the MDP policy is only defined on the bounded observability regions. For simplicity, only the basic framework will be considered and such a recovery mechanism is left for future physical experimentation.

Consistent models for reward and of sensing mean that solving this task with our MDP+POMDP framework is directly comparable to the solution from utilizing a single POMDP, as they have exactly the same information available. However, in the combined

MDP+POMDP framework, when a bounded observation region is entered, all past observations are forgotten. This is because each MDP action is executing a fresh POMDP policy that does not incorporate any prior information. As such, the MDP+POMDP framework will necessarily lose any optimality properties that either level separately may provide. However, given a reasonably large task, although the POMDP is asymptotically optimal, it cannot find a policy with a useful rate of success for many tasks. The MDP+POMDP framework, by means of breaking the curse of history occasionally, can achieve significant success rates in reasonable time.

If we are only interested in answering a single query, the start region may be reduced to only outgoing policies, and the goal region may be reduced to only incoming policies. For tasks involving multiple task queries, or sequences of goal regions, the robot will need a policy to get to a goal region as well as policies that treat each goal as a start region as well. Both types of tasks will be investigated in the upcoming sections.

5.5 Environments

The MDP+POMDP framework is designed to apply in environments that are larger than current POMDP solvers such as MCVI can directly handle. The abstraction that it makes is in the time domain, splitting one large task into several smaller ones. In this particular setting with doorframes, it also splits the environment in the space domain, though the algorithm does not require this. To support this split, the addition of a bounded observability sensor that can guarantee being in a closed region is assumed, but we propose that this is an acceptable model of a sensor that can detect a doorframe.

The first environment will be a general set of rooms with one start region (uniformly sampled) and one goal region, depicted in Figure 5.3 and referred to as “Rooms.” There are

11 regions of bounded observation: 9 doorframes between rooms, the start, and the goal. 21 local policies between these regions are needed for this environment (given the particular adjacencies in this task), assuming a fixed start to goal task (policies to the start or from the goal are not computed). Rooms that are exact duplicates (three across the bottom) can re-use policies between them. This is why only 21 policies are needed even though there are 33 pairs of adjacent regions.

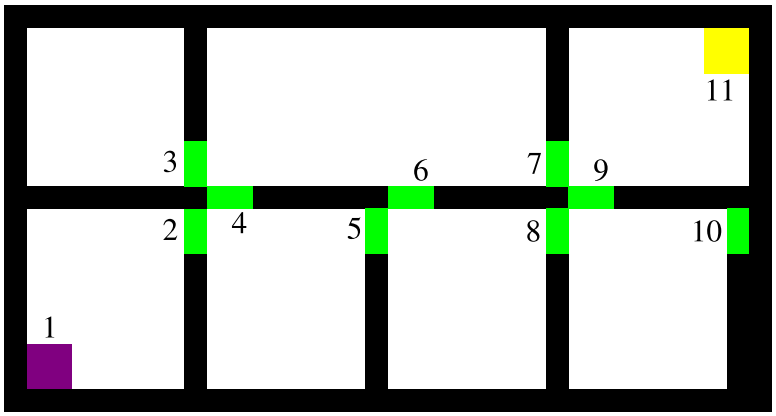


Figure 5.3 : A task, called "Rooms," for a single start and goal region. The start region is in the lower left, colored purple. The goal region is in the upper right and is colored yellow. All other regions that support bounded observation are marked in green. Each action moves the agent an expected distance equal to the smallest region width. The task can be naturally decomposed into 7 rooms with 11 labeled regions. 3 rooms are exact duplicates of each other. The 5 unique rooms require computing 21 distinct policies from region to region.

The second environment, denoted "office" is recreated from existing planning literature [77], and was selected to highlight a potential benefit of the MDP+POMDP algorithm in a multi-query example, depicted in Figure 5.4. This environment was used in [77] to support planning for paths of both sequences and coverings of goal regions that were randomly selected for a given number of total goals. Therefore, there are seven regions that are not doorframes that are bounded observability regions, and may be selected from for start and goal locations. In Figure 5.4, only one start and goal combination is shown. However, there are a total of $7 \cdot 6 = 42$ possible start and goal combinations. In this environment, there are

15 regions with bounded observability, and 59 adjacent pairs of these regions. Unlike the "Rooms" example, there are no symmetric rooms that allow policy re-use. Computing the 59 local policies will include policies to and from every possible start and goal. Therefore, the MDP+POMDP solution can solve all start and goal combinations without solving any additional POMDPs. Only the high level MDP that takes an insignificant amount of time to solve needs to be run for each of the 42 possible tasks. However, a global POMDP solution only solves one of these – we will use the one depicted. As discussed in the results, the computational time to test them all is prohibitive and likely uninformative.

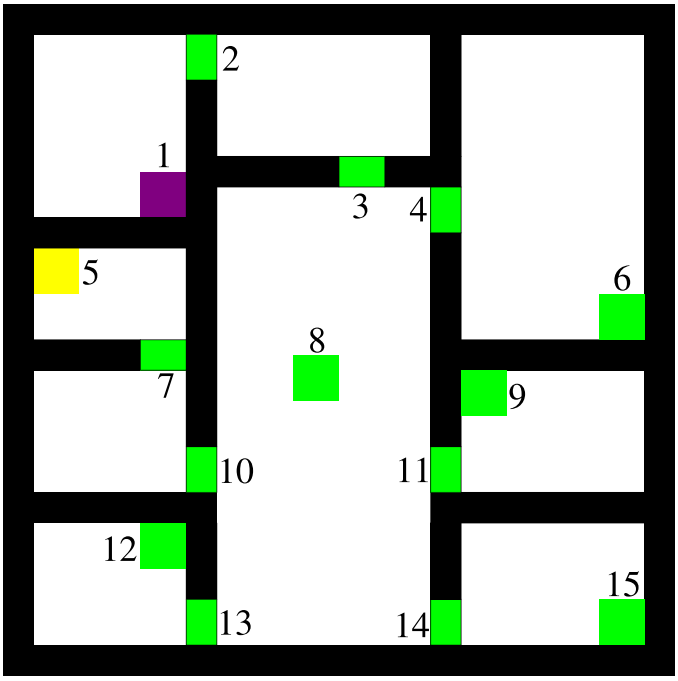


Figure 5.4 : A task, called "Office," for a multiple start and goal regions. One such combination of start and goal region is shown colored purple and yellow as before. All other regions that support bounded observation are marked in green as in the prior example. Each action moves the agent an expected distance equal to the smallest region width. The task can be naturally decomposed into 9 rooms with 15 regions, where all rooms unique. The 9 rooms require computing 59 distinct policies from region to region. The square regions (1,5,6,8,9,12,15) are the possible start and goal regions.

Both tasks need a measure of the expected upper bound on the number of actions that might be required for a good solution, the planning horizon. The maximum planning hori-

zon for each of the small rooms in both problems is estimated as $L = \frac{l+w}{m} \cdot 2 \approx 40$, where l and w are the length and width of the environment respectively and m is the nominal motion length. This approximation is from the knowledge that in a navigation task, we are unlikely to need to travel longer than twice the width plus the length of the room. The larger room is thus given $L = 80$ and the whole environment uses $L = 200$. As discussed in Chapter 1, the number of reachable belief points are upper bounded by the number of possible histories, then, for each of the three task sizes, the number of reachable belief points is at most

$$|B| = (|A| \cdot |O|)^L \approx \begin{cases} 10^{110} & \text{for } L = 40 \\ 10^{220} & \text{for } L = 80 \\ 10^{552} & \text{for } L = 200 \end{cases}$$

This can be intuited as the maximum number of possible histories that can be observed. Clearly, $10^{110} \cdot 5 + 10^{220} \ll 10^{552}$ (using the "Rooms" environment as an example), indicating that we expect to see significant performance benefit from this two-stage approach.

5.6 Evaluation Strategy

The performance of the MDP+POMDP framework will be evaluated with respect to the two indoor navigation tasks described above. The results from the MDP+POMDP framework will be compared against the results from solving a single, larger POMDP. The comparison metrics will be time spent to compute the policy and percentage of the time that the policy gets to the goal.

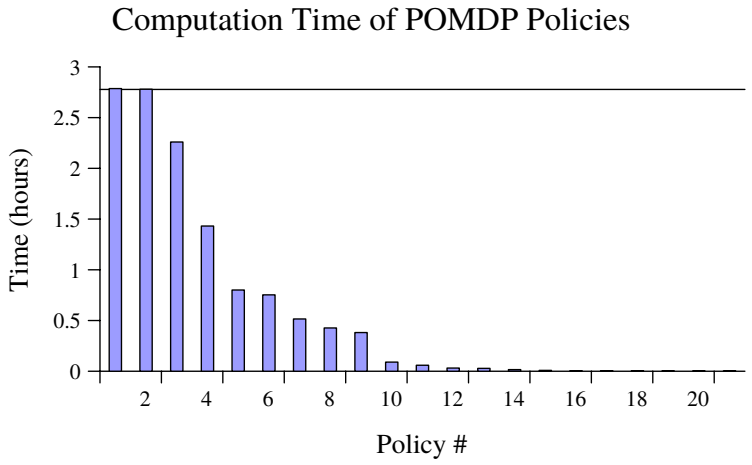


Figure 5.5 : The time to compute each of the 21 POMDP subtask policies for the "Rooms" environment, sorted by computation time. The line marks the timeout time. The x-axis is an arbitrary numbering of each unique policy.

5.7 Experimental Results

In the "Rooms" environment, a time limit of 10,000 seconds ($\approx$ 2.7 hours) was provided to each small policy. Although the 21 policies each get 2.7 hours, meaning the maximum computation time is $21 \cdot 2.7 = 56.7$ hours, the small policies often take much less than the time limit, as shown in Figure 5.5. The total computation time for all 21 policies was 12.4 machine-hours. An added benefit of this approach is that the 21 policies can be computed in parallel; the total time (assuming 21 machines were available) would be only 2.7 hours. To compare to a traditional solution, a single large POMDP instance was constructed and given 27 hours of runtime. The single large POMDP cannot take advantage of multiple machines with the current implementation of MCVI. The single large POMDP was given a time limit of 27 hours, 10 times the computation time of a single policy, and well over the total time spent on all smaller policies. The MDP is only over 12 states (the 11 regions described above, plus one dead state with only a self-loop), and takes less than one second to solve for an optimal policy, so is neglected for the time comparison.

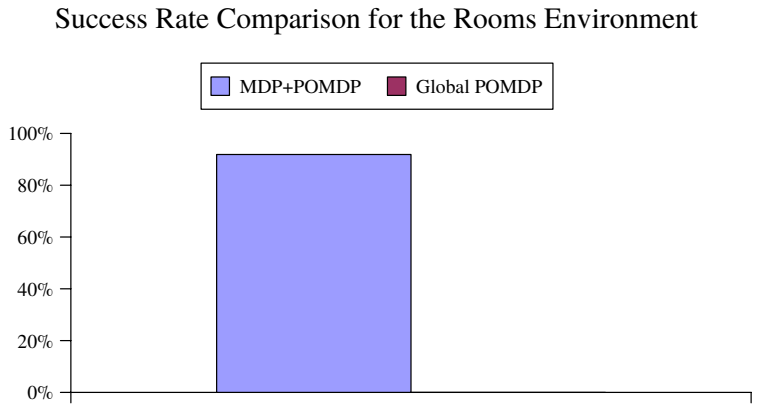


Figure 5.6 : The success rate of the MDP+POMDP solution compared to the single, global POMDP. The global POMDP was not able to find even an approximately optimal solution, seen as a 0% success rate.

This MDP+POMDP policy success rate compared to the global POMDP solution is shown in Figure 5.6. Although in theory, the global POMDP should be able to achieve higher overall success rates, the exponential complexity of the computation requires so much time that the policy computed over 27 hours is still insufficient to find a policy with a non-zero success rate, while MDP+POMDP provided a very good success rate using a composition of local policies that took 12.4 hours to compute.

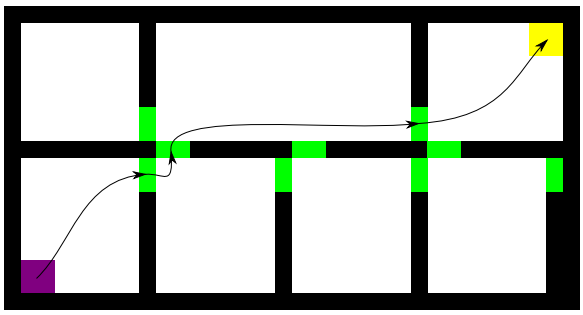


Figure 5.7 : The most probable sequence of regions of the MDP policy.

The high level (MDP) policy computed by the MDP+POMDP solution is analyzed to determine the most probable path, shown in Figure 5.7. As all of the transitions throughout

the space have a high probability of success, so the high level behavior is expected to contain the fewest transitions possible.

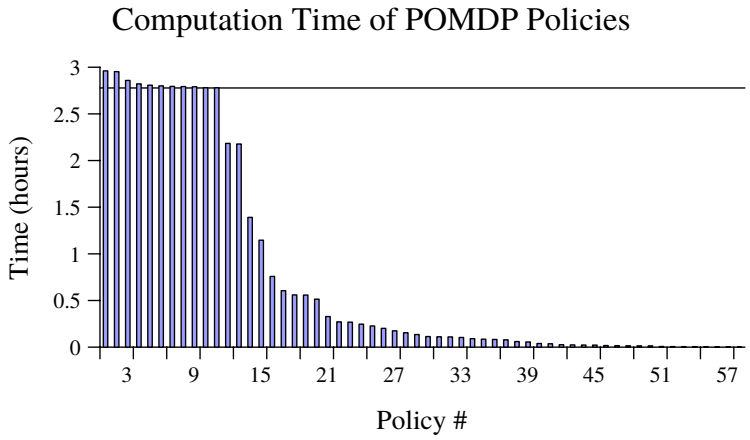


Figure 5.8 : The time to compute each of the 59 POMDP subtask policies for the "Office" environment, sorted by time. The line marks the timeout time (only checked once per iteration, therefore some times are longer). The x-axis is an arbitrary numbering of each unique policy.

In the second "Office" environment, many similar trends are seen. As in the prior results, Figure 5.8 shows that some policies are computationally difficult, but many are easy. The overall solution time is much less than the maximum $59 \cdot 2.7 \approx 160$ hours. In this environment, the total time to compute the 59 small POMDP policies was 44 hours. For a fair comparison, the global POMDP was run for 45 hours to solve this task.

As illustrated in Figure 5.9, the solution found using MDP+POMDP achieved a higher probability of success as compared to the global POMDP solution given the same amount of time. Unlike the rooms example, the global POMDP did find a reasonably successful policy. This is because although the environment is more complex, the particular start/goal pair did not require as long of a time horizon to solve. Even though the maximum time horizon specified was the same, the heuristics in MCVI were able to find that it was unnecessary to search that far into the future.

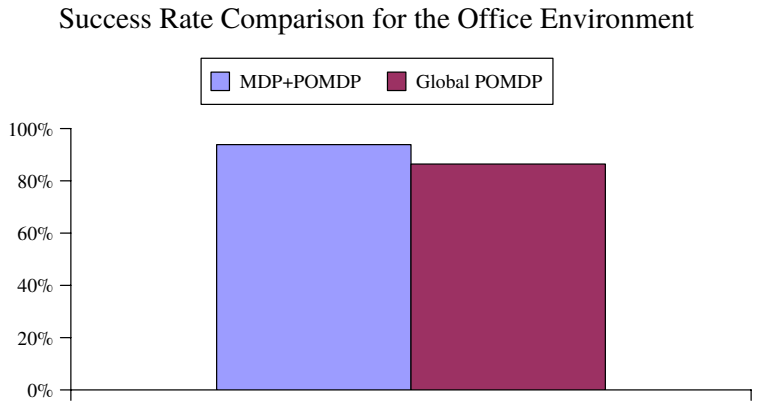


Figure 5.9 : The success rate of the MDP+POMDP solution compared to the single, global POMDP. The MDP+POMDP framework took 44 hours to compute a solution with 94% success rate; given 45 hours, the global POMDP policy had an 86% success rate.

However, this global solution is only for this particular start/goal combination, and does not provide any help to solve any other start/goal pair. Because of the excessive time required (80 machine-days) and the low expected utility of doing so, the global POMDP was not run for 45 hours per each of the 42 start/goal combinations. Rather, we simply will report the MDP+POMDP solution success rate for each of these possible tasks in Figure 5.10. The time involved to run all 42 tasks at the MDP level was under one half of a second, and the success rate for each task was not below 80% for any of the tasks.

5.8 Summary

The experimental evidence indicates that this novel technique of combining an MDP with many small POMDPs is an effective tool to combat the curse of history. In particular, this technique shines in the presence of multi-query tasks, as illustrated by the Office example. All of the computational benefits are driven by the presence of the proposed doorframe sensor model that can provide bounded observations. We believe that this is a reasonable

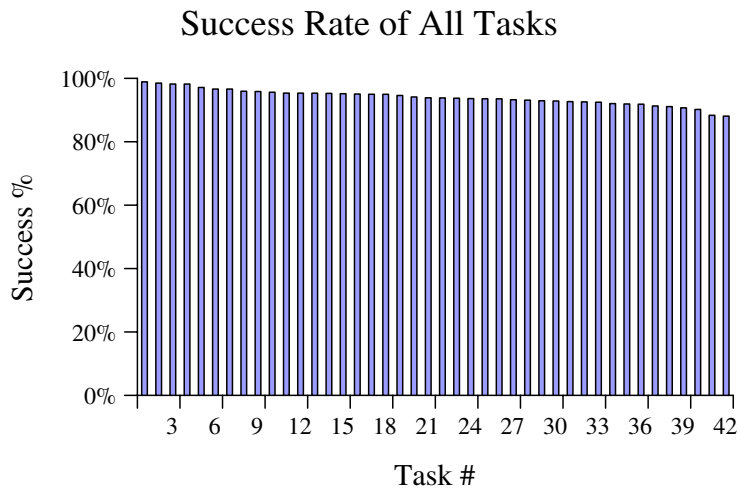


Figure 5.10 : The success rate of the MDP+POMDP solution across each of the 42 possible start/goal combinations, sorted by success rate. The X-axis is an arbitrary numbering of the possible start/goal combinations that define different tasks.

sensor model of a doorframe detector, however, physical experiments characterizing the true noise model of such a device would help validate the applicability of this technique.

Possible extensions of this framework into other sensing tasks are expected. Solving tasks other than navigation that are motivated by the example of a cheap, sensor-enhanced wheelchair under human actuation would enhance the generality and utility of this novel formulation. The task space itself is an interesting blend of cognition-assistance and robotic navigation under uncertainty. As future work gets closer to a physical implementation, human-computer interaction becomes a significant factor that would need to be considered.

Chapter 6

Image Manifold Verification as a Robotic Sensing Task

6.1 Motivation

Considering uncertainty with POMDPs is a very general approach that can apply to a wide variety of robotic tasks. Although POMDPs are the focus of this thesis, we have additionally investigated two other problems that include some sensing and uncertainty. These two additional problems will be very focused on solving a particular task and not be as concerned with the generality of the approach as we were when using the very general POMDP task specification.

In many robotics applications, including search and rescue, mobile robots often use sophisticated algorithms to extract information from sensor measurements. The research described in this chapter was published in [78], and is part of the progression in this thesis from a focus on single sources of uncertainty to the full POMDP formulation. In an information-gathering task, the goal locations for the motion planner are not known other than as a function of the sensor model. Often, there exist many locations of equal sensing value, but computing a perfect tour of exactly the necessary sensing locations is an intractable problem, particularly in a distributed multi-robot team. To maximize the utility of available resources robots need to decide *when* and *where* sensor measurements should be taken and *how* they can navigate to positions that offer the most informative measurements. The contributions of the thesis author in this project focused on the multi-robot planning

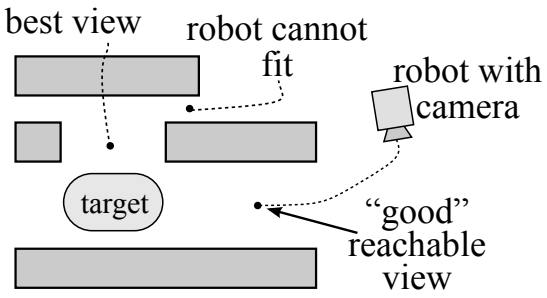


Figure 6.1 : Predicting informativeness of sensor measurements and reasoning about reachability are both essential for efficient target classification. While the best view of a target may be unreachable, there may exist multiple informative views that are easily reachable.

framework, the communication model, and integrating the communicated information into the planner cost graph.

6.2 Problem Definition

The task the robots need to solve is a cooperative sensing task, where they must gather enough information to verify an uncertain target class. We propose a distributed sensing strategy that exploits the *intrinsic structure* of an ensemble of camera measurements to *predict* which future measurements may be useful (and worth navigating the robots to obtain) and which ones are redundant (and not worth obtaining); this is similar to the *next best view problem* (NBV) [79]. To accomplish this, we also propose a measure of informativeness of viewpoints based on the *predictability* of images obtained at the viewpoints given a set of known images. Our solution enables the robots to navigate to viewpoints that are both *informative* and *reachable*.

Fig. 6.1 illustrates the importance of predictive sensing and reachability for the NBV problem. Traditionally, the cost to reach the NBV has been ignored, or has been arbitrarily set

to the distance from the current position. For car-like robots under differential constraints, this is problematic. The robots cannot simply use a reactive, control-based approach to follow a gradient based on the cost map. Furthermore, the NBV may be unreachable, or positions near it may not be informative at all (due to, e.g., occlusion). On the other hand, there may exist many reachable positions that are only marginally less informative than the best view. Since computing the next best *reachable* view is generally undecidable, we use an approximate solution and an online replanning framework to select informative views and reachable paths.

The contributions of this chapter are as follows. We propose a novel formulation for multi-robot multi-target verification for car-like robots. We employ image manifolds to represent the images sensed by the robots which enables predictions of informativeness. These predictions allow us to avoid *uninformative* views, which leads to substantial savings in power usage, bandwidth, and computation. We use the concept of cost maps to encode informativeness (and lack thereof) of viewpoints. These cost maps are used by the motion planning algorithm along with constraints on reachability. By exploiting reachability robots are more likely to find short informative paths rather than potentially long paths to the "best" view point (and avoid unreachable "best" views altogether).

6.3 Related Work

The work presented in this chapter brings together ideas from diverse topics in robotics, including NBV selection, motion planning, and multi-robot coordination.

Next-best-view selection Computing the NBV is a prominent area of research (also called "active vision") that is primarily concerned with the online 3D reconstruction of target

models[80, 81, 82] or entire scene environments [83, 84]. Usually, the problem setup considers one sensor, one target, and motion. In [85], a sonar-based approach that uses a cost map to encode the utility of unexplored areas is presented, but reachability is not considered. An algorithm to find points with large visibility polygons at frontiers of explored areas of unknown environments is presented in [83]. While reachability *is* considered, the view point selection and path planning are treated as separate problems. A similar approach is presented in [82] for model reconstruction; there, a sampling-based planner is used to compute collision-free paths to the best viewpoints, but no differential constraints are considered. The motivation of using cost maps to characterize informativeness is similar to a Bayesian formulation [86]. However, our notion of informativeness is geometrically meaningful as we evaluate it in the context of an image manifold, while Bayesian approaches are statistically motivated and do not necessarily capture the geometric constraints underlying the problem. In [87], a statistically motivated informativeness measure is combined with geometric reasoning. By computing the image articulation manifold, we are directly geometrically motivated. Because of this, we do not estimate the correlation of images, but compute it using optical flow, discussed in Section 6.5.

Kinodynamic motion planning For systems with second-order dynamics, such as the car-like robots considered in this work, it is hard to find optimal solutions for motion planning problems [38, 39], but many practical approaches have been proposed. Most relax guarantees on optimality and provide weaker notions of completeness. Sampling-based algorithms [39] provide probabilistic completeness guarantees and have been successfully used for decentralized replanning for dynamic systems [62]. Such algorithms have also been combined with cost maps to find paths that minimize some cost function over paths [45].

Multi-robot coordination There have been many approaches to let robots safely operate in the same space. They range from reactive, control-theoretic approaches to AI-based negotiation/trading-based schemes (see [88] for a detailed overview). Here, we focus on the coordination framework presented in the previous chapter that can provide strong safety guarantees for kinodynamic robots in environments with obstacles [62]. This scheme is flexible enough to let robots coordinate on higher level tasks such as mapping unknown environments or, as in this paper, navigating to informative viewpoints in a distributed fashion. Unlike [89], we do not attempt to address the global scheduling problem, but instead let local decisions tend to drive the system to success.

6.4 Overall Framework

The primary focus of this paper is the interaction between sensing and planning. Each robot executes a sense/plan/act loop, where it obtains a sensor measurement, plans for the next cycle, and executes the plan computed in the previous cycle. A robot may choose to *not* obtain a sensor measurement in a particular cycle if its location is predicted to be uninformative. At the heart of this sense/plan/act cycle is a geometric NBV algorithm to propose novel views to sense. In contrast to conventional NBV algorithms, we characterize the informativeness of viewpoints using a cost map to accommodate reachability constraints (candidate NBV points might be unreachable). We therefore need to use a planning algorithm to produce dynamically feasible motions. We assume the robots have a map of the environment and are able to self-localize on this map.

Our framework has two main parts: an *offline* model building effort, where we build manifold models for each of the targets that we are interested in identifying (see Sec. 6.5), and an *online* real-time processing step wherein the sense/plan/act loop is executed (see Sec. 6.6).

Offline The goal of our offline computations is to build a framework wherein informativeness of views is characterized. As we will see later, the NBV problem can be mapped to an elegant maximization of our notion of the informativeness of viewpoints (see Sec. 6.5). We characterize the informativeness of a viewpoint by the size of the neighborhood on the image manifold that is predicted by the image obtained at that viewpoint.

Online The online processing part executes a sense/plan/act loop. We focus on two processing stages: (a) an image analysis step where-in we use advances in object detection and pose estimation to locate target(s) of interest in the sensed image and suggest potential NBVS using cost maps, and (b) a path planning process that coordinates the robots to optimize some desired objective with reachability and inertial constraints. The image analysis process is non-myopic and does not simply pick the NBV according to some metric. Instead, a cost map is created to guide the planning process towards informative views. The cost map is defined by an informativeness score for all locations in the workspace and is updated at each iteration cycle by processing the aggregate set of measurements acquired by the mobile robots.

The planning process biases the growth of a search tree containing feasible paths towards areas that have a high informativeness. This accomplishes the goal of finding informative viewpoints that are also reachable. Whenever robots are within communication range, robots not only exchange any information about the targets that they have acquired so far but also communicate their plans for the next cycle. This process is explained in [62]. The plans from each neighboring robot are transformed into another cost map. The robots then use a composite cost map, formed by combining their own cost map, as well as cost maps for the neighbors' plans, to formulate a plan that balances finding an informative path with staying out of the way of neighboring robots.

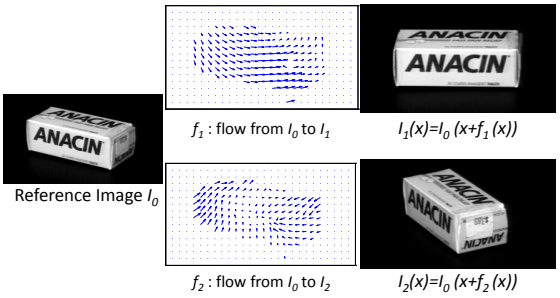


Figure 6.2 : Optical flow between a pair of images is defined as the pixel displacement field that maps one image onto another. Shown above is a reference image and optical flow to other images belonging to an image manifold. Note that I_0 and I_2 are flow predictable from each other. In contrast, while I_1 is flow predictable from I_0 the reverse is not true due to occlusion.

6.5 Offline model building

When the context is clear, we will interchangeably use the terms "sensor" and "camera"; "sensing" and "imaging"; and "measurements" and "images".

Image manifolds Suppose we consider images as elements of some high-dimensional space. Under certain conditions it has been shown [90, 91] that, if we acquire images of a moving object with K-degrees of freedom of motion, then the ensemble of N-pixel images obtained can be modeled as lying on a K-dimensional *image articulation manifold* $\mathcal{I} = \{I_\theta : \theta \in \Theta\}$ (IAM) embedded in $\mathbb{R}^N$, where Θ is the K-dimensional articulation parameter space and $I_\theta : \Theta \to \mathbb{R}^N$. Manifold models for images have been extensively studied in the context of semi-supervised classification, information fusion, and data visualization, among other applications.

Our goal is to build a notion of informativeness of images belonging to the manifolds. We achieve this by constructing operators on the manifold that link image pairs; we refer to such operators as *transport operators*.

Transport operators A transport operator $f : \mathbb{R}^2 \mapsto \mathbb{R}^2$ is a map that enables us to "traverse" the manifold from one image I_0 to another image I_1 as follows:

$$I_1 = f(I_0) \implies I_1(\mathbf{x}) = I_0(\mathbf{x} + f(\mathbf{x})).$$

We consider the space of all such transport operators denoted as $\mathcal{F}$. Note that given I_0, this space serves as a *predictive model* for images on an IAM.

We observe that the *optical flow* between a pair of images is a natural instance of transport on an IAM (see Fig. 6.2); consequently, optical flow can be used for *predicting* images belonging to an IAM. Optical flow computation between a pair of images is a well-studied problem in computer vision [92, 93, 94]. Given two images I_0 and I_1, the optical flow between them (if it exists) is defined as the tuple $(\mathbf{v}_x, \mathbf{v}_y)$, where $\mathbf{v}_x \in \mathbb{R}^N$ and $\mathbf{v}_y \in \mathbb{R}^N$, such that

$$I(x, y) = I_0(x + \mathbf{v}_x(x, y), y + \mathbf{v}_y(x, y)). \tag{6.1}$$

Given two images I_0 and I_1, we say that I_1 is *flow predictable* from I_0 if there exists an optical flow-based transport operator such that $I_1 = f \circ I_0$. Note that this notion of flow predictability is not commutative due to self-occlusions (see Fig. 6.2). When I_1 is flow predictable from I_0, then there is no need to sense both images; indeed, given I_0 and the transport operator f, we can obtain I_1 using (6.1).

We exploit the idea of flow predictability to define a notion of *informativeness* of an image. Given an image $I \in \mathcal{I}$, we define $B(I)$ as the largest subset of the IAM that is flow predictable given I:

$$B(I) = \{I_\theta \in \mathcal{I} \mid \exists f \text{ s.t } I_\theta = f(I)\}.$$

We call $B(I)$ as the *flow neighborhood* of I. The cardinality of the flow neighborhood, $|B(I)|$ is the measure of the informativeness in the image I. The key idea here is that images

associated with higher cardinality are more desirable to be sensed as they can predict a larger neighborhood of the IAM. We exploit this in Section 6.6 to build cost maps the reflect the relative informativeness of sensing at a particular location given previously sensed data.

Model building The initial preprocessing phase consists of building an efficient model of the IAM $\mathcal{I}$ for each target from sufficient training data. Given training images $\{I_1, \ldots, I_M\}$ and their articulation parameter values $\{\theta_1, \ldots, \theta_M\}$, we first build an atlas $\{(I_k^{\text{ref}}, B(I_k^{\text{ref}})); k = 1, \ldots, M^{\text{ref}}\}$ such that $\cup_k B(I_k^{\text{ref}}) \equiv \widehat{\mathcal{I}} \supset \{I_1, \ldots, I_M\}$. The images I_k^{ref} are denoted as the *reference images*. The exact number of reference images is not critical but more images will produce a more accurate IAM model at a higher (offline) computational cost. The reference images should be as close to the expected articulation parameters that will be seen in the online phase, to help with image registration. The above atlas can be built using a greedy algorithm in $O(M)$ time [95]; this algorithm relies on k-means clustering extended to the IAM. In addition to the atlas, we also compute and store the optical flows from I_k^{ref} to all images in their flow neighborhood, as well as the inverse flow from the images in the flow neighborhoods to their corresponding reference images. Note that the flow maps may suffer suffer significantly from occlusion due to obstacles in the scene. In such cases, we store binary indicator images denoting the occluded pixels as well; we term these images *occlusion maps*.

6.6 Online processing

The online processing framework consists of a sense/analyze/act loop. As mentioned earlier in Section 6.4, there are two distinct processing stages: an image analysis step and a motion planning step. The main elements of these two stages are discussed next.

Preprocessing A series of image analysis algorithms need to be performed to identify the location and the pose of the target(s) from the sensed imagery; we assume that state-of-the-art computer vision techniques for object detection provide us a bounding box around the target. Typically, such algorithms use a bag-of-features model [96] to describe target(s) in terms of their SIFT (or any other) features [97]. The end goal of these algorithms is to identify the location of the target and potentially its pose. For our applications, once a bounding box is obtained around a target, we use our precomputed manifold models to predict the NBV as follows.

Model validation Given a set of processed images $\{I^s\}$ (images of the target separated from the background) and the precomputed models for the IAM $\mathcal{I}$, the next phase is to predict the NBV to guide the navigation. We achieve this using the following procedure: given a sequence of sensed images $\{I_k^s\}$ including the newly sensed images, we compute $\bigcup_k B(I_k^s)$, which represents the portion of the IAM that is flow-predictable from the sensed images. We observe that for $k = 1, \ldots, M^{\text{ref}}$: (1) If I_k^{ref} is flow predictable from I_k^s for some k, then $B(I_k^{\text{ref}}) \subset B(I^s)$; (2) If I_k^s is flow predictable from I_k^{ref}, then we compute the optical flow and corresponding occlusion map from I^s to I_k^{ref}. The set $B(I^s)$ includes all images in $B(I_k^{\text{ref}})$ whose occlusion map with respect to I_k^{ref} is a superset of the occlusion map of $B(I^s)$; and (3) If $\bigcup_k B(I_k^s) - \widehat{\mathcal{I}} = \emptyset$, then we have charted the IAM using the sensed images and declare the target identity as validated.

In our implementation, these steps are done in a decentralized manner with each robot processing only the images that it has acquired. Each robot builds the flow neighborhoods $B(I^s)$ over its own images. Computing the union $\cup_k B(I_k^s)$ is done easily by local communication of the set of image articulation parameters that each robot has captured; this avoids the need to exchange the images between the robots.

Online cost map computation When $B_{\text{diff}} = \cup_k B(I_k^s) - \widehat{\mathcal{I}} \neq \emptyset$, then there are neighborhoods in the IAM that we need to chart and corresponding neighborhoods in the workspace that should be explored. To facilitate this, we first define the vector $\mathbf{s} \in \mathbb{R}^M$ such that $\mathbf{s}(k) = 1$ if $I_k \in B_{\text{diff}}$ and $\mathbf{s}(k) = 0$ otherwise, where $\mathbf{s}(k)$ is the $k-$th component of $\mathbf{s}$. Then, we define a real-valued function $C(x, y)$ on the *workspace*:

$$C(x, y) = \sum_{k=1}^{M} \mathbf{s}(k) c(|B(I_k) - B_{\text{diff}}|) \text{ Ind}((x, y) \in W(\theta_k)), \tag{6.2}$$

where $\text{Ind}(\cdot)$ is the indicator function and $W(\theta_k)$ is the set of physical (workspace) coordinates that correspond to the articulation parameter θ_k. The quantity $c(|B(I_k) - B_{\text{diff}}|)$ is a payoff term associated with acquiring the image I_k that depends on the cardinality of the additional portion of the IAM $\mathcal{I}$ that we are able to chart by sensing the image I_k. In this paper, we use $c(k) = \frac{k}{M}$, where M is the number of images in our training ensemble. We have freedom in our choice of $c(\cdot)$ and any monotonic increasing function can be used.

The above formulation is invariant to the order in which the images are acquired. Indeed, it does not matter if the images were captured by more than one camera. Thus, our approach can be used in the mobile *multi-camera* setup. Further, the framework can be extended to the case of *multiple targets*. In this setting, the IAM dimension increases linearly with the number of targets, but all other aspects remain the same.

Online implicit coordination through cost maps The cost map derived from the measurements is used to guide the navigation of multiple robots. Each robot executes a replanning loop, in which each robot takes a sensor measurement iff the cost map indicates it is informative. Since the focus of this paper is on the interaction between sensing and planning, and not on communication, we will assume that updates to the cost map can be

propagated relatively quickly through the robots' communication network. The robots thus have a consistent view of the world. (Communication delays, dynamic network topologies, transmission errors, and uncertainties will be addressed in future work.)

First, each robot analyzes its immediate vicinity to determine if targets can be seen or if it still needs to navigate to such an area. It does so by computing the percentage of grid cells in the cost map in small neighborhood around the current position that contain informative viewpoints (i.e., $B(I^s)$ is above some threshold value). If this percentage is above a certain prescribed value p (e.g., $p = 20\%$ in our experiments), then the robot will use the new cost map as-is. On the other hand, if this threshold is not met, then the robot considers itself as far away from useful views and switches to using a slightly different cost map. In this case, the robot identifies informative areas in the sensing cost map and uses a brushfire algorithm to compute the workspace distance to these areas. This brushfire-based cost map is then used to guide the robot closer to informative areas. Finally, when a robot receives paths from other, neighboring robots, it will increase the costs around those paths since there is no need for two robots to sense the same areas. The computation of the effective cost map from the sensing cost map is extremely simple and can be performed just prior to motion planning in each replanning cycle. The end result is that each robot finds paths that balance the informativeness of a viewpoint with the cost to reach it. Simultaneously, the cost map assists with the coordination among the robots so that they do not compete to get to the same viewpoint (since the cost to cross paths from nearby robots causes a robot to navigate to other viewpoints). Note also that there is no mapping from robots to targets. This is a huge advantage when multiple targets can be seen from a single viewpoint, as we will demonstrate in Section 6.9.

Multi-robot motion planning Due to the complex dynamics of the robots, they cannot simply follow a potential based on the cost map but rather need to plan to obtain controls that drive the robots through low-cost cells in the cost map. Each ground robot has second-order, car-like dynamics:

$$\dot{x} = v\cos\phi\cos\theta, \quad \dot{y} = v\cos\phi\sin\theta,$$
$$\dot{\theta} = vl\sin\phi, \qquad \dot{v} = u_0, \qquad \dot{\phi} = u_1,$$

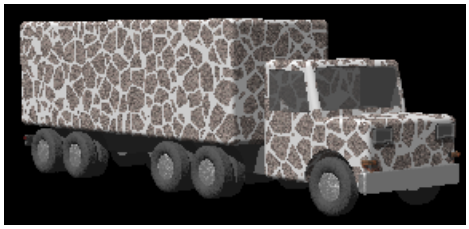
where (x, y, θ) is the pose of a robot, v is its velocity, l is its length, ϕ is its steering angle, and the control vector $u = (u_0, u_1)$ controls the acceleration and turning rate. The velocities and controls are bounded, so the robots cannot stop or change direction instantaneously. This makes navigation in environments with obstacles very challenging; the robots cannot rely on purely reactive behaviors and need to plan ahead.

Each robot generates a series of candidate paths using a sampling-based planner [39]. Such planners have been demonstrated to be effective in finding valid paths for constrained, dynamic systems [43] and can be guided in their search for good paths by cost maps. The robots communicate with their neighbors not only to exchange sensor updates, but also to exchange proposed plans. We use the motion coordination framework for car-like robots presented in the prior chapter, with the second-order dynamics described above.

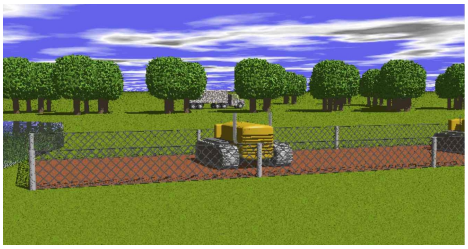
6.7 Environments

We have constructed a series of environments of increasing difficulty to test our approach. They are shown in Fig. 6.3. In the empty environment (Fig. 6.3(c)), we used 3 robots to verify k distinct targets ($k = 1, 2, 3$). In the more realistic environments

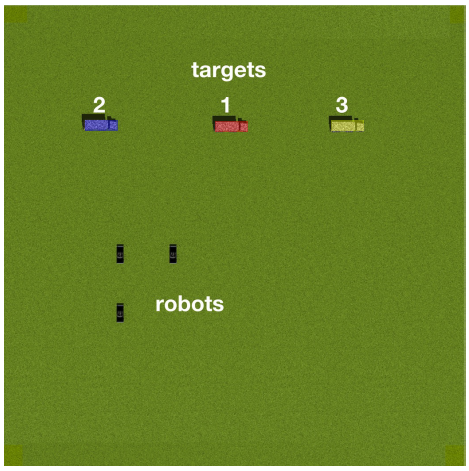
(Fig. 6.3(d), there are two targets and two robots. In the simplest version of this environment, there is only a river separating the left half of the scene from the right half. The river is not an occluding obstacle, but only a navigational obstacle. An additional level of complexity is introduced by adding trees around the targets, so that the robots cannot get close to the targets, but still have (partial) views of the target. In this scenario ("+trees" below) the robot cannot individually sense the targets completely, but together they can. In the most complex version (referred to as "+barriers" below), we have added some more navigational obstacles in the form of a few fenced-in barriers that introduce additional oc-



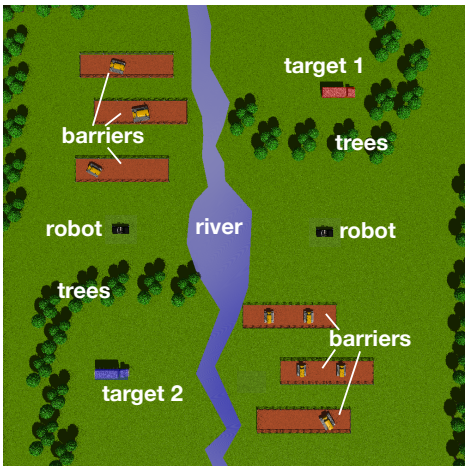
(a) Sample target template



(b) Robot's view of a barrier, trees, and a target



(c) The empty environment



(d) The outdoor environment

Figure 6.3 : Simulation setup. (a) Example image from the target (model) manifold. (b) Example image captured by a robot. (c) Top view of the empty scene. (d) Top view of the outdoor scene. The different colors for the targets denote that the targets are distinct; they do not need to be the same truck.

clusion. Fig. 6.4 shows the cost map $C(x, y)$ (see (6.2)) for the "+trees" scenario after the two robots have taken a few measurements.

6.8 Evaluation Strategy

Simplifications A ground-based mobile camera is parameterized by 3 degrees of freedom (its 2D location (x, y) and its 1D orientation ϕ) and thus characterized by a 3D articulation space Θ. However, we assume that the target can be centered in the image whenever it is within the field of view, thereby reducing the articulation space by one dimension. Second, if we ignore perspective effects, then translation of the camera towards the target results in a simple scaling of the target. By scaling images to a preset canonical size we reduce the problem by an additional dimension. Thus, we have reduced a 3D articulation space to a simple 1D articulation space, parameterized by a circle around the target $\Theta = \mathbb{S}^1$. Given $\phi \in \Theta$, the image I_ϕ corresponds to a camera located at $(r\cos\phi, r\sin\phi)$ viewing the target. We leverage this simplification in the computation of the cost map (6.2) of each robot at every time step as follows: we can redefine $\text{Ind}((x, y) \in W(\phi_0)) = 1$ provided $\text{atan}\frac{y-y_t}{x-x_t} = \phi_0$ and zero otherwise, where (x_t, y_t) is the location of the target. In addition to this, we add a penalty term to account for loss in resolution due to scaling of the form $e^{-\kappa((x-x_t)^2+(y-y_t)^2)}$.

Comparison with other approaches We analyze the performance of our algorithm using two metrics: (a) time necessary to verify all targets, and (b) number of sensing measurements taken. We use the following stopping criterion: when 75% of the IAM is charted, we declare success in target verification. Our approach is compared to two simpler approaches. In the "naïve" approach the robots simply need to get a view of the target from every angle. In other words, the robots do not compute $B(I^s)$ to predict images from nearby viewpoints.

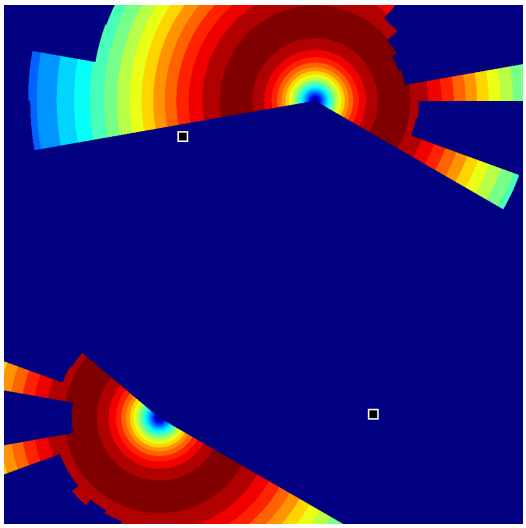


Figure 6.4: Sensor cost map for the "+trees" scenario in figure 6.3(d). The robots' current positions are indicated by black squares. Blue corresponds to uninformative, while red corresponds to highly informative. Note that areas from which views are explainable by previous measurements as well as areas from which the targets are occluded by trees are both marked as uninformative.

In the other approach (called NBV below) the robots are repeatedly trying to navigate to the k next best views. For the NBV approach we compute the k next best views from the sensing cost map and let those be targets for k robots (without assigning targets to robots). Here, the effective cost map is defined by the sum of Euclidean distances to the k views. In all cases the planning algorithm that is guided by the cost map remains the same. The comparison with the naïve approach shows the advantage of predictive sensing, while comparing with the NBV approach will highlight the importance of exploiting reachability.

6.9 Experimental Results

Fig. 6.5 summarizes the results from all simulations. In the outdoor scene, the naïve and the NBV algorithms fail to complete the validation tasks; in these cases occlusion and navigational obstacles causes the robots to get stuck trying to get to unreachable positions. As one would expect, the NBV approach is competitive in terms of completion time with our approach in the empty scene, but it cannot handle more complex scenes. From the measurements plot in Fig. 6.5 we see that our approach requires fewer measurements in

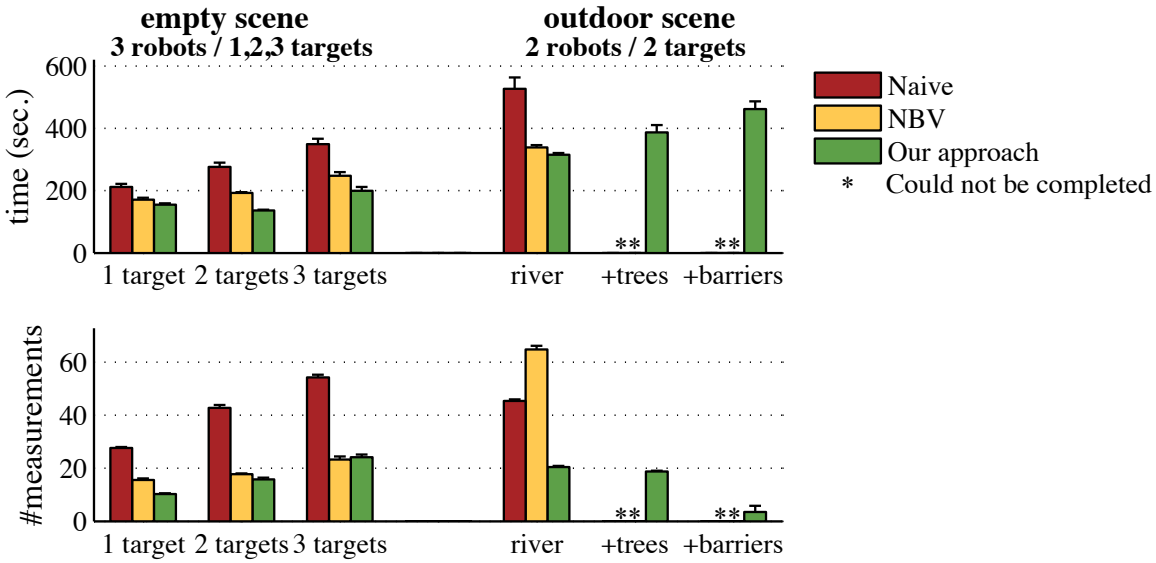


Figure 6.5 : Simulation results. (top) The mean time to verify the identity of all targets using the three different approaches for the two different scenes. For each scene there are three variations of increasing complexity, as indicated along the X-axis. For each variant we computed the mean over 10 runs. The error bar corresponds to one standard error of the mean. (bottom) A similar plot for the mean total number of measurements taken.

almost all cases. Also, the number of measurements decreases with the complexity of the outdoor scene variations. This is due to occlusion: there are simply fewer unobstructed informative views available along a path. In contrast, our approach is still able to find all the necessary viewpoints to verify the targets.

6.10 Summary

We presented a novel approach to multi-robot target verification of multiple static targets. This problem was cast as a distributed next-best-view problem with differential motion constraints on the robots. By reasoning about the image manifold we can predict which measurements are expected to be useful. Through simulations we demonstrated substantial savings in both the time required and the number of measurements needed when compared

with alternative approaches. Through the use of cost maps the approach achieves both tight integration between sensing and planning as well as implicit coordination among the robots. The coordination is used to avoid collisions and to distribute the robots over multiple areas of interest. The unknown parameter, if a high reward sensing locations was reachable, was successfully addressed through a novel framework built around sampling-based replanning.

Chapter 7

Replanning for a Partially-Known Sensing Task

7.1 Introduction

This chapter presents the second focused, non-POMDP technique we developed to address the problem of robotic planning with imperfect information. Specifically, we will investigate a single robot with a task that depends directly on the imperfect information from a sensor. A classic task for a robot is that of motion planning. Given a start state A and a destination region B, the problem is to find a continuous path that the robot can execute to reach B. The robot is in general governed by a set of differential constraints [98, 49]. In this chapter, we consider a variation of the problem above. The robot computes a plan to move from A to region B. However, the robot is equipped with a sensor that can alert it if an anomaly (a target of interest to be sensed) is detected within some range while the robot is moving. In that case, the robot tries to deviate from its computed path and gather more information about the target without incurring a big delay while fulfilling its primary mission, which is to move to destination B (see Figure 7.1). The chapter develops a framework in which this problem can be studied. A powerful state-of-the art sampling based planner (SyCLoP [99]) is modified and used in a replanning loop. The generality of the planner allows the consideration of many realistic robots with complex dynamics. However, this thesis will continue its focus on car-like robots. The contributions of the thesis author in this project focused on the integration of the planning framework and the information provided by the sensor model.

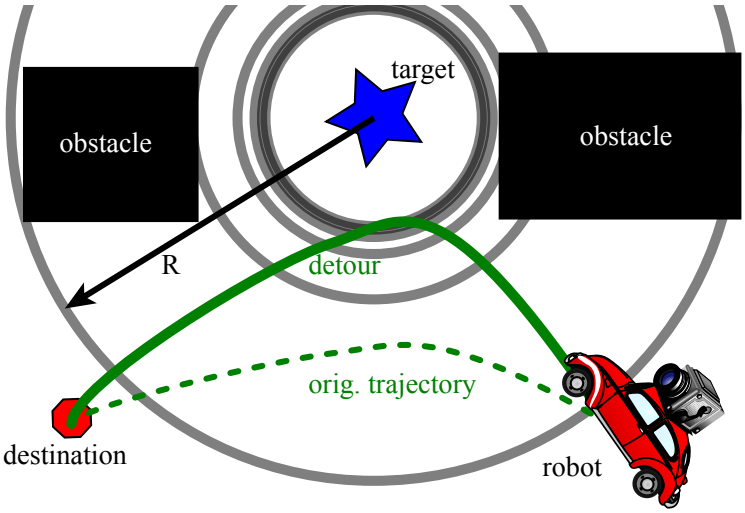


Figure 7.1 : A schematic representation of a car-like robot making a detour from a path towards its primary destination to opportunistically gather additional information about a secondary target (indicated by a blue star) once the presence of the latter has been detected at distance R. The concentric circles indicate isocontours of probability of gathering enough information to, e.g., classify a target. The probability decreases quadratically with the distance to the target with the sensor model used in this chapter.

The novelty of this work lies in the fact that the planner automatically computes deviations from the original path without asking the user to specify viewpoints and without forcing the robot to always go close to the point of interest. This is achieved by modifying a key step of the planner and providing a single parameter that can be easily tuned to control the trade-off between gathering information about the secondary target and the time required to complete the primary mission.

This work touches upon the broader topic of the need to balance multiple objectives mainly for mobile robots. It would be ideal if a planning algorithm managed this balancing process with minimal or no human intervention. Implementation on a physical platform would introduce additional complexity that we do not explore, and is well described in [100]. Applications exist in search and rescue operations, reconnaissance missions, maintenance of mobile wireless networks, mine clearing, and others. What complicates multi-

objective planning in these domains is that some of the objectives may not be known beforehand, but are triggered by sensor readings (e.g., when anomalies are detected). It is challenging to balance the primary objectives stated at the start of the mission with secondary objectives as they arise. In this chapter we consider moving a robot to a final destination as the primary task, and maximizing sensor information gain about a detected anomaly as the secondary task. A more formal definition of this scenario is presented in Section 7.4. Although the problem is presented in the case of a simple robot, it also makes sense (and is more challenging) in a multi-robot setting.

7.2 Related Work

This chapter, similarly to the previous one, brings together several ideas across robotics, including kinodynamic motion planning, replanning, and view selection. As has been previously deicussed, it is hard to find solutions for motion planning problems under second-order car-like dynamics [38, 39, 98, 49], but many practical approaches have been proposed. Sampling-based algorithms [39] have been successfully used for replanning for dynamic systems [101, 102, 62]. The typical approach in replanning is to use relatively short time periods called *planning cycles* to plan a robot's motion for the next planning cycle while at the same time the robot is executing the plan computed in the previous cycle.

Sampling-based algorithms have also been combined with cost maps to find paths that minimize some cost function over paths (such as the amount of work) [45]. Cost maps typically consist of a grid decomposition of the workspace with a traversal cost associated with each grid region. Cost maps are very relevant to our work. They have been used successfully in [46] in combination with a rich set of precomputed maneuvers to effectively plan motions for a car in the DARPA Urban Challenge. In our work a generalization of cost

maps will be used that not only depend on distance or path length, but also on sensor-derived information about possible viewpoints of a target. Unlike many other grid-based planning techniques, the completeness of the planner is not limited by the grid resolution.

As in the previous chapter, we emphasize that most of the work on viewpoint selection ignores whether a viewpoint is reachable for a given robot. This is problematic, since a robot may be asked to go to a position that is unreachable. At the same time, many other viewpoints that are almost as good may be easily reachable. Previously, we discussed an algorithm to find points with large visibility polygons at frontiers of explored areas of unknown environments [83], where reachability is considered, but viewpoint selection and path planning are decoupled and treated as separate problems. A similar approach has been presented in [82] for model reconstruction; here, the authors use a sampling-based planner to compute collision-free paths to the best viewpoints, but no differential constraints are considered. The approach in [81] combines the next best view (NBV) with cost maps that include secondary costs. In that work, a candidate path from the current location to the NBV is recursively deformed to obtain good views "on the way" to the best view, as opposed to directly constructing a sensor-based plan. Bayesian approaches often do not capture the motion constraints underlying the problem. In [103], a Bayesian sensor model computes a cost map and waypoints are selected. Unlike our work, the considered motion planner does not receive this cost information. It receives only the waypoints, and reachability is assumed. However, the proposed Bayesian sensor model could be used in our framework with minimal modification and this is a possibility for future work.

7.3 Overall approach

Our work relies on a sampling-based planner called SYCLOP [99]. It poses few constraints on the underlying dynamics of the robot. SYCLOP works by automatically defining

a decomposition of the workspace, creating an adjacency/abstraction graph, and searching that graph for a high-level guide. A low-level planning layer computes the actual dynamically feasible paths and informs the upper layer for how to assign informative weights to the edges of the abstraction graph, so that new meaningful leads can be recomputed if needed. This is critical for (probabilistic) completeness. SyCLoP has been shown to perform well experimentally with a variety of widely-used sampling-based planners such as RRT [40] and EST [104]. In this chapter, the weights of the abstraction graph are further influenced via cost factors that encode the presence of new information as explained in Section 7.5. Our work does not explicitly utilize a cost *map*, but rather a cost *graph* between regions that form an arbitrary decomposition of the configuration space. The costs in our approach encode both the quality of potential viewpoints as well as the travel time required to reach them with a car-like robot.

7.4 Problem Definition

The problem addressed in this chapter can be formulated as follows: given a car-like robot in an initial workspace pose A, compute a dynamically feasible path to region B while optimizing for both low path length and high information gain about an unexpected target, if the latter is detected while executing an initial trajectory that leads to B. Getting to B is considered the robot's primary objective, while collecting information about a target T is considered its secondary objective. Depending on the application, the information is used to, e.g., classify, recognize, or model the target. The robot is able to detect a target's position when it is within a distance R, but a stochastic model (described below) is used to define when sufficient information about the target has been obtained to, e.g., classify it. Due to differential constraints on the robot's motion, the robot cannot simply drive straight toward

a good viewpoint for T, but needs to carefully plan a path that respects these constraints, avoids obstacles, gets close to T and quickly reaches B.

Because this work relies on the interaction between a sensing framework and a navigation framework, we need to carefully define several terms that we will use to describe our methods. The robot *discovers* the target T when it is within R of it. The robot *senses* the target when a sensor measurement of the target is deemed to be of high quality, as described in Section 7.4.2. The mission is *complete* when the robot arrives at B. The mission is *successful* when it is *completed* and the robot has *sensed* every target that was *discovered*.

7.4.1 Robot Model

The robot used in this chapter has the same second-order, car-like dynamics presented in Chapter 6. The velocities and controls are bounded, so the robot cannot stop or change direction instantaneously. As stated before, these dynamics make navigation in environments with obstacles very challenging; the robot cannot rely on purely reactive behaviors and needs to plan ahead. While only car-like robots are considered in this chapter, our planning algorithm is applicable to robots with even more general differential constraints.

As discussed above, due to the complex dynamics, the robot cannot simply follow a potential based on some function of sensor payoff and distance to goal, but needs to *plan* to obtain control inputs that drive it through low-cost areas.

7.4.2 Sensor Model

This chapter uses a simple model of a sensor to provide a proof of concept on the method, but our approach generalizes to other kinds of sensors. There are two types of sensor measurements:

1. Presence and location of a discovered target within distance R from the sensor/robot, and

2. Determination if a high-quality sensor measurement of the target has been achieved.

Both measurements are obtained once every planning cycle as described in Section 7.5. High-quality measurements (type 2) are more complicated because the modeled sensor parameters must be taken into account.

Our sensing model is similar to the models that can be found in field robotics research [105]. We model the event of acquiring a high-quality measurement as a Gaussian process: the robot could fail to detect a target even if, with a perfect sensor, it is in theory able to do so. This is more realistic than simply choosing a distance cut-off within which a high-quality measurement is always obtained.

We have used the parameters from an amateur-level digital still camera as our sensor. Specifically:

$$\begin{aligned} \text{sensor size} &= \text{APS-C standard} \\ &= 23.6\text{mm} \times 15.7\text{mm} \\ \text{image pixels} &= 4592 \times 3056 \\ \text{focal length} &= 24\text{mm} \\ \text{target pixels} &= 100 \times 100 \end{aligned}$$

Additionally, we need to choose a typical target size, which was arbitrarily set to 1m^2. We assume that the camera is pointed directly at the target once it is within discovery range. For comparison, the environment that the robot is operating in is $400\text{m} \times 400\text{m}$ and the robot is approximately $0.5\text{m} \times 0.25\text{m}$.

Once these parameters are defined, we define a high-quality sensing event (e.g., for classifying a target) as occurring when a sample from a Gaussian distribution centered at 0 has a magnitude greater than 1. The standard deviation σ of this Gaussian takes into account the distance r between the robot and the target as follows:

$$\sigma = \frac{4592}{r \cdot \frac{0.0236}{0.024}} \cdot \frac{3056}{r \cdot \frac{0.0157}{0.024}} \cdot \frac{1}{22500} \approx \frac{980}{r^2} \tag{7.1}$$

The first two terms in Equation 7.1 describe the number of pixels that the modeled camera sensor will capture that are of the target. There are 4592 horizontal pixels, which is divided by r times the horizontal field of view of the lens. The horizontal field of view is computed as the horizontal sensor size of 23.6mm divided by the focal length of 24mm. The result of the first term is the horizontal size of the target in pixels for a single image frame. Vertical size is calculated similarly. The target is assumed to present a square visible region to our sensor to make this calculation easier to understand, however, this is certainly not a requirement. Once the total number of target pixels in a particular image frame is computed, we compare it to our set goal of $22,500$ target pixels. The factor of $22,500$ pixels is chosen using the following heuristic: suppose that the pixels on target can be robustly identified using foreground-background subtraction. Then, an affine-invariant feature extraction algorithm (e.g., SIFT [106]) can be employed on these pixels for robust target detection. For highly textured images, it is typical to obtain around 100 reliable SIFT feature descriptors in a 100×100 pixel image [107], plus we add a buffer for the fact that not all images are highly textured and therefore desire a 150×150 image. Therefore, at a range of approximately 30m, $\sigma = 1$, and the robot has a 32% chance to obtain a high-quality measurement of the target. When the robot gets closer, success probability increases as r^2, and of course decreases in a similar fashion as the robot moves farther away. There is a hard cutoff at R, assuming that if the robot cannot tell the location of the target, it cannot be expected to

classify it either.

7.5 Algorithm

To build a solution framework, we extend a previously proposed sampling-based planner called SyCLoP [99], implemented in the Open Motion Planning Library (OMPL) [47]. The use of SyCLoP is critical to our solution because its structure allows information to flow between a high-level discrete planning layer and low-level continuous sampling-based planning layer. An illustration is offered in Figure 7.2. The discrete layer of SyCLoP can be used to encode information about coarse characteristics of the workspace (e.g., where the obstacles are) but also, as shown in this chapter, other information such as the information gathered by sensors. A critical advantage of SyCLoP is that it incorporates a structured way to pass information from the discrete layer to the continuous layer and vice-versa, an advantage that boosts its performance as shown in [99]. In the scenario investigated in this chapter, the bidirectional information flow enables the motion planner to take into account both reachability and sensor information at the same time. Our implementation of SyCLoP uses RRT [40] as the underlying continuous planner.

The inputs to our modified SyCLoP are a motion planning problem with dynamics (workspace obstacles, current and goal positions), and expected sensor informativeness of regions in the workspace supplied by the sensor model. SyCLoP and the sensor model operate in a replanning loop as explained in [62]. The duration of each iteration of the loop is called the planning cycle. During each cycle the robot executes the plan from the previous cycle, while it plans a new path to the destination with the sensor measurements passed to SyCLoP at the end of the previous planning cycle. The sensor model estimates

help guide the robot to interesting areas as described below. The replanning loops continues until the mission is complete.

In the original SyCLoP, the discrete guide (or high-level plan as indicated in Figure 7.2) is recomputed at fixed small intervals to take into account new reachability information "discovered" by the analysis done by the continuous sampling-based motion planner. The discrete model is typically chosen to be a 2D grid decomposition of the workspace. A directed acyclic graph (DAG) is then constructed by the high-level discrete planner between adjacent regions of the decomposition, starting from the region that the robot is currently in. Edge weights are assigned to this graph of adjacent regions, starting with all equal values. These edge weights will be modified by the progress estimates that the sampling-based planner provides. A guide path (high-level plan) in the discrete model is computed. The sampling-based motion planning tree is mapped to the regions of the discrete model, and tree nodes in regions further along the guide path are more likely to be selected for a exploration by the sampling-based planner. This exploration runs RRT for k iterations, where random samples are only drawn from the subset of the state space that projects to the selected region. Starting from these samples, RRT operates without modification. The same discrete guide path is sampled for a region to expand from in this manner 100 times. After the total of $100k$ RRT expansions, the reachability information is updated in the progress estimation module, changing the weights on the discrete layer, and a new guide path is computed, taking these new weights into account. SyCLoP repeats this series of steps until the end of the planning cycle has been reached. The discrete guide path on the DAG is computed by a shortest path algorithm 95% of the time, and by a random depth first search, ignoring edge weight, the other 5%. All SyCLoP internal parameters were left at their defaults. For additional details on the inner workings of SyCLoP see [99].

We incorporate predicted sensor payoff in the above scheme by adding a new multi-

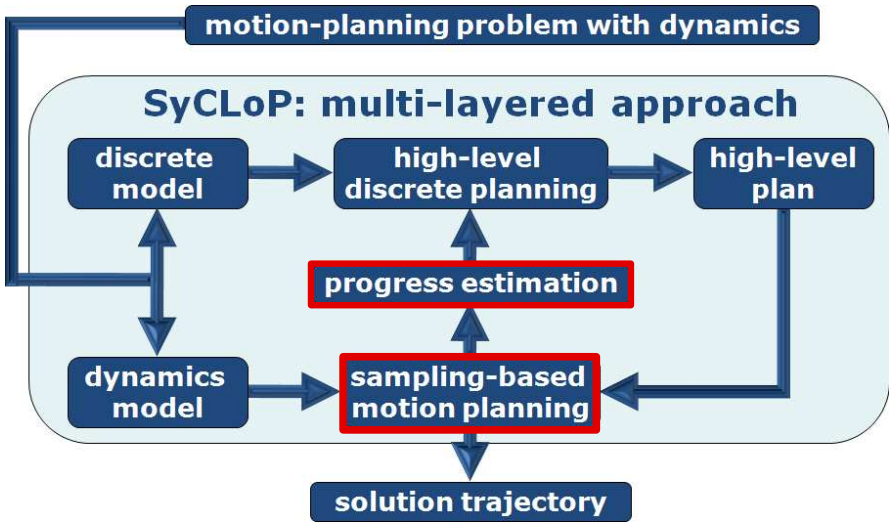


Figure 7.2 : Diagram illustrating the interaction between the modules in the SyCLoP planner. The highlighted central modules have been modified in incorporate information about the expected sensor payoff. Adapted from [99].

plicative edge weight factor in the progress estimation module for the discrete layer. The same multiplicative factor is used for the distance function in the continuous sampling-based motion planning. This new factor causes the discrete guide path to prefer regions of interest, while simultaneously considering reachability. The idea behind this multiplier, simply called *factor* from here on, is to allow the incorporation of a secondary objective, while the planner still only explicitly considers the primary objective. The secondary objective is seen as an online optimization problem, where *factor* determines the relative costs of poor performance between the primary and secondary objectives. It typically causes a deviation of the robot from its original path.

Each planning cycle, a new sensor measurement is available. This is used to modify the edge weights of the discrete model and help guide the robot to areas where the sensor is likely to be most effective. The sensor model associates a payoff value to each region of the discrete model. The payoff of a region is calculated as the estimated σ value of the sensor model, as described in Section 7.4.2, as if the robot were at the center of the region,

and normalized to be between 0 and 1. A payoff of 0 means that no sensor information is expected and 1 means that this is the best possible location to take a sensor reading. If no anomaly is in range, the sensor model reports the same value everywhere. In this case, *factor* will be equal across every region and not influence SyCLoP at all. Otherwise, *factor* will be calculated as in Equation 7.2.

$$factor = \left(1.5 - \frac{\text{payoff}\,[region1] + \text{payoff}\,[region2]}{2}\right)^{c} \tag{7.2}$$

A region with sensor payoff greater than 0.5 causes *factor* to be less than one, and therefore the DAG edge will have a decreased weight. Regions where the sensor payoff is less than 0.5 will yield an increase in edge weight. The values of *factor* fall in the range $[0.5^c, 1.5^c]$. The c parameter controls how important the sensor information is compared to path length and reachability. Another way of looking it is that c controls the relative importance of the secondary target with respect to the primary target. There is no easy way to decide this automatically and, thus, c has to be specified by the user. Although in our sensing model the payoffs do not change other than when targets enter or leave the maximum range, there is no restriction against a sensor model that provides different payoffs for every planning cycle.

Finally, to bias the sampling-based motion planner towards informative paths, the distance function used in this motion planner is similarly multiplied by *factor*, turning it into a problem-specific cost function. This is because the discrete guide may encourage the continuous layer in the correct direction, but the randomized continuous layer finds and selects a path that is closer to the goal than one that acquires extra sensor information. By weighting distance as $distance \times factor$, we ensure that the distance from the goal is correctly balanced against the potential payoff of greater sensor information at both planning levels.

The exponent c is hence a control parameter of the system, where $c = [0, \infty)$. This is used to define how much the sensor information influences SyCLoP by specifying the balance of how much sensor information is worth compared to extra distance traveled. $c = 0$ reduces to the case where sensor information is not utilized in the planning, and we use this for a baseline benchmark.

The output at the end of each planning cycle is a sequence of control inputs for the robot that last for the duration of the next planning cycle. The robot then begins executing that plan and takes a new sensor measurement. For the first planning cycle, the robot does not move, and all subsequent cycles follow until the robot arrives at the destination.

7.6 Experimental Results

To provide proof of concept, we change c in our scenario. As the exponential control parameter c increases, we expect to see that the robot takes longer to reach the goal, but successfully senses the target more often.

In each experimental run below, the robot executes a replanning loop with cycle time of 2s. The world model, robot, and visualization utilize the interprocess communication infrastructure available in ROS [108].

Figure 7.3 shows the environment that was simulated with three paths overlaid on it. The first, with $c = 1.0$, shows no meaningful deviation from the $c = 0.0$ case. We do not show $c = 0.0$ because of the significant overlap it would have. The second, with $c = 1.7$, is a path with clear deviation from the case with little or no sensor information, but still does not deviate too far from the initial path of the robot. Finally, the third path, with $c = 2.0$, has a very large deviation, where the planner leads the robot almost to the target. When the

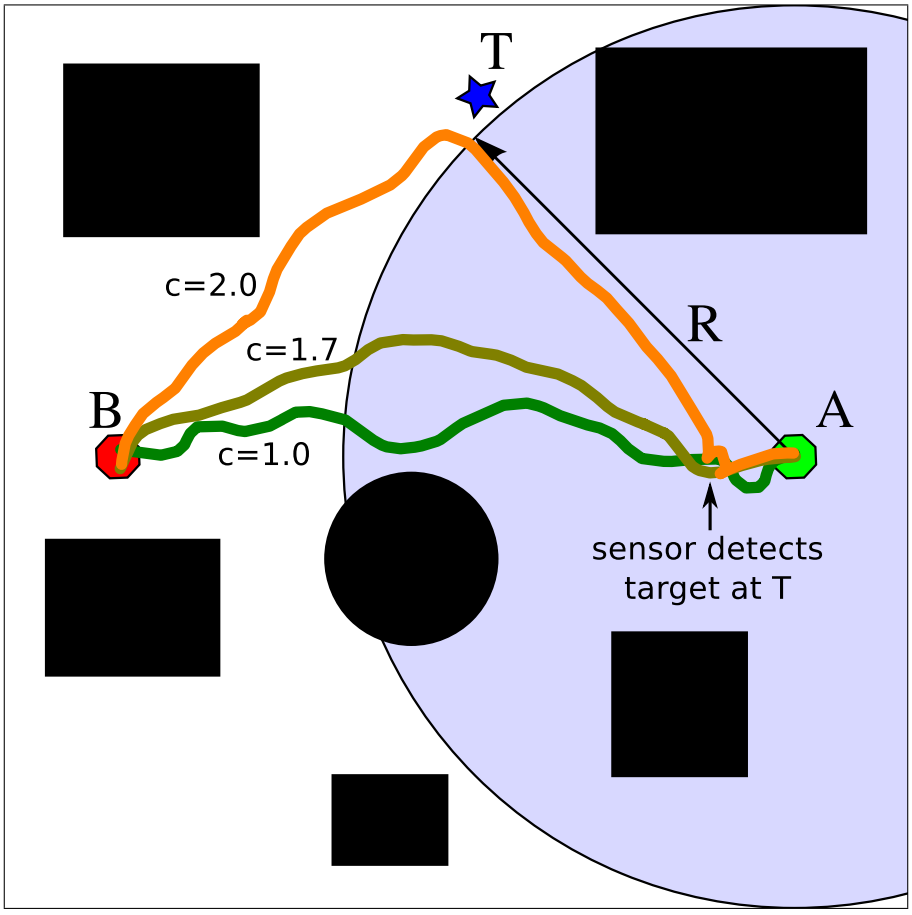


Figure 7.3 : Representative execution paths of the robot with $c = 1.0, 1.7, 2.0$. The robot starts at position A, needs to travel to destination B, and discovers a secondary objective to sense a target at position T.

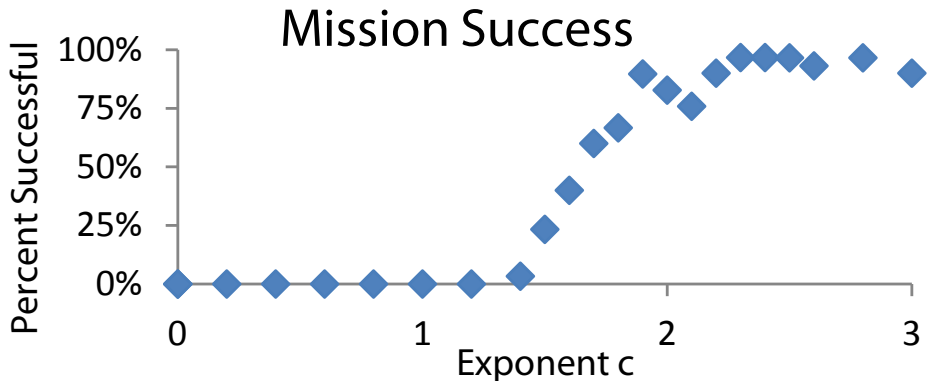


Figure 7.4 : The percentage of successful missions, leveling off at approximately $c = 2.2$

robot gets close, it has a very high probability of sensing the target as defined in Section 7.4.2. Once the sensor model detects a high quality measurement, it no longer considers the target to be an anomaly, and reports that no region in the workspace is expected to provide sensor information. Thus, after a success, the subsequent replanning operations head to the goal without further influence by the sensor.

An important graph to present is Figure 7.4, showing the success of missions as the c exponent is varied across different mission executions. Each data point represents the mean of 30 independent runs. As can be seen, the average mission success probability increases when we increase c to make the expected sensor information more valuable. It is interesting to note that success rate is near 100% at $c \approx 2.0$ and thus does not increase at higher c values. This is because the system already deviates enough to acquire good sensor information, so more deviation does not help.

The data shows that under the Gaussian sample sensor criteria, the robot is never successful at c values under 1.5. Above 1.5, success rates very quickly jump to 90%. Upon observing this jump, more experiments were run at intermediate values. It is still a very fast jump, and the authors plan to investigate whether this behavior is a property of the environment, the weighting factor, the robot dynamics, or another unforeseen interaction.

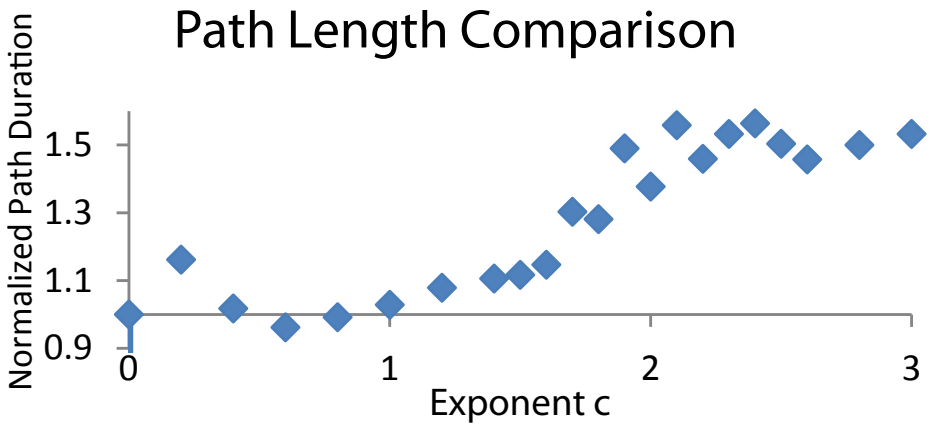


Figure 7.5 : Higher c values result in higher completion times, as expected.

We can see from Figure 7.5 that, as expected, modifying the distance function and DAG weights guide the robot along longer paths. At around $c = 2.0$ and above, the system is heading directly for the target until the target has been in a high quality sensor measurement and then it heads away towards the destination. This causes a plateau in the completion time, as the sensor values stop influencing the behavior of the planner once the target has been determined to be successfully sensed. The Euclidean distance between the start and destination is 300m, while the distance from start to target to destination is $\approx$ 424m. The plateau seen in this graph is at approximately 1.5×, which is near the theoretically expected $\frac{424}{300} \approx 1.42\times$, if the robot could turn instantly. At the intermediate value of $c = 1.7$ we observe that the average deviation is approximately 1.3×, and the system is successfully sensing the target 60% of the time. At a high value of $c = 2.0$ we see that the average deviation is approximately 1.4×, and the system is successfully sensing the target more than 80% of the time. This makes sense in the context of success probability changing as $\frac{1}{r^2}$.

To demonstrate the generality of our approach, we also simulated the same start, goal and target locations in a different workspace. There, the obstacles are much harder to navigate around for a car-like robot. Figure 7.6 shows solution paths for the $c = 0$ and

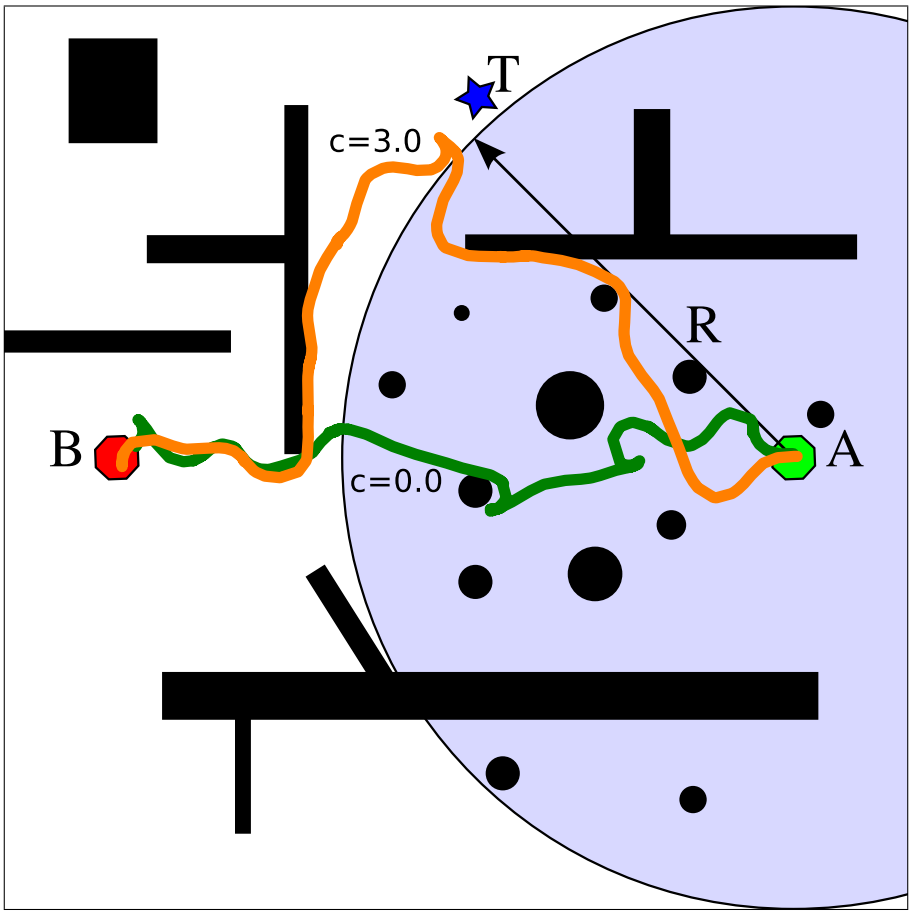


Figure 7.6 : A complex environment, with sensor range (R), start (A), destination (B) and target location (T) marked, as well as two example execution paths.

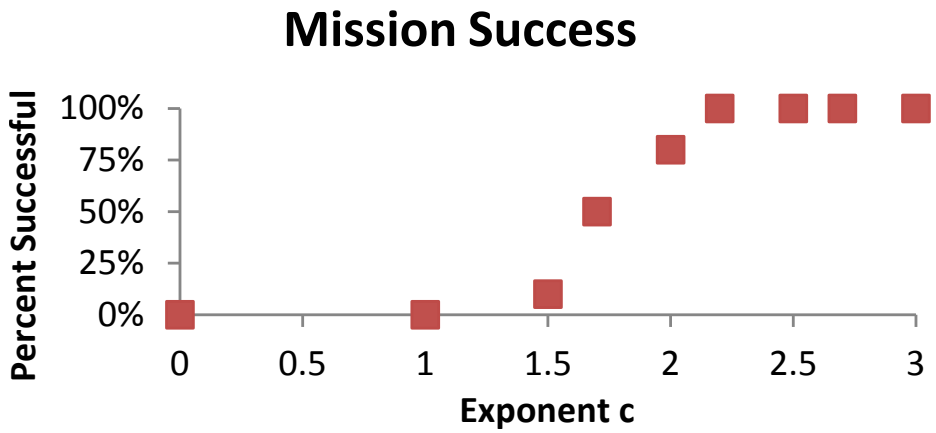


Figure 7.7 : A very similar trend across c values can be seen when success rates are plotted in the complex environment.

$c = 3.0$ cases. In both paths, the car-like robot has to make several changes in direction, which are costly because the robot reverses direction and turns to a new heading. The system does not collide with any obstacles although it does get quite close—the line drawn along the path is thicker than the robot is wide. Figure 7.7 shows a very similar trend as in the simpler environment, where each data point is the success rate across 10 executions from start to destination.

7.7 Summary

We have developed a structured way of taking into account secondary sensing objectives that are not completely specified when the robot starts its mission. Depending on how important the secondary objective is, a parameter c can be used for the robot to automatically adjust its path an appropriate amount to capture sensor information about the target once it has been discovered. More secondary objectives could be considered in a number of ways, e.g., by making the payoff of a region the weighted sum of information gain related to each objective or by making *factor* a weighted sum of terms for each objective.

The method presented here can be improved by incorporating more realistic sensor models. This can be done in a fairly straightforward fashion if sensor values can be translated into payoff values for the planner.

Chapter 8

Discussion

This thesis has presented several tasks for robotic planning under uncertain information and solution frameworks for each of them. The main goal of this thesis was to develop computational techniques to extend the range of tasks and size of problems that could be solved when formulated as a POMDPs. In general, the computational complexity involved in solving a POMDP limits the utility of POMDPs, and our abstraction techniques have demonstrated that they extend these limits.

We have presented three separate methods to extend the utility of the POMDP problem formulation through careful abstraction of the task considered. These three techniques for reducing the computational complexity of the solution were developed to utilize the structure inherent in each of several robotic planning problems. Each technique was implemented, and an empirical evaluation of each method in turn showed that they extend the range of solvable POMDP tasks.

The first method abstracted the action space of the POMDP. This was accomplished by only considering a subset of the actions available. A solution on the subset of actions was used to inform the construction of a model that included additional actions only where they could be expected to improve the policy. The new POMDP with a slightly larger subset of actions was then solved. It was shown that, given a specific time budget, solving the pair of simplified POMDPs yielded an action policy that could gain significantly higher reward than the exact POMDP modeling the same task.

Continuing on this success, we then developed a method to abstract the action model (action space and state space) in a much more drastic fashion. In the previous method, a policy on an abstracted model could always be executed on the exact model because the only abstraction was to take a subset of the actions. This new framework was also designed specifically for continuous state spaces. Another significant change from the previous technique is that only one abstracted PODMP is solved, rather than a progression of model refinements. Due to the significant changes to the action model in this formulation, a policy computed with this abstraction of a POMDP cannot be executed with the underlying motion constraints. It is shown through several experiments across several tasks that utilizing a state-of-the-art robotic planning system in a replanning loop is an effective technique to address this action model mismatch. Again, it was found that considering the complete model directly in a POMDP makes the computation required intractable and cannot solve several tasks that the abstracted model can.

Each of these techniques has been built around reducing the effect of the curse of dimensionality. The final abstraction technique introduces the notion that there may be natural boundaries between subtasks in many robotic problems, particularly ones that have long time horizons. This new method requires a specific sensor that can achieve bounded localization at natural boundary regions. Given this new sensor type, it is shown that "forgetting" the history of beliefs at each of these regions can be an effective way to break the curse of history. Smaller subtasks are each solved with a POMDP and combined at the global level using a fully-observable MDP, defined across the bounded observability regions. This MDP+POMDP framework allows the solution of tasks that are well outside the reach of modern POMDP solvers, even if the global POMDP formulation is also given access to this new sensor type.

The POMDP framework cannot currently be used to directly address these tasks due to

the computational complexity involved. However, the techniques presented in this thesis allow the consideration of the underlying general sensing and expected reward optimization task using an abstracted POMDP. Furthermore, each technique to abstract POMDPs includes a method to apply the resulting abstract policy back to the true underlying problem. Although each technique cannot guarantee optimality, it is shown empirically that given reasonable time limits, the solution to the abstracted model is a much better approximation of an optimal policy than attempting to solve the true problem directly with a larger POMDP for the same amount of time. By increasing the size of the problem instances that can be solved with POMDPs, we have taken a step towards pushing this powerful theoretical framework further into the realm of applicability into practical, physical robotics.

Finally, two additional tasks with perfect localization and sensing were tested, using more focused techniques that do not utilize the general POMDP formulation. The first of these tasks was navigating to acquire the unknown sensor information in a multi-robot team with a fully-known task. The second task was single-robot navigation with sensing, and the task was partially unknown at the outset of execution. For each, a solution was developed and experimentally validated.

8.1 Open Problems

Robotic planning with uncertain information is a broad area of research. It is an open question if directly solving the POMDP is the appropriate technique, as opposed to reinforcement learning or neural nets that are explicitly learning a policy without full knowledge of the underlying POMDP model [76, 75, 34, 33]. In a very real sense, the techniques presented in this thesis sit in a middle ground – rather than directly solve the POMDP or perform model-free policy learning, we have built model abstractions to capture the essence

of the task and then solve these abstracted models. Another open challenge is to algorithmically characterize when the various abstraction techniques can be successfully applied. Following on this, if tasks are amenable to multiple abstraction techniques, which should be applied – or perhaps they could all be applied (which opens the question of if abstraction order matters).

It is also a very interesting question to see if a direct comparison of reinforcement learning techniques using neural nets [76, 75] or policy graphs [34, 33] provides any insight into the structure of robotic POMDP tasks. Finally, developing a more structured notion of transfer learning as it is known in the AI community could be a useful technique to apply to POMDPs. Rather than the simple bootstrapping in Chapter 3, or PBPT [61], or room equivalence in Chapter 5, we are asking about a general formulation that can encode the exactly how a policy for a particular task may apply later, in a similar situation (and defining a similarity measure for tasks is itself an interesting problem). In this scenario, it might be useful or even critical to use a representation of the policy designed for modification, as in the reinforcement learning techniques.

Bibliography

[1] J. Blanco, J. Fernandez-Madrigal, and J. Gonzalez, "A Novel Measure of Uncertainty for Mobile Robot SLAM with Rao Blackwellized Particle Filters," *Intl. Journal of Robotics Research*, vol. 27, no. 1, pp. 73–89, Jan. 2008. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364907082610

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. February 1998, pp. 99–134, May 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437029800023Xhttp://linkinghub.elsevier.com/retrieve/pii/S000437029800023X

[3] S. Zhang and M. Sridharan, "Active visual sensing and collaboration on mobile robots using hierarchical POMDPs," in *AAMAS*, 2012, pp. 181–188. [Online]. Available: http://dl.acm.org/citation.cfm?id=2343602

[4] L. P. Kaelbling and T. Lozano-Perez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, Jul. 2013. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364913484072

[5] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," in *Intl. Symp. on Robotics Research*, 2009.

[6] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards robotic assistants in nursing homes: Challenges and results," *Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 271–281, Mar. 2003. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0921889002003810

[7] C. Papadimitriou and J. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987. [Online]. Available: http://www.jstor.org/stable/10.2307/3689975

[8] P. Sosík and A. Rodríguez-Patón, "Membrane computing and complexity theory: A characterization of PSPACE," *Journal of Computer and System Sciences*, vol. 73, no. 1, pp. 137–152, Feb. 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0022000006001048

[9] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 3rd ed. Prentice hall Englewood Cliffs, 2009, vol. 74.

[10] John L. Kelley, *General Topology*. Springer, 1975.

[11] E. J. Sondik, "The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs," *Operations Research*, vol. 26, no. 2, pp. 282–304, Apr. 1978. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/opre.26.2.282

[12] T. Lee and Y. J. Kim, "GPU-based Motion Planning under Uncertainties using POMDP," in *IEEE Intl. Conf. on Robotics and Automation*, Karlsruhe, Germany, 2013. [Online]. Available: http://graphics.ewha.ac.kr/GPOMDP/Data/GPOMDP2013.pdf

[13] M. Hauskrecht, "Value-Function Approximations for Partially Observable Markov Decision Processes," *Journal of Articial Intelligence Research*, vol. 13, pp. 33–94, Jun. 2000. [Online]. Available: http://arxiv.org/abs/1106.0234

[14] Z. Zhang and X. Chen, "Accelerating Point-Based POMDP Algorithms via Greedy Strategies," *Simulation, Modeling, and Programming for Autonomous Robots*, vol. 6472, pp. 545–556, 2010. [Online]. Available: http://www.springerlink.com/index/DJ61P5304196208V.pdf

[15] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Uncertainty in Artificial Intelligence*, Edinburgh, Scotland, Jul. 2005. [Online]. Available: http://arxiv.org/abs/1207.1412http://uai.sis.pitt.edu/papers/05/p542-smith.pdf

[16] J. M. Porta, N. Vlassis, and P. Poupart, "Point-based value iteration for continuous POMDPs," *Journal of Machine Learning Research*, vol. 7, no. Nov, pp. 2329–2367, 2006. [Online]. Available: http://digital.csic.es/bitstream/10261/30568/1/doc1.pdf

[17] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Conf. on Uncertainty in Artificial intelligence*, Banff, Canada, 2004, pp. 520–527. [Online]. Available: http://dl.acm.org/citation.cfm?id=1036906

[18] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, 2008. [Online]. Available: http://www.roboticsproceedings.org/rss04/p9.pdf

[19] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Intl. Joint Conf. on Artificial Intelligence*, vol. 18, 2003,

pp. 1025–1032. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.1777&rep=rep1&type=pdf

[20] Z. Lim, D. Hsu, and W. S. Lee, "Monte Carlo Value Iteration with Macro-Actions," in *Advances in Neural Information Processing Systems*, 2011. [Online]. Available: https://www-new.comp.nus.edu.sg/$\sim$leews/publications/nips2011.pdf

[21] J. S. Dibangoye, A.-I. Mouaddib, and B. Chai-draa, "Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs," in *Intl. Conf. on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, 2007, pp. 569–576. [Online]. Available: http://dl.acm.org/citation.cfm?id=1558013.1558092

[22] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[23] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping POMDPs," in *IEEE Intl. Conf. on Robotics and Automation*, Apr. 2007, pp. 4685–4692. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4209819

[24] N. Roy and S. Thrun, "Coastal Navigation with Mobile Robots," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1043–1049, 2000.

[25] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, "Online Planning Algorithms for POMDPs," *Journal of Artificial Intelligence Research*, vol. 32, no. 2, pp. 663–704, Jul. 2008. [Online]. Available: http://www.aaai.org/Papers/JAIR/Vol32/JAIR-3217.pdf

[26] J. Ballesteros, L. Merino, M. A. Trujillo, A. Viguria, and A. Ollero, "Improving the Efficiency of Online POMDPs by using Belief Similarity Measures," in *IEEE Intl. Conf. on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 1784–1790.

[27] S. Ross and B. Chaib-Draa, "AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs," in *Intl. Joint Conf. on Artificial Intelligence*, 2007, pp. 2592–2598. [Online]. Available: http://www.aaai.org/Papers/IJCAI/2007/IJCAI07-417.pdf

[28] J. van den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using iterative local optimization in belief space," *Intl. Journal of Robotics Research*, vol. 31, no. 11, pp. 1263–1278, Sep. 2012. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364912456319

[29] J. Van den Berg, S. Patil, and R. Alterovitz, "Efficient approximate value iteration for continuous Gaussian POMDPs," in *AAAI Conf. on Artificial Intelligence*, 2012, pp. 1832–1838. [Online]. Available: http://www.aaai.org/ocs/index.php/aaai/aaai12/paper/download/5159/5340

[30] A.-a. Agha-mohammadi, S. Chakravorty, and N. M. Amato, "FIRM: Feedback controller-based information-state roadmap - A framework for motion planning under uncertainty," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, vol. 1, no. 1, Sep. 2011, pp. 4284–4291. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6048702

[31] A. Undurti and J. P. How, "An online algorithm for constrained POMDPs," in *IEEE Intl. Conf. on Robotics and Automation*. IEEE, May 2010, pp. 3966–3973. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5509743

[32] M. Wang and R. Dearden, "Improving Point-Based POMDP Policies at Run-Time," in *nternational Joint Conference on Artificial Intelligence*, Middlesbrough, UK,

August 2013. [Online]. Available: http://ijcai.org/papers13/Papers/IJCAI13-354.pdf

[33] C. Cai, X. Liao, and L. Carin, "Learning to explore and exploit in pomdps," *Advances in Neural Information Processing Systems*, pp. 1–9, 2009. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2009_1107.pdf

[34] X. Liao, H. Li, R. Parr, and L. Carin, "Regionalized policy representation for reinforcement learning in POMDPs," *The Snowbird Learning Workshop*, pp. 1–2, 2007. [Online]. Available: http://snowbird.djvuzone.org/2007/abstracts/155.pdf

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[36] T. Jaakkola, S. P. Singh, and M. I. Jordan, "Reinforcement learning algorithm for partially observable Markov decision problems," *Advances in Neural Information Processing Systems: 7*, vol. 7, p. 345, 1995.

[37] J. Kober, J. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, Aug. 2013. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364913495721

[38] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1987.

[39] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[40] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/02783640122067453

[41] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=508439

[42] C. M. Clark, S. M. Rock, and J.-C. Latombe, "Motion planning for multiple mobile robots using dynamic networks," in *IEEE Intl. Conf. on Robotics and Automation*, Sep. 2003, pp. 4222–4227.

[43] K. E. Bekris, K. I. Tsianos, and L. E. Kavraki, "Safe and distributed kinodynamic replanning for vehicular networks," *ACM/Springer Mobile Networks and Applications*, vol. 14, no. 3, pp. 292–308, 2009.

[44] D. Grady, K. E. Bekris, and L. E. Kavraki, "Asynchronous distributed motion planning with safety guarantees under second-order dynamics," in *Workshop on the Algorithmic Foundations of Robotics*, 2010.

[45] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Trans. on Robotics*, vol. 26, no. 4, pp. 635 –646, Aug. 2010.

[46] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Intl. J. of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[47] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012. [Online]. Available: ompl.kavrakilab.org

[48] L. Jaillet, J. Cortes, and T. Siméon, "Transition-based RRT for path planning in continuous cost spaces," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Sep. 2008, pp. 2145–2150. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4650993http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4650993

[49] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://msl.cs.uiuc.edu/planning/

[50] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation*, May 2006, pp. 1243–1248.

[51] O. Brock and O. Khatib, "Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths," in *IEEE International Conference on Robotics and Automation*. IEEE, 2000, pp. 550–555. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=844111

[52] K. Hauser, "Adaptive Time Stepping in Real-Time Motion Planning," in *Workshop on the Algorithmic Foundations of Robotcs*. Springer, 2011, pp. 139–155. [Online]. Available: http://www.springerlink.com/index/K1350J4820K1M148.pdf

[53] R. B. Rusu, I. A. Sucan, B. P. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-Time Perception-Guided Motion Planning for a Personal Robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, 2009, pp. 4245–4252.

[54] D. Grady, M. Moll, and L. E. Kavraki, "Automated Model Approximation for Robotic Navigation with POMDPs," in *IEEE Intl. Conf. on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 78–84.

[55] O. Madani, S. Hanks, and A. Condon, "On the Undecidability of Probabilistic Planning and Infinite-Horizon Partially Observable Markov Decision Problems," in *National Conf. on Artificial Intelligence*, no. Littman 1997, 1999.

[56] W. S. Lovejoy, "Computationally Feasible Bounds for Partially Observed Markov Decision Processes," *Operations Research*, vol. 39, no. 1, pp. 162–175, Jan. 1991. [Online]. Available: http://www.springerlink.com/index/10.1007/BF02055574http://or.journal.informs.org/cgi/doi/10.1287/opre.39.1.162

[57] A. Foka and P. Trahanias, "Real-time hierarchical POMDPs for autonomous robot navigation," *Robotics and Autonomous Systems*, vol. 55, no. 7, pp. 561–571, Jul. 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0921889007000279

[58] J. Pineau, N. Roy, and S. Thrun, "A hierarchical approach to POMDP planning and execution," in *Workshop on Hierarchy and Memory in Reinforcement Learning*, 2001. [Online]. Available: http://www.cs.mcgill.ca/$\sim$jpineau/files/jpineau-icml01.pdf

[59] H. Kurniawati, T. Bandyopadhyay, and N. M. Patrikalakis, "Global motion planning under uncertain motion, sensing, and environment map," *Autonomous Robots*, Jun. 2012. [Online]. Available: http://www.springerlink.com/index/10.1007/s10514-012-9307-y

[60] R. Kaplow, A. Atrash, and J. Pineau, "Variable resolution decomposition for robotic navigation under a POMDP framework," in *IEEE Intl. Conf. on Robotics and Automation*, May 2010, pp. 369–376. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5509188

[61] H. Kurniawati and N. M. Patrikalakis, "Point-based policy transformation : Adapting policy to changing POMDP models," in *Workshop on the Algorithmic Foundations of Robotics*, 2012, pp. 1–16.

[62] K. E. Bekris, D. K. Grady, M. Moll, and L. E. Kavraki, "Safe distributed motion coordination for second-order systems with different planning cycles," *International Journal of Robotics Research*, vol. 31, no. 2, pp. 129–149, 2012.

[63] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," in *International Symposium of Robotics Research*, ser. Springer Tracts in Advanced Robotics, S. Thrun, R. Brooks, and H. Durrant-Whyte, Eds., vol. 28. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 239–253–253. [Online]. Available: http://www.springerlink.com/content/34h70073l47vq206/

[64] H. Bai, D. Hsu, W. S. Lee, and V. Ngo, "Monte Carlo value iteration for continuous-state POMDPs," in *Workshop on the Algorithmic Foundations of Robotics*, 2010, pp. 175–191.

[65] H. Bai, D. Hsu, and W. S. Lee, "Planning How to Learn," in *IEEE Intl. Conf. on Robotics and Automation*, Karlsruhe, Germany, 2013. [Online]. Available: http://bigbird.comp.nus.edu.sg/pmwiki/farm/motion/uploads/Site/icra13.pdf

[66] S. Candido and S. Hutchinson, "Minimum uncertainty robot navigation using information-guided POMDP planning," in *IEEE Intl. Conf. on Robotics and*

Automation, May 2011, pp. 6102–6108. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5979695

[67] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957. [Online]. Available: http://www.jstor.org/stable/2372560

[68] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990. [Online]. Available: http://www-personal.acfr.usyd.edu.au/spns/motion/ReedsShepp1990.pdf

[69] H. A. Yanco, "Wheelesley: A robotic wheelchair system: Indoor navigation and user interface," in *Assistive technology and artificial intelligence*. Springer, 1998, pp. 256–268.

[70] E. Prassler, J. Scholz, and P. Fiorini, "A robotics wheelchair for crowded public environment," *Robotics & Automation Magazine, IEEE*, vol. 8, no. 1, pp. 38–45, 2001.

[71] R. E. Kalman and Others, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[72] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proc. Adaptive Systems for Signal Processing, Communications, and Control Symposium*. IEEE, 2000, pp. 153–158. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=882463

[73] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. Wan, "The unscented particle filter," in *NIPS*, 2000, pp. 584–590.

[74] A. Doucet, N. De Freitas, and N. Gordon, "An introduction to sequential Monte Carlo methods," in *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 3–14.

[75] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving Deep Memory POMDPs with Recurrent Policy Gradients," in *Artificial Neural Networks ICANN 2007*, J. M. de Sá, L. A. Alexandre, W. Duch, and Danilo Mandic, Eds. Springer Berlin Heidelberg, 2007, ch. Solving De, pp. 697–706.

[76] R. P. N. Rao, "Decision making under uncertainty: a neural model based on partially observable markov decision processes." *Frontiers in computational neuroscience*, vol. 4, p. 146, Jan. 2010. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2998859&tool=pmcentrez&rendertype=abstract

[77] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments," in *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, ACM. Philadelphia, PA, USA: ACM, 2013, pp. 353–362.

[78] D. K. Grady, M. Moll, C. Hegde, A. C. Sankaranarayanan, R. G. Baraniuk, and L. E. Kavraki, "Multi-Robot Target Verification with Reachability Constraints," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, IEEE. College Station, TX: IEEE, 2012, pp. 1–6.

[79] J. E. Banta, L. R. Wong, C. Dumont, and M. A. Abidi, "A next-best-view system for autonomous 3-D object reconstruction," *IEEE Trans. Systems, Man and Cybernetics,*

Part A, vol. 30, no. 5, pp. 589–598, 2000.

[80] R. Pito, "A solution to the next best view problem for automated surface acquisition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 21, no. 10, pp. 1016–1030, 1999.

[81] E. Dunn, J. van den Berg, and J.-M. Frahm, "Developing visual sensing strategies through next best view planning," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Oct. 2009, pp. 4001–4008.

[82] L. Torabi and K. Gupta, "Integrated view and path planning for an autonomous six-dof eye-in-hand object modeling system," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Oct. 2010, pp. 4516–4521.

[83] H. H. González-Baños and J.-C. Latombe, "Navigation strategies for exploring indoor environments," *Intl. J. of Robotics Research*, vol. 21, no. 10-11, pp. 829–848, 2002.

[84] J. Faigl, M. Kulich, and L. Přeučil, "A sensor placement algorithm for a mobile robot inspection planning," *Journal of Intelligent & Robotic Systems*, pp. 1–25, 2010.

[85] R. Grabowski, P. Khosla, and H. Choset, "Autonomous exploration via regions of interest," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Oct. 2003.

[86] C. Laporte and T. Arbel, "Efficient discriminant viewpoint selection for active bayesian recognition," *International Journal of Computer Vision*, vol. 68, no. 3, pp. 267–287, 2006.

[87] J. Velez, G. Hemann, A. Huang, I. Posner, and N. Roy, "Planning to perceive: Exploiting mobility for robust object detection," in *Proc. ICAPS*, 2011.

[88] L. E. Parker, "Path planning and motion coordination in multiple mobile robot teams," in *Encyclopedia of Complexity and System Science*, R. Meyers, Ed. Springer Verlag, 2009.

[89] L. Matignon, L. Jeanpierre, and A. Mouaddib, "Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes," in *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[90] H. Murase and S. Nayar, "Visual Learning and Recognition of 3D Objects from Appearance," *Intl. J. of Computer Vision*, vol. 14, no. 1, 1995.

[91] D. Donoho and C. Grimes, "Image manifolds which are isometric to Euclidean space," *J. Math. Imaging and Computer Vision*, vol. 23, no. 1, July 2005.

[92] B. Horn and B. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[93] C. Liu, "Beyond pixels: exploring new representations and applications for motion analysis," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[94] T. Brox and J. Malik, "Large displacement optical flow: descriptor matching in variational motion estimation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 33, 2010. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2010.143

[95] A. C. Sankaranarayanan, C. Hegde, S. Nagaraj, and R. G. Baraniuk, "Go with the flow: Optical flow-based transport operators for image manifolds," in *Allerton Conference on Communication, Control and Computing*, Allerton, IL/USA, September 2011.

[96] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. Ieee, 2006, pp. 2169–2178.

[97] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. J. of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[98] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[99] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion Planning With Dynamics by a Synergistic Combination of Layers of Planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, Jun. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5477241

[100] R. Rusu, I. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, Oct. 2009, pp. 4245 –4252.

[101] J. R. Bruce and M. M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. 2002 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2002, pp. 2383–2388.

[102] K. E. Bekris and L. E. Kavraki, "Greedy but Safe Replanning under Kinodynamic Constraints," in *IEEE International Conference on Robotics and Automation*. Rome, Italy: IEEE, Apr. 2007, pp. 704–710. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2007.363069

[103] J. Velez, G. Hemann, A. S. Huang, I. Posner, and N. Roy, "Planning to perceive: Exploiting mobility for robust object detection," in *Intl. Conf. on Automated Planning and Scheduling*, 2011.

[104] J. M. Phillips, N. Bedrosian, and L. E. Kavraki, "Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces," in *Proc. 2004 IEEE Intl. Conf. on Robotics and Automation*. New Orleans, LA: IEEE Press, April 2004, pp. 3968–3973. [Online]. Available: http://www.kavrakilab.org/sites/default/files/phillips2004guided-expansive-spaces.pdf

[105] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting wilderness search and rescue using a camera-equipped mini UAV," *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 89–110, Jan. 2008. [Online]. Available: http://doi.wiley.com/10.1002/rob.20226

[106] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. J. of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[107] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1615–1630, 2005.

[108] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS : an open-source Robot Operating System," in *Open-Source Software workshop of the International Conference on Robotics and Automation*, 2009. [Online]. Available: http://ai.stanford.edu/~mquigley/papers/icra2009-ros.pdf