An Extended Kalman Filter Extension of the Augmented Markov Decision Process

by

Peter Hans Lommel

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005 [June 2005]

Author: [illegible]
Department of Aeronautics and Astronautics
May 20, 2005

Certified by: [illegible]
Nicholas Roy, Ph.D.
Assistant Professor, Department of Aeronautics and Astronautics
Thesis Supervisor

Certified by: [illegible]
Marc W. McConley, Ph.D.
Principal Member of the Technical Staff
The Charles Stark Draper Laboratory, Inc.
Thesis Supervisor

Accepted by:
Jaime Peraire, Ph.D.
Professor, Department of Aeronautics and Astronautics
Chairman, Department Committee on Graduate Students
Chair, Committee on Graduate Students

AERO

[This page intentionally left blank.]

An Extended Kalman Filter Extension of the Augmented Markov Decision Process

by

Peter Hans Lommel

Submitted to the Department of Aeronautics and Astronautics
on May 20, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

As the field of robotics continues to mature, individual robots are increasingly capable of performing multiple complex tasks. As a result, the ability for robots to move autonomously through their environments is a fundamental necessity. If perfect knowledge of the robot's position is available, the robot motion planning problem can be solved efficiently using any of a number of existing algorithms. Frequently though, the robot's position can only be estimated using incomplete and imperfect information from its sensors and an approximate model of its dynamics. Algorithms which assume perfect knowledge of the robot's position can still be applied by treating the mean or maximum likelihood estimate of the robot's position as certain. However, unless the uncertainty in the agent's position is very small, this approach is not reliable. In order to perform optimally in this situation, planners, such as the partially observable Markov decision process, plan over the entire set of beliefs (distributions over the robot's position). Unfortunately, this approach is only tractable for problems with very few states. Between these two extreme approaches, however, lies a continuum of possible planners which plan over a subset of the belief space. The difficulty that these planners face is choosing and representing a minimal subset of the belief space which spans the set of beliefs that the robot will actually experience. In this paper, we show that there exists a very natural such set, the set of Gaussian beliefs. By combining an extended Kalman filter with an augmented Markov decision process, we create a path planner which efficiently plans over a discrete approximation of the set of Gaussian beliefs. The resulting planner is demonstrated via simulation to be both computationally tractable and robust to uncertainty in the robot's position.

Thesis Supervisor: Nicholas Roy, Ph.D.
Title: Assistant Professor, Department of Aeronautics and Astronautics

Thesis Supervisor: Marc W. McConley, Ph.D.
Title: Principal Member of the Technical Staff

The Charles Stark Draper Laboratory, Inc.

Acknowledgments

This thesis is the product of so much support, motivation and encouragement, that I can only begin to express my gratitude to the many people who have made it possible.

I would like to thank the C.S. Draper Laboratory for giving me the opportunity to study at one of the worlds greatest technical institutions. Specifically I would like to thank Milt Adams, Brent Appleby, Tim Brand and George Schmidt.

Next I would like to thank my advisors, Nick Roy and Marc McConley, for the many discussions which have made my research enjoyable and from which I learned a great deal, and for the guidance which made this thesis possible.

I would also like to thank the advisors, supervisors and professors who inspired me to continue my academic pursuit into graduate school and then opened the doors so that I could. My thanks go to Mike Elgersma, Sonja Glavaski, Dale Enns, George Papageorgiou, Jorge Tierno, Datta Godbole, Kartik Ariyur, Richard Phillips, Krishnan Mahesh and Eric Feron.

My thanks also go to my classmates and friends who helped me through so many exams and projects and who have made MIT fun: Aron Cooper, Len Wholey, Craig Van Beusekom, Gordon Thompson, Geoff Huntington, Theresia Becker, Rachel Thessin, Alisa Hawkins and Dave Woffinden.

Most of all I would like to thank my family, who never let me doubt myself. It is their undying support and encouragement that gives me the drive and motivation to always strive to be the best. Without my sister, it would have been impossible for me to finish my undergraduate studies. Without my brother, I could not possibly be in graduate school right now. Lastly, and most importantly, I want to thank my girlfriend Louie and our two amazing children Tia and Bryce who have made my experience that much more wonderful. This thesis is for them.

Assignment

Draper Laboratory Report Number T-1518

Peter Hans Lommel　　　　　　　　　　Date

Contents

List of Figures

List of Tables

[This page intentionally left blank.]

Chapter 1

Introduction

There are two basic capabilities that any system must possess in order to perform a task autonomously. First, the system must have some means of knowing what is happening in the world around it (i.e., it must be able to estimate its state with respect to its environment). Second, given the state of the world around it, it must have some way of reasoning about how it should act in order to achieve the task it was designed to perform. This second problem is called the decision making problem and it will be the focus of this thesis. Specifically, we will focus on the path planning problem which is an application of a class of decision making problems called sequential decision making problems. A sequential decision making problem is one in which it may be necessary for the system to perform more than one action sequentially in order to perform the desired task. [27, 21]

1.1 Motivating the Path Planning Problem

As the field of robotics continues to mature, the tasks that a single robot is capable of performing become more numerous as well as more complex. One of the fundamental necessities for performing these tasks is the ability to move autonomously through the environment. For example, Nursebot [22] is capable of providing assistance to elderly patients, but in order to do so it must be able to move autonomously throughout the home. Another example is unmanned air vehicles (UAVs) [1]. Many UAV applications, such as search and surveillance, require the UAV to move autonomously through the environment.

In addition to being a necessity for mobile robots, an effective path planner can serve as a useful aide to humans performing a task in a high stress environment such as firefighting, search and rescue, law enforcement or military missions. Assessing complex risks and costs in these situations can require detailed and time consuming analysis. A path planner that can represent these risks and costs appropriately can help human agents, whose main focus may lie elsewhere, to choose a safe and effective path through their environment. Since our path planner may have applications in robotics or as an aide to human navigation, we will refer to robots and humans collectively as agents in this paper.

The motivation for this research also stems from the fact that virtually every autonomous agent that interacts with its environment requires some form of sequential decision making. Although the path planning problem is one specific class of sequential decision making problems, it is the author's hope that this research will lead to some additional insight into quick approximate solutions for a more general class of problems.

1.2 Uncertainty in Path Planning

Under the assumption that the environment and the agent's state within the environment are fully known, numerous optimal and sub-optimal planning algorithms have been developed [19]. Unfortunately though, a mobile agent moving about in the real world generally has no way of knowing its exact position. In this case, the agent's position is usually estimated using an approximate mathematical model of its dynamics and imperfect information from its sensors. Since an agent's sensing capabilities are limited by cost, space and weight restrictions, the assumption of perfect knowledge of the agent's location is frequently a poor one.

The simplest approach to path planning under uncertainty is to neglect the uncertainty that is modeled by the statistical inference and simply treat the best available estimate of the map and agent location as certain. Intuitively, this approach is quite successful in situations where the uncertainty remains small, but becomes much less reliable as the uncertainty grows. The most complete approach to path planning in an uncertain environment is to represent the uncertainty using probability distributions, or beliefs, and then plan over the entire set of possible beliefs [30, 21, 7, 26]. This approach, however, has complexity that is prohibitive to solving real world problems. The intent of this research is to develop a path planner that is robust to uncertainty but avoids the complexity issues that uncertainty generally dictates.

Uncertainty in the path planning problem can come in two basic forms. The first is uncertainty in the map; the location of everything in the environment other than the agent. The problem of accurately locating objects within an environment is called the mapping problem [31]. The second source of uncertainty in the environment is uncertainty in the location of the agent. The problem of accurately locating an agent within an environment is called the localization, or state estimation, problem [12, 15]. The problem of simultaneous localization and mapping (SLAM) [29, 28] has recently received much attention [31, 33, 32, 23]. In our planner, we will retain the assumption that the map is known, but we will not assume that the agent's position is known. Since accurate maps can be created from resources such as aerial photos and building plans, the assumption of a known map is significantly less restrictive than the assumption of a known agent location. The problem of uncertain maps is not to be neglected though, and it will be a focal area in the future of this research.

1.3 Thesis Overview

1.3.1 Thesis Statement

We first note that as a consequence of the central limit theorem, uncertainty in the position of an agent acting in the real world can be accurately modeled by the set of Gaussian distributions. The thesis of this document, then, is that we can exploit this fact in order to robustly and efficiently plan in the face of uncertainty by formulating an augmented Markov decision process whose states are Gaussian distributions, and describing the evolution of these states using the extended Kalman filter equations.

1.3.2 Document Structure

We begin development of the robust planner in chapter two by describing the planning problem more completely. We create a very generic formulation for the planning problem (without uncertainty) and describe each element of the formulation. We then see how this formulation can be slightly modified to include uncertainty in the outcome of the agent's actions, but still assumes that the state is known at all times. This problem formulation is called a Markov decision process (MDP). We discuss the MDP in detail, including basic algorithms for solving MDPs. We then describe another planning problem formulation, which does not assume that the state is known, called the partially observable Markov decision process (POMDP). The POMDP is able to plan optimally under uncertainty in the agent's position by planning over the set of beliefs (probability distributions over the agent's position). Complexity analysis of the POMDP, however, shows that, for many problems, its exact solution is not computationally tractable. This motivates an approximate solution which we will develop in chapters three and four.

Chapter three discusses an optimal state estimator which is commonly applied in the field of navigation, the extended Kalman filter. The robustness of the extended Kalman filter to real world uncertainty comes despite the fact that it represents uncertainty only in the form of Gaussian distributions. The extended Kalman filter equations will be described in detail, since they will not only be used to estimate the agent's states, but they will be a fundamental part of the path planner developed in chapter four.

In chapter four, we show that by pairing a Markov decision process with an extended Kalman filter, we can exploit the structured uncertainty information provided by the extended Kalman filter. We construct an augmented Markov decision process whose states are Gaussian distributions which reflect the uncertainty information provided to the agent by the extended Kalman filter. The result is a path planner that is robust to uncertainty and is computationally tractable for problems that reflect real world situations.

In chapter five, a simulation is developed for comparing the augmented Markov decision process path planner to a basic Markov decision process path planner.

Chapter 2

The Planning Problem

In this chapter we will provide a detailed discussion of planning under uncertainty in the form of the Markov decision process (MDP) and the partially observable Markov decision process (POMDP). First though, a brief background of the classical planning problem (without uncertainty) will be provided. For a more extensive background in planning, the reader is referred to [27, 19].

The planning problem is made up of a set of states, $\mathbb{S}$, which describe the world in which we are planning, and a set of actions, $\mathbb{A}$, which affect the states of the world. We will describe the effects of the actions by the function $T(\cdot)$:

$$\text{T: } (\mathbb{S} \times \mathbb{A}) \mapsto \mathbb{S} \tag{2.1}$$

so that $T(s, a) \in \mathbb{S}$ is the state that results from having executed action $a \in \mathbb{A}$ from state $s \in \mathbb{S}$.

Now, in order to choose the optimal sequence of actions, we need to define optimality for the planning problem. This is done in terms of an objective function, $z(\cdot)$. Let the solution to the planning problem be described by an ordered list (or vector) containing the optimal sequence of actions. Then define $\mathbb{P}_n$ to be the complete set of such lists of length n:

$$\mathbb{P}_n = \mathbb{A}^n \tag{2.2}$$

and define $\mathbb{P}$ to be the union of all such sets:

$$\mathbb{P} = \bigcup_n \mathbb{P}_n \tag{2.3}$$

Then the objective function, $z(\cdot)$, will then map this set of possible action sequences to the set of real numbers:

$$z : \mathbb{P} \mapsto \mathbb{R} \tag{2.4}$$

This objective function is generally taken to be the sum (or integral) of rewards that are collected as a result of executing each action. In general, these rewards will be a function of the state as well as the action. We will describe these rewards by a function $R(\cdot)$:

$$R : (\mathbb{S} \times \mathbb{A}) \mapsto \mathbb{R} \tag{2.5}$$

So, for some $p \in \mathbb{P}$, we have[1]:

$$z(p) = \sum_{i=1}^{|p|} R(s_i, p_i) \tag{2.6}$$

where:

$$s_1 = \text{initial state, and } s_i = T(s_{i-1}, p_{i-1}) \text{ for } i \in [2, |p|] \tag{2.7}$$

There are two common sources for these rewards. First, desirable states in the state space can be associated with positive rewards and undesirable states can be associated with negative rewards. The second source of rewards may be better described as costs. A cost (or small negative reward) is incurred each time an action is invoked. In general, this cost may depend on the state as well as the action because the action may be more costly to execute from some states than it is from other states.

Once we have defined the state space, the actions and the objective function, the planning problem reduces to choosing the path, or sequence of actions, that maximizes the objective function:

$$\mathrm{p}^* = \arg\max_{p\in\mathbb{P}} \quad \mathrm{z(p)} \tag{2.8}$$

[1]In this report, the operator $|\cdot|$ will apply to vectors and sets, and it will denote the number of elements in the vector or set.

Alternatively, the solution to the planning problem may be represented by an optimal policy. A policy, π, is a mapping from the state space to the action space:

$$\pi : \mathbb{S} \mapsto \mathbb{A} \tag{2.9}$$

The optimal action sequence, p^*, from any initial state in the feasible space can be extracted from the optimal policy, π^*, iteratively:

$$p_i^* = \pi^*(T(s_{i-1}, p_{i-1}^*)) \tag{2.10}$$

where:

$$s_0 = \text{initial state, and } p_0^* = \pi^*(s_0)$$

The planning problem is commonly solved by constructing a connected graph whose nodes are the states and whose arcs are the actions. The arc costs are taken to be the action costs described above, and the graph is then searched, beginning at the initial state, for the lowest cost path to a goal state [19, 9, 18, 20].

2.1 Planning Under Uncertainty

In this section, we will incorporate uncertainty into the planning problem in two steps. First, we will allow uncertainty in the effects of the actions but assume that the state is known with certainty at all times. This type of planning problem is described by a Markov decision process (MDP). Then, building from our discussion of the MDP, we will develop the partially observable Markov decision process (POMDP) in which we finally drop the assumption that the state is known.

2.1.1 Markov Decision Processes

The generic MDP [14, 25] is described by the tuple $\langle \mathbb{S}, \mathbb{A}, T, R \rangle$, where:

$\mathbb{S}$ = the set of possible states.

$\mathbb{A}$ = the set of possible actions.

$T : (\mathbb{S} \times \mathbb{A}) \mapsto \Pi(\mathbb{S})$ is the transition function which maps states and actions to distributions over posterior states. $T(s, a, s')$ is the probability of ending up in state s', given that action a was executed from state s.

$R : (\mathbb{S} \times \mathbb{A}) \mapsto \mathbb{R}$ is a reward function which maps states and actions to real numbers.

The initial state, s_0 is sometimes included in the MDP tuple as well. Note that the only change between the classical planner described in the previous section and the MDP is that the transition function T is now allowed to be stochastic. The classical planner, then, is actually a special case of the MDP.

Before we attempt to solve the MDP we will need to redefine the objective function. The objective function for an MDP is often referred to as the value function (V). Due to the stochastic nature of the MDP, we cannot simply specify its solution as an optimal action sequence (since the transitions are not necessarily deterministic, we do not know what state the agent will end up in after the first action, so the optimal second action depends on the outcome of the first action, etc.). Instead, we will specify the solution as an optimal policy, and the value function will depend on the policy rather than the action sequence. Specifically, the value function, given the policy and evaluated at some initial state s_0, $V_\pi(s_0)$, will be the expected discounted future reward:

$$V_\pi(s_0) = E\Big[\sum_{i=0}^{n-1} \gamma^i R(s_i, \pi(s_i))\Big] \tag{2.11}$$

where:

$\gamma \in (0, 1)$ is the discounting factor

We may think of the discounting factor as the relative value of future rewards as compared to immediate rewards. The motivation for the discounting factor may be physical, the actual value of the rewards may depreciate over time (like money due to inflation), or it may be mathematical as we will see shortly. The upper limit, n, in the sum in equation (2.11) is what is called the horizon length of the MDP. The horizon length may be finite, which means that the agent should maximize the expected reward assuming that exactly n actions will be executed. In this case, the discounting factor, γ, may be less than *or equal to* one, depending on whether the rewards depreciate, and the sum in equation (2.11) will be finite. The horizon length may also be infinite, in which case the agent should maximize the expected reward assuming an indefinite number of actions will be executed. In this case, the discounting factor must be strictly less than one for the sum in equation (2.11) to be finite.

An important concept that arises from the horizon length of the MDP is stationarity of the solution. A stationary optimal policy is independent of time; the optimal policy after having executed zero actions is the same as the optimal policy after having executed k actions, for all k. For an infinite horizon, the number of actions that remain to be executed does not change (it is always indefinite), so there is no reason for the policy to change, therefore the optimal policy is stationary. For a finite horizon however, the optimal behavior when only one action remains may be very different than the optimal behavior when many actions remain, so the optimal policy is non-stationary. For the non-stationary case, the MDP solution is actually a set of policies, $\delta^* = \{\pi_n^*, \pi_{n-1}^*, ..., \pi_2^*, \pi_1^*\}$, where π_i^* is the optimal i step policy.

Now we can take a closer look at equation (2.11), beginning with a one step policy for which (2.11) reduces to:

$$V_{\pi_1}(s) = E[R(s, \pi_1(s))] = R(s, \pi_1(s)) \tag{2.12}$$

We can then evaluate a two step policy using (2.11) and (2.12):

$$\begin{aligned} V_{\pi_2}(s) &= E[R(s, \pi_2(s)) + R(s', \pi_1(s'))] \\ &= R(s, \pi_2(s)) + \gamma \sum_{s' \in \mathbb{S}} T(s, \pi_2(s), s') V_{\pi_1}(s') \end{aligned} \tag{2.13}$$

Then, iterating on (2.13), we get[2]:

$$V_{\pi_n}(s) = R(s, \pi_n(s)) + \gamma \sum_{s' \in \mathbb{S}} T(s, \pi_n(s), s') V_{\pi_{n-1}}(s') \tag{2.14}$$

For an infinite horizon MDP, since we have assumed the optimal policy is stationary, we have $V_{\pi_n}(s) = V_{\pi_{n-1}}(s) = V_\pi(s)$, so:

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathbb{S}} T(s, \pi(s), s') V_\pi(s') \tag{2.15}$$

which is a set of $|\mathbb{S}|$ equations in $|\mathbb{S}|$ unknowns which can be solved for $V_\pi(s)$ for each s.

Based on (2.14) and (2.15) we can develop a set of algorithms for solving MDPs. The algorithms will be shown for the infinite horizon case, but they can be easily generalized to the finite horizon case. The two most basic algorithms are called value iteration and policy iteration. Value iteration works by iteratively improving the value function (pushing it toward the optimal value function) and computing the associated policy. The value iteration algorithm is shown in Table 2.1.

Policy iteration also iteratively improves the value function, but uses a slightly different approach. In policy iteration, we compute the optimal policy at each iteration with respect to the value function from the previous iteration and then re-compute the value function according to this policy. The policy iteration algorithm is shown in Table 2.2.

In Algorithm 2, the function getVfromπ() returns the solution of equation (2.15).

The advantage of value iteration over policy iteration is that each iteration of value iteration has complexity O($|\mathbb{S}|^2|\mathbb{A}|$) and each iteration of policy iteration has complexity O($|\mathbb{S}|^3$) (due to the solution of $|\mathbb{S}|$ linear equations in step 12 of Algorithm 2) and generally $|\mathbb{S}| \gg |\mathbb{A}|$. The advantage of policy iteration over value iteration is that policy iteration converges in less than or equal to the number of iterations that value iteration requires [25].

MDPs can also be solved by formulating them as linear programs [10]. When the environment is not known beforehand, i.e., the transition function and reward function

[2]By maximizing the right hand side of equation (2.14) over the set of possible actions, we obtain Bellman's equation [4], which constitutes lines (9) and (11) of the value iteration algorithm (Table 2.1)

Table 2.1: MDP Value Iteration

```
Algorithm 1: Value iteration
Input: an MDP, a discounting factor, a stopping criterion
Output: A policy
ValueIteration($\langle \mathbb{S}, \mathbb{A}, T, R \rangle$, $\gamma$, $\epsilon$)
(1)  $\Delta V = \infty$
(2)  foreach ($s \in \mathbb{S}$)
(3)     $V(s) = \max_{a \in \mathbb{A}} R(s,a)$
(4)     $\pi(s) = \arg\max_{a \in \mathbb{A}} R(s,a)$
(5)  while ($\Delta V > \epsilon$)
(6)     $\Delta V = 0$
(7)     foreach ($s \in \mathbb{S}$)
(8)        foreach ($a \in \mathbb{A}$)
(9)           $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathbb{S}} T(s,a,s') V(s')$
(10)       $\Delta V = \max(\Delta V, (\max_{a \in \mathbb{A}} Q(s,a)) - V(s))$
(11)       $V(s) = \max_{a \in \mathbb{A}} Q(s,a)$
(12)       $\pi(s) = \arg\max_{a \in \mathbb{A}} Q(s,a)$
(13) return $\pi$
```

Table 2.2: MDP Policy Iteration

```
Algorithm 2: Policy iteration
Input: an MDP, a discounting factor
Output: A policy
PolicyIteration($\langle \mathbb{S}, \mathbb{A}, T, R \rangle$, $\gamma$)
(1)  $\Delta \pi = 1$
(2)  foreach ($s \in \mathbb{S}$)
(3)     $V(s) = 0$
(4)  while ($\Delta \pi = 1$)
(5)     $\Delta \pi = 0$
(6)     foreach ($s \in \mathbb{S}$)
(7)        foreach ($a \in \mathbb{A}$)
(8)           $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathbb{S}} T(s,a,s') V(s')$
(9)        if $\neg(\pi(s) = \arg\max_{a \in \mathbb{A}} Q(s,a))$
(10)          $\Delta \pi = 1$
(11)       $\pi(s) = \arg\max_{a \in \mathbb{A}} Q(s,a)$
(12)    $V(s) = \text{getVfrom}\pi(\pi)$
(13) return $\pi$
```

are not known, the MDP solution can be approximated by learning the value function from a set of reward/transition samples [34].

The work presented in this thesis used an implementation of the value iteration algorithm. The contribution of this paper, however, is an extension to the augmented Markov decision process, and no restriction is placed on the method of solving the resulting (A)MDP.

2.1.2 Partially Observable Markov Decision Processes

Although the MDP provides a means for considering uncertainty in the state transitions, it still assumes that the agent's state is fully known at all times. The partially observable Markov decision process (POMDP), however, provides a formulation for the path planning problem whose solution is optimal (to within the resolution of the grid) even under uncertainty in the agent's position. This is accomplished by addressing uncertainty explicitly. Rather than mapping states to actions, the POMDP solution is an optimal policy that maps beliefs to actions. A belief is a probability distribution over the set of possible states, $\mathbb{S}$. The belief of a particular state, $b(s)$, is the probability that the agent occupies that state.

The POMDP is described by the tuple $\langle \mathbb{S}, \tilde{\mathbb{A}}, \tilde{T}, \tilde{R}, \Omega, O \rangle$, where:

$\mathbb{S}$ = the set of possible states (same as MDP).

$\tilde{\mathbb{A}}$ = the set of possible actions.

$\tilde{T} : (\mathbb{S} \times \tilde{\mathbb{A}}) \mapsto \Pi(\mathbb{S})$ is the transition function which maps states and actions to distributions over posterior states.

$\tilde{R} : (\mathbb{S} \times \tilde{\mathbb{A}}) \mapsto \mathbb{R}$ is a reward function which maps states and actions to real numbers.

Ω = the set of possible observations

$O : (\mathbb{S} \times \tilde{\mathbb{A}} \times \Omega) \mapsto [0, 1]$ is an observation function which maps states, actions and observations to probabilities. $O(s', a, z)$ is the probability of receiving observation $z \in \Omega$ after transitioning, via action $a \in \tilde{\mathbb{A}}$, into state $s' \in \mathbb{S}$.

We have added a new set, Ω, and a new function, $O(s', a, z)$, to the tuple. Ω is the set of actual observation values that the agent might receive from its sensors. For example, if the sensor is a laser range finder taking discrete measurements of distance in a grid world, the set of possible observations would be $\Omega = \Omega_c = [0, r_{max}]$ where r_{max} is the maximum range of the sensor. In general, Ω is taken to be finite and countable, so in the laser range finder example, we might use $\Omega = \Omega_d = \{0, \Delta, 2\Delta, 3\Delta, \ldots, \text{floor}(\frac{r_{max}}{\Delta})\Delta\}$ where Δ is the width of a single grid cell. If the agent has multiple sensors, the set of possible observations is $\Omega = \prod_i \Omega_i$ where Ω_i is the set of possible observations from the i^{th} sensor, and $\prod_i$ denotes the Cartesian product over all of the sets. The observation function, $O(s', a, z)$, is a conditional probability density function (PDF) (or probability mass function (PMF) in the discrete case) over Ω, conditioned on arriving in state s' via action a. If the uncertainty in the laser range finder from above is due to Gaussian white noise, then its continuous observation function (corresponding to Ω_c) is a Gaussian PDF (figure 3-1). The mean of the PDF would be the true range (from s' to the nearest object directly in front of the sensor) and the standard deviation of the PDF would be the standard deviation of the white noise. In this case the observation is independent of the action. The observation function for the discrete model of this sensor (corresponding to Ω_d), will be a PMF which approximates the Gaussian PDF.

The rest of the tuple, $\langle \mathbb{S}, \tilde{\mathbb{A}}, \tilde{T}, \tilde{R} \rangle$, is the same as the MDP tuple with the exception of a possible modification to the action space, $\mathbb{A}$. The POMDP action space, $\tilde{\mathbb{A}}$, contains all of the MDP actions, but it may also contain some additional actions that would be irrelevant to the MDP. These additional actions are actions with the specific intent of gathering information about the agent's state. While most of the information that the agent receives will come from observations which are received from its sensors automatically at some regular frequency, there may be some observations which can only be received as the result of the agent having taken a specific action. Finding the range to a specific landmark in the agent's environment would be an example of this kind of action. If measurements from a sensor are expensive (in terms of energy, time, stealth, etc.), we may also wish to add them to this category so that the agent can choose to use the sensor only when it improves the solution. If we refer to the set of information gathering actions as $\mathbb{Z}$, we have $\tilde{\mathbb{A}} = \mathbb{A} \cup \mathbb{Z}$. Since the MDP assumes the agent's state is known at all times, the actions in $\mathbb{Z}$ have no effect, so they are not useful in the MDP formulation. The state transition function, $\tilde{T}$, and the reward function, $\tilde{R}$, are the same as the MDP transition and reward functions except they are defined over the additional set of actions, $\mathbb{Z}$.

Having defined POMDPs, we can start to think about solving them. Due to the complex nature of POMDPs, there are many algorithms, both exact and approximate, for solving them. In this section, we will show the POMDP version of value iteration, which is the basis for all of the exact POMDP algorithms. We will then briefly discuss the complexity of solutions to the generic POMDP problem. For a complete discussion of POMDP algorithms the reader is referred to [30, 21, 7, 26].

We begin developing the POMDP version of value iteration by defining a policy tree. A one-step policy tree is simply a single action, just like a one step policy. A two-step policy tree consists of an action at its root which has one branch for each possible observation in Ω. At the end of each of these branches is a one-step policy tree. An n-step policy tree consists of a root action with a branch for each observation, at the end of which is an (n-1)-step policy tree. At execution time, the agent executes the action at the root of the policy tree and then receives an observation. The agent then selects the (n-1)-step policy tree corresponding to the observation and executes the action at its root. This is repeated for n steps. Example one, two and three-step policy trees are shown in figure 2-1.

Next, we re-write the expected discounted future reward equation, (2.14), for a policy tree, p:

$$V_p(s) = \tilde{R}(s, p_0) + \gamma \sum_{s' \in \mathbb{S}} \sum_{z \in \Omega} p(s', z | a, s) V_{p_z}(s') \tag{2.16}$$

where p_0 is the action at the root of p, $p(s', z|a, s)$ is the probability of executing action a from state s, ending up in state s' and receiving observation z, and p_z is the policy tree at the end of the branch corresponding to observation z. Writing equation (2.16) in terms of the transition and observation functions, we get:

$$V_p(s) = \tilde{R}(s, p_0) + \gamma \sum_{s' \in \mathbb{S}} \tilde{T}(s, a, s') \sum_{z \in \Omega} O(s', a, z) V_{p_z}(s') \tag{2.17}$$

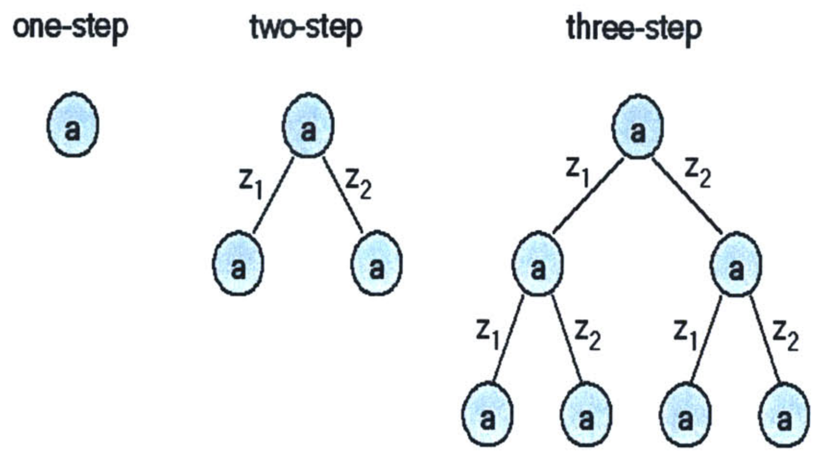


Figure 2-1: Example policy trees with two possible observations. The 'a' on each node does not refer to a specific action (each one may be different)

Next we define two vectors:

$$\alpha_p = [V_p(s_1), V_p(s_2), \dots, V_p(s_{|S|})] \tag{2.18}$$

and

$$b = [b(s_1), b(s_2), \dots, b(s_{|S|})] \tag{2.19}$$

Combining (2.17) and (2.18), we get:

$$\alpha_p(s) = \tilde{R}(s, p_0) + \gamma \sum_{s' \in \mathbb{S}} \tilde{T}(s, a, s') \sum_{z \in \Omega} O(s', a, z) \alpha_{p_z}(s') \tag{2.20}$$

The expected discounted future reward for a given belief, b, and a given policy, p, denoted $V_p(b)$, is then the dot product of the belief and the alpha vector corresponding to the policy:

$$V_p(b) = \sum_{s \in \mathbb{S}} b(s) V_p(s) = b \circ \alpha_p \tag{2.21}$$

As a result of (2.20) and (2.21), the value of an n-step policy tree is completely specified by its root action and the α vectors of the (n-1)-step policy trees at the ends of the root action's branches. This means that the root action and the α vector are all we will need to store for each policy tree in order to perform value iteration. We will denote the complete set of n-step alpha vectors (corresponding to n-step policy trees) by Γ_n. In order to construct Γ_n in the value iteration algorithm, we also define one more vector set, $\mathbb{J}_\Gamma$. Given a set of α vectors, Γ, define the vector ρ to be a permutation (with repetition), of length $|\Omega|$, of α vectors in Γ (so, $\rho_i \in \Gamma$). Then we define $\mathbb{J}_\Gamma$ to be the set of all possible such ρ vectors.

Given the complete set of n-step α vectors (and their associated root actions), Γ_n, the optimal value function is:

$$V^*(b) = \max_{\alpha \in \Gamma_n} V_p(b) = \max_{\alpha \in \Gamma_n} b \circ \alpha_p \tag{2.22}$$

from which the optimal policy can be computed to be:

$$\pi_n^*(b) = \left[\arg\max_{\alpha \in \Gamma_n} b \circ \alpha\right]_0 \tag{2.23}$$

where $[\ ]_0$ denotes the root action corresponding to the α vector inside of the brackets.

So, basically, POMDP value iteration just amounts to constructing the set of (n+1)-step α vectors from the set of n-step α vectors. The infinite horizon version of the actual algorithm is shown in Table 2.3. As was the case with the MDP, generalizing the algorithm to the finite horizon case is straightforward.

At execution time, the optimal action is chosen according to equation (2.23), with Γ_n replaced by the set of α vectors returned by the algorithm, Γ^+.

From Algorithm 3, we see that evaluating a single α vector takes $O(|\mathbb{S}|^2|\Omega|)$ operations. We also see that the size of the set Γ_n is $O(|\mathbb{A}||\mathbb{J}_{\Gamma_{n-1}}|) = O(|\mathbb{A}||\Gamma_{n-1}|^{|\Omega|}) = O(|\mathbb{A}|^{|\Omega|^{(n-1)}})$. Combining these, the asymptotic complexity of Algorithm 3 is $O(|\mathbb{S}|^2|\Omega||\mathbb{A}|^{|\Omega|^{(n-1)}})$ (for n iterations). As a result, this algorithm is only computationally tractable for problems with very small action and observation spaces. For some special classes of POMDPs, better complexity results can be obtained [21].

From equation (2.22), we see that the optimal POMDP value function is piecewise linear in the belief, and that each linear segment corresponds to a specific α vector. Some of the α vectors are dominated at all belief points by other α vectors; i.e. there may exist

Table 2.3: POMDP Value Iteration

```
Algorithm 3: POMDP Value Iteration
Input: a POMDP, a discounting factor
Output: A policy
POMDPValueIteration($\langle \mathbb{S}, \tilde{\mathbb{A}}, \tilde{T}, \tilde{R}, \Omega, O \rangle$,$\gamma$)
(1)  $\Gamma^+ = \{\}$
(2)  foreach ($a \in \mathbb{A}$)
(3)     foreach ($s \in \mathbb{S}$)
(4)        $\alpha(s) = \tilde{R}(s,a)$
(5)     $\Gamma^+ = \Gamma^+ \cup \alpha$
(6)  $\Delta V = 2\epsilon$
(7)  while ($\Delta V > \epsilon$)
(8)     $\Gamma^- = \Gamma^+$
(9)     $\Gamma^+ = \{\}$
(10)    foreach ($a \in \mathbb{A}$)
(11)       foreach ($\rho \in \mathbb{J}_{\Gamma^-}$)
(12)          foreach ($s \in \mathbb{S}$)
(13)             $\alpha(s) = \tilde{R}(s,a) + \gamma \sum_{s' \in \mathbb{S}} \sum_{i=1}^{|\Omega|} \tilde{T}(s,a,s') O(s',a,z_i) \rho_i(s')$
(14)          $\Gamma^+ = \Gamma^+ \cup \alpha$
(15)    $\Delta V = \max_b \Big[ (\max_{(\alpha \in \Gamma^+)} b \circ \alpha) - (\max_{(\alpha \in \Gamma^-)} b \circ \alpha) \Big]$
(16) return $\Gamma^+$
```

$\alpha_i \in \Gamma_n$ such that $\arg\max_{\alpha \in \Gamma_n} b \circ \alpha \neq \alpha_i$ for any b. Exact algorithms have been created which take advantage of this by pruning the set Γ_n to include only the useful α vectors, which slows the exponential growth of $|\Gamma_n|$ [16, 6]. However, although empirically they are much faster, these algorithms still have the same asymptotic complexity as Algorithm 3 and, in general, they are still only tractable for problems with small action and observation spaces.

As a result of the prohibitive complexity of exact solutions, many POMDP applications use approximate solutions. Since the purpose of the approximate solutions is to mitigate the complexity of the POMDP, it is important to understand the underlying cause of the complexity. There are two basic differences between the MDP and the POMDP, each of which contribute to the additional complexity of the POMDP. First, the POMDP maps beliefs to actions, and the belief space for n discrete states is an $(n - 1)$ dimensional simplex. This is commonly referred to as the "curse of dimensionality" [16]. Second, since the belief at any point in time depends on all of the observations previous to that time, even if we are given the initial belief, the number of possible beliefs grows exponentially in the horizon length. This is commonly referred to as the "curse of history" [24]. The way to reduce the complexity then is to simplify our representation of the belief space. Some methods that are employed to this end are: planning over a subset of the belief space, policy search, applying heuristics to the MDP solution, hierarchical methods (grouping states), etc. Roy [26] defines a continuum of increasing complexity and robustness, between the MDP and the exact POMDP, in which all of these approximations lie.

In chapter four, an augmented MDP path planner is developed which takes advantage of the structure of the information which is available to the agent (by assuming the information comes from an extended Kalman filter) to plan over a small set of beliefs which is representative of the complete set of beliefs that the agent will ever see. First though, in chapter three, we will develop the structure and rules of the extended Kalman filter, so that we can apply them to the augmented MDP.

Chapter 3

Extended Kalman Filter

In this chapter we develop the extended Kalman filter (EKF) [5, 11, 3, 17]. The EKF is an important and widely employed tool in the field of estimation. The EKF is a means of combining partial knowledge of an agent's initial state, knowledge of the agent's dynamics and information provided by observations to maintain the maximum *a posteriori* (MAP) estimate of the agent's state at all times. Further, as a consequence of the central limit theorem[1], the EKF is able to robustly estimate states in real world problems using only a small portion of the possible belief space, specifically the set of normal (or Gaussian) distributions. As we will see in chapter four, we can exploit this result to modify the MDP developed in chapter two in order to create a planner that is robust to uncertainty in the observations as well as the agent's position without affecting the MDP's complexity.

We will begin by deriving the Kalman filter equations for a linear system that is evolving continuously over time and receives measurements from its sensors at discrete time intervals. Then we will modify these equations to get the discrete time, discrete measurement version of the equations which are easier to implement. We then develop the extended Kalman filter which is able to accommodate non-linearities in the dynamics as well as the observations. In chapter five we will see the EKF implemented on a simulation of a real world application.

[1]The central limit theorem states that the distribution of the sum of independent random variables tends to be Gaussian, regardless of the distribution of the independent variables. Since uncertainty in navigation usually comes from the superposition of many sources, it tends to be normally distributed.

3.1 The Gaussian Distribution

The uncertainty in the Kalman filter is represented by a normal (or Gaussian) distribution, and the optimality of the Kalman filter depends on the uncertainty (noise) in the agent's dynamics and observations being normally distributed. Therefore, before developing the Kalman filter, a brief summary of the Gaussian distribution is in order. The Gaussian density function is:

$$f_X(x) = \frac{1}{(2\pi)^{n/2}|C|^{1/2}} \exp\left[-\frac{1}{2}(x-\bar{x})^T C^{-1}(x-\bar{x})\right] \tag{3.1}$$

where:

$$x \in \mathbb{R}^{n\times 1}$$

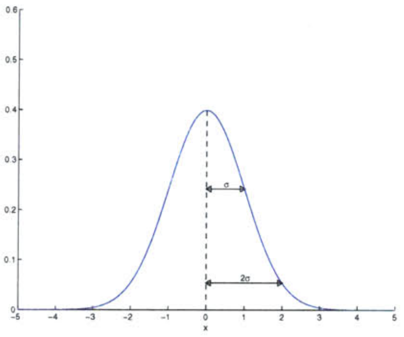
$$\bar{x} = E[x]$$

$$C = \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \cdots \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

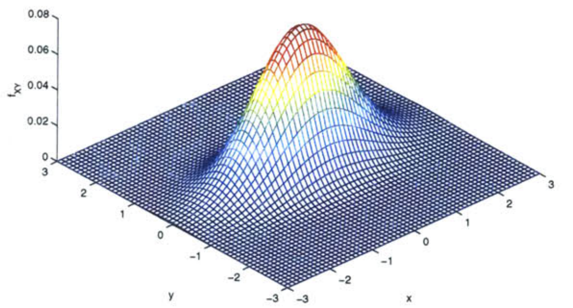
Note that the Gaussian probability density at any point, x, is fully determined by the vector mean, $\bar{x}$, and the covariance, C, of the distribution. The standard deviation of the distribution of state i is σ_i, the variance of the distribution of state i is σ_i^2 and the correlation coefficient between states i and j is ρ_{ij}. Example Gaussian density functions are shown for one and two variables in figure 3-1.

3.2 Kalman Filter Basics

In chapter one of this report, the author claimed that an autonomous agent required two basic capabilities, the ability to take actions which affect its environment and the ability to reason about which actions it should choose. We may think of estimation as a third capability whose necessity is generally implied by the ability to reason. An agent's ability to reason about which actions it should take in order to achieve some desirable state in its environment will be severely limited if the agent has no knowledge of its state. In many



(a) Scalar Gaussian



(b) Bi-variate Gaussian

Figure 3-1: (a) Scalar Gaussian density function with $\bar{x} = 0$ and $\sigma_x = 1$, (b) Bi-variate Gaussian density function with $\bar{x} = \bar{y} = 0$, $\sigma_x = 1$, $\sigma_y = .5$ and $\rho_{xy} = 0$

applications it is unreasonable to assume that the agent's state will be known. Rather, the agent will likely have some model of how its states change over time, as well as a set of sensors which provide it with imperfect measurements, or observations, which contain information about its states. Given this information, the estimator's job is to provide the agent with the best possible guess of its current state and possibly some measure of how reliable that guess is.

We begin with the problem of estimating the agent's state without any observations. All we are given is a model of the agent's dynamics which describe how the agent's states change over time. In some cases, we may even have to estimate this model. This is generally referred to as the identification problem [8]. For our purposes however, we will assume that the model is known. Further, we will assume that the model is in the form of a set of linear differential equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = Ax(t) + Bu(t) + w(t), \quad x(0) = x_0 \tag{3.2}$$

where:

$x(t) \in \mathbb{R}^{n \times 1}$ is the state vector at time t (x_0 = initial state)

$u(t) \in \mathbb{R}^{m\times 1}$ is the input vector at time t

$A \in \mathbb{R}^{n\times n}$ is the differential coefficient matrix

$B \in \mathbb{R}^{n\times m}$ is the input coefficient matrix

$w(t)$ is zero mean Gaussian white noise with intensity (or power spectral density) W. w accounts for uncertainty in the agent's dynamics.

If the model does not naturally take this form, we can generally approximate it in this form using a technique called linearization. We discuss linearization in section 3.3. We have also made the assumption that all of the states are continuous. In general, our agent may have some states that only take discrete values, such as discrete modes of operation or vehicle health states (which might take on the values 'broken' or 'ok'). Systems with both continuous and discrete states are called hybrid systems, and estimation of hybrid systems is called hybrid estimation [13]. Future work on this topic will investigate inclusion of discrete states in the planner by using an EKF based hybrid estimator.

Let $\hat{x}(t)$ be the estimated state vector at time t. Then, since we don't know w (but we know that $E[w] = 0$), and we have no observations available, the best state estimate at time t is given by its expected value, which gives:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{x}(t) = A\hat{x}(t) + Bu(t), \quad \hat{x}(0) = \hat{x}_0 \tag{3.3}$$

where:

$\hat{x}_0$ is the best available estimate of the initial state

Define $e(t)$ to be the error in the estimate:

$$e(t) = \hat{x}(t) - x(t) \tag{3.4}$$

Differentiating (3.4) and substituting in (3.2) and (3.3), we get:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}e(t) &= \Big(A\hat{x}(t) + Bu(t)\Big) - \Big(Ax(t) + Bu(t) + w\Big), \quad e(0) = e_0 = \hat{x}_0 - x_0 \\ &= A\Big(\hat{x}(t) - x(t)\Big) - w\end{aligned}$$

$$= \quad Ae(t) - w \tag{3.5}$$

The solution to (3.5) is[2]:

$$e(t) = \exp(At)e_0 + \int_0^t \exp\Big(A(t-\tau)\Big)w(\tau)\mathrm{d}\tau \tag{3.6}$$

and, since $E[w] = 0$:

$$E[e(t)] = \exp(At)E[e_0] \tag{3.7}$$

This means that, as long as the original system, (3.2), is stable [11, 3], the effect of the initial errors will die out and the errors will go to zero. Mathematically speaking (for stable system):

$$\lim_{t \to \infty} E[e(t)] = 0 \tag{3.8}$$

In general though, there is no guarantee that the system is stable, and, even if it is, we have no control over the rate of convergence (i.e. if $\exp(At)$ is only small for large t, the initial error will take a long time to die out).

We can better characterize the estimation error using its covariance matrix:

$$P(t) = E\Big[\Big(e(t) - E[e(t)]\Big)\Big(e(t) - E[e(t)]\Big)^T\Big] = E\Big[e(t)e^T(t)\Big] \tag{3.9}$$

In Appendix A, the following equation, describing the dynamics of the covariance matrix, is derived:

$$\frac{\mathrm{d}}{\mathrm{d}t}P(t) = AP(t) + P(t)A^T + W, \quad P(0) = P_0 \tag{3.10}$$

If the system is stable, equation (3.10) converges to a steady state, and we can obtain the steady state error covariance by setting the right side of equation (3.10) equal to zero and solving for $P(t)$.

Now we can incorporate the information that we have available in the form of discrete time observations from the agent's sensors. In general, the observations are not direct measurements of the states (ex: a GPS measurement gives the range from the agent to the

[2] $exp(At) = I + At + \frac{A^2t^2}{2!} + \frac{A^3t^3}{3!} \dots$ is the matrix exponential

satellite, it does not give the agent's position directly). In order to be useful though, the observations must have some dependence on the states. We will assume this dependence is linear (non-linear observations will be addressed in section 3.3):

$$z(\tau_i) = Hx(\tau_i) + v \tag{3.11}$$

where:

τ_i is a discrete observation time

v is sampled from zero mean white noise with variance V

Now we define the innovation:

$$q(\tau_i) = z(\tau_i) - H\hat{x}(\tau_i) \tag{3.12}$$

The innovation is an indirect measure of the error in the state estimate (note: the innovation itself has some error due to the measurement noise, v). By choosing an appropriate blending factor, K (sometimes called the filter gain), we can use the innovation to update the estimate:

$$\hat{x}(\tau_i^+) = \hat{x}(\tau_i^-) + K[q(\tau_i^-)] \tag{3.13}$$

Where the superscripts ($-$) and ($+$) denote the time instants before and after the measurement update respectively. We also need to update the covariance matrix when we incorporate a measurement. Plugging (3.13) into (3.4) and then plugging the result into (3.9) we get:

$$P(\tau_i^+) = E\Big[\Big\{\hat{x}(\tau_i^-) + K\Big(z(\tau_i) - H\hat{x}(\tau_i^-)\Big) - x(\tau_i)\Big\}\Big\{\text{same}\Big\}^T\Big] \tag{3.14}$$

Substituting (3.11) into (3.14) we get:

$$\begin{aligned} P(\tau_i^+) &= E\Big[\Big\{\hat{x}(\tau_i^-) + K\Big(Hx(\tau_i) - H\hat{x}(\tau_i^- + v)\Big) - x(\tau_i)\Big\}\Big\{\text{same}\Big\}^T\Big] \\ &= E\Big[\Big\{\Big(I - KH\Big)\Big(\hat{x}(\tau_i^-) - x(\tau_i)\Big) + Kv\Big\}\Big\{\text{same}\Big\}^T\Big] \end{aligned}$$

$$= (I - KH)P(\tau_i^-)(I - KH)^T + KVK^T \tag{3.15}$$

All that remains now is to choose an optimal filter gain. With respect to the EKF, optimal means that we would like to choose a filter gain that minimizes the uncertainty in (or covariance of) the estimates of each state after incorporating each measurement. Since the estimate uncertainties in an EKF are always represented by normal distributions, the estimate that minimizes the uncertainty after each measurement is also the MAP estimate. Minimizing the covariance of the estimate of each state after a measurement is equivalent to minimizing the trace of $P(\tau_i^+)$ from equation (3.15). Since $P(\tau_i^-)$ and V are both positive definite (by definition of a covariance matrix), the trace of $P(\tau_i^+)$ is a convex function of K. Therefore, we can find the optimal gain by differentiating the trace of $P(\tau_i^+)$ with respect to K, setting the result equal to zero and solving for the Kalman gain K^*. The result is:

$$K^* = P(\tau_i^-)H^T\Big(HP(\tau_i^-)H^T + V\Big)^{-1} \tag{3.16}$$

By propagating the estimate and covariance using equations (3.3) and (3.10) between measurements and incorporating measurements when they become available using equations (3.13), (3.15) and (3.16), we maintain an optimal state estimate at all times. In practice though, it is often desirable to modify equations (3.3) and (3.10) in order to update the estimate and covariance at discrete time intervals. This procedure is outlined in the following section.

3.2.1 Discrete Time Kalman Filter

In the previous section, we developed the Kalman Filter equations for continuous time dynamics and discrete time observations. In this form, it is necessary to integrate equations (3.3) and (3.10) in order to propagate the estimate and covariance between measurements. In most estimation and controls applications, however, the dynamics are propagated in small discrete time-steps in order to make software faster and easier to program.

We begin derivation of the discrete time filter equations by looking at the discrete form of equation (3.2):

$$x(t+\Delta t) = A_{\Delta t}x(t) + B_{\Delta t}u(t) + w_{\Delta t} \tag{3.17}$$

To compute the $A_{\Delta t}$ and $B_{\Delta t}$ matrices and the vector $w_{\Delta t}$ we use the solution of equation (3.2):

$$\begin{aligned} x(t) &= \exp[At]x(0) + \int_0^t \exp[A(t-\tau)]Bu(\tau)\mathrm{d}\tau \\ &\quad + \int_0^t \exp[A(t-\tau)]w(\tau)\mathrm{d}\tau \end{aligned} \tag{3.18}$$

Taking $x(0) = x(t)$ and propagating for one time-step, Δt, we get:

$$\begin{aligned} x(t+\Delta t) &= \exp[A\Delta t]x(t) + \int_t^{t+\Delta t} \exp[A(t+\Delta t-\tau)]Bu(\tau)\mathrm{d}\tau \\ &\quad + \int_t^{t+\Delta t} \exp[A(t+\Delta t-\tau)]w(\tau)\mathrm{d}\tau \end{aligned} \tag{3.19}$$

Comparing equation (3.19) to equation (3.17), we obtain the exact equalities:

$$A_{\Delta t} = \exp[A\Delta t] \tag{3.20}$$

$$B_{\Delta t}u(t) = \int_t^{t+\Delta t} \exp[A(t+\Delta t-\tau)]Bu(\tau)\mathrm{d}\tau \tag{3.21}$$

$$w_{\Delta t} = \int_t^{t+\Delta t} \exp[A(t+\Delta t-\tau)]w(\tau)\mathrm{d}\tau \tag{3.22}$$

In practice, we may choose to use equation (3.20) directly to compute $A_{\Delta t}$, or, for small Δt, we may choose to use a first order Taylor series expansion of equation (3.20) about the point $\Delta t = 0$:

$$A_{\Delta t} \approx I + A\Delta t \tag{3.23}$$

If $u(\tau)$ takes an appropriate functional form, we may choose to evaluate the individual elements of the vector integrand in equation (3.21) and then compute the exact value of the integral. More commonly though, we make the assumption that Δt is small enough so that $u(t)$ may be taken to be constant over the interval $[t, t + \Delta t]$. In this case, equation (3.21) reduces to[3]:

$$B_{\Delta t} = \int_{t}^{t+\Delta t} \exp\left[A(t + \Delta t - \tau)\right] B \mathrm{d}\tau \tag{3.24}$$

If A is not invertible, it is still necessary to solve for the individual elements of the integrand in equation (3.24) and then compute the integral. However, if A is invertible, the integral has a closed form solution:

$$\begin{aligned} B_{\Delta t} &= \Big[-A^{-1} \exp\left[A(t + \Delta t - \tau\right] B\Big]_{t}^{t+\Delta t} \\ &= A^{-1}\Big(-I + \exp\left[A\Delta t\right]\Big) B \end{aligned} \tag{3.25}$$

We may also use a first order Taylor series approximation of $\exp\left[A\Delta t\right]$ here. If we do, equation (3.25) reduces to:

$$B_{\Delta t} \approx B\Delta t \tag{3.26}$$

In order to obtain a useful form for $w_{\Delta t}$, we make the assumption that Δt is small enough so that $\exp\left[A(t + \Delta t - \tau)\right] \approx I$ for $\tau \in [t, t + \Delta t]$. In this case, equation (3.22) reduces to:

$$w_{\Delta t} = \int_{0}^{\Delta t} w(\tau) \mathrm{d}\tau \tag{3.27}$$

A property of Gaussian white noise is that its integral also has a Gaussian distribution. From equation (3.27) we can see that $E[w_{\Delta t}] = 0$. So, all that we need to finish characterizing $w_{\Delta t}$ is its covariance:

[3]If we physically restrict the control inputs to be constant over $[t, t + \Delta t]$, equation (3.24) is still exact

$$
\begin{aligned}
W_{\Delta t} = E[w_{\Delta t} w_{\Delta t}^T] &= E\Big[\Big(\int_0^{\Delta t} w(\tau_1)\mathrm{d}\tau_1\Big)\Big(\int_0^{\Delta t} w(\tau_2)\mathrm{d}\tau_2\Big)^T\Big] \\
&= E\Big[\int_0^{\Delta t}\int_0^{\Delta t} w(\tau_1)w^T(\tau_2)\mathrm{d}\tau_1\mathrm{d}\tau_2\Big] \\
&= \int_0^{\Delta t}\int_0^{\Delta t} E[w(\tau_1)w^T(\tau_2)]\mathrm{d}\tau_1\mathrm{d}\tau_2 \\
&= \int_0^{\Delta t} E[w(\tau_2)w^T(\tau_2)]\mathrm{d}\tau_2 \\
&= W\Delta t
\end{aligned}
\tag{3.28}
$$

Now equation (3.17) is fully defined, and, analogous to the continuous case, we can define our estimator to be:

$$\hat{x}(t+\Delta t) = A_{\Delta t}\hat{x}(t) + B_{\Delta t}u(t) \tag{3.29}$$

The error in the estimate is still defined as in equation (3.4). So, using equations (3.9) and (3.17) we can compute the covariance update corresponding to equation (3.29):

$$
\begin{aligned}
P(t+\Delta t) &= E[e(t+\Delta t)e^T(t+\Delta t)] \\
&= E\Big[\Big(A_{\Delta t}e(t) - w_{\Delta t}\Big)\Big(A_{\Delta t}e(t) - w_{\Delta t}\Big)^T\Big] \\
&= E[A_{\Delta t}e(t)e^T(t)A_{\Delta t}] - E[A_{\Delta t}e(t)w_{\Delta t}^T] - E[w_{\Delta t}e^T(t)A_{\Delta t}^T] + E[w_{\Delta t}w_{\Delta t}^T] \\
&= A_{\Delta t}P(t)A_{\Delta t}^T + W_{\Delta t}
\end{aligned}
\tag{3.30}
$$

3.2.2 Summary of the Discrete Kalman Filter Equations

Although the derivation of the Kalman filter equations is rather complex and involved, their implementation is quite simple. We begin with an initial state estimate, $\hat{x}(0)$, and an initial covariance matrix, $P(0)$. We then use equations (3.29) and (3.30) (repeated here) to propagate the estimate and covariance until we receive an observation:

$$\hat{x}(t+\Delta t) = A_{\Delta t}\hat{x}(t) + B_{\Delta t}u(t)$$

$$P(t+\Delta t) = A_{\Delta t}P(t)A_{\Delta t}^T + W_{\Delta t}$$

Then, when we receive an observation, we use equations (3.13), (3.15) and (3.16) (repeated here) to update the estimate and covariance[4]:

$$\hat{x}(\tau_i^+) = \hat{x}(\tau_i^-) + K[q(\tau_i^-)]$$

$$P(\tau_i^+) = (I - KH)P(\tau_i^-)(I - KH)^T + KVK^T$$
$$K^* = P(\tau_i^-)H^T\Big(HP(\tau_i^-)H^T + V\Big)^{-1}$$

3.3 The Extended Kalman Filter

Thus far, we have assumed that the dynamics of our system are described by a set of linear differential equations, (3.2), and that the observations are linear combinations of the states, (3.11). When we write the differential equations that describe a real system (or real observations), however, they are rarely linear. Fortunately though, for small changes in the system, non-linear systems can be approximated very well by linear systems. In this section, we will show how the Kalman filter equations which we have already developed can be applied to a very general class of non-linear systems.

We begin by assuming new functional forms for the system dynamics and observations which include non-linear cases[5]:

$$\frac{\mathrm{d}}{\mathrm{d}t}x = f(x,u) + w \tag{3.31}$$

$$z(t) = g(x) + v \tag{3.32}$$

Where the only restriction is that the functions f and g are continuous and differentiable over the set of feasible values of x and u.

Next we consider the Taylor series expansion of the functions f and g about the

[4]When we use the discrete time equations to propagate the estimate and covariance between observations, we can only incorporate an observation at an integer multiple of the time-step, Δt

[5]To reduce clutter, the (t) has been dropped from $x(t)$ and $u(t)$.

arbitrary state x_0:

$$f(x,u) = f(x_0,u) + \frac{\partial}{\partial x} f(x,u)\Big|_{x_0} (x - x_0) + o(x - x_0) \tag{3.33}$$

$$g(x) = g(x_0) + \frac{\partial}{\partial x} g(x)\Big|_{x_0} (x - x_0) + o(x - x_0) \tag{3.34}$$

where $o(x - x_0)$ represents higher order terms, and:

$$\lim_{h \to 0} \frac{o(h)}{h} = 0 \tag{3.35}$$

So, for small $(x - x_0)$, we have:

$$f(x,u) \approx f(x_0,u) + J_f(x - x_0) \tag{3.36}$$

$$g(x) \approx g(x_0) + J_g(x - x_0) \tag{3.37}$$

where:

$$J_f = \frac{\partial}{\partial x} f(x,u) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad \text{and} \quad J_g = \frac{\partial}{\partial x} g(x) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

are the Jacobian matrices of f and g with respect to x. Unless otherwise noted, the Jacobian matrices will always be evaluated at $x = x^*$.

Now we define two new variables, the deterministic state x^*, and the perturbation state δx, so that:

$$x = x^* + \delta x \tag{3.38}$$

The meaning of these two variables will be discussed shortly but for now their definitions will suffice. We will also rewrite the state estimate as:

$$\hat{x} = x^* + \delta \hat{x} \tag{3.39}$$

where $\delta \hat{x}$ is the estimate of the perturbation state. The estimate error can also be updated to reflect the new states:

$$e = \hat{x} - x = \delta\hat{x} - \delta x \tag{3.40}$$

Now we are ready to try to estimate the states of the non-linear system. As in the linear case, in the absence of measurements, the best available estimate of the state is its expected value:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{x} = f(\hat{x}, u) \tag{3.41}$$

Assuming that the perturbation states are small and taking the first order Taylor series expansion of equations (3.31) and (3.41), we get:

$$\frac{\mathrm{d}}{\mathrm{d}t}x = f(x, u) + w \approx f(x^*, u) + J_f(\delta x) + w \tag{3.42}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{x} = f(\hat{x}, u) \approx f(x^*, u) + J_f(\delta\hat{x}) \tag{3.43}$$

Then, taking the difference of equations (3.43) and (3.42), we get a linear system of differential equations describing the error:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\hat{x} - \frac{\mathrm{d}}{\mathrm{d}t}x &\approx f(x^*, u) + J_f(\delta\hat{x}) - f(x^*, u) - J_f(\delta x) - w \\ \frac{\mathrm{d}}{\mathrm{d}t}e &\approx J_f(e) - w \end{aligned} \tag{3.44}$$

Note that equation (3.44) is equivalent to equation (3.5) with A replaced by J_f. As a result, we can reuse the derivation in Appendix A to get the differential equation for the error covariance:

$$\frac{\mathrm{d}}{\mathrm{d}t}P = J_f P + P J_f^T + W \tag{3.45}$$

In order to incorporate observations into our estimate, we first define a perturbation observation, δz, using the first order Taylor series expansion of the original non-linear observation:

$$\begin{aligned} z &= g(x) + v \approx g(x^*) + J_g(\delta x) + v \\ \delta z &= z - g(x^*) \approx J_g(\delta x) + v \end{aligned} \tag{3.46}$$

Now we have an observation which is linear in the perturbation states and we can use equations (3.12), (3.13), (3.15) and (3.16) with H replaced by J_g and z replaced by δz to incorporate the measurements:

$$q = \delta z - J_g \delta \hat{x} \tag{3.47}$$

$$\delta \hat{x}^+ = \delta \hat{x}^- + Kq \tag{3.48}$$

$$P^+ = (I - KJ_g)P^-(I - KJ_g)^T + KVK^T \tag{3.49}$$

$$K = P^- J_g^T \left(J_g P^- J_g^T + V \right)^{-1} \tag{3.50}$$

To implement the EKF, we begin with $x^* = \hat{x} = \hat{x}_0$ and $\delta x = 0$, where $\hat{x}_0$ is the initial state estimate. We then use equations (3.41) and (3.45) (or their discrete time versions) to propagate the estimate until we receive an observation. In some cases, if integration of equation (3.41) is difficult, we may use its linear approximation, (3.43), to propagate the estimate. In this case we would assume x^* and u, and thus $f(x^*, u)$, are constant during the period of integration, and $\delta \hat{x}$ evolves according to $\delta \hat{x} = \hat{x} - x^*$. When we receive an observation, we use equations (3.47) through (3.50) to update the perturbation estimate. During this step we assume x^* is constant.

In order to keep $\delta \hat{x}$ small, so that the Taylor series approximations remain accurate, any time $\delta \hat{x} \neq 0$ (after a measurement update or after propagating forward in time) we update the deterministic state, $x^* \leftarrow x^* + \delta \hat{x}$, and reset the perturbation estimate, $\delta \hat{x} \leftarrow 0$. Whenever we update the deterministic state, we also reevaluate the Jacobian matrices, J_f and J_g.

Having described the EKF implementation, we can now clarify the definition of the deterministic state and the perturbation state. The deterministic state is really just the best estimate of the true state at any given time, and the perturbation state is just the difference between the estimated state and the true state, or the error. The states of the EKF then, are the perturbation estimates which are estimates not of the actual non-linear states, but rather of the error in the estimate of the non-linear states. The filter attempts to drive these errors toward zero by using them to update the non-linear state estimate.

Intuition suggests that the EKF will work well whenever the Taylor series approximations are accurate and fall apart when the approximations are inaccurate (because the perturbation states get too large making the higher order terms relevant, and/or due to extreme non-linearities). This intuition is correct, however, the EKF tends to be more robust to inaccurate approximations than might be expected. The reason for this is that the EKF works by adding updates to the state estimate which push it toward the true state. Even if the approximations become quantitatively inaccurate, as long as they remain relatively accurate qualitatively, the updates will still push the estimate in the right direction.

[This page intentionally left blank.]

Chapter 4

Augmented Markov Decision Processes

In chapter two, we developed a number of path planners, all of which lacked robustness to uncertainty in the agent's position, with the exception of the POMDP which has prohibitive complexity for most problems. We noted, however, that we can approximate the POMDP solution by considering a subset of the complete belief space. For example, we might choose to estimate the agent's states using an extended Kalman filter, resulting in the closed loop system shown in figure 4-1. By pairing our planner with an EKF, we insure that the only beliefs that the planner will ever experience will be Gaussian distributions. Therefore, we can fully describe every belief that the planner will experience by its mean (which lies in $\mathbb{S}$), and its covariance. This suggests that by simply augmenting the state vector of the original MDP with states that represent the covariance of the Gaussian distributions, we can construct an augmented MDP (AMDP) whose states represent (exactly), the beliefs of the POMDP. Therefore, under the constraint that the belief space of the original POMDP is restricted to the set of Gaussian beliefs, the exact solution of the AMDP is an exact solution of the POMDP (herein, we will only solve the AMDP approximately; we will approximate the set of Gaussian distributions using a set of discrete mean values, the centers of the grid cells, and a discrete set of standard deviation values). We also have the result that the problem of planning under uncertainty for any system whose states can be effectively estimated by an EKF falls into this constrained class of

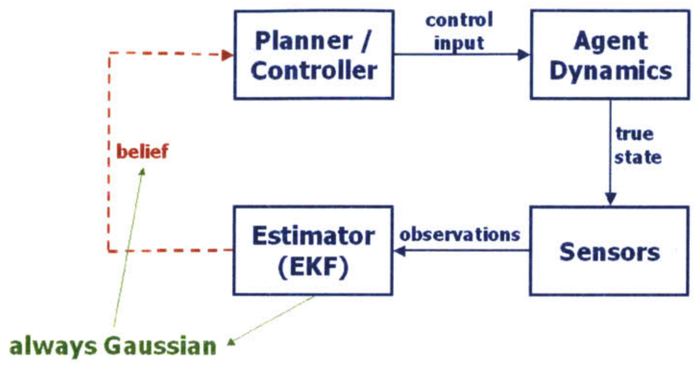


Figure 4-1: Closed-loop agent control system. The signal on the dashed line between the estimator and the planner is always a Gaussian distribution.

POMDPs. This means that the wide range of applicability of the EKF implies a wide range of applicability of the AMDP.

The planner described in this chapter is most similar to the work done by Roy in citeRoy03, Roy99. The main difference is the use of the analytical EKF equations, rather than direct application of Bayes' rule over discrete grid cells, to describe the transitions of the augmented states. The advantage of the EKF equations over application of Bayes' rule is that the transitions resulting from Bayes' rule are specific to the set of grid cells, while the EKF equations can be used to calculate the transitions for any set of grid cells including the limiting case of continuous space. This will provide an opportunity for future work to investigate an AMDP planner that can begin with a coarse approximate solution, and then refine the solution only in the proximity of states that the agent expects to experience. The disadvantage is a loss of generality; the EKF equations can only describe Gaussian distributions.

In the remainder of this chapter, we will describe in detail each element of the AMDP tuple $\langle \mathbb{S}', \mathbb{A}', R', T' \rangle$. Then in chapter five, we will develop a model of a system with uncertainty and complexity which are representative of a real world application. Then we will develop an AMDP path planner for this model and demonstrate its application via simulation.

4.1 The AMDP State Space

As stated earlier, each state of the AMDP is actually a Gaussian distribution, and, since we have limited our planner to Gaussian distributions, it is possible to construct the AMDP state space so that it includes every distribution that the planner could possibly experience. We will begin by constructing this state space, and then we will approximate it with a set of states that fits into the MDP framework from chapter two. We will consider the case of motion in two dimensional space. The extension to three dimensional space is straightforward.

Each AMDP state will take the form:

$$x = [m, c] \tag{4.1}$$

where m is a vector mean and c is a vector containing the elements of a covariance matrix.

As was the case with all of the planners in chapter two, we will begin by assuming that the agent remains within a bounded region of two (or three) dimensional Euclidean space. Call this set $\mathbb{X}$. Next, from section (3.1), we see that a two dimensional covariance matrix can be constructed from three elements[1], two standard deviations, σ_x and σ_y, and a correlation coefficient, ρ_{xy}. Now, we can define the complete set of AMDP states as:

$$\mathbb{S}_c = \{[m, \sigma_x, \sigma_y, \rho_{xy}] | m \in \mathbb{X}, \sigma_x \in (0, \infty), \sigma_y \in (0, \infty), \rho_{xy} \in (-1, 1)\} \tag{4.2}$$

Future work will explore algorithms which will find (or approximate) policies which are optimal mappings from the continuous set $\mathbb{S}_c$ to the action space. In order to apply the MDP algorithms which we defined in chapter two, we define discrete sets of standard

[1]A three dimensional covariance matrix can be constructed using three standard deviations and three correlation coefficients

deviation values and correlation coefficient values:

$$\Sigma = \{0, \Delta_\sigma, 2\Delta_\sigma, \ldots, \sigma_{max}\} \tag{4.3}$$

$$\Theta = \{-1, -1+\Delta_\rho, -1+2\Delta_\rho, \ldots, 1\} \tag{4.4}$$

where σ_{max} characterizes the largest uncertainty that the agent expects to see and Δ_σ and Δ_ρ are small discrete increments.

Then we define the discrete set of AMDP states as:

$$\mathbb{S}' = \{[s, \sigma_x, \sigma_y, \rho_{xy}] | s \in \mathbb{S}, \sigma_x \in \Sigma, \sigma_y \in \Sigma, \rho_{xy} \in \Theta\} \tag{4.5}$$

or

$$\mathbb{S}' = \mathbb{S} \times \Sigma \times \Sigma \times \Theta \tag{4.6}$$

In order to make the AMDP algorithm faster, we may choose to further our approximation of $\mathbb{S}_c$ by omitting ρ_{xy} from the state vector and only include distributions whose principal axes are the x and y axes. We may choose to go one step further and include only circular distributions by omitting one of the standard deviation values from the state vector.

4.2 The AMDP Action Space

The action space for the AMDP is the same as the action space for the POMDP described in chapter two:

$$\mathbb{A}' = \tilde{\mathbb{A}} \tag{4.7}$$

The information gathering actions which had no effect in the basic MDP now have meaning because the AMDP has observability into the uncertainty in the agent's state.

4.3 The AMDP Reward Function

According to the MDP definition, the reward function must map states and actions to real numbers. Since the AMDP state space is different than the MDP state space, the reward function used in the MDP and POMDP cannot be directly applied to the AMDP. However, in defining the expected discounted reward for the POMDP (equation (2.22)), we implicitly defined the reward for a belief. By combining equations (2.17) and (2.22) and considering only the portion due to the immediate reward, we get:

$$R(b,a) = \sum_{s \in \mathbb{S}} b(s) R(s,a) \tag{4.8}$$

Using equation (4.8) we can define the reward for an AMDP state, $x \in \mathbb{S}'$, as:

$$R'(x,a) = \sum_{s \in \mathbb{S}} b_x(s) R(s,a) \tag{4.9}$$

where $b_x(s)$ is the integral[2] over the grid cell corresponding to $s \in \mathbb{S}$, of the Gaussian probability density function defined by the state $x \in \mathbb{S}'$. A visual interpretation of the computation of $b_x(s)$ is shown in figure 4-2. The dotted line is the Gaussian probability density function corresponding to the state $s \in \mathbb{S}'$ and the solid line is the probability mass function which approximates the probability density function. The area inside of the boxes give the values of $b_x(s)$ for $s \in \mathbb{S}$.

4.4 The AMDP State Transition Function

Referring to the MDP definition again, the AMDP state transition function must map states and actions to posterior states (or posterior distributions).

$$T' : (\mathbb{S}' \times \mathbb{A}') \mapsto \Pi(\mathbb{S}') \tag{4.10}$$

[2]Since the integral of the Gaussian probability density function does not have a closed form solution, $b_x(s)$ must be computed numerically

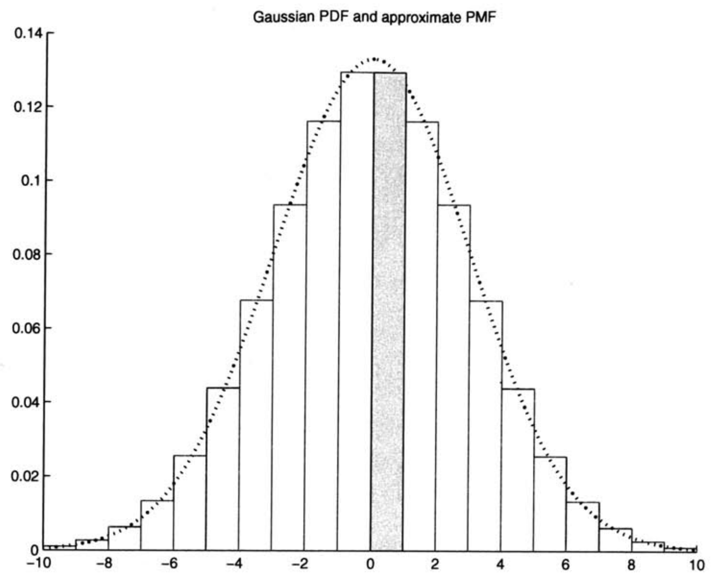


Figure 4-2: The dotted line is the Gaussian probability density function corresponding to the state $s \in \mathbb{S}'$ and the solid line is the probability mass function which approximates the probability density function. The shaded area is the value of $b_x(1)$.

Since the MDP formulation does not include observations, the effect of observations must be captured by the transition function as well. That is, $T'(x, a)$ describes the evolution of the Gaussian distribution x, as a result of temporal dynamic effects as well as observation updates, conditional on the action a. In chapter three, we derived a set of equations which accomplish precisely this, the EKF state and covariance propagation and measurement update equations:

Time propagation:

$$\hat{x}(t + \Delta t) = A_{\Delta t}\hat{x}(t) + B_{\Delta t}u(t) \tag{4.11}$$

$$P(t + \Delta t) = A_{\Delta t}P(t)A_{\Delta t}^T + W_{\Delta t} \tag{4.12}$$

Measurement updates:

$$K = P(\tau_i^-)H^T\Big(HP(\tau_i^-)H^T + V\Big)^{-1} \tag{4.13}$$

$$\hat{x}(\tau_i^+) = \hat{x}(\tau_i^-) + K[z(\tau_i) - H\hat{x}(\tau_i^-)] \tag{4.14}$$

$$P(\tau_i^+) = (I - KH)P(\tau_i^-)(I - KH)^T + KVK^T \qquad (4.15)$$

Before continuing, we make an observation which will simplify the remaining derivations. The element of the MDP which makes state transitions non-deterministic is uncertainty in the outcome of the actions. In the POMDP, and also the AMDP, the effect of the uncertainty in the action outcomes has a deterministic effect on the belief transitions: given some prior belief, in the absence of additional sensor information, we know exactly what the belief will be after any action. However, the POMDP belief transitions are made non-deterministic by the uncertainty in the observations. In the AMDP, the uncertainty in the observations enters in the form of the measurement noise matrix, V, via equations (4.13) and (4.15), which describe a deterministic transition in the covariance matrix. The reason this is possible is because of the assumption that the uncertainty in the measurements is Gaussian distributed white noise. Therefore, the measurement update equation, (4.14), gives the posterior position estimate as the sum of two Gaussian distributed random variables (the prior position estimate and the scaled innovation), and there is a closed form solution for the covariance of the sum of two Gaussian random variables (namely equation (4.15)) which depends only on the variance of those two variables. As a result, the AMDP state transitions are deterministic, and the function $T(s, a)$ will return a single deterministic state, s'.

Now, in order to compute the posterior state, $s' = T(s, a)$, using equations (4.11) through (4.15), we need to specify four things: the initial conditions, $\hat{x}(0)$ and $P(0)$, the measurement values, $z(t)$, the total duration of action a, Δt_a (this determines how far into the future we need to propagate the state and covariance, this is not the same as Δt in the above equations), and the input vector $u(\tau), \tau \in [0, \Delta t_a)$.

The initial mean and covariance, $\hat{x}(0)$ and $P(0)$ are taken directly from the initial state s. Since the measurement values are not available until execution time, we will use the expected value of the measurements for planning: $z(t) = E[z(t)] = E[Hx(t)+v] = H\hat{x}(t)$. As a result, equation (4.14) reduces to $\hat{x}(\tau_i^+) = \hat{x}(\tau_i^-)$. The total time, Δt_a, and the input vector, $u_a(\tau)$ are specified by the action, a. This is best shown via an example. We will compute the total time and input vector for the action 'move right'. Assume for planning

Table 4.1: AMDP State Transition Algorithm

```
Algorithm 4: AMDP State Transitions
Input: AMDP state and action space, characteristic velocity uncertainty
Output: AMDP state transitions
AMDPTRANSITIONS($\mathbb{S}', \mathbb{A}'$)
(1)  foreach ($a \in \mathbb{A}'$)
(2)     foreach ($s \in \mathbb{S}'$)
(3)        $u_a(t), \Delta t_a$ = ActionSpecs($a$)
(4)        $\bar{x}_0, P_0$ = InitialConditions($s, \sigma_v$)
(5)        $\bar{x}, P$ = EKF($u_a(t), \Delta t_a, \bar{x}_0, P_0$)
(6)        $T(s,a)$ = ConvertState($\bar{x}, P$)
(7)  return $T(s,a)$
```

purposes that the input vector, $u(t)$, in equation (4.11) is the velocity of the agent, i.e., $B_{\Delta t} = I\Delta t$. The basic assumption here is that at execution time, the agent will employ a controller which regulates velocity with respect to a coordinate system fixed to the map. For the action 'move right', we will assume that the agent moves at a constant speed, u, to the right. Then the input vector is $u_{moveright}(t) = [u, 0]^T$, and the total time is $\Delta t_{moveright} = \frac{\Delta_x}{u}$ where Δ_x is the width of a single grid cell.

An algorithm for computing the AMDP transition function is given in Table 4.1

In Algorithm 4, the function ActionSpecs($\cdot$), returns the sequence of control inputs and duration for action a. The function InitialConditions($\cdot$) computes the mean and covariance to be used in the EKF equations from the initial AMDP state and the characteristic velocity uncertainty. The function EKF($\cdot$) returns the mean and covariance that result from application of the EKF equations given the initial conditions and control sequence, and the function ConvertState($\cdot$) converts a mean and a covariance matrix into an AMDP state.

The complete AMDP algorithm is given in Table 4.2.

Table 4.2: AMDP algorithm

```
**Algorithm 5:** AMDP Algorithm
**Input:** a POMDP,$\gamma$,$\epsilon$
**Output:** an AMDP policy
AMDP(POMDP)
(1)    Compute the AMDP state space, $\mathbb{S}'$, per section 4.1
(2)    Compute the AMDP action space, $\mathbb{A}'$, per section 4.2
(3)    Compute the AMDP reward function, $R'$, per section 4.3
(4)    $T'$=AMDPtransitions($\mathbb{S}', \mathbb{A}'$)
(5)    $\pi'$ = ValueIteration($\langle \mathbb{S}', \mathbb{A}', T', R' \rangle$, $\gamma$,$\epsilon$)
(6)    return $\pi'$
```

[This page intentionally left blank.]

Chapter 5

An Implementation of the Augmented Markov Decision Process

In this chapter we will develop a simulation with two basic purposes. First, grounding the AMDP path planner in a specific problem will clarify the general purpose algorithm from the previous chapter. Second, the simulation will verify that the AMDP is more robust to uncertainty than the basic MDP but is still computationally tractable for a realistic problem.

In order to accomplish these goals, we first build a model of an agent's dynamics which is generic enough so that it is representative of path planning problems as a whole, but complex enough to be realistic. Then we will develop an EKF state estimator and MDP and AMDP path planners for our model. Finally, we will compare the MDP and AMDP path planners by running pairs of simulations where only the path planner is varied. We will show the results of a large set of simulations which provide statistics to support our claim that the AMDP path planner is more robust than the MDP path planner under agent pose uncertainty, and we will show results from specific simulation scenarios which showcase the differences between the MDP and AMDP planners.

5.1 A 6-DOF Generic Mobile Agent Simulation

In this section, we will construct a mathematical model which describes the motion of an agent. To be complete, we will construct a model which allows both translational and rotational motion in three dimensions (i.e. the model will have six degrees of freedom (6-DOF)). The motion of an agent is due to forces and moments acting on the agent's body, which depend on the agent's states and control inputs. For example, the forces and moments acting on an aircraft are due to the engine thrust, which depends on the throttle input, and due to aerodynamic forces, which depend on the states of the aircraft and the control surface locations (inputs).

We would generally construct our model by describing the forces and moments mathematically in terms of the states and inputs and writing Newton's law for translational and rotational motion in terms of the forces and moments:

$$F = ma = m\ddot{x} \tag{5.1}$$

$$M = J\alpha = J\ddot{\theta} \tag{5.2}$$

where F is the force acting on the agent, m is the mass of the agent, a is the acceleration of the agent, M is the moment acting on the agent, J is the moment of inertia of the agent and α is the angular acceleration of the agent. By replacing F and M in equations (5.1) and (5.2) with their functional equivalents, we end up with the differential equations of motion [2]. In doing this though, we have to take careful note of the fact that Newton's law only holds in an inertial reference frame.

Here though, in the interest of generality, we will not consider parameters such as the mass or moment of inertia of a specific agent. Instead, we will treat the acceleration and the angular acceleration[1] as inputs. If we are given the force and moment equations, we can solve equations (5.1) and (5.2) for the acceleration and angular acceleration and substitute the result into our equations of motion.

We will write the equations of motion in the (assumed inertial) Earth-centered Earth-

[1]In our model we actually omit the angular acceleration and take the angular rate as input.

fixed (ECEF) reference frame. To be complete, we will not neglect the fact that the forces and moments, and by extension the inputs to our system, are defined in the non-inertial reference frame attached to the body of the agent. In Appendix A.2 the equations of motion in the ECEF reference frame are derived in terms of the acceleration and angular rate in the body reference frame (the skew(·) function is also defined in Appendix A.2):

$$\dot{x}_E = v_E \tag{5.3}$$

$$\dot{v}_E = R_{EB}a_B + 2\text{skew}(R_{EB}\omega_B)v_E \tag{5.4}$$

$$\dot{R}_{EB} = \text{skew}(R_{EB}\omega_B)R_{EB} \tag{5.5}$$

where R_{EB} is the coordinate transformation matrix from body coordinates to ECEF coordinates[2], so that:

$$r_E = R_{EB}r_B \tag{5.6}$$

for an arbitrary vector r.

Since our simulation will actually run in small discrete time increments, Δt, we need to compute discrete time versions of the equations of motion. We begin by assuming that a_B and ω_B are constant over the small time intervals. Then we integrate equations (5.3) and (5.4) to get their discrete time equivalents:

$$x_E(t+\Delta t) = x_E(t) + v_E(t)\Delta t + \frac{1}{2}[R_{EB}(t)a_B(t) + 2\text{skew}(R_{EB}(t)\omega_B(t))v_E(t)]\Delta t^2 \tag{5.7}$$

$$v_E(t+\Delta t) = v_E(t) + [R_{EB}(t)a_B(t) + 2\text{skew}(R_{EB}(t)\omega_B(t))v_E(t)]\Delta t \tag{5.8}$$

Then we take the first order Taylor series expansion of equation (5.5) to get its discrete time approximation:

$$R_{EB}(t+\Delta t) = [I + skew(R_{EB}\omega_B(t)\Delta t)]R_{EB}(t) \tag{5.9}$$

In order to simulate the motion of the agent using our model, we just need to specify

[2]If we wish to track the actual Euler angles that describe the rotations from the ECEF frame to the body frame, we can compute them directly from the elements of R_{EB}.

the values of the inputs, a_B and ω_B. In order to do this, we will assume that the agent is equipped with a velocity and attitude regulating feedback controller, such that the closed loop system behaves like a first order system. Then we specify the velocity, v_r, and attitude, θ_r, of a desired reference trajectory[3] (or the planner specifies the desired velocity and attitude) and we compute the acceleration and angular rate for each discrete time-step using the following equations:

$$a_B = \frac{1}{\tau_v}(v_r - v_B) \tag{5.10}$$

$$\omega_B = \frac{1}{\tau_\theta}(\theta_r - \theta_B) \tag{5.11}$$

where τ_v and τ_θ are first order time constants which are chosen to be characteristic of the actual closed loop system response.

The last element of our simulation is a description of the environment (or map). We will describe the environment by a set of polygonal obstacles, $\mathbb{O}$. The motion of the agent will be constrained by the obstacles via equation (5.10). The updated form of the equation is:

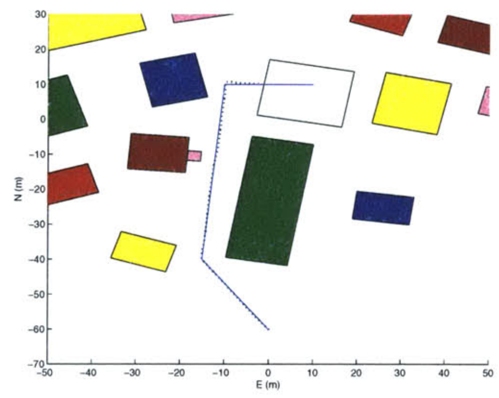
$$a_B = \frac{1}{\tau_v}(v_r - v_B) + a_{OBST} \tag{5.12}$$

where a_{OBST} is a function of the position of the agent. a_{OBST} is zero everywhere except very near the obstacles. Near the obstacles, a_{OBST} takes on a large magnitude and points away from the obstacle, simulating the acceleration resulting from a collision.

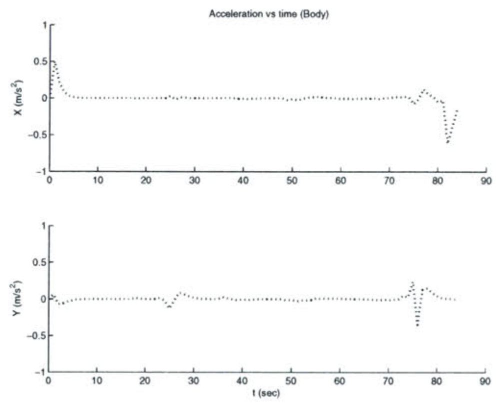
In figure 5-1, the results of a simulation in which the velocity and azimuth commands where chosen manually are shown. The commands were chosen such that the agent would collide with an obstacle so that we could see the effect of equation (5.12). The agent's trajectory through the map is shown as well as time traces of the agent's acceleration, velocity and azimuth angle.

follows a trajectory which ends inside of a building (so there is a collision). figure 1: trajectory overlaid on the map (including reference trajectory) figure 2: accelerations vs time

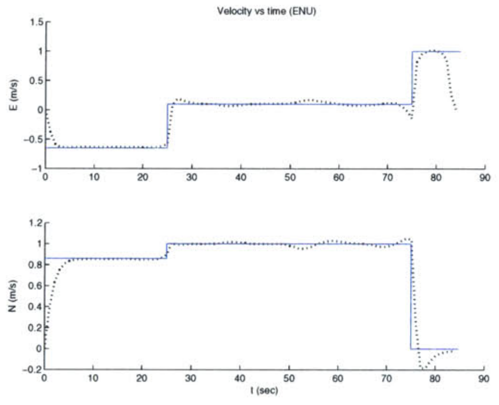
[3]In our simulation, we will specify the velocity and attitude in an East-North-Up (ENU) coordinate system and translate them to body coordinates during the simulation.



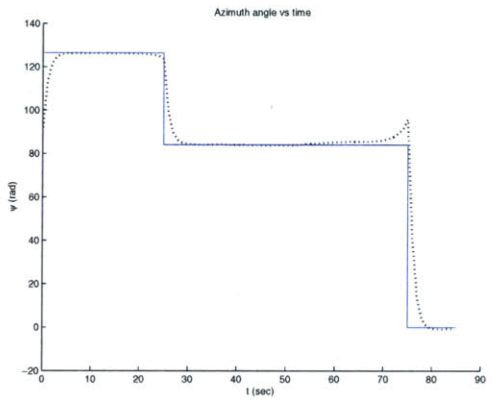
(a) Trajectory



(b) Acceleration



(c) Velocity



(d) Azimuth

Figure 5-1: (a) The simulated trajectory of the agent through the map. (b) The simulated acceleration of the agent, note the negative spike in the body x-axis acceleration at 81 seconds due to the collision. (c) The commanded and actual velocity of the agent. (d) The commanded and actual azimuth angle of the agent. In all of these plots, the solid line is the commanded (or reference) value and the dotted line is the actual value.

5.2 The Extended Kalman Filter Equations

Now that we have defined the equations of motion for our model we can develop an extended Kalman filter to estimate the states of the model. First we will define the states of the filter. Next we will develop the time propagation equations for the filter states and the error covariance matrix. Then we will define a set of sensors for the agent. We will create a mathematical model and the corresponding filter update equations for each sensor.

5.2.1 The States of the EKF

The non-linear states of the EKF will be the same as the states of the model with the exception that the acceleration and angular rate will be considered states instead of inputs. Even though the acceleration and angular rates were taken to be inputs for the purpose of modeling, it is not reasonable to assume that they will be known. Since these states have been added to the EKF, we also need to add the following two stochastic equations of motion:

$$\dot{a}_B = w_a \tag{5.13}$$

$$\dot{\omega}_B = w_\omega \tag{5.14}$$

where w_a and w_ω are Gaussian white noise processes with power spectral densities W_a and W_ω respectively. The discrete time versions of (5.13) and (5.14) are:

$$a_B(t + \Delta t) = a_B(t) + w_{a,\Delta t} \tag{5.15}$$

$$\omega_B(t + \Delta t) = \omega_B(t) + w_{\omega,\Delta t} \tag{5.16}$$

where $w_{a,\Delta t}$ and $w_{\omega,\Delta t}$ are Gaussian white sequences with variances $W_{a,\Delta t}$ and $W_{\omega,\Delta t}$ respectively. From equations (5.15) and (5.16), we can see that $w_{a,\Delta t}$ and $w_{\omega,\Delta t}$ have an intuitive physical interpretation. They are simply the amount of change, during one time-step, in the acceleration and angular rate of the agent.

As in section 3.3, we will also define perturbation states for the EKF:

$$\delta x = \hat{x}_E - x_E \tag{5.17}$$

$$\delta v = \hat{v}_E - v_E \tag{5.18}$$

$$\delta a = \hat{a}_B - a_B \tag{5.19}$$

$$\begin{aligned} \delta\theta &= \hat{\theta}_E - \theta_E, \text{ such that:} \\ R_{EB} &= (I + \text{skew}(\delta\theta))\, \hat{R}_{EB} \end{aligned} \tag{5.20}$$

$$\delta\omega = \hat{\omega}_B - \omega_B \tag{5.21}$$

where the state estimates are denoted by the hat notation.

In order to define the EKF error covariance matrix, we define the vector δ, which contains all of the perturbation states, $\delta^T = [\delta x^T \quad \delta v^T \quad \delta a^T \quad \delta\theta^T \quad \delta\omega^T]$. Then the covariance matrix is defined to be $P = E[\delta\delta^T]$.

The EKF states are initialized to the best available initial estimates ($\hat{x}_E(0) = \hat{x}_{E,0}$, etc.). The uncertainty in each of these estimates is given by a zero mean Gaussian distribution which is characterized by specifying its variance ($\sigma^2_{x,0}$, etc.). We then initialize the perturbation states to their expected values, which are all zero because, given the above definition of the initial uncertainty, the expected values of equations (5.17) through (5.21) are all zero[4]. Then we assume that the initial errors are uncorrelated, so that the initial error covariance matrix is:

$$P_0 = \begin{pmatrix} \sigma^2_{x,0} & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{v,0} & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{a,0} & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{\theta,0} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{\omega,0} \end{pmatrix} \tag{5.22}$$

[4]Expanding the expected value of equation (5.20) and solving gives $E[\text{skew}(\delta\theta_0)] = 0$, so $E[\delta\theta_0] = 0$.

5.2.2 EKF Time Propagation Equations

Now that we have defined the EKF states, and their initial conditions, we can define the equations which describe their evolution through time. We begin by applying equation (3.41) to the discrete time equations (5.7), (5.8), (5.9), (5.15) and (5.16) to obtain the discrete time equations of motion for the full non-linear state estimates:

$$\hat{x}_E(t+\Delta t) = \hat{x}_E(t) + \hat{v}_E(t)\Delta t + \frac{1}{2}[\hat{R}_{EB}(t)\hat{a}_B(t) + 2\text{skew}(\hat{R}_{EB}(t)\hat{\omega}_B(t))\hat{v}_E(t)]\Delta t^2 \quad (5.23)$$

$$\hat{v}_E(t+\Delta t) = \hat{v}_E(t) + [\hat{R}_{EB}(t)\hat{a}_B(t) + 2\text{skew}(\hat{R}_{EB}(t)\hat{\omega}_B(t))\hat{v}_E(t)]\Delta t \quad (5.24)$$

$$\hat{R}_{EB}(t+\Delta t) = [I + skew(\hat{R}_{EB}(t)\hat{\omega}_B(t)\Delta t)]\hat{R}_{EB}(t) \quad (5.25)$$

$$\hat{a}_B(t+\Delta t) = \hat{a}_B(t) \quad (5.26)$$

$$\hat{\omega}_B(t+\Delta t) = \hat{\omega}_B(t) \quad (5.27)$$

We propagate the EKF states between measurements by direct application of equations (5.23) through (5.27).

In order to propagate the perturbation states and the covariance matrix through time, we first derive the perturbation state equations of motion (see Appendix A.3). The perturbation state equations of motion are shown here in the vector function form of equation (3.31):

$$\begin{pmatrix} \delta\dot{x} \\ \delta\dot{v} \\ \delta\dot{a} \\ \delta\dot{\theta} \\ \delta\dot{\omega} \end{pmatrix} = \underbrace{\begin{pmatrix} \delta v \\ f_{\delta v} \\ 0 \\ \hat{R}_{EB}\delta\omega + 2\text{skew}(\hat{R}_{EB}\hat{\omega}_B)\delta\theta \\ 0 \end{pmatrix}}_{f} + \begin{pmatrix} 0 \\ 0 \\ w_a \\ 0 \\ w_\omega \end{pmatrix} \quad (5.28)$$

where:

$$f_{\delta v} = \text{skew}(\hat{R}_{EB}\hat{\omega}_B)\delta v + \hat{R}_{EB}\delta a$$

$$+[\text{skew}(\hat{R}_{EB}\hat{a}_B) - \text{skew}(\hat{v}_E)\text{skew}(\hat{R}_{EB}\hat{\omega}_B)]\delta\theta$$
$$-\text{skew}(\hat{v}_E)\hat{R}_{EB}\delta\omega \tag{5.29}$$

We propagate the perturbation state estimates between measurements by direct application of equation (5.28). Since the function f is zero if all of the perturbation states are zero, and $E[w_a] = E[w_\omega] = 0$, the perturbation state estimates remain zero between measurements.

To propagate the covariance matrix, we compute the Jacobian of the function f, J_f. Since f is linear in the perturbation states we can compute the Jacobian directly:

$$J_f = \begin{pmatrix} 0 & I & 0 & 0 & 0 \\ 0 & \text{skew}(\hat{R}_{EB}\hat{\omega}_B) & \hat{R}_{EB} & J_{\delta v,\delta\theta} & -\text{skew}(\hat{v}_E)\hat{R}_{EB} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\text{skew}(\hat{R}_{EB}\hat{\omega}_B) & \hat{R}_{EB} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.30}$$

where:

$$J_{\delta v,\delta\theta} = [\text{skew}(\hat{R}_{EB}\hat{a}_B) - \text{skew}(\hat{v}_E)\text{skew}(\hat{R}_{EB}\hat{\omega}_B)] \tag{5.31}$$

Now we apply the discrete time version of equation (3.45) (i.e. equations (3.23) and (3.30)):

$$P(t + \Delta t) = (I + J_f\Delta t)P(t)(I + J_f\Delta t)^T + W_{\Delta t} \tag{5.32}$$

where:

$$W_{\Delta t} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & W_{a,\Delta t} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & W_{\omega,\Delta t} \end{pmatrix} \tag{5.33}$$

5.2.3 Sensors and Measurement Updates

Now, to complete our EKF, we will define the measurement update equations. In order to define the measurement update equations, we will define a set of sensors for our agent. For each sensor, we will define the (possibly non-linear) measurement function, g, and the measurement noise variance, V. We will also compute the Jacobian of each measurement function, J_g, so that, to incorporate the measurements, we simply apply equations (3.47) through (3.50).

One sensor that we will equip our agent with is an inertial measurement unit (IMU). The IMU consists of two parts, an accelerometer and a gyroscope. The accelerometer takes measurements of the agent's acceleration at a very high rate and integrates these values to obtain a change in velocity, Δv which it reports to the agent at a slower rate (we will use 100 Hz). The other sensors will all have rates which are slower than 100 Hz (except the gyroscope which we will also assume operates at 100 Hz). As a result, we will take Δt in the discrete time equations above to be $(\frac{1}{100\text{Hz}}=)$ .01 seconds. Next, since we have assumed that the acceleration is approximately constant during these small time-steps, we will divide Δv by Δt to get the acceleration. However, even though we will assume the accelerometer is rigidly attached to the body reference frame, it does not provide a direct measurement of a_B. This is because the accelerometer measures the specific force which includes the actual acceleration plus gravity and inertial effects. As a result, the measurement equation for the accelerometer is:

$$z_{acc} = a_B + \omega_B \times v_B + R_{EB}^T \frac{x_E}{||x_E||} g + v_{acc} \tag{5.34}$$

where g is the gravitational constant of Earth ($g \approx 9.81\frac{\text{m}}{\text{s}^2}$), and v_{acc} is a Gaussian white noise sequence with variance $V_{acc} = \sigma_{\text{acc}}^2 I$. The Jacobian of the accelerometer measurement is:

$$J_{acc} = \begin{bmatrix} J_{acc,x_E} & \text{skew}(\hat{\omega}_B)\hat{R}_{EB}^T & I & J_{acc,\delta\theta} & -\text{skew}(\hat{R}_{EB}^T\hat{v}_E) \end{bmatrix} \tag{5.35}$$

where:

$$J_{acc,x_E} = \hat{R}_{EB}^T(I - \frac{1}{||\hat{x}_E||^2}\hat{x}_E\hat{x}_E^T)\frac{g}{||\hat{x}_E||} \tag{5.36}$$

$$J_{acc,\delta\theta} = \text{skew}(\hat{\omega}_B)\hat{R}_{EB}^T\text{skew}(\hat{v}_E) + g\hat{R}_{EB}^T\frac{\hat{x}_E}{||\hat{x}_E||} \tag{5.37}$$

The gyroscope takes measurements of the agent's angular rate at a high frequency, and integrates these measurements to obtain a change in attitude, $\Delta\theta$, which it reports to the agent at a slower rate. Similar to the accelerometer, we will divide $\Delta\theta$ by Δt to get the angular rate, which we have assumed to be constant during the time-steps. In this case however, the agent's angular rate expressed in an inertial frame is the same as it is in the body frame, so the gyroscope measurement is a direct measurement of the agent's angular rate:

$$z_{gyro} = \omega_B + v_{gyro} \tag{5.38}$$

where v_{gyro} is a white noise sequence with variance $V_{gyro} = \sigma_{gyro}^2 I$. The Jacobian of the gyroscope measurement is:

$$J_{gyro} = \begin{bmatrix} 0 & 0 & 0 & I & 0 \end{bmatrix} \tag{5.39}$$

We have assumed that the IMU is a strapdown system which means that the acceleration and angular rate measurements are given in body coordinates and must be converted to inertial coordinates before they can be integrated. Since this conversion is done via a transformation matrix which relies on information provided by the gyroscope, the gyroscope error results in an acceleration error in the inertial frame which integrates into error in the velocity and position. The main reason that we included all six degrees of freedom in our model was to capture this effect.

The next sensor that we will add is a radio which communicates with a receiver at a known location. We will refer to the receiver as a sensor node, and we will denote its position by $x_{E,sn}$. By timing the signal delay between the sensor node and the agent, we obtain a measurement of the range from the agent to the sensor node. The resulting measurement equation is:

$$z_{sn} = ||x_{E,sn} - x_E|| + v_{sn} \tag{5.40}$$

where v_{sn} is a white noise sequence with variance $V_{sn} = \sigma_{sn}^2$ which is intended to model the uncertainty in the signal delay as well as the position of the sensor node. The Jacobian of the sensor node measurement is:

$$J_{snr} = \begin{bmatrix} \frac{(\hat{x}_E - x_{E,sn})}{||x_{E,sn} - \hat{x}_E||} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.41}$$

The sensor node range measurements will be received at a frequency of 1 Hz.

The last sensor that we will consider is a camera. The agent employs the camera by standing so that the camera is pointed directly forward (along the body x-axis), such that a known landmark is in the view of the camera[5]. Then, based on the location of the landmark within the image, the angle between the agent's body x-axis and the vector from the agent to the landmark can be computed. We will refer to this measurement as a landmark bearing measurement, and we will denote the position of the landmark by $x_{E,lm}$. The landmark bearing measurement equation is:

$$z_{lmb} = \arctan\left(\frac{h_{lm}}{g_{lm}}\right) + v_{lm} \tag{5.42}$$

where:

$$h_{lm} = [0 \quad 1 \quad 0] \circ R_{EB}^T(x_{E,lm} - x_E) \tag{5.43}$$

$$g_{lm} = [1 \quad 0 \quad 0] \circ R_{EB}^T(x_{E,lm} - x_E) \tag{5.44}$$

and v_{lm} is a white noise sequence with variance $V_{lm} = \sigma_{lm}^2$. The Jacobian of the sensor node measurement is:

$$J_{lm} = \begin{bmatrix} J_{lm,x_E} & 0 & 0 & J_{lm,\delta\theta} & 0 \end{bmatrix} \tag{5.45}$$

where:

$$J_{lm,x_E} = \frac{g_{lm} f_{lm}^{x_E} - f_{lm} g_{lm}^{x_E}}{g_{lm}^2 + f_{lm}^2} \tag{5.46}$$

[5]We will assume perfect data association, which means that the agent always knows exactly which landmark it is looking at.

$$J_{lm,\delta\theta} = \frac{g_{lm} f_{lm}^{\delta\theta} - f_{lm} g_{lm}^{\delta\theta}}{g_{lm}^2 + f_{lm}^2} \tag{5.47}$$

$$h_{lm}^{x_E} = \frac{\partial}{\partial x_E} h_{lm} = -[0 \quad 1 \quad 0]\hat{R}_{EB}^T \tag{5.48}$$

$$g_{lm}^{x_E} = \frac{\partial}{\partial x_E} g_{lm} = -[1 \quad 0 \quad 0]\hat{R}_{EB}^T \tag{5.49}$$

$$h_{lm}^{\delta\theta} = \frac{\partial}{\partial \delta\theta} h_{lm} = [0 \quad 1 \quad 0]\text{skew}\Big(\hat{R}_{EB}^T (x_{E,lm} - \hat{x}_E)\Big) \tag{5.50}$$

$$g_{lm}^{\delta\theta} = \frac{\partial}{\partial \delta\theta} g_{lm} = [1 \quad 0 \quad 0]\text{skew}\Big(\hat{R}_{EB}^T (x_{E,lm} - \hat{x}_E)\Big) \tag{5.51}$$

The landmark bearing measurement will not be received at a regular frequency. Instead it will be taken to be an information gathering action. For simulation purposes though, we will assume that it takes the agent ten seconds to take a landmark bearing measurement.

5.3 The Motion Planners

Now that we have defined a map, described the motion of an agent through the map and equipped the agent with an EKF for state estimation, our motion planner test bed is complete. We are now ready to compare our AMDP path planner to the basic MDP path planner. We will begin by describing the states, actions, state transition function and reward function for the MDP, and then we will describe how each of these is modified to build the AMDP.

5.3.1 The Markov Decision Process

To construct the MDP state space, we begin by designating some point within our map to be the origin, $(x_0, y_0) = (0, 0)$. We then describe the region of the map over which we wish to plan, $\mathbb{S}_0$, by four parameters, x_{min}, x_{max}, y_{min} and y_{max}:

$$\mathbb{S}_0 = \{(x, y) | x \in (x_{\text{min}}, x_{\text{max}}), y \in (y_{\text{min}}, y_{\text{max}})\} \tag{5.52}$$

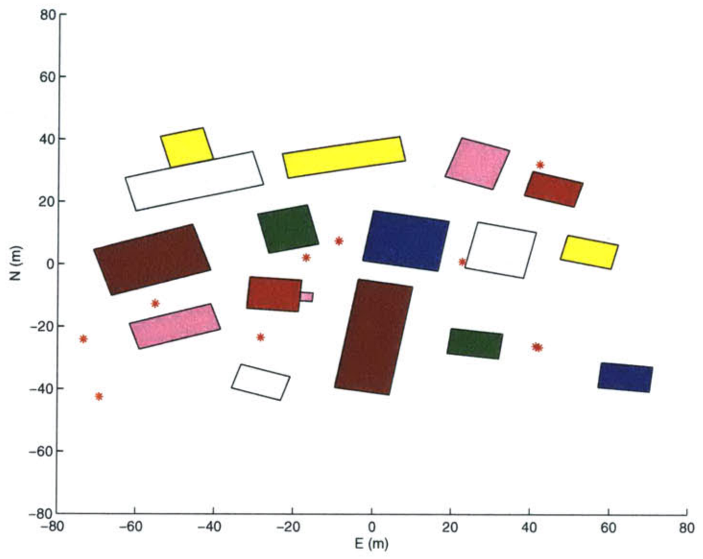


Figure 5-2: The space over which we wish to plan is a subset of two dimensional Euclidean space which is occupied by polygonal obstacles (buildings) and hazards which are indicated by the stars.

Figure 5-2 shows the space over which we wish to plan. In order to make the path planning problem more interesting, we have also included a set of stationary hazards, which we may think of as stationary adversarial agents or hazardous terrain.

We then construct a uniform grid over this space made up of grid cells with width Δ_x and height Δ_y. Each grid cell is a state of the MDP. We will represent the grid cells by the coordinates of their centers, so that the set of MDP states, $\mathbb{S}$, is:

$$\mathbb{S} = \{(x, y) | x \in \mathbb{X}, y \in \mathbb{Y}\} \tag{5.53}$$

$$\mathbb{X} = \{x_{\text{min}} + \frac{\Delta_x}{2}, x_{\text{min}} + \frac{3\Delta_x}{2}, \ldots, x_{\text{max}} - \frac{\Delta_x}{2}\}$$
$$\mathbb{Y} = \{y_{\text{min}} + \frac{\Delta_y}{2}, y_{\text{min}} + \frac{3\Delta_y}{2}, \ldots, y_{\text{max}} - \frac{\Delta_y}{2}\}$$

In order to compute the reward and transition function it will be necessary to represent the locations of the obstacles, specified as polygons, and hazards, specified as points, in terms of the set of discrete states, $\mathbb{S}$. This is done by specifying for each cell of the grid, whether that cell is empty, contains an obstacle, or contains a hazard. An algorithm which

Table 5.1: Algorithm to descretize a subset of two dimensional Euclidean space which is occupied by polygonal obstacles and point-based hazards.

```
Algorithm 6: AssignCellContents
Input: a set of grid cells, a set of obstacles, a set of hazards
Output: contents of each grid cell
ASSIGNCELLCONTENTS($\mathbb{S}, \mathbb{O}, \mathbb{H}$)
(1)  foreach ($s \in \mathbb{S}$)
(2)     $contents(s)$='empty'
(3)  foreach ($o \in \mathbb{O}$)
(4)     foreach (connected pair of vertices, $(v_1, v_2) \in o$)
(5)        $LIST = [\ ]$
(6)        $contents(s(v_1))$='obstacle'
(7)        $contents(s(v_2))$='obstacle'
(8)        $append(LIST, neighbors(s(v_1)))$
(9)        $append(LIST, neighbors(s(v_2)))$
(10)       while ($LIST$ is not empty)
(11)          $s = pop(LIST)$
(12)          if ($intersect(s, v_1, v_2)$)
(13)             $contents(s)$='obstacle'
(14)             $append(LIST, neighbors(s))$
(15) foreach ($h \in \mathbb{H}$
(16)    $contents(s)$='hazard'
(17) return contents
```

specifies the contents of each cell is shown in Table 5.1. The following functions are used in Algorithm 6: $s(p)$ returns the grid cell that contains the point p, $append(LIST, cells)$ adds the grid cells, $cells$, to the end of $LIST$, $neighbors(cell)$ returns the eight (or less if $cell$ is on the edge of the grid) cells which surround $cell$, $pop(LIST)$ returns the first element of $LIST$ and removes it from $LIST$, $intersect(cell, v_1, v_2)$ returns $true$ if the line from point v_1 to point v_2 intersects $cell$ and false otherwise. The results of Algorithm 6 for the map shown in figure 5-2 is shown in figure 5-3, where cells containing obstacles are indicated by circles and cells containing hazards are indicated by a +. The resolution of this grid is 2 meters in both the East and North directions.

Next we define the actions for the MDP. First we will define the set of actions, then, in table 5.2, we specify, for each action, a corresponding reference velocity, v_r, and attitude, θ_r, so that the actions can be executed in the simulation via equations (5.10) and (5.11). v_r

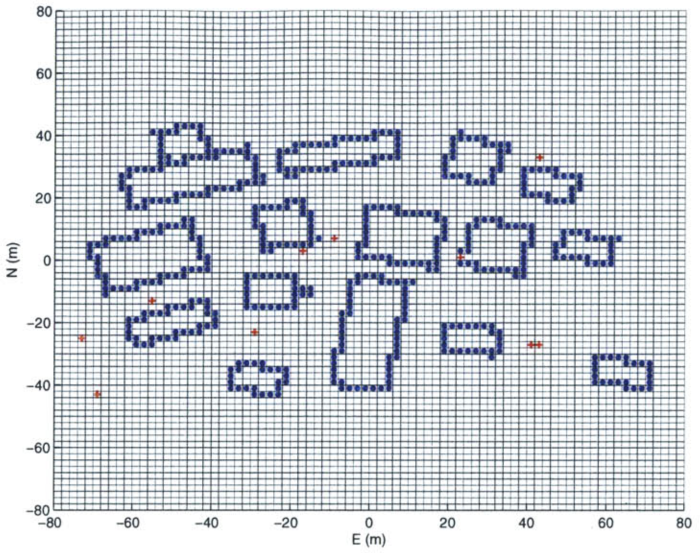


Figure 5-3: MDP Grid Contents: The contents of the grid corresponding to figure 5-2 are indicated with circles for obstacles and a + for hazards.

Table 5.2: MDP actions with corresponding reference velocity and attitude

Action	v_r	θ_r
'move north'	$(0 \quad 1 \quad 0)$	$(0 \quad 0 \quad \frac{\pi}{2})$
'move east'	$(1 \quad 0 \quad 0)$	$(0 \quad 0 \quad 0)$
'move south'	$(0 \quad -1 \quad 0)$	$(0 \quad 0 \quad -\frac{\pi}{2})$
'move west'	$(-1 \quad 0 \quad 0)$	$(0 \quad 0 \quad \pi)$
'stop'	$(0 \quad 0 \quad 0)$	$(0 \quad 0 \quad 0)$

is specified by a velocity vector in the East-North-Up coordinate system, and θ_r is specified by the Euler angles which describe the transformation from East-North-Up coordinates to body coordinates[6].

The set of MDP actions, $\mathbb{A}$, is:

$$\mathbb{A} = \{\text{'move north'}, \text{'move east'}, \text{'move south'}, \text{'move west'}, \text{'stop'}\} \tag{5.54}$$

The 'stop' action is intended to be invoked when the agent believes it has reached the goal state.

[6]We will assume that the agent is always upright in the East-North-Up coordinate system, which means $\theta_r = (0 \quad 0 \quad \psi)$

Next we will define the state transition function for the MDP. First we define the transition function for the action 'stop', because it is the simplest. The transition for this action will be deterministic and, regardless of which state the action is invoked from, the agent will transition to a terminal state which we will denote $s = \mathit{null}$, with probability one. So we have:

$$T(s, \text{'stop'}, s') = \left\{ \begin{array}{cc} 1 & \text{if } s' = \mathit{null} \\ 0 & \text{otherwise} \end{array} \right\} \tag{5.55}$$

The 'move' actions will be described by a stochastic transition which we obtain by assuming that the uncertainty in the velocity during the 'move' action is Gaussian with some characteristic variance[7], σ_{v*}^2. Since the velocity is taken to be constant during the action, the deterministic part of the state transition obeys the following linear relation:

$$\begin{bmatrix} x(\Delta t) \\ y(\Delta t) \\ v_x(\Delta t) \\ v_y(\Delta t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ y(0) \\ v_x(0) \\ v_y(0) \end{bmatrix} \tag{5.56}$$

Then, in the same way that we developed equation (3.30), we can show:

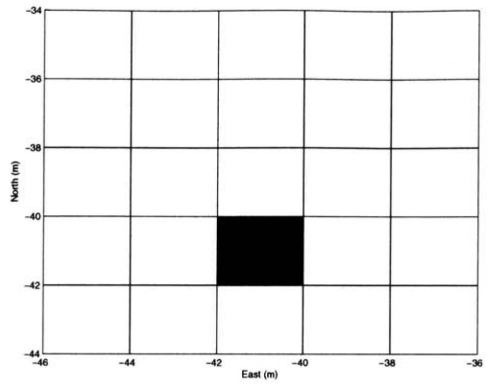
$$P(\Delta t) = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} P(0) \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{5.57}$$

with

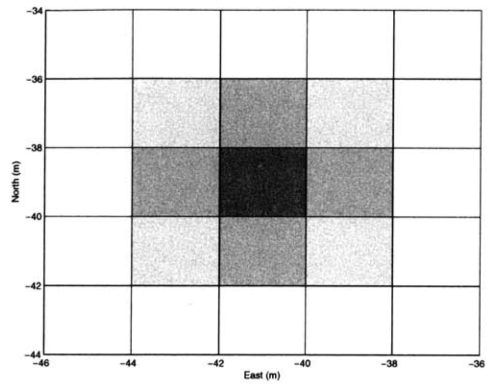
$$P(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{v*}^2 & 0 \\ 0 & 0 & 0 & \sigma_{v*}^2 \end{bmatrix} \tag{5.58}$$

which results in the following covariance matrix for the position after the action (top left

[7]The characteristic uncertainty in the velocity depends on the quality of the sensors, and it is obtained for a specific set of sensors empirically by observing the results of simulations run with that set of sensors.



(a) Distribution before 'move north'



(b) Distribution after 'move north'

Figure 5-4: (a) Distribution before action 'move north'. (b) Distribution after action 'move north' with characteristic velocity variance $= .5\frac{\mathrm{m}^2}{\mathrm{s}^2}$. Black indicates probability = 1, white indicates probability = 0.

2x2 portion of $P(\Delta t)$):

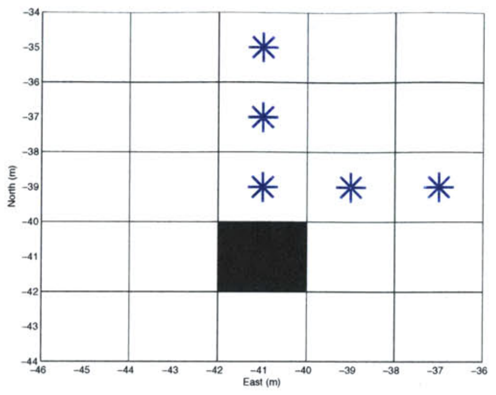
$$P_{xy}(\Delta t) = \begin{bmatrix} \sigma_{v*}^2 \Delta t^2 & 0 \\ 0 & \sigma_{v*}^2 \Delta t^2 \end{bmatrix} \tag{5.59}$$

To obtain Δt, we take $\Delta t = \frac{\Delta_x}{||v_r||} = \Delta_x$ for actions 'move east' and 'move west', and $\Delta t = \Delta_y$ for actions 'move north' and 'move south'. Equations (5.56) and (5.59) show that the distribution of the agent's position after the 'move' actions is a Gaussian distribution with the mean at the center of the intended posterior grid cell, and with variance $\sigma_{v*}^2 \Delta t^2$. So, denoting the intended posterior state by s_i', we have:

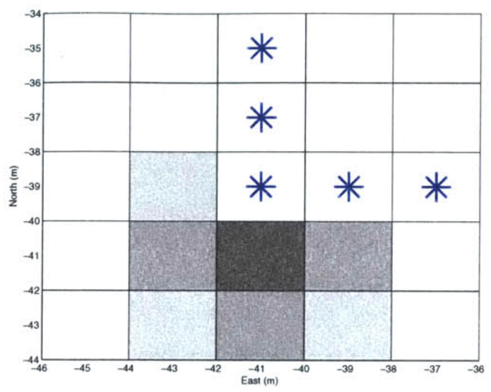
$$T(s, \text{'move *'}, s') = \left\{ \begin{array}{ll} 0 \text{ if grid cell } s' \text{ contains an obstacle} & \\ \int_{s'} N(s_i', P_{xy}(\Delta t)) & \text{otherwise} \end{array} \right\} \tag{5.60}$$

For simulation purposes, we compute $T(s, \text{'move *'}, s')$ only for the intended posterior state and the eight surrounding states, and then we re-scale the probabilities so that we have a well formed distribution. Figure 5-4 shows the distribution resulting from the action 'move north' within the grid from figure 5-2, with $\sigma_{v*}^2 = .5\frac{\mathrm{m}^2}{\mathrm{s}^2}$.

When a 'move *' action would result in a posterior state which is occupied by an obstacle, we replace s_i' in equation (5.60) with s. Figure 5-5 shows the distribution



(a) Distribution before 'move north'



(b) Distribution after 'move north'

Figure 5-5: (a) Distribution before action 'move north'. (b) Distribution after action 'move north' with characteristic velocity variance $= .5\frac{\text{m}^2}{\text{s}^2}$, with obstacles, indicated by stars, preventing the action 'move north'. Black indicates probability = 1, white indicates probability = 0.

resulting from the action 'move north' when obstacles are present.

The last element of the MDP is the reward function. We have three basic objectives: (1) get to the goal, (2) get there safely (avoiding hazards), and (3) get there in the minimum amount of time. To achieve these three objectives we will define three reward functions and the actual reward will be their sum. To achieve the first objective, we will give the agent a large reward for executing the action 'stop' from the goal state, and zero reward for executing that action from any other state:

$$R_1(s,a) = \left\{ \begin{array}{ll} 10000 & \text{if } s \text{ is the goal state and } a = \text{'stop'} \\ 0 & \text{otherwise} \end{array} \right\} \tag{5.61}$$

To achieve the second objective, we will assign a large negative reward to grid cells that contain hazards:

$$R_2(s,a) = \left\{ \begin{array}{ll} -10000\Delta t_a & \text{if } s \text{ contains a hazard} \\ 0 & \text{otherwise} \end{array} \right\} \tag{5.62}$$

where Δt_a is the duration of the action. To achieve the third objective, we will assign a cost, or negative reward, to each of the 'move' actions equal to the duration of the action:

$$R_3(s,a) = \left\{ \begin{array}{ll} -\Delta_x & \text{if } a = \text{'move east' or 'move west'} \\ -\Delta_y & \text{if } a = \text{'move north' or 'move south'} \\ 0 & \text{otherwise} \end{array} \right\} \tag{5.63}$$

During implementation, we will make one addition to the MDP planner. In order to take advantage of the landmark bearing measurements which are not received automatically, we will employ a heuristic which tells the agent to take a set of landmark bearing measurements (one facing each direction; north, east, south and west) any time its position variance gets above some threshold. To insure that the agent does not get stuck in one place taking landmark bearing measurements over and over because the measurements do not cause its position variance to go below the threshold, these sets of measurements will be constrained to have at least one minute between them. It is important to note that this heuristic applies only during execution and does not change the MDP planner itself. It is simply intended to keep the uncertainty somewhat bounded so that its effect is reduced. It is also important to note that such a heuristic is unnecessary for the AMDP; the AMDP policy will choose to use this action only when it is necessary.

To solve the MDP, we employ the value iteration algorithm from chapter 2, Algorithm 1.

5.3.2 The Augmented Markov Decision Process

The procedure for constructing an AMDP from an MDP was covered in detail in chapter 4. As a result, in this section we will only discuss parts of the procedure which are specific to this example.

Just as in section 4.1, the AMDP state space is constructed by simply augmenting the MDP state space with covariance states so that each state of the AMDP represents a Gaussian distribution. We have chosen to include two additional states, a standard deviation in the East direction, σ_E, and a standard deviation in the North direction, σ_N. For the time being, the author has chosen to omit the correlation coefficient, potentially degrading performance, in order to make the algorithm run faster. It is the author's belief that the effect of neglecting the correlation coefficient will be small enough so that this trade-off will be beneficial. The space of possible standard deviation values, Σ, is constructed similarly to the way that the MDP state space was constructed. We begin by constructing a continuous set of standard deviation values:

$$\Sigma_0 = [0, \sigma_{max}] \tag{5.64}$$

where σ_{max} is chosen to represent the largest standard deviation value that we expect the agent to experience. We then create a type of uniform grid over this set with segments of width Δ_σ. Then the discrete set of standard deviation values is taken to be the centers of these grid cells:

$$\Sigma = \{\frac{\Delta_\sigma}{2}, \frac{3\Delta_\sigma}{2}, \ldots, \sigma_{max} - \frac{\Delta_\sigma}{2}\} \tag{5.65}$$

The selection of Δ_σ has important ramifications. Choosing Δ_σ to be very small causes the solutions to be costly to compute in terms of time. However, choosing Δ_σ to be too large causes the planner to misrepresent the effects of the uncertainty which can result in very poor performance. For example, imagine an agent which has only an IMU and no other sensors. Since the IMU only bounds the acceleration error, the agent's position error will grow unbounded. However, the rate of growth in the position error is finite, so the amount that the position error grows during an action (a 'move' action for example) is finite. If this amount is less than $\frac{\Delta_\sigma}{2}$, the error growth will not cause the discrete standard deviation state in the planner to change, and, rather than unbounded growth in the position error, the planner will expect a constant position error.

The only change in the action space between the MDP and the AMDP is the addition of a new type of action. The landmark bearing measurement, which was described in section 5.2.3, will be taken by the AMDP to be an information gathering action. In fact, we will add four landmark bearing measurement actions, one facing each direction (North, East, South and West).

The state transition function and reward function are computed using (Algorithm to be added to chapter 4) and equation (4.9) respectively. Also, equation (5.63) from the

MDP reward function is slightly modified to include the landmark bearing measurement action before applying equation (4.9):

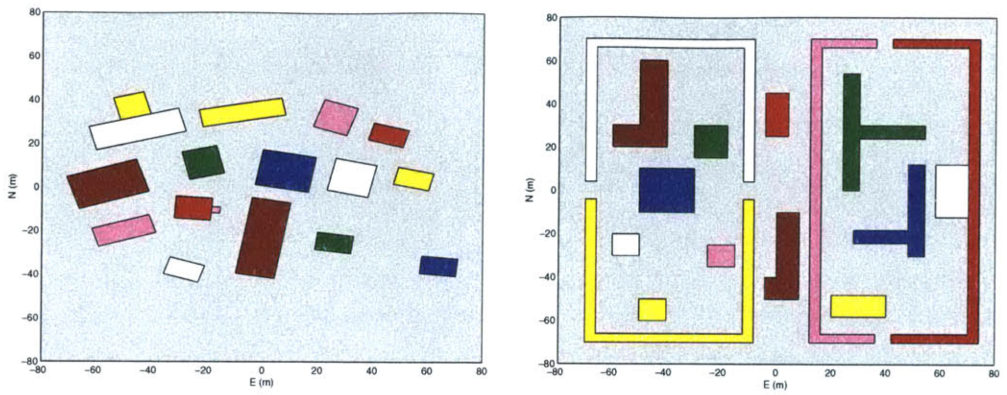
$$R_3(s,a) = \left\{ \begin{array}{ll} -\Delta_x & \text{if } a = \text{`move east' or `move west'} \\ -\Delta_y & \text{if } a = \text{`move north' or `move south'} \\ -10 & \text{if } a = \text{`landmark bearing measurement'} \\ 0 & \text{otherwise} \end{array} \right\} \tag{5.66}$$

We also use the value iteration algorithm from chapter 2 to solve the AMDP.

5.4 Preliminary Experimental Results

In this section we will analyze the results of a set of simulations that were run using the motion model, EKF and AMDP and MDP planners described above. First we will statistically compare the AMDP and MDP planners using a large set of simulations in which the map and sensor qualities are varied. Then we will take a look at some specific simulation scenarios which provide further insight into the results of the larger set of simulations, and which showcase the differences between the MDP and the AMDP.

Before each simulation begins, all of the free parameters, such as the obstacle, hazard and sensor node locations and the measurement noise parameters, are assigned values. Then, given these parameter values, the planner creates and stores a policy (if the appropriate transition function has already been created and stored the planner will use it, otherwise it will create a new one and store it). Then the initial conditions are specified, including the simulation states and the EKF states and covariance matrix. Then the simulation is propagated in small discrete time-steps according to the flow diagram in figure 4-1, with the 'Agent Dynamics' box replaced by the model described in section 5.1. At each time-step, the action is selected using the policy generated by the planner and the estimate from the EKF. The reference velocity and attitude corresponding to the selected action are then fed into the 6-DOF simulation, where the time and states are then propagated. The EKF estimate is updated and the next action is selected. This loop continues to run until the agent chooses to execute the 'stop' action, or until the simula-



(a) Warehouse Site (b) Two Fortresses

Figure 5-6: Half of the simulations where run on each of these maps. (a) Warehouse site map, (b) Two Fortresses map.

tion time reaches ten minutes, whichever comes first. The performance of the planner is measured for each simulation by computing the actual cumulative reward for that simulation. This is done by initializing the cumulative reward to zero and then incrementing it at each time-step by $(R_1 + R_2 + \Delta t)$ from equations equations (5.61) and (5.62), with Δt_a replaced by the simulation time-step, Δt, in equation (5.62).

5.4.1 Statistical Analysis

A set of 960 simulations was conducted to compare the effect of uncertainty on the MDP and AMDP planners. Each simulation was run two times, once using the MDP planner and once using the AMDP planner. The simulations are broken into six subsets which are shown in table 5.3. These subsets vary in that they were simulated using two different maps, which are shown in figure 5-6, and using three different simulated sensor qualities, which are described in table 5.4. By varying the sensor qualities, we can vary the amount of uncertainty that the agent will experience and we can test the extent to which the AMDP is more robust to the uncertainty than the MDP.

Within each simulation subset, eight different hazard scenarios are randomly selected, and for each hazard scenario twenty pairs of initial positions and goal positions are chosen

Table 5.3: Map and sensor quality for each simulation subset.

Subset	Map	Sensor Quality
1-a	Warehouse site	a
1-b	Warehouse site	b
1-c	Warehouse site	c
2-a	Two Fortresses	a
2-b	Two Fortresses	b
2-c	Two Fortresses	c

Table 5.4: The standard deviation of the measurement noise (as seen in section 5.2.3) is given for each sensor, for each quality level.

Quality Level	σ_{acc}	σ_{gyro}	σ_{sn}	σ_{lm}
a	200 μg	.05 $\frac{deg}{sec}$	1 m	3 deg
b	800 μg	.17 $\frac{deg}{sec}$	4 m	3 deg
c	1500 μg	.57 $\frac{deg}{sec}$	8 m	3 deg

at random[8], resulting in a total of 160 simulations per subset. Hazard scenarios one and two contain ten hazards, three and four contain twenty hazards and five and six contain forty hazards. In each of these cases, the hazards are randomly distributed throughout the map. Hazard scenarios seven and eight contain a different type of hazard which we will call line-of-sight hazards. The objective is for the agent not to be seen by these hazards. This new objective is reflected by modifying the MDP reward function. We first define the function $\zeta(s_1, s_2)$ which returns 0 if the straight line from s_1 to s_2 intersects an obstacle and 1 otherwise. Then we replace equation (5.62) of the MDP rewards with:

$$R_2(s,a) = -\Delta t_a \sum_{h \in \mathbb{H}} -1000\zeta(s,h) \exp\left(-\frac{||h-s||}{10}\right) \tag{5.67}$$

Hazard scenarios seven and eight contain five randomly placed line-of-sight hazards.

In figure 5-7, a histogram of the cumulative reward values is shown for each subset of simulations. From figures 5-7(a) and 5-7(d), we see that when the uncertainty is small, the MDP and AMDP planners both avoid hazards and succeed in reaching the goal position with high probability. When the uncertainty is increased by using the quality level 'b'

[8]When the random initial position and goal position, and also the random hazard positions, are chosen, they are tested to see if they lie within a building and, if so, they are discarded.

sensors, we see two changes. First, the percentage of simulations in which the MDP planner successfully reaches the goal position (indicated by a cumulative reward greater than zero) drops from 89.4% to 50.3%, a drop of 39.1%, while the AMDP planner drops only 16.8% from 90.9% to 74.1%. Second, increasing the uncertainty also increases the number of simulations in which the MDP planner runs into hazards, while the number of simulations in which the AMDP planner runs into hazards stays relatively constant. Using the quality level 'a' sensors, the percentage of simulations in which the accumulated reward was less than -2000 was 7.5% for the MDP and 5.94% for the AMDP. Using the quality level 'b' sensors, the MDP percentage increased to 16.25%, while the AMDP only increased to 6.57%.

When the uncertainty is increased further by using the quality level 'c' sensors, the results are somewhat unexpected. Although the AMDP planner still outperforms the MDP planner, we see the performance gap between the two planners narrow. The AMDP reaches the goal 25% of the time, which is only 7.8% more than the MDP which reaches the goal 17.2% of the time. This is the result of a flaw in the current AMDP path planning algorithm. This flaw comes from the fact that the AMDP path planner assumes deterministic transitions which are based on an imperfect model of the agent's sensors and dynamics. If, during implementation, the belief states don't transition as the planner expects them too based on these models, the performance of the planner can be greatly compromised. In our case, the model occasionally overestimates the effectiveness of the landmark bearing measurements. As a result, the planner ends up choosing the landmark bearing measurement from belief states which, during implementation, actually self transition when the landmark bearing measurement is executed (the discrete belief state does not change as a result of the measurement). If the agent experiences such a belief state, it will get stuck in an infinite loop and never reach the goal position. We will see an example of this in the following section on specific scenarios. Fortunately, there appears to be an easy patch to the algorithm which will fix this flaw. By simply using conservative models of the sensors (which underestimate their effectiveness), we can insure that the situation described above does not occur. Also, in spite of this flaw, the AMDP path planner still provides a significant increase in robustness with respect to avoiding hazards. Using the quality level 'c' sensors, the percentage of simulations in which the accumulated reward was less than -2000 was 22.2% for the MDP and 5.94% for the AMDP. That is a 5.95% increase for the MDP planner over the quality level 'b' sensors and a .63% decrease for the AMDP planner.

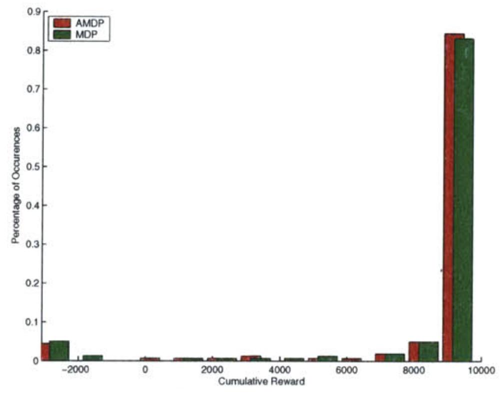
A histogram of the cumulative reward values for the complete set of simulations is shown in figure 5-8.

Figure 5-9 shows how the mean cumulative reward changes with the sensor quality for the MDP and the AMDP. Once again, we see that, as the uncertainty is increased, the performance of the MDP planner suffers while the AMDP planner remains significantly more consistent.

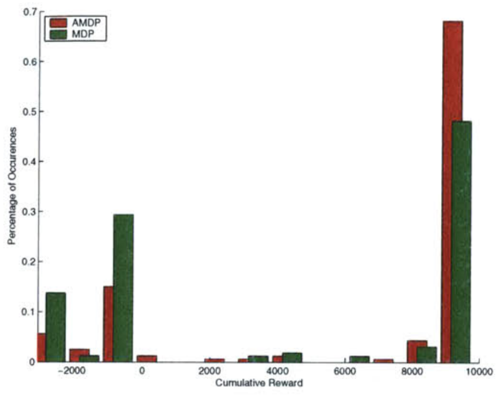
Of course, the increase in robustness of the AMDP comes at a cost. Although the asymptotic complexity of the AMDP algorithm is the same as the MDP algorithm, the AMDP state space is much larger than the MDP state space (by a factor of $|\Sigma|^2$ in this case). As a result, value iteration takes much longer for the AMDP than for the MDP. Furthermore, computing an AMDP state transition generally requires multiple matrix inversions (to propagate the EKF states), which means that each AMDP state transition takes much longer to compute than an MDP state transition. This, along with the increase in the number of states, means that the AMDP state transition function takes much longer to compute than the MDP state transition function. Even though the transition function can be computed off-line, this increase in computation time is so significant that it may become prohibitive as the size of the MDP state space is increased. Yet another increase in computation time comes from the fact that the AMDP reward function must be calculated by computing the belief corresponding to each AMDP state and applying equation (4.9). The mean computational times for the AMDP and MDP, over the set of 960 simulations, are shown in table 5.5. All computations were performed on a Dell Precision 340 desktop computer with a 2.2 GHz Intel Pentium 4 processor, using code written in c++.

5.4.2 Specific Scenarios

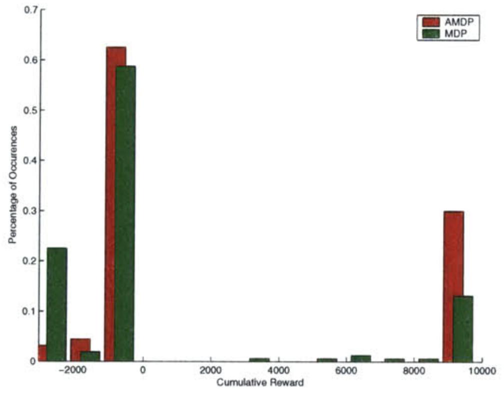
In this section we will take a look at four specific scenarios. Each of these scenarios will showcase a hypothesis that was confirmed or a lesson that was learned from the simulated



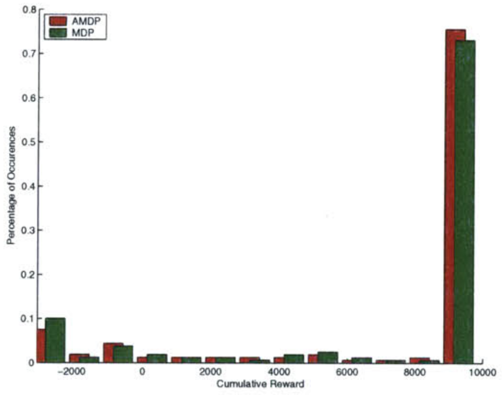
(a) Subset 1-a



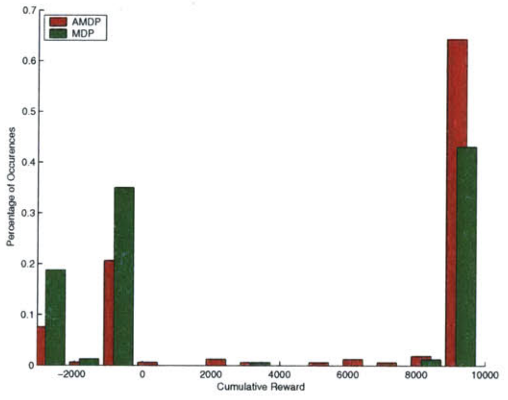
(b) Subset 1-b



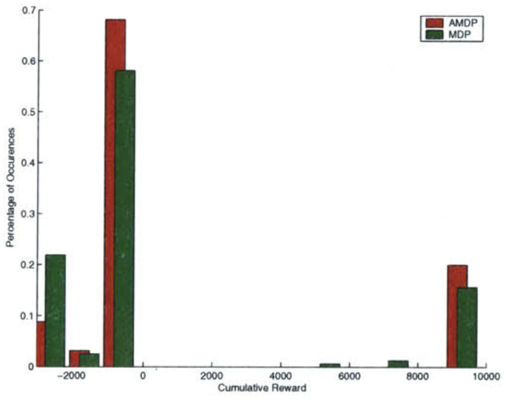
(c) Subset 1-c



(d) Subset 2-a



(e) Subset 2-b



(f) Subset 2-c

Figure 5-7: A histogram of cumulative reward values is shown for each of the simulation subsets in table 5.3. The bars in the foreground represent simulations which were run with the MDP planner and the bars in the background represent simulations which were run with the AMDP planner. The bar on the far left of each histogram represents all simulations in which the cumulative reward was less than -2000.

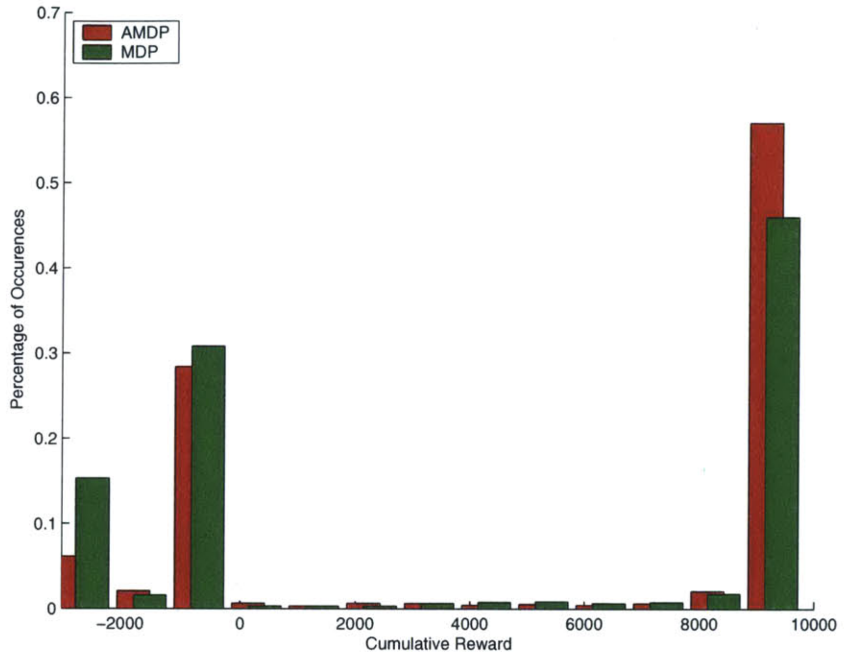


Figure 5-8: The histogram of cumulative reward values for the complete set of simulations. The bars in the foreground represent simulations which were run with the MDP planner and the bars in the background represent simulations which were run with the AMDP planner. The bar on the far left of the histogram represents all simulations in which the cumulative reward was less than -2000.

Table 5.5: The mean computational times for the AMDP and MDP. The AMDP planner contained 2.56 million states, and the MDP contained 6400 states. *The reward computation times are shown for the line-of-sight hazard simulations.

	AMDP	MDP
Transitions (off-line)	4×10^4 sec	1 sec
Rewards (off-line)*	4800 sec	76 sec
Value Iteration	175 sec	2 sec

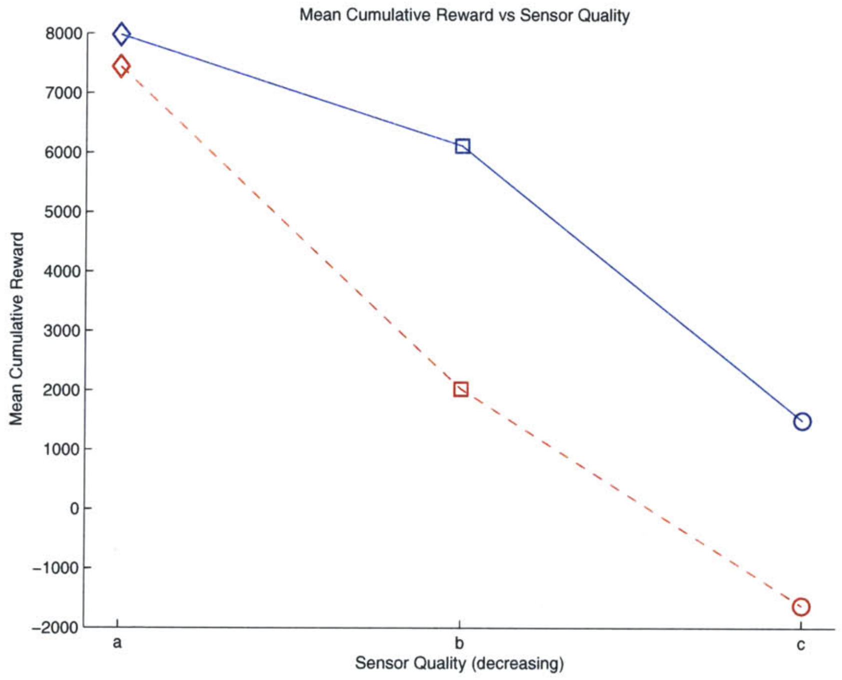


Figure 5-9: Mean cumulative reward vs. sensor quality. The AMDP planner is represented by the solid blue line and the MDP planner is represented by the dashed red line. The diamonds represent the mean for sensor quality 'a' (from table 5.4), the squares represent sensor quality 'b' and the circles represent sensor quality 'c'.

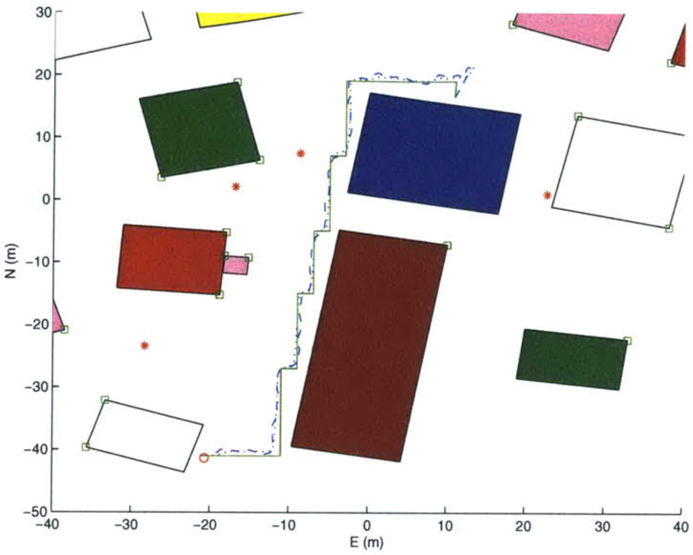


Figure 5-10: The MDP path with small uncertainty. The solid line is the planned path, the dashed line is the estimated path and the dotted line is the actual path.

experiments.

The first scenario confirms the hypothesis that the MDP planner will perform very well when the uncertainty remains small, and demonstrates that the AMDP planner behaves very similar to the MDP planner when the uncertainty is small. Figures 5-10 and 5-11 show the planned, actual and estimated paths for the MDP and AMDP in a simulation that was run using the quality level 'a' sensors. As expected, the AMDP and MDP paths are very similar, and the planned, actual and estimated paths are very similar as well. In this simulation, the cumulative reward for the MDP planner was 9900.95 and the cumulative reward for the AMDP planner was 9897.64.

The next two scenarios showcase the two advantages that the AMDP gains over the MDP as the uncertainty increases. Figures 5-12 and 5-13 show the planned, actual and estimated paths for the MDP and AMDP in a simulation that was run using the quality level 'b' sensors. Although the uncertainty in the stochastic MDP transitions was increased by using a larger characteristic velocity uncertainty, the MDP is not able to capture the fact that the uncertainty continues to grow beyond one transition. As a result, it is overly confident that it will not run into the hazards and it chooses a path that goes dangerously

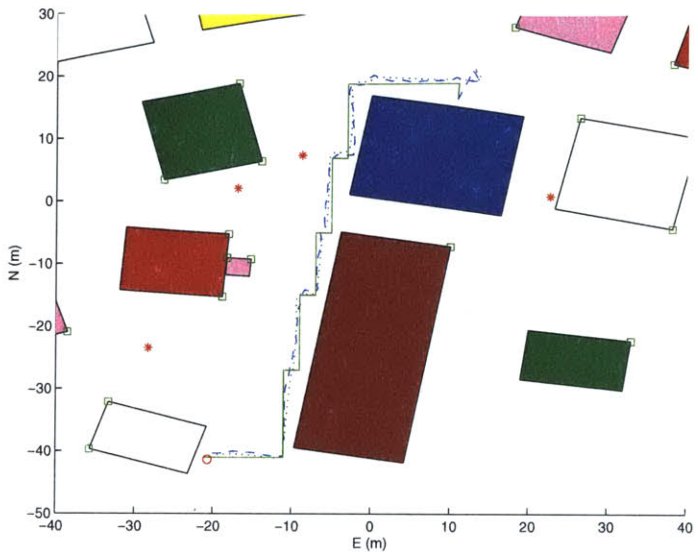


Figure 5-11: The AMDP chooses a very similar path to the MDP (figure 5-10) when the uncertainty is small. The solid line is the planned path, the dashed line is the estimated path and the dotted line is the actual path.

close to them. In this simulation, as a result of the uncertainty, the agent strays from the planned path and runs right over a hazard. The AMDP planner, however, is able to capture the fact that the agent's position uncertainty will be moderately large by the time it reaches the obstacles. As a result, it is willing to choose a slightly longer path in order to maintain a very low probability that the agent will run into the hazards. In the simulation that is shown in figures 5-12 and 5-13, the cumulative reward for the MDP planner was -21,486.6 and the cumulative reward for the AMDP planner was 9802.67.

The second advantage that the AMDP planner gains over the MDP planner under uncertainty is an increased likelihood that the agent will successfully reach the desired goal location. In figures 5-14 through 5-17 the results of a simulation are shown in which the quality level 'c' sensors were used. The poor quality sensors result in a large amount of uncertainty in the agent's position as it approaches the goal position. As a result, the AMDP planner chooses to use the information gathering action (the landmark bearing measurement), to localize before executing the 'stop' action. In figure 5-17, we can see the agent's position uncertainty converge as it gets near the goal location. Since the MDP

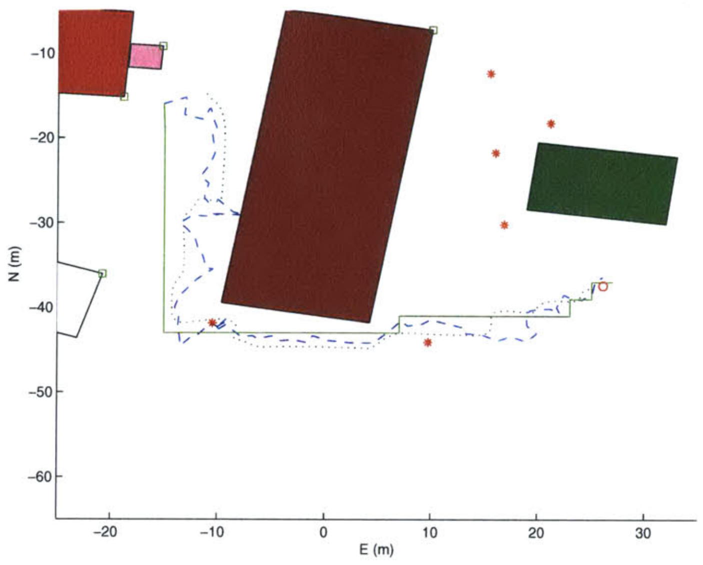


Figure 5-12: Since the MDP has no observability into the uncertainty in the agent's position, it chooses shorter paths that go dangerously close to hazards. The solid line is the planned path, the dashed line is the estimated path and the dotted line is the actual path.

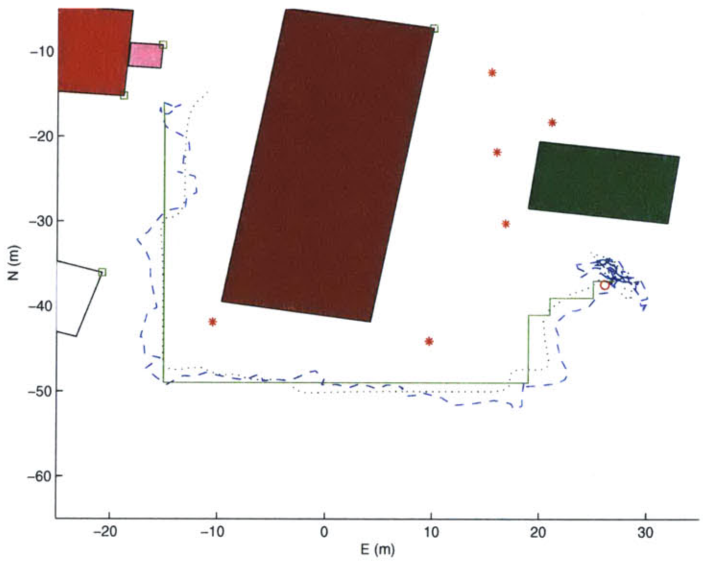


Figure 5-13: The AMDP chooses paths that maintain a safer distance from the obstacles so that, even under uncertainty it is very unlikely that the agent will run into a hazard. The solid line is the planned path, the dashed line is the estimated path and the dotted line is the actual path.

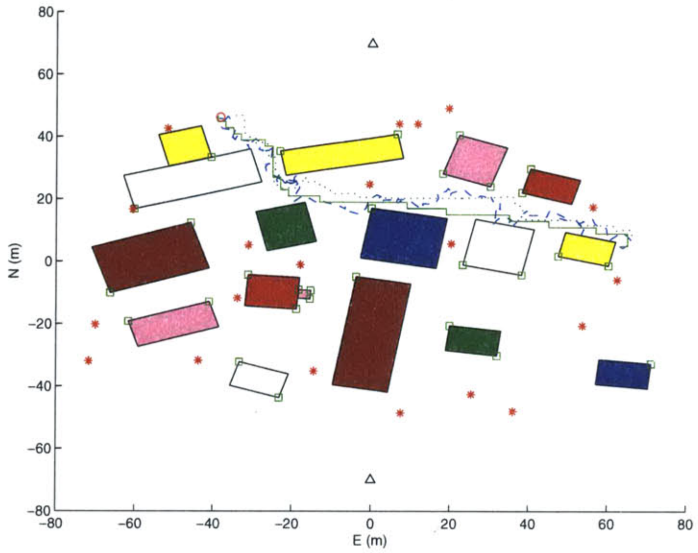


Figure 5-14: Since the MDP has no observability into the uncertainty in the agent's position, it is not able to choose to localize before executing the 'stop' action. As a result it does not successfully arrive at the goal position. The solid line is the planned path, the dashed line is the estimated path and the dotted line is the actual path.

has no observability into the agent's position uncertainty it cannot choose to take actions to localize before choosing the 'stop' action. In figure 5-17, we see that agent's position uncertainty remains large throughout the end of the simulation. As a result, the AMDP planner successfully reaches the goal position and the MDP planner does not.

The final simulation scenario showcases the flaw in the AMDP planner which caused it to perform slightly poorer than expected in simulations in which the uncertainty was large. Due to the various approximations (including discretization and the characteristic velocity uncertainty), the state transitions which where calculated by the AMDP do not always accurately reflect the true evolution of the states and their uncertainties. As a result, the AMDP planner can get stuck in a loop, repeating the same action over and over. Figure 5-18 shows a simulation in which the quality level 'c' sensors where used. Following the AMDP policy, the agent gets near the goal position and then attempts to reduce the uncertainty in its position by taking a landmark bearing measurement. The distribution that results from the measurement is not the distribution that was expected by the planner. As a result, the AMDP policy tells the agent to take another landmark

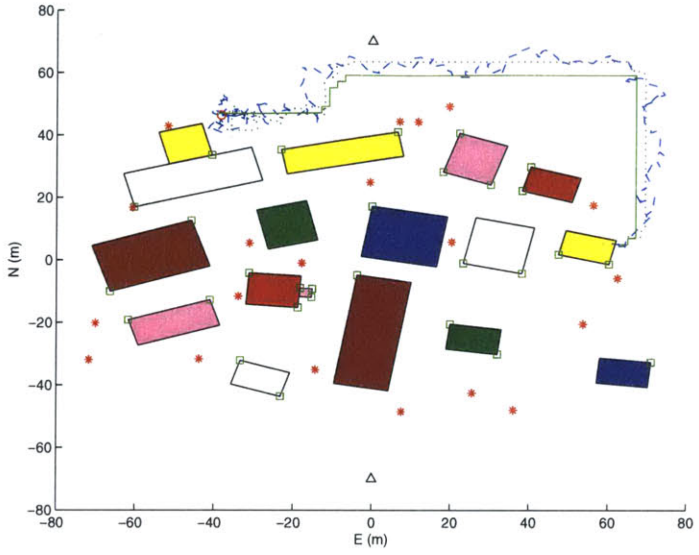


Figure 5-15: Since the AMDP does have observability into the uncertainty in the agent's position, it chooses to use the information gathering action (landmark bearing measurement) in order to localize before executing the 'stop' action. As a result it is able to successfully reach the goal position. The solid line is the planned path, the dashed line is the estimated path and the dotted line is the actual path.

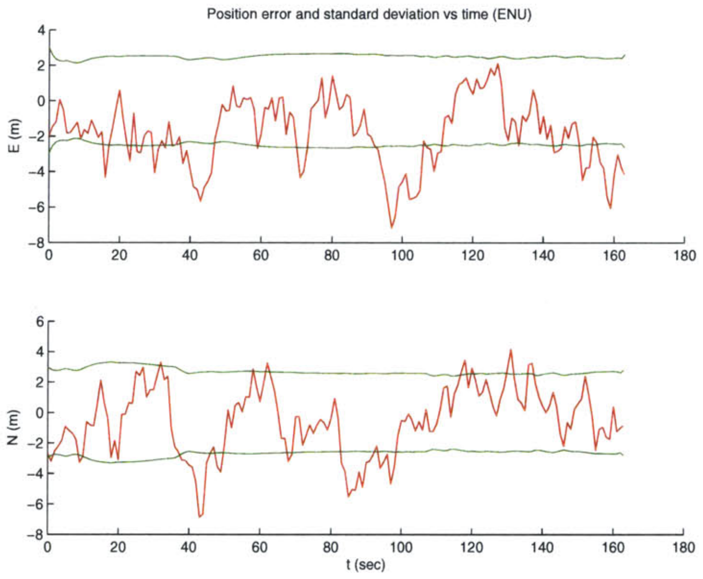


Figure 5-16: In this plot we can see that when using the MDP planner, the agent's position uncertainty remains large throughout the simulation.

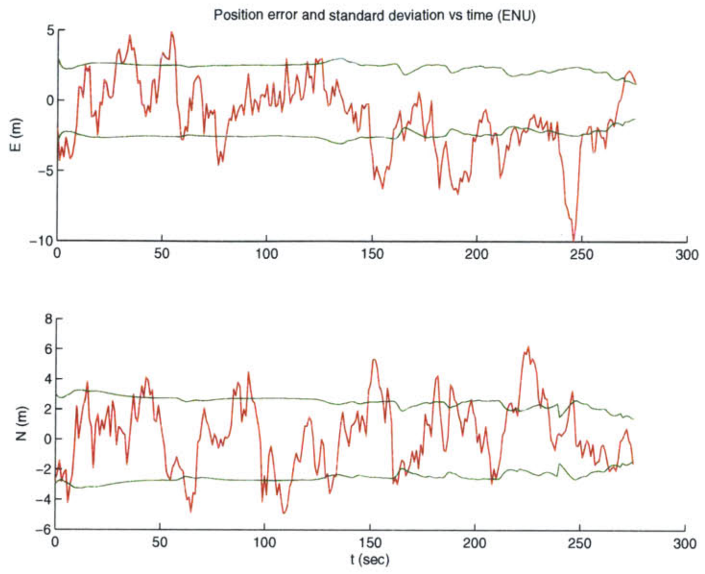


Figure 5-17: When using the AMDP planner, the position uncertainty converges before the end of the simulation as a result of the information gathering action.

bearing measurement, which results in the same distribution again. This is repeated until the ten minute limit is reached. Figure 5-19 shows a plot of the standard deviations of the position uncertainties (as calculated by the EKF) versus time. We can see the position uncertainty repeatedly converging as a result of the landmark bearing measurements and then diverging as a result of the velocity and acceleration uncertainty, beginning at approximately 500 seconds. However, the uncertainty never converges to the point that the planner expects it to.

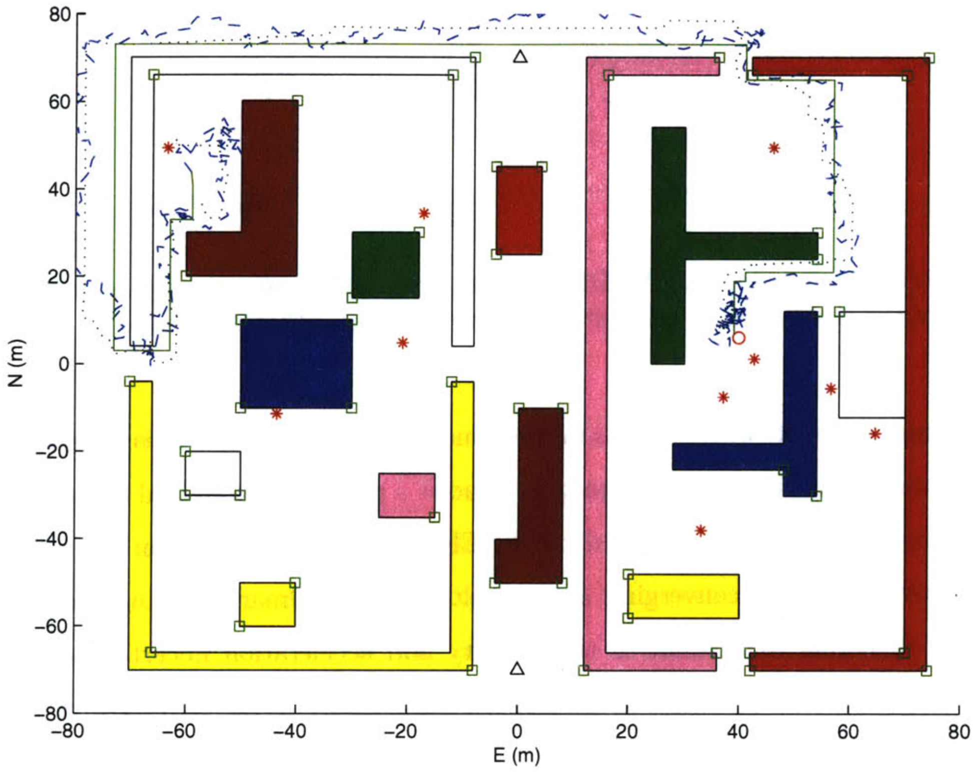


Figure 5-18: In this simulation, the AMDP planner causes the agent to get near the goal position and then repeatedly take landmark bearing measurements, which do not have their expected effect. As a result the agent never reaches the goal position.

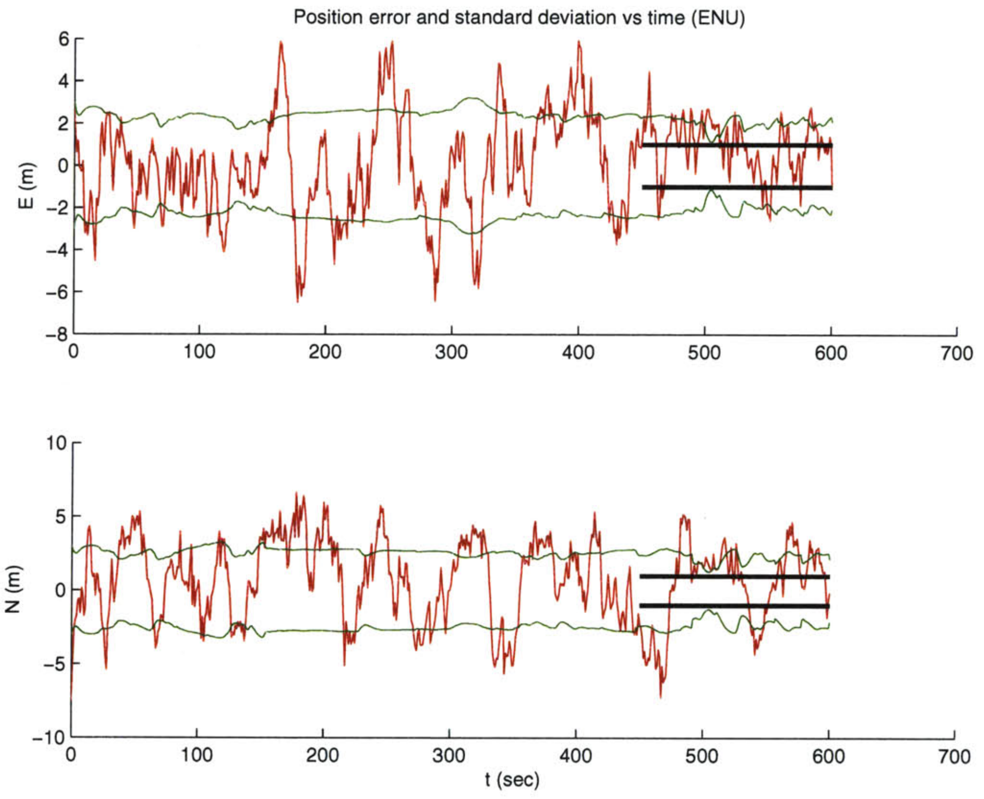


Figure 5-19: In this figure we can see the effects of the repeated landmark bearing measurements on the position uncertainty. However the uncertainty never reaches the point that the planner expects it to. The standard deviation that the planner expects to reach is shown by the straight solid line beginning at 450 seconds.

[This page intentionally left blank.]

Chapter 6

Conclusion

The intent of this research was to develop an efficient path planner which is robust to the types of uncertainty that an agent might experience in the real world. In chapter 2, we saw that the POMDP is optimal under uncertainty, but is not computationally tractable for state and action spaces which are indicative of the path planning problem. We also noted that the underlying causes of the complexity of the POMDP stem from the fact that the POMDP plans over the set of beliefs, which is an (n-1) dimensional simplex for n discrete states. This insight has allowed developers to create tractable algorithms which approximate the solution to the POMDP by planning over a subset of the belief space. If this subset is chosen to be representative of the set of beliefs that the agent will experience, the approximate solution will be nearly optimal (and therefore very robust) under uncertainty. This raises the question: What approximation of the belief space is appropriate for the path planning problem? To answer this question, we considered the problem of state estimation; the beliefs that the planner will experience are received from the state estimator. We then noted that one state estimator which is commonly employed in the field of navigation, the EKF, represents uncertainty in the agent's states using the set of Gaussian beliefs. Therefore, if we pair our planner with an EKF, a natural approximation of the belief space is the set of Gaussian distributions. In chapter 3, we looked at the EKF and the set of Gaussian distributions in detail. We found that the Gaussian distributions can be described parametrically, by a mean and a covariance matrix, and that the EKF equations provide a description of how the Gaussian distributions evolve. Then, in chapter 4, we exploited these characteristics to create a deterministic MDP whose states are Gaussian distributions and whose state transitions are described by the EKF equations. The result is an Augmented MDP whose solution approximates the solution of the planning problem formulated as a POMDP.

In chapter 5, we demonstrated, via simulation, that if an agent's states can be estimated using an EKF, then the EKF extension to the AMDP provides a computationally tractable planner which is significantly more robust to uncertainty than the basic MDP planner.

We found that one disadvantage of the AMDP is that, although the problem can be solved relatively efficiently using the value iteration algorithm, computation of the state transition function is very costly and must be done off-line. This means that our representation of the map cannot change during implementation, which prohibits us from solving problems in which parts of the map are dynamic, or parts of the map are estimated on-line. Another disadvantage of the AMDP is its heavy reliance on the fidelity of the sensor models and its sensitivity to the resolution of the discrete state approximations.

6.1 Future Work

Future work on the EKF extension to the AMDP will focus in three areas. One of these areas is further verification of the algorithm. Although the simulations in chapter 5, showed that the algorithm provided a significant increase in robustness for a model whose complexity is representative of a real world application, many real world implementation issues cannot be captured by a simulation. In order to verify that these issues are not prohibitive to implementation of the algorithm, future work will involve testing the algorithm on a real system. In addition, before implementing the algorithm on a real system, a study will be done to verify that the infinite loop problem in the simulations, caused by sensor modeling errors, can be eliminated by insuring that the sensor models are conservative, and to gain insight into how conservative the models need to be.

Another focal area of future work will be improvements to the algorithm. These improvements will include a variable resolution version of the algorithm which will make the solution faster, and make the algorithm applicable to larger maps (larger state spaces). Another improvement to the algorithm will be the ability to plan in dynamic and uncertain environments, which will allow the agent to do both planning and mapping on-line. Preliminary studies indicate that both of these improvements can be accomplished by learning (or estimating) the AMDP value function using samples from the model, similar to Q-learning [34].

The third area of future research will be application of the algorithm to more difficult problems. These will include problems with multiple cooperative and/or adversarial agents, and problems with additional tasks such as target tracking or games (like soccer for example).

[This page intentionally left blank.]

Bibliography

[1] Unmanned aerial vehicles roadmap: 2002-2027. Technical report, Office of the Secretary of Defence, December 2002.

[2] H. Baruh. *Analytical Dynamics.* McGraw-Hill, Boston, MA, 1999.

[3] P.R. Bélanger. *Control Engineering: A Modern Approach.* Saunders College Publishing, Fort Worth, TX, 1995.

[4] R. Bellman. *Dynamic Programming.* Princeton University Press, NJ, 1957.

[5] R.G Brown and P.Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering.* John Wiley & Sons, Inc., New York, NY, 1997.

[6] A. Cassandra, M.L. Littman, and N.L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, 1997.

[7] A.R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Brown University, Providence, RI, 1998.

[8] P.R. Chandler, M. Pachter, and M. Mears. System identification for adaptive and reconfigurable control. *Journal of Guidance, Control, and Dynamics*, 18:516–524, 1995.

[9] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, 19(2):96–125, 2000.

[10] F. D'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10:98–108, 1963.

[11] P. DeRusso, R.J. Roy, C.M. Close, and A.A. Desrochers. *State Variables for Engineers.* John Wiley & Sons, Inc., New York, NY, 1998.

[12] A. Gelb. *Applied Optimal Estimation.* The MIT Press, 1984.

[13] M.W. Hofbaur and B.C. Williams. Hybrid estimation of complex systems. *IEEE Transactions on Systems, Man and Cybernetics - Part B:Cybernetics*, 34(5):2178–2191, October 2004.

[14] R. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, MA, 1960.

[15] A.H. Jazwinski. *Stochastic Processes and Filtering Theory*, volume 64 of *Mathematics in Science and Engineering.* Academic Press, Inc, New York, New York, 1970.

[16] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[17] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[18] L.E. 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*, 12(4):566–580, 1996.

[19] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[20] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[21] M.L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Brown University, Providence, RI, 1996.

[22] M. Montemerlo, J. Pineau, N. Roy, S. Thrun, and V. Verma. Experiences with a mobile robotic guide for the elderly. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[23] Michael Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* PhD thesis, Carnegie Mellon University, 2003.

[24] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

[25] M.L. Puterman. *Markov Decision Processes-Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

[26] N. Roy. *Finding Approximate POMDP Solutions Through Belief Compression.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2003.

[27] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[28] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. Technical Report TR 4760 & 7239, SRI, 1985.

[29] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In Ingemar J. Cox and Gordon T. Wilfong, editors, *Autonomous Robotic Vehicles.* Springer-Verlag, Orlando, FL, 1990.

[30] E. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes.* PhD thesis, Stanford University, Stanford, CA, 1971.

[31] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann, 2002.

[32] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A.Y. Ng. Simultaneous mapping and localization with sparse extended information filters: Theory and initial results. *The International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[33] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 2004. To appear.

[34] Christopher J.C.H. Watkins. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

Appendix A

Additional Calculations and Equations

A.1 Error Covariance Differential Equation

In chapter 3, the differential equation for the linear filter error was defined as:

$$\frac{\mathrm{d}}{\mathrm{d}t}e(t) = Ae(t) - w \tag{A.1}$$

Where w is white noise with power spectral density W. We also defined the estimate error covariance matrix to be:

$$P(t) = E\Big[\Big(e(t) - E[e(t)]\Big)\Big(e(t) - E[e(t)]\Big)^T\Big] = E\Big[e(t)e^T(t)\Big] \tag{A.2}$$

In order to continue, it will be helpful to define an auxiliary variable:

$$\frac{\mathrm{d}}{\mathrm{d}t}s = w \tag{A.3}$$

The following properties will be useful:

$$\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau \quad = \quad s(t+\Delta t) - s(t) \tag{A.4}$$

$$E\Big[s(t)\Big(\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big)^T\Big] = E\Big[\Big(\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big)s(t)^T\Big] = 0 \tag{A.5}$$

$$E\Big[e(t)\Big(\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big)^T\Big] = E\Big[\Big(\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big)e(t)^T\Big] = 0 \tag{A.6}$$

$$E\Big[s(t+\Delta t)s^T(t)\Big] = E\Big[s(t)s^T(t+\Delta t)\Big] = E\Big[s(t)s^T(t)\Big] \tag{A.7}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}E[s(t)s^T(t)] = W \tag{A.8}$$

The last four equations above are properties of white noise.

Now, in order to propogate the error covariance through time, we take a look at how the error changes over an infinitesimally small time increment:

$$e(t+\Delta t) \approx e(t) + \frac{\mathrm{d}}{\mathrm{d}t}e(t)\Delta t = e(t) + Ae(t)\Delta t + \int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau \tag{A.9}$$

Substituting (A.9) into (A.2) we get:

$$\begin{aligned}
P(t+\Delta t) &= E\Big[e(t+\Delta t)e(t+\Delta t)^T\Big] \\
&\approx E\Big[\Big(e(t) + Ae(t)\Delta t + \Big\{\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big\}\Big)\Big(\text{same}\Big)^T\Big] \\
&= E\Big[e(t)e^T(t)\Big] + E\Big[e(t)e^T(t)A^T\Delta t\Big] + E\Big[Ae(t)e^T(t)\Delta t\Big] \\
&\quad + E\Big[Ae(t)e^T(t)A^T\Delta t^2\Big] + E\Big[\Big\{\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big\}\Big\{\int_t^{t+\Delta t} w(\tau)\mathrm{d}\tau\Big\}^T\Big] \\
&= P(t) + AP(t)\Delta t + P(t)A^T\Delta t + AP(t)A^T\Delta t^2 \\
&\quad + E\Big[\Big(s(t+\Delta t) - s(t)\Big)\Big(s(t+\Delta t) - s(t)\Big)^T\Big] \\
\frac{P(t+\Delta t) - P(t)}{\Delta t} &= AP(t) + P(t)A^T + AP(T)A^T\Delta t \\
&\quad + \frac{E\Big[\Big(s(t+\Delta t)s^T(t+\Delta t)\Big)\Big] - E\Big[\Big(s(t)s^T(t)\Big)\Big]}{\Delta t}
\end{aligned} \tag{A.10}$$

Taking the limit as $\Delta t \to 0$ the approximation becomes an equality and we get:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}P(t) &= AP(t) + P(t)A^T + \frac{\mathrm{d}}{\mathrm{d}t}E\Big[s(t)s^T(t)\Big] \\
&= AP(t) + P(t)A^T + W, \quad P(0) = P_0
\end{aligned} \tag{A.11}$$

A.2 Inertial Reference Frame Equations of Motion

To obtain the equations of motion in the inertial reference frame, we begin with the position, whose time derivative is just the velocity in the inertial frame:

$$\dot{x}_E = v_E \tag{A.12}$$

To obtain the time derivative of the velocity, we apply Coriolis' theorem twice:

$$\begin{aligned}
\dot{v}_E &= R_{EB}\Big[\frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\mathrm{d}}{\mathrm{d}t}x_B\Big)\Big] \\
&= R_{EB}\Big[\frac{\mathrm{d}}{\mathrm{d}t}(v_B + \omega_B \times x_B)\Big] \\
&= R_{EB}\Big[a_B + \omega_B \times v_B + \dot{\omega}_B \times x_B + \omega_B \times v_B + \omega_B \times (\omega_B \times x_B)\Big] \\
&= R_{EB}\Big[a_B + 2\omega_B \times v_B + \dot{\omega}_B \times x_B + \omega_B \times (\omega_B \times x_B)\Big]
\end{aligned} \tag{A.13}$$

We obtain the time derivative of the angular velocity in the same way:

$$\begin{aligned}
\dot{\omega}_B &= \frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\mathrm{d}}{\mathrm{d}t}\theta_B\Big) \\
&= \frac{\mathrm{d}}{\mathrm{d}t}(\theta_B + \omega_B \times \theta_B) \\
&= \alpha_B + \omega_B \times \omega_B + \dot{\omega}_B \times \theta_B + \omega_B \times \omega_B + \omega_B \times (\omega_B \times \theta_B) \\
&= \alpha_B + \dot{\omega}_B \times \theta_B + \omega_B \times (\omega_B \times \theta_B)
\end{aligned} \tag{A.14}$$

Now we define the body reference frame to be fixed to the center of mass of the agent's body. This has two implications. First, x_B, which is the position of the agent's center of mass in the body fixed coordinate system, and θ_B which is the rotation of the agent's body with respect to the body fixed coordinate system, are always zero. Setting $x_B = 0$ and $\theta_B = 0$ in equations (A.13) and (A.14) we get:

$$\dot{v}_E = R_{EB}a_B + 2R_{EB}\omega_B \times v_B \tag{A.15}$$

$$\dot{\omega}_B = \alpha_B \tag{A.16}$$

Second, R_{EB} is changing over time as the agent's body rotates, so we will have to derive the equation of motion for R_{EB}. To do this, we first define the function skew(r), which returns the skew-symmetric matrix for a vector, r:

$$\text{skew}(r) = \begin{pmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{pmatrix} \tag{A.17}$$

so that for arbitrary vectors, a and b, we have:

$$a \times b = \text{skew}(a)b \tag{A.18}$$

Next we compute the derivative of a vector, r, using the chain rule for derivatives:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} r_E &= \frac{\mathrm{d}}{\mathrm{d}t}[R_{EB} r_B] \\ &= R_{EB}\dot{r}_B + \dot{R}_{EB} r_B \end{aligned} \tag{A.19}$$

and using Coriolis' theorem:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} r_E &= R_{EB}\dot{r}_B + \text{skew}(\omega_E) r_E \\ &= R_{EB}\dot{r}_B + \text{skew}(\omega_E) R_{EB} r_B \end{aligned} \tag{A.20}$$

and then set the results equal:

$$\begin{aligned} R_{EB}\dot{r}_B + \dot{R}_{EB} r_B &= R_{EB}\dot{r}_B + \text{skew}(\omega_E) R_{EB} r_B \\ \dot{R}_{EB} &= \text{skew}(\omega_E) R_{EB} \end{aligned} \tag{A.21}$$

Since α_B only appears in equation (A.16), we will omit both α_B and equation (A.16) from our model and take ω_B to be an input. Then the equations of motion in the inertial reference frame are given by (A.12), (A.15) and (A.21).

A.3 EKF Perturbation State Equations of Motion

To obtain the EKF perturbation state equations of motion, we differentiate equations (5.17) through (5.21) and then substitute equations (5.3) through (5.5) and (5.13) through (5.14) into the result:

$$\begin{aligned}
\delta\dot{x} &= \dot{\hat{x}}_E - \dot{x}_E \\
\delta\dot{x} &= \hat{v}_E - v_E \\
\delta\dot{x} &= \delta v
\end{aligned} \tag{A.22}$$

$$\begin{aligned}
\delta\dot{v} &= \dot{\hat{v}}_E - \dot{v}_E \\
\delta\dot{v} &= [\hat{R}_{EB}\hat{a}_B + \text{skew}(\hat{R}_{EB}\hat{\omega}_B)\hat{v}_E] - [R_{EB}a_B + \text{skew}(R_{EB}\omega_B)v_E] \\
\delta\dot{v} &= \hat{R}_{EB}\hat{a}_B + \text{skew}(\hat{R}_{EB}\hat{\omega}_B)\hat{v}_E - [I + \text{skew}(\delta\theta)]\hat{R}_{EB}(\hat{a}_B - \delta a) \\
&\quad -\text{skew}\Big([I + \text{skew}(\delta\theta)]\hat{R}_{EB}(\hat{\omega}_B - \delta\omega)\Big)(\hat{v}_E - \delta v) \\
\delta\dot{v} &= \text{skew}(\hat{R}_{EB}\hat{\omega}_B)\delta v + \hat{R}_{EB}\delta a \\
&\quad +[\text{skew}(\hat{R}_{EB}\hat{a}_B) - \text{skew}(\hat{v}_E)\text{skew}(\hat{R}_{EB}\hat{\omega}_B)]\delta\theta \\
&\quad -\text{skew}(\hat{v}_E)\hat{R}_{EB}\delta\omega
\end{aligned} \tag{A.23}$$

$$\begin{aligned}
\delta\dot{a} &= \dot{\hat{a}}_B - \dot{a}_B \\
\delta\dot{a} &= w_a
\end{aligned} \tag{A.24}$$

$$\begin{aligned}
\delta\dot{\theta} &= \dot{\hat{\theta}}_E - \dot{\theta}_E \\
\delta\dot{\theta} &= [\hat{R}_{EB}\hat{\omega}_B + \dot{\hat{R}}_{EB}\hat{\theta}_B] - [R_{EB}\omega_B + \dot{R}_{EB}\theta_B] \\
\delta\dot{\theta} &= \hat{R}_{EB}\hat{\omega}_B + \text{skew}(\hat{R}_{EB}\hat{\omega}_B)\hat{\theta}_E - [I + \text{skew}(\delta\theta)]\hat{R}_{EB}(\hat{\omega}_B - \delta\omega) \\
&\quad -\text{skew}(\hat{R}_{EB}\omega_B)(\hat{\theta}_E - \delta\theta) \\
\delta\dot{\theta} &= \hat{R}_{EB}\delta\omega + 2\text{skew}(\hat{R}_{EB}\hat{\omega}_B)\delta\theta
\end{aligned} \tag{A.25}$$

note: θ_E is actually the error in the transformation matrix R_{EB}, so, since there is no error in the 'true' rotation matrix, $\theta_E = 0$ (however, $\hat{\theta}_E \neq 0$).

$$\begin{aligned} \delta\dot{\omega} &= \dot{\hat{\omega}}_B - \dot{\omega}_B \\ \delta\dot{\omega} &= w_\omega \end{aligned} \tag{A.26}$$

6075-54