FINDING APPROXIMATE POMDP SOLUTIONS THROUGH BELIEF COMPRESSION

Nicholas Roy

CMU-RI-TR-03-25

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

August 2003

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Thesis Committee:
Tom Mitchell, Co-Chair
Sebastian Thrun, Co-Chair
Geoffrey Gordon
Leslie Kaelbling (Massachusetts Institute of Technology)
Reid Simmons

ABSTRACT

RECENT research in the field of robotics has demonstrated the utility of probabilistic models for perception and state tracking on deployed robot systems. For example, Kalman filters and Markov localisation have been used successfully in many robot applications (Leonard & Durrant-Whyte, 1991; Thrun et al., 2000). There has also been considerable research into control and decision making algorithms that are robust in the face of specific kinds of uncertainty (Bagnell & Schneider, 2001). Few control algorithms, however, make use of full probabilistic representations throughout planning. As a consequence, robot control can become increasingly brittle as the system's perceptual uncertainty, and state uncertainty, increase.

This thesis addresses the problem of decision making under uncertainty. In particular, we use a planning model called the partially observable Markov decision process, or POMDP (Sondik, 1971). The POMDP model computes a policy that maximises the expected future reward based on the complete probabilistic state estimate, or belief. Unfortunately, finding an optimal policy exactly is computationally demanding and thus infeasible for most problems that represent real world scenarios.

This thesis describes a scalable approach to POMDP planning which uses low-dimensional representations of the belief space. We demonstrate how to make use of a variant of Principal Components Analysis (PCA) called Exponential family PCA (Collins et al., 2002) in order to compress certain kinds of large real-world POMDPs, and find policies for these problems. By finding low-dimensional representations of POMDPS, we are able to find good policies for problems that are orders of magnitude larger than those solvable by conventional approaches.

ACKNOWLEDGEMENTS

My thanks are first due to my advisors, Sebastian Thrun and Tom Mitchell. This thesis would not have been possible without their guidance, in all matters technical and otherwise. I am fortunate to have had such scholars as colleagues and friends. I am owe many thanks to Geoff Gordon, who not only provided insight and algorithms but also corrected my math, at which I am clearly no good at all. I am grateful to Reid Simmons and Leslie Kaelbling for their insight, suggestions and questions throughout my Ph. D. career.

I have had the tremendous good fortune to have Joelle Pineau and Mike "Magic" Montemerlo as my partners in crime. I could not have asked for two better colleagues throughout the craziness of POMDPs, the Nursebot and Carmen. My only regret is that we did not take the batteries out of Pearl *before* driving to Michigan.

My thanks to Drew Bagnell, Aaron Courville and Rosemary Emery-Montemerlo for years of fruitful discussions, advice, support and friendship.

My thanks to Dieter Fox and Wolfram Burgard for their advice, technical expertise and friendship over the years, but above all, for being Weicheier. It has also been my pleasure to work with Dirk "Foliengriller" Hähnel, Cyrill Stachniss and Maren Bennewitz, who have provided Carmen patches and much more.

My thanks to Sara Kiesler, Martha Pollack, Jacqueline Dunbar-Jacob, Judy Matthews, and Don Chiarulli, for assistance and support on the Nursebot project.

My thanks to the authors of the Sphinx & Festival speech-recognition and speech-synthesis packages for making these packages publicly available, and especially Kevin Lenzo and Alan Black for their technical assistance.

My thanks to Greg Armstrong for the bottlecap inside Flo.

My thanks to Suzanne Lyons-Muth, Diane Stidle and Sharon Woodside for fabulous administrative support. And chocolate.

My appreciation of a larger order of magnitude are due to my family. My parents and brothers have been enormously supportive of me for my entire life. This thesis is as much theirs as it is mine.

Finally, and most of all, my thanks to my wife Kristine, for so much. In my Master's thesis, I reported that a biologist's eye was essential for computer science theses. Six years later, I am happy to report it is still true.

This thesis was funded by the National Science Foundation under ITR grant # IIS-0121426 and by a Bourse de Doctorat provided by Fonds Québécois de la Recherche sur la Nature et les Technologies (FCAR). Their support is gratefully acknowledged.

DEDICATION

For Kristine, who is the same kind of crazy as me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION

- $\mathcal{S}$: the state space
- s_i : the ith state in $\mathcal{S}$
- s^t : the state at time t
- $\mathcal{A}$: the action space
- a_i : the ith action in $\mathcal{A}$
- $\mathcal{Z}$: the observation space
- z_i : the ith observation in $\mathcal{Z}$
- $p(s^t = s_i)$: the probability that the state at time t is state s_i.
- B : the space of beliefs
- $\tilde{B}$: a low-dimensional representation of B
- $b \in B$: a full belief, a high dimensional representation of $p(s)$
- b^t : the belief at time t
- $\tilde{b} \in \tilde{B}$: a low-dimensional representation of b
- $\tilde{R}$: a reward function defined over a low-dimensional representation of the model
- $\tilde{T}$: a transition function for the low-dimensional representation of the model
- U : a basis matrix
- $V(s)$: the MDP value function
- $V(b)$: the POMDP value function
- $\tilde{V}(\tilde{b})$: the value function over the low-dimensional belief space
- $H(b)$: the entropy of b
- $\overline{H}(b)$: the normalised entropy of b
- $X = \{b_1, \dots, b_{|X|}\}$: a set of sample beliefs
  also, the matrix of column vectors $\left[b_1 | \dots | b_{|X|}\right]$
- $\tilde{X} = \{\tilde{b}_1, \dots, \tilde{b}_{|X|}\}$: the low-dimensional representation of X
  also, the matrix of column vectors $\left[\tilde{b}_1 \mid \dots \mid \tilde{b}_{|X|}\right]$
- $\tilde{B}^* = \{\tilde{b}^*_1, \dots, \tilde{b}^*_i\}$: a discretisation of the low-dimensional space $\tilde{B}$ into grid cells

CHAPTER 1

Introduction

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning", the king said, gravely, "and go on until you come to the end: then stop."

– Lewis Carroll
"Alice in Wonderland"

DECISION making is one of the central problems of artificial intelligence and robotics. Most robots are deployed into the world to accomplish specific tasks, but the real world is a difficult place in which to act, as actions can have serious consequences.

Figure 1.1. A mobile robot navigating in a public space.

The real world is also characterised by uncertainty; sensors such as cameras, range finders, etc. are noisy, and the entire world is not always observable. Probabilistic state estimation explicitly recognises the impossibility of always correctly identifying the true state of the world, such as the location of a robot, or a person in the world. Instead, probabilistic state estimators provide a distribution over possible states of the world.

Planners in the real world (such as motion planners, dialogue systems, etc.) rarely model the same notions of uncertainty. Such planners typically assume a known state of the world. If the provided state estimate is a probability distribution, then a heuristic is used to extract a single "best" state estimation, such as the distribution mean or mode. Some planners attempt to compensate for estimation errors through robust control, but few deployed systems incorporate a full probability distribution as a state estimate in planning.

The disadvantage to using a single state estimate for planning is that when the distribution over possible states is very uncertain, the single state estimate is likely to be wrong, and the planner is equally likely to choose a "poor" (e.g., unreasonable) action. Furthermore, when the system must take explicit action to disambiguate the possible situations, most planners are incapable of identifying these actions. In comparison, a planner that incorporates full probability distributions can choose "reasonable" actions when the true state (and "best" action) are very uncertain. Such planners not only know how to disambiguate uncertainty, but also when to do so.

This thesis is concerned with the problem of how a physical system, such as a mobile robot, can make decisions that allow it to act as reliably and as effectively as possible. We will address the particular problem of handling uncertainty, for example where the current state of the world may not be known, and the outcome of any action is difficult to predict. We will use a probabilistic framework in order to model this uncertainty and develop algorithms for decision making that are robust to the uncertainty characteristic of real world problems.

1.1. Markov Processes

One framework for modelling uncertainty is the "Markov process". The Markov process describes how the state of the world evolves, and rests upon two key assumptions. The first assumption is that the world can be described probabilistically. That is, each possible state of the world can be identified, and there exist well-defined, stationary probability distributions that describe how the world can change as a consequence of each action.

The second assumption is known as the "Markov assumption", where the future is assumed to be independent of the past conditioned upon knowledge of the present. If a robot knows its current state, then nothing about how it arrived at its current state contains further information that can help predict what can happen next. (Note that the Markov assumption says nothing about *how* the current state is known, and the history of the robot may have a great deal to say about how the state evolved.) Once the current state is known, the past is of no predictive value.

1.2. Markov Decision Processes

If an agent (such as a robot) chooses the actions that change the state, then the decision making problem can be described as a "Markov decision process" (MDP). The advantage of MDPs is that decision theory can be used to reason quantitatively about decisions in the face of action uncertainty. This allows us not only to compute the value of actions, but also to find the "best" or optimal action for any states. A complete policy specifies the action to take for each state, and we will describe in chapter 2 how to compute the optimal policy.

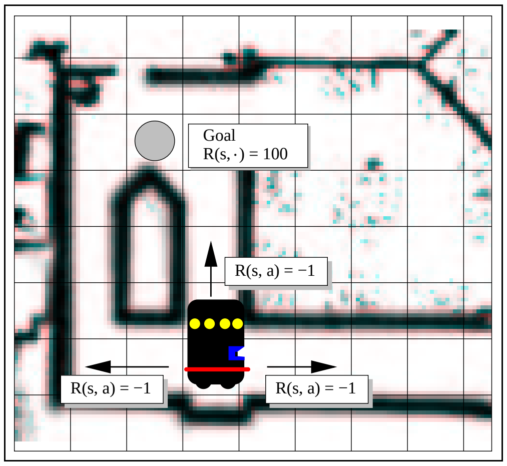


Figure 1.2. One simple way to model mobile robot navigation as a Markov decision process. Each grid cell corresponds to a state that the robot could be in. Black regions are walls, white regions are free space, and grey regions are regions of uncertainty in the map; the sensor data did not indicate with high certainty whether these regions are free space, for example due to sensor noise or occlusion. The arrows describe possible actions. Not shown is the probability of moving from state to state as a consequence of each action. The objective is for the controller to choose actions that maximise its reward.

Figure 1.2 describes one way to pose mobile robot motion planning as an MDP. The possible positions of the robot are described by a discrete state space, such as the grid shown here laid over the map. The actions are also given by a discrete action space that describe how the robot can move from state to state. In this example, each action has some small negative reward except actions that arrive at the goal state, which have a large positive reward. The optimal motion policy chooses actions that maximise the reward the robot receives according to this reward function.

The simple model shown in figure 1.2 is not entirely correct, in that it ignores the issue of state estimation. This model assumes that the robot can always know its true state; in practice, this is achieved by comparing sensor information from the environment with an *a prior* model of the environment. In this figure, the robot can see nearby environmental features easily, and therefore should be able to track its true position easily. Figure 1.3 shows the actual full environment, a map of the retirement facility Longwood, near Pittsburgh. This environment consists of many large open spaces that contain few sensor cues for position tracking. There are substantial environmental symmetries. The environment is a very public place, which means that there are often many people present, changing the perceptual structure of the environment substantially. These factors can introduce errors in the estimate of the robot's position.

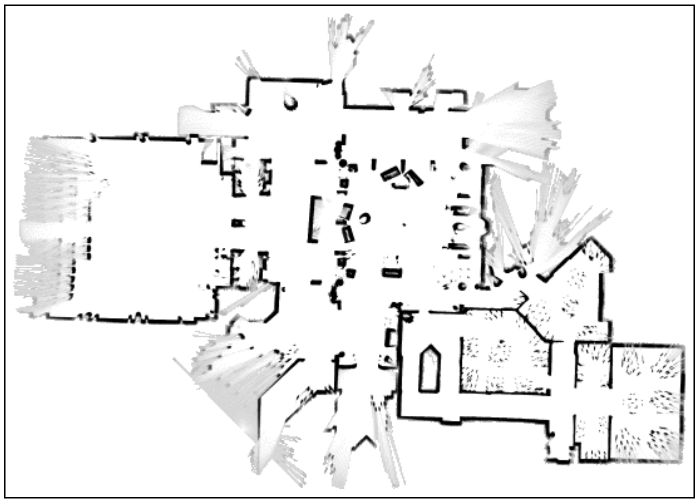


Figure 1.3. A map of the Longwood at Oakmont retirement facility in Pittsburgh. The white areas are free space, the black pixels are obstacles, and the grey areas again are regions of map uncertainty. Notice the large open spaces, and many symmetries that can lead to ambiguity in position. The map is 53.6m $\times$ 37.9m.

1.3. Partial Observability

The practical limitation of the Markov decision process is that the exact state of the world is assumed to be known at all times. A policy dictates which action to take from each state, but it is not always possible to know what state the world is in. In most real situations, the true state of the world is not directly observable. For example, the absolute position of a robot can often not be known. In such situations, the state is described as "hidden". Instead of directly observing the state of the world, the agent receives observations (e.g., sensor measurements) that permit possible states to be inferred. But, these observations rarely allow the agent to infer the current state uniquely and correctly. The concept of "partial observability" describes this scenario in which the state is not known.

A "partially observable Markov process" (Sondik, 1971) (or "hidden Markov model" (Rabiner, 1990)) extends the notion of Markov processes to encompass observations by providing a model of how the world generates observations, conditioned upon the current state (and possibly the last action). This model allows the controller to infer a probability distribution over world states; we call this distribution a "belief". Furthermore, a belief is a sufficient statistic over the agent's history; therefore we can reason equally about beliefs as about agent histories without loss of information about the current state. As we will see in chapter 2, future beliefs are independent of past beliefs conditioned upon the current belief, which corresponds to the Markov assumption as applied to beliefs instead of states. Therefore, partially observable Markov processes are equivalent to "belief state" Markov processes.

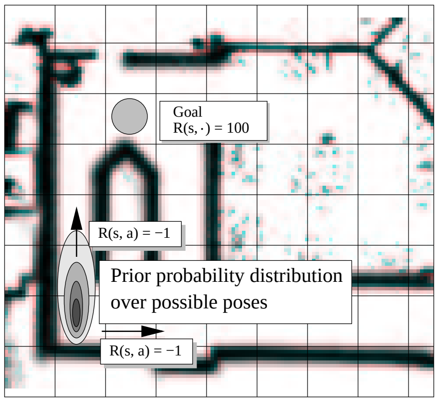
1.4. Partially Observable Markov Decision Processes

If our agent chooses the actions that change the state (just as in the MDP), then we describe the problem of decision making as a "partially observable Markov decision process", or POMDP. Figure 1.4 shows how the mobile robot motion planning problem can be described as a POMDP, extending the MDP depicted in figure 1.2. The state space is again the possible positions of the robot, and the action space again describes how the robot can move from state to state. Observations are added to the model, for example as discrete laser range measurements. The probability of such measurements can be calculated from the map of the environment and knowledge of the robot's sensing capabilities. The objective of the controller is again to choose actions that maximise the reward the controller receives.

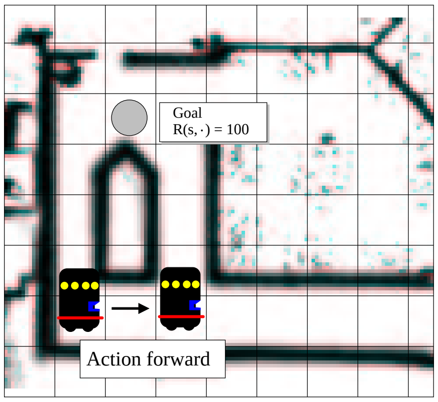
The advantage to using POMDPs for decision making is that POMDP policies handle uncertainty well. For instance, if the current belief indicates that the world is in one of a wide range of states, the POMDP policy will handle this ambiguity appropriately. The POMDP planning process can take advantage of actions that gather information to reduce uncertainty, even if the problem specification (e.g., the reward function) does not make explicit the advantages of gathering such information. The POMDP planning process also assigns value to information; an optimal POMDP policy not only uses actions to reduce uncertainty, but also only when necessary. The optimal policy contains an explicit trade-off between resolving state ambiguity against and maximising reward (Russell & Norvig, 1995), accepting high levels of uncertainty when the uncertainty will have no impact on the reward received.

1.5. Solving Large POMDPs

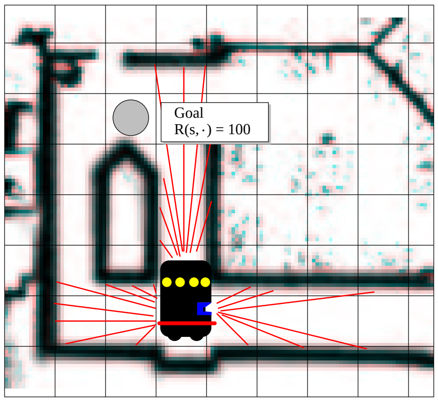
Standard POMDP policies are universal plans (Schoppers, 1987) in that they describe an action for every possible belief. Pre-computing universal plans is reasonable in domains



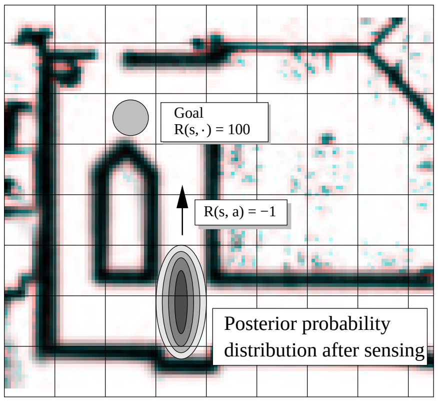
(a) Prior belief



(b) Action Taken



(c) Observation received



(d) Posterior belief

Figure 1.4. Modelling mobile robot navigation as a partially observable Markov decision process. Figure (a) shows the initial distribution over where the robot might be. Figure (b) shows the robot taking an action, and in (c) receiving laser range data as an observation. The action and observation are integrated into the prior. Figure (d) shows the posterior distribution over where the robot might be. An optimal POMDP policy chooses actions based on such distributions, rather than an estimate of the current state.

where the planning time can be amortised over the execution time of the plan. However, explicit planning over the full space of beliefs leads to an unacceptably high computational cost for finding full POMDPs policies. Techniques for finding exact optimal plans for POMDPs typically cannot handle problems with more than a few hundred states at the most (Hauskrecht, 2000; Zhang & Zhang, 2001). If the observation space is relatively complex, or the beliefs have a strong dependency on history, then existing exact algorithms do not scale even that far, handling planning problems with at most a few dozen states (Pineau et al., 2003a). Unfortunately, few planning problems involving real, physical systems can

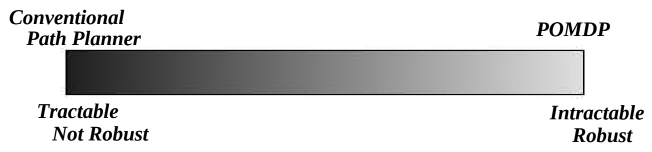


Figure 1.5. The most useful planner lies somewhere on a continuum between the MDP-style approximations and the full POMDP solution.

be expressed so compactly (Burgard et al., 1999). We would like to deploy robots that plan over thousands of possible states of the world (e.g., map grid cells), with thousands of possible observations (e.g., laser range measurements) and actions (e.g., velocities). Existing exact POMDP solution algorithms do not appear to scale to these problems.

One common solution is to (falsely) assume that the state of the world *is* knowable, using some set of heuristics or assumptions (Cassandra, 1998; Burgard et al., 1999; Singh et al., 1999). Many algorithms make decisions based on some statistic of the belief, such as the maximum likelihood state or mean of the distribution. For some problems this approach works well. If the uncertainty is bounded and small, the optimal action (with respect to the MDP policy) may be approximately the same for all possible states. When an agent makes decisions based on a notion of the world that is wrong, the best planning algorithms can fail badly. Imagine a mobile robot that does not know which of two different but similar corridors it is in. A planner that "guesses" which corridor it is in for motion planning could fail catastrophically.

Instead of the computational intractability of the full POMDP solution, or the sub-optimal performance of MDP solutions, we seek an intermediate solution that can plan to gather information while maintaining a degree of tractability. This forms a kind of continuum as depicted in figure 1.5.

1.6. The Curses of POMDPs

There are two competing views of why POMDPs are generally so computationally intractable, as well as the appropriate way to overcome this intractability. These views can be referred to as the "Curse of Dimensionality" (Kaelbling et al., 1998) and the "Curse of History" (Pineau et al., 2003a). The "Curse of Dimensionality" refers to the fact that, for a POMDP with n states, the optimal POMDP policy must be computed over an n–1-dimensional belief space, a substantially difficult task. The "Curse of History" refers to the fact that each belief is in essence a sufficient statistic over a history of actions and observations. The number of different such histories grows exponentially with the planning horizon, leading to computational intractability.[1]

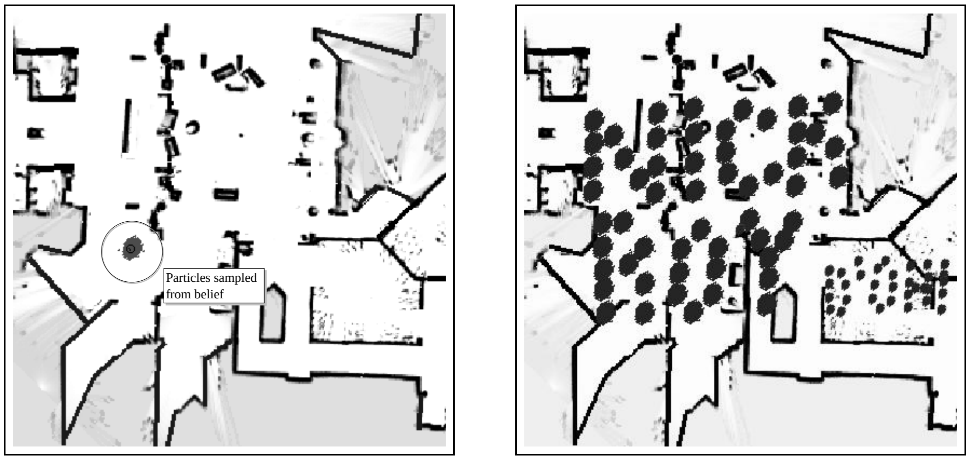
There is tension between these two confounding factors, in that algorithms that overcome one curse often fall victim to the second. For example, some algorithms attempt to overcome the curse of history by clustering similar beliefs (and therefore clustering similar histories) together, using various grid-based schemes (Lovejoy, 1991; Brafman, 1997; Hansen & Zhou, 2003). However, the complexity of these clustering methods is invariably exponential in the size of the state space, and becomes unreasonably unwieldy with large dimensional belief spaces. Similarly, algorithms have also attempted to overcome the curse of history through reachability analysis, identifying only the beliefs that can be reached and converting the high-dimensional, continuous belief space into a low-dimensional, discrete space (Hauskrecht, 2000; Zhou & Hansen, 2001). However, the number of histories to be considered in reachability analysis can be exponential in the planning horizon, becoming again unwieldy for problems that require even a modest planning horizon.

1.7. Dimensionality Reduction

The tension between these two approaches does suggest a possible solution. Irrespective of initial distribution, action and observation sequence, it is true that in many problems certain beliefs are very unlikely. In fact, for real world problems, there space of likely, or *relevant* beliefs is much smaller than the full belief space. If the POMDP solver can identify a low-dimensional space of relevant beliefs, the computational complexity of the solution process might be dramatically reduced. This is distinct from identifying the (possibly infinite) *set* of reachable beliefs. Instead, we wish to find a representation of the low-dimensional space in which they are embedded.

Figure 1.6 suggests what we mean by relevant or plausible belief. Imagine a mobile robot navigating in a real world. The state space and observation space are such that, even with low-noise sensors, the set of reachable beliefs is intractably large. Regardless, it is clear that there are certain kinds of beliefs that the robot is more likely to encounter than others. On the left, figure 1.6 (a) shows a typical belief that a mobile robot might experience while navigating in the nursing home environment of figure 1.3. The distribution is unimodal and compact. On the right is a very different kind of belief; in this domain it is extremely unlikely, if not impossible, to find a sequence of actions and observations that would result in such a checker-board belief ever occurring.

[1]Whether these two curses are related to the well-known "Curse of the Mummy's Tomb" (Carreras, 1964), is an open question.



(a) A plausible belief (b) An implausible belief

Figure 1.6. Two example probability distributions over robot pose. On the left is a plausible belief, a distribution that might be expected to occur in this domain. The probability distribution is depicted as the output of a particle filter. The probability distribution is concentrated on a single mode, around the true position of the robot. On the right is an implausible belief, one that could is very unlikely to occur in the robot navigation domain as consequence of actions and observations. Again the distribution is depicted as the output of a particle filter, but in this case, the probability mass is distributed as a regular pattern across space.

The difficulty is finding the relevant belief space before planning over it. There is little, if any, information in the model specification that can be used to determine this low-dimensional belief space. There are two possible courses of action that will be explored in this thesis. One approach is to use domain knowledge to restrict the space of possible beliefs, and plan over that space. The second possible approach is to collect a set of sample beliefs, and subsequently infer a representation that somehow describes these collected beliefs in a principled manner. This second approach assumes that low-dimensional representations can be inferred based on a small sample of beliefs, as opposed to representations based on reachability analysis that must sample all possible beliefs and therefore fall victim to the curse of history.

1.8. Thesis Statement

The central thesis of this document is that probabilistic state estimation can be integrated into decision making for real world problems tractably while still producing good policies if compact representations of the probabilistic estimates are used. We will demonstrate how to find these representations and how to plan with them in order to find better policies for real world problems than could be found previously.

1.9. Contribution

The contributions of this thesis are three algorithms that can be used to solve large partially observable Markov decision processes. These algorithms are predicated on two assumptions. Firstly, large POMDPs are difficult to solve because of the dimensionality of the belief space; by reducing the belief space, good policies can be found more easily. Secondly, POMDPs can be modelled as discrete belief space MDPs, i.e., there exists a function defined over a discrete low-dimensional representation of the belief space that is an acceptable approximation of any function defined over the continuous high-dimensional belief space.

The Augmented MDP

The first algorithm demonstrates a way to solve POMDPs by summarising probability distributions using two statistics: the mean and entropy. These statistics are jointly sufficient under certain restricted classes of distributions, for example, gaussians with a single variance parameter. For other distributions, this representation can be viewed as lossy.

One consequence of this representation is that only certain kinds of beliefs can be easily distinguished, for example, beliefs that place the probability mass in substantially different areas of the state space, or beliefs that have substantially different levels of uncertainty. The kinds of plans that this planner tends to produce are ones that choose actions that reduce the uncertainty as quickly as possible, and then keep the uncertainty low in pursuit of reward. One good analogy is a boat hugging the coastline as it navigates. For this reason, the algorithm is sometimes referred to as "Coastal Navigation" (Roy et al., 1999; Roy & Thrun, 1999). Just as sailors can more easily maintain a fix on their current position by staying close to recognisable landmarks along the coast, the robot can keep its positional uncertainty low by staying close to recognisable environmental features. However, the applicability is not limited to actual navigation, or motion planning, tasks, and we will see its utility in different tasks, such as dialogue management.

Exponential Family PCA POMDPs

The second algorithm demonstrates a way to find low-dimensional representations of a set of high-dimensional probability distributions automatically, through a technique similar to Principal Components Analysis (PCA) (Joliffe, 1986). PCA is a technique for finding some low-dimensional, linear subspace that passes through, or close to, a set of data. The particular distance metric used by conventional PCA is not well-suited for representing probability distributions; however, PCA can be easily modified for other distance metrics.

This thesis demonstrates how to modify PCA appropriately for probability distributions, a technique called exponential-family PCA (E-PCA), and how to plan using this representation.

Variable Resolution POMDPs

By finding good low-dimensional representations of the probability distributions generated by a POMDP, the POMDP model itself can be compressed to the same dimensionality. Unfortunately, E-PCA introduces a non-linearity between the high and low-dimensional representation, which means that conventional value function representations are no longer valid, and an alternate method of solving the low-dimensional POMDP must be used.

The third algorithm will demonstrate a variant of existing techniques for planning in POMDPs that cannot be solved using conventional algorithms. We will see how to find a good discrete approximation of the belief space, and plan over that representation. This approach takes advantage of function approximation using a variable resolution grid. Previous approaches have suffered from a number of limitations, such as assuming deterministic transitions (Moore & Atkeson, 1995; Munos & Moore, 1999), or assumptions on the shape of the continuous space (Hauskrecht, 2000; Zhou & Hansen, 2001). These limiting assumptions have so far prevented their application to POMDPs or their scaling to real-world sized problems.

Experimental Results

We will discuss three real world problems and how they can be cast in the framework of MDPs and POMDPs. The three problems are mobile robot navigation, dialogue management and people finding. We will use these three problems as the basis for developing new algorithms because they are representative of many real-world problems. These problems are typically high-dimensional in both the state space and the observation space. The navigation problem is an example of one where the probability mass is compact; the uncertainty is largely due to sensor noise. This will allow us to develop one kind of approximate POMDP algorithm in chapter 3. The people-finding problem is qualitatively different to both navigation and dialogue management in that the uncertainty is due not to control and sensor errors, but a fundamental lack of knowledge about a feature in the world. The dialogue problem has uncertainty due both to an initial lack of knowledge and sensor noise; additionally, the dialogue management problem is an example with a non-metric state space. We use these three problems to exemplify a wide variety of real world problems and motivate the algorithms.

1.10. Document Outline

In chapter 2, we will begin by describing Markov processes more formally, and examine how conventional value iteration is used to find policies for POMDPs. We will also describe how some real world problems have been described in terms of Markov decision processes, and how these problems extend to POMDPs. We will also examine prior art in other approximate methods in solving large POMDPs.

In chapter 3, we will develop the Coastal Navigation algorithm, and evaluate its performance in chapter 4 on some real world problems.

In chapter 5, we will develop the Exponential Family PCA algorithm, and how it pertains to solving large POMDPs.

In chapter 6, we will examine the problems encountered in solving POMDPs in the previous chapters. We will develop the variable resolution approach that allows us to solve large POMDPs more efficiently, and demonstrate the solution of even larger POMDPs.

Finally, we conclude with a summary of the work, and a discussion of some of the more interesting results. We discuss some of the unresolved problems of planning under uncertainty and remaining open questions.

CHAPTER 2

Sequential Decision Making

To judge what one must do to obtain a good or avoid an evil, it is necessary to consider not only the good and the evil in itself, but also the probability that it happens or does not happen; and to view geometrically the proportion that all these things have together.

– Antoine Arnauld
"Port-Royal Logic"

IN this chapter, we will describe how to pose the decision making problem as a Markov decision process (MDP), and how to use this framework for choosing actions for real world problems. We begin by describing the simple Markov decision process and how to use this model to find the optimal policy. Next, we describe how to extend MDPs to problems with uncertain information, or partial observability. We describe the partially observable Markov decision process (POMDP), and algorithms for finding good policies. This description will cover not only conventional approaches to solving POMDPs, but also many approximate methods. Few POMDP solution algorithms have been applied to real world problems of any size, especially as large as we would like to tackle and with the kinds of uncertainty we wish to handle. Examining these earlier approaches will lend some insight into why they have not scaled to large real world problems, and how a new algorithm might improve upon them.

2.1. Markov Decision Processes

A Markov decision process (MDP) is a model for deciding how to act in "an accessible, stochastic environment with a known transition model" (Russell & Norvig (1995), pg 500.). A Markov decision process is described by the following:

- a set of states $\mathcal{S} = \{s_1, s_2, \dots s_n\}$
- a set of actions $\mathcal{A} = \{a_1, a_2, \dots , a_m\}$
- a set of transition probabilities $T(s_i, a, s_j) = p(s_j|s_i, a)$

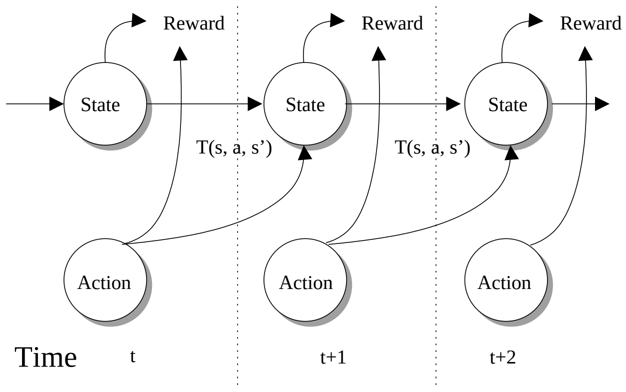


Figure 2.1. A graphical model of the Markov decision process. The Markov assumption models the conditional independence of future states from past states given knowledge of the current state.

- a set of rewards $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$
- a discount factor $\gamma \in [0, 1]$
- an initial state $s_0 \in \mathcal{S}$

Figure 2.1 depicts the MDP as a graphical model. The controller chooses the actions that cause the state transitions, but the particular posterior state is chosen according to the transition function $T(s, a, s')$. The graphical model also makes explicit the Markov assumption, that future states are conditionally independent of past states given knowledge of the current state.

A full description of the notation throughout this thesis is given in the preface, however, to eliminate confusion:

- s_i is a specific state, the i^{th} state in the state space
- s^t is the state at time t
- $p(s^t = s_i)$ is the probability that the true state at time t is state s_i.

In this document, superscripts will always refer to time, subscripts will always refer to specific instances in a set.

Solving Markov Decision Processes

The objective of the controller is to choose actions that maximise the reward it receives, as described by the reward function. A policy as a mapping from state to actions, and an optimal policy is one that receives the maximum sum of rewards. In a stochastic world, the outcome of any action, and therefore the reward, is not known, but because the transitions are described probabilistically, we can compute the *expected* sum of rewards (or utility) of any policy; the optimal policy is therefore one that maximises the expected sum of rewards.

```
1.  t = 0
2.  $\forall s_i \in \mathcal{S} : V^0(s_i) = 0$
3.  do
4.      change = 0
5.      $\forall s_i \in \mathcal{S}$:
6.          $V^t(s_i) = \max_a \left( R(s_i, a) + \gamma \sum_{j=1}^{|\mathcal{S}|} p(s_j|s_i, a) \cdot V^{t-1}(s_j) \right)$
7.          change = change + $V^t(s_i) - V^{t-1}(s_i)$
8.  while not converged
```

Table 2.1. An algorithm for MDP value iteration.

The expected reward of a policy is defined over a horizon of t steps. MDPs can be defined for infinite horizon ($t \rightarrow \infty$) problems, in which case a discount factor may also be used to ensure reasonable behaviour in the face of unlimited time. The discount factor can be viewed as a "smoothed out version of the limited-horizon algorithms" (Russell & Norvig (1995) pg. 507); in a sense, it represents the probability that the agent survives to take another step. An optimal policy is known to always exist in the discounted ($\gamma < 1$) case with bounded immediate reward (Howard, 1960).

Value Iteration for MDPs

One common way of finding the optimal policy for an MDP is by computing a value function over the state space. The converged value function gives the long-term expected reward for that state, and is computed iteratively using Bellman's equation (Bellman, 1957):

$$V^t(s_i) = \max_a \left(R(s_i, a) + \gamma \sum_{j=1}^{|\mathcal{S}|} p(s_j|s_i, a) \cdot V^{t-1}(s_j) \right). \tag{2.1}$$

Bellman's equation is defined recursively, and gives the value function for each state at horizon t, based on the existing value function at horizon $t - 1$. The value iteration algorithm using Bellman's equation is given in table 2.1.

After t iterations of Bellman's equation, the value function corresponds to the utility function of tth step policy. Given the appropriate conditions, as $t \rightarrow \infty$ the value function will converge to the optimal utility function. The stopping criterion in step 8 of table 2.1 can be a fixed number of iterations or when the change between iterations falls below a specific level ϵ (Russell & Norvig, 1995). The optimal policy is given as the action that corresponds to the largest expected reward for each state:

$$\pi(s_i) = \operatorname*{argmax}_a \left(R(s_i, a) + \gamma \sum_{j=1}^{|\mathcal{S}|} p(s_j|s_i, a) \cdot V(s_j) \right). \tag{2.2}$$

In addition to value iteration, there are other methods of solving MDPs. For example, policy iteration assumes a fixed policy and computes the value function of that policy (Howard, 1960; Russell & Norvig, 1995). A new policy is then computed to improve that value function. The advantage of policy iteration is that policies converge faster than value functions; however, each step of computing a given policy may still take a long time to converge. Value functions may also be computed using linear programming (Bertsekas, 1976; Ross, 1983).

2.2. Partially Observable Markov Decision Processes

MDPs explicitly model the uncertainty of actions, but the assumption is that the state of the world is always known exactly; this is rarely the case. Imagine a navigating mobile robot: the robot cannot intrinsically know its location exactly, but instead receives sensor information, from which it should infer its current location.[1] Or, imagine a person talking to a mobile robot; the true intentions of the person cannot be assumed to be known exactly.

When the true state of the world cannot be observed directly (or is "inaccessible"), the world is "Partially Observable." In partially observable domains, decisions cannot be made based on the state of the world; the best one can do is make decisions based on our observations of the world. The Markov decision process is extended to partially observable Markov decision processes (POMDPs) (Sondik, 1971) by adding observations and a probabilistic model of how observations are generated. The POMDP is described formally as:

- a set of states $\mathcal{S} = \{s_1, s_2, \ldots s_n\}$
- a set of actions $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$
- a set of transition probabilities $T(s_i, a, s_j) = p(s_j|s_i, a)$
- a set of observations $\mathcal{Z} = \{z_1, z_2, \ldots, z_l\}$
- a set of observation probabilities $O(s_i, a_j, z_k) = p(z \mid s, a)$
- an initial distribution over states, $p(s^0)$
- a set of rewards $R : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \mapsto \mathbb{R}$

The reward function for a POMDP is often specified as $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ (Littman, 1996; Kaelbling et al., 1998), however, the reward function for a POMDP as given above is the most general form (Cassandra, 1998).

Figure 2.2 depicts the graphical model of POMDPs. As in figure 2.1, the robot chooses

[1]Occasionally, outdoor robots are assumed to be able to always identify their location correctly and uniquely. For example, GPS is assumed to work well for large-scale, outdoor navigation problems, such as navigating boats on the open sea. However, such sensors are limited in resolution (especially for some satellite configurations (Miller et al., 1999)) and can experience errors such as multi-path bounce problems in densely structured environments.

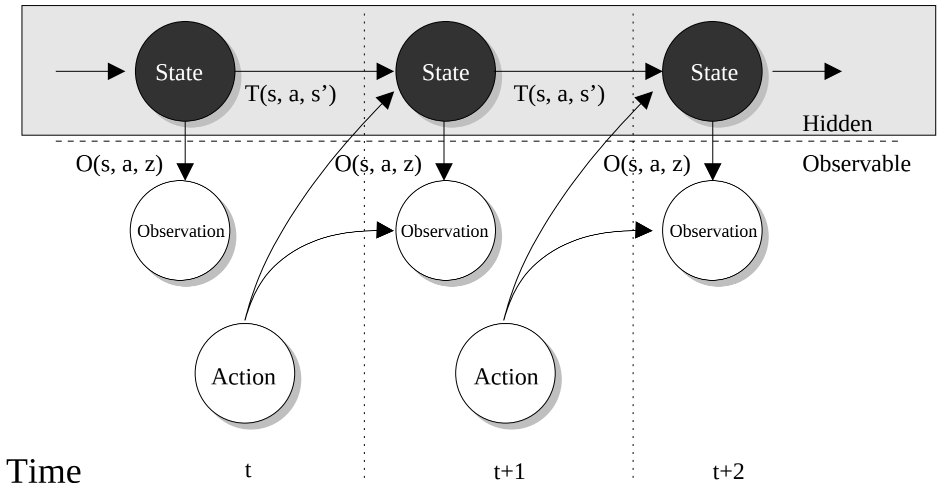


Figure 2.2. A graphical model of the POMDP. Like the MDP, the Markov assumption models the conditional independence of future states and observations from past states and observations given knowledge of the current state. Unlike the MDP, the state is hidden from the controller (these nodes are greyed out).

the actions that cause the state to evolve over time. The rewards have been dropped from this diagram for clarity, without loss of generality to the discussion in this chapter.

Notice that we still maintain the Markov assumption: future states and observations are conditionally independent from past states and observations given knowledge of the current state. But in contrast to the MDP, the current state is not accessible—only the actions and observations are known. Consequently, future states and observations are *not* independent of past observations. Predicting the observation at time t is not possible because s^t is not known. Instead, the state is estimated using

$$p(s^t | z^t, a^t, z^{t-1}, a^{t-1}, z^{t-2}, a^{t-2}, \dots, z^0, a^0, s^0), \tag{2.3}$$

where $p(s^t)$ is the distribution over states at time t, and $p(s^0)$ is the initial distribution over states. Maintaining the entire history of the controller can become computationally difficult as the lifetime of the controller increases. Fortunately, there exists a sufficient statistic for the history of actions and observations: the probability distribution over states. This distribution is called a *belief* b (also sometimes referred to as an "information state" (Littman, 1996; Cassandra, 1998)). We compute the belief at time t, b^t, using

$$b^t = p(s^t | z^t, a^t, z^{t-1}, a^{t-1}, z^{t-2}, a^{t-2}, \dots, z^0, a^0, s^0) \tag{2.4}$$

which we can compute recursively so long as we maintain the prior distribution over states, $p(s^{t-1})$. Figure 2.3 describes this graphical model that depicts a new form of the Markov assumption: future observations and beliefs are conditionally independent from the past, given the current belief. There is a strong correlation between the structure of the model in

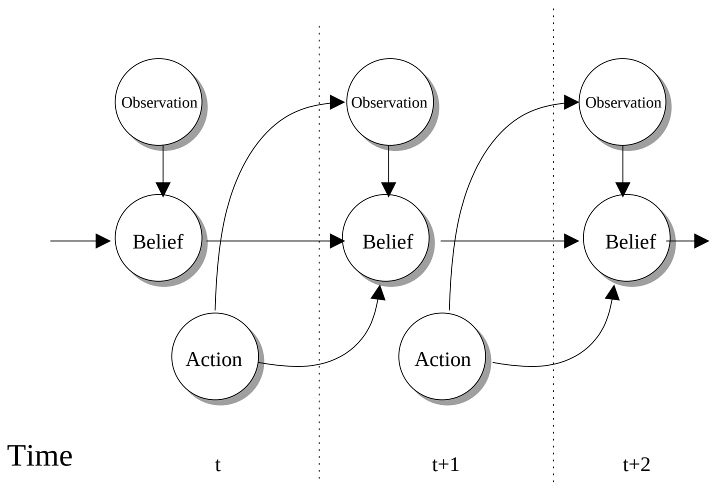


Figure 2.3. The graphical model of the POMDP using beliefs. The Markov assumption as applied to beliefs models the conditional independence of future beliefs and observations from past beliefs and observations given the current belief.

figure 2.1 and figure 2.3; in essence, a POMDP is a belief state MDP, that is, an MDP with a state space defined by the set of beliefs.

The probability distribution can be updated at each time step using a Bayes' filter, as in equation (2.8):

$$\begin{aligned} b^t(s_i) &= p(s_i|z^t, a^t, s^{t-1}) && (2.5)\\ &= \alpha\, p(z^t|s_i, a^t)\, p(s_i|a^t, s^{t-1}) && (2.6)\\ &= \alpha\, p(z^t|s_i, a^t) \sum_{j=1}^{|\mathcal{S}|} p(s_i|s_j, a^t) p(s^{t-1} = s_j) && (2.7)\\ &= \alpha\, O(s_i, a^t, z^t) \sum_{j=1}^{|\mathcal{S}|} T(s_j, a^t, s_i) p(s^{t-1} = s_j) && (2.8) \end{aligned}$$

where equation (2.5) holds from the Markov property, equation (2.6) holds from Bayes' rule (α is a normalising constant) and equation (2.7) from the definition of transitions.

2.3. Solving POMDPs

The notion of belief is at the crux of how a POMDP operates. Whereas an MDP policy dictates an action for every state, a POMDP policy dictates an action for every possible belief. If we think of a POMDP as a belief MDP then it is tempting to try and apply equation (2.1) directly to the belief space instead. But, the belief space is continuous and has dimensionality $\mathbb{R}^{|\mathcal{S}|-1}$ which would mean applying this equation to an infinitely large state space. We will instead use a recursive procedure similar to equation (2.1) for computing a value function over the belief space.

Value Iteration for POMDPs

There are a number of algorithms for solving POMDPs exactly (Sondik, 1971; Monahan, 1982; Littman, 1996; Zhang & Zhang, 2001) which are guaranteed to converge to the optimal value function for certain classes of POMDPs as computation time $t \to \infty$. These algorithms rely on the insight that the value function for a POMDP can be constructed from the supremum of a set of hyperplanes defined over the simplex of the belief space, which results in a piece-wise linear convex value function. The class of POMDPs this applies to is finite-horizon POMDPs with bounded reward. Perhaps the best explanation for constructing a value function was given by Littman (1996) in chapter 7 of his thesis, an explanation we will summarise here.

We will begin by describing policies in terms of policy trees, rather than value functions. A policy tree is a tree describing sequences of actions and observations. A policy tree of size t consists of all possible action-observation sequences of length t. Each arc in the tree is labelled with an observation, and each node with an action to be performed when the observation leading to that node is received. A complete policy is found by assigning different policy trees to different regions of the belief space. A policy can then be executed by looking at the current belief, determining which is the appropriate corresponding policy tree, and executing the action in the root node of that tree. The algorithm for computing a policy of horizon t is given using a recursive procedure. We first show how to compute the one-step policy, and then how to recursively compute the two-step policy.

One-step Planning. Consider a POMDP with a model consisting of 2 states, 2 actions and 2 observations, and its one-step optimal plan. A policy tree for this problem and its value function are shown in figure 2.4. This policy tree is *not* the complete policy; recall

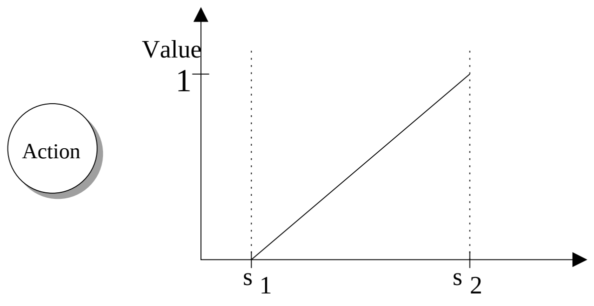


Figure 2.4. A one-step policy tree and its value function. For a two-action model, there are two such policy trees that describe the full policy. The horizontal axis of the graph is the belief b, and the vertical axis is the expected value for each belief.

that we associate different policy trees with different regions of belief space. The optimal policy consists of a division of belief space into convex regions each with its own policy tree.

The one-step policy tree is trivial: the choice of a single action. The value function for this policy tree is also very simple, and is the expected value of the immediate reward function under this policy tree, where the immediate reward for this action (specified in the model) is $R(s_1, a_1) = 0, R(s_2, a_1) = 1$. The value function for the one-step policy tree is a single plane, because the value function is the *expected* immediate reward and expectation is linear in the belief. We can depict the belief space for a 2 state problem using a single axis if we parameterise the belief b as the pair $(p(s_0), 1 - p(s_0))$. In general, a belief space of $|\mathcal{S}|$ states has $|\mathcal{S}| - 1$ dimensions, as does the value function.

The complete one-step policy is generated by assigning one-step policy trees to regions of belief space; the action at the tree root describes the action to take for each belief in that region of belief space. The complete value function is the supremum of the value functions for the trees and determines which policy tree is assigned to which region of belief space. Figure 2.5 depicts the complete policy as the set of one-step policy trees, and the complete one-step value function as the supremum of the one-step policy tree value functions. On the left of figure 2.5, we see two one-step policy trees and their value func-

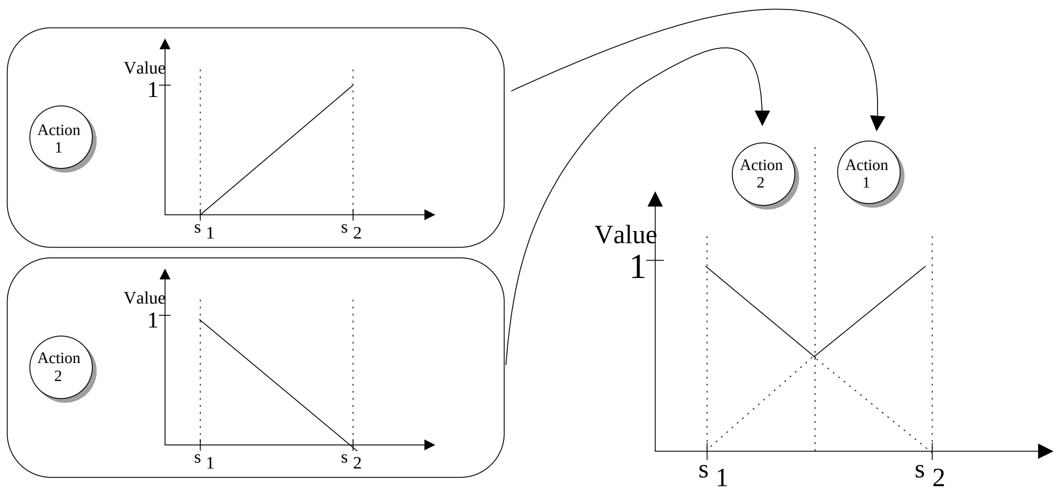


Figure 2.5. For the horizon 1 problem, the complete policy is specified by a set of policy trees, and their value functions. On the left is each one-step policy tree and its value function. On the right is the complete value function.

tions. The lower policy tree is for action 2, and has a different value function because the immediate reward for this action is different $(R(s_1, a_2) = 1, R(s_1, a_2) = 0)$. The graph on the right of figure 2.5 shows the complete value function for the horizon 1, and we see that the two policy trees have different regions of superiority. When the belief is closer to s_1, a_2 has higher value, and when the belief is closer to s_2, a_1 has higher value.

The value function for a finite-horizon POMDP will be the supremum of the value functions for each policy tree, and the value of each tree is a hyperplane, as described above. The value function can therefore be represented by the set of hyperplanes in $\mathbb{R}^{|\mathcal{S}|-1}$ defined by the policy trees. The hyperplanes are often called α-vectors (Kaelbling et al., 1998) (not to be confused with the normaliser of equation (2.8)) or less often γ (Cassandra, 1998) (not to be confused with the discount factor γ). The set of α-vectors we call Γ. The vectors in Γ define a piece-wise linear, convex value function (that the supremum of a set of vectors is convex follows from the definition of convexity). We will abuse the nomenclature in this section by using α-vector to refer to both the specific co-efficients of the value hyperplanes and the value hyperplanes themselves.

Two-step Planning. Having solved the one-step planning problem, we can use this solution to solve the two-step planning problem. Let us first see how to develop a two-step policy tree (such as on the left of figure 2.6), and then see how to compute the entire two-step policy. The entire two-step policy will be given by computing all possible two-step trees and assigning each tree to the region of belief space where its associated α-vector dominates the α-vectors from all other trees. Once the two-step policy is created, we can dispense with the trees and α-vectors from the one-step policy. The $t-1$ policy trees are only useful for computing the t policy.

Figure 2.6 depicts a horizon 2 policy tree on the left. The policy tree consists of an initial action, an (unpredictable) observation, and a subsequent action. We can compute the value of this policy tree from the value functions of the one-step policy trees. Recall that Bellman's equation for MDPs (equation (2.1)) defines the value of an action as the immediate reward of the action plus the expected future reward of the policy after that action. A similar recursive equation exists for iteratively computing value functions for policy trees of increasing horizons.

We already have (from the previous step) the value function of each subtree of depth 1. For each one-step policy tree p_i, the value function for each p_i is

$$V^1_{p_i}(s) = R(s, a_{p_i}), \tag{2.9}$$

where a_{p_i} is the action at the root of p_i.

The value function $V^2_{p_i}$ of a two-step policy tree p_i is the sum of the immediate reward of the root action and the future rewards (as in figure 2.6) where the future rewards are given by the one-step policy subtrees. We cannot know the future reward because we do

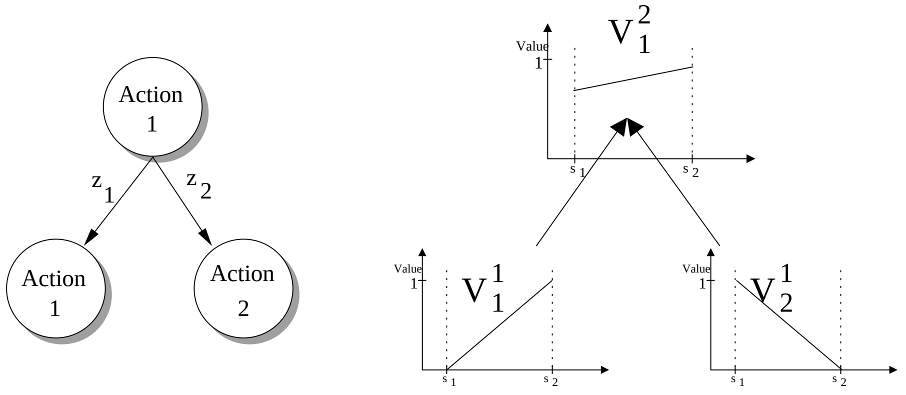


Figure 2.6. A two-step policy tree and its value function. The value of this tree V_1^2 is computed from the weighted combination of subtrees V_1^1 and V_2^1.

not know the observation we will receive or the posterior state, but we can compute an *expectation* over rewards.

We need only compute the value function at each state (the corners of the belief space), and the value at each state is the immediate reward at that state, plus the sum of the values of each next state weighted by the expected probability of that next state, where the expectation is with respect to the observations, as in:

$$V_{p_i}^2(s) = R(s, a_{p_i}) + \gamma \sum_{j=1}^{|\mathcal{S}|} \sum_{k=1}^{|\mathcal{Z}|} p(s_j|s, a_p, z_k) V_{p,z_k}^1(s_i) \tag{2.10}$$

Substituting $p(s_j|s, a_p, z_k)$ using equation (2.8):

$$= R(s, a_{p_i}) + \gamma \sum_{j=1}^{|\mathcal{S}|} T(s, a_{p_i}, s_j) \sum_{k=1}^{|\mathcal{Z}|} O(s_j, a_{p_i}, z_k) V_{p,z_k}^1(s_j), \tag{2.11}$$

where a_p is the action at the root of the tree as before; $V_{p_{z_j}}$ is the value of the one-step policy sub-tree rooted at the observation z_j in the two-step tree p_i. Recall that $T(\cdot)$ and $O(\cdot)$ are respectively the transition and observation probability functions. This update equation gives rise to a recursive algorithm for finding the POMDP value function just as equation (2.1) did for MDPs.

Equation (2.11) tells us how to compute the value function at the root of a single policy tree from its subtrees, but this is again *not* the complete policy. A single policy tree is associated with only a region of belief space. We therefore compute all possible two-step trees, and then assign each tree to the region of belief space where its value dominates all other trees. The set of all possible trees is given by a separate policy tree for each possible root action and for each possible assignment of subtree to observation. Figure 2.7 shows the four possible assignments of the two subtrees for the two-step policy tree in figure 2.6.

The model we are discussing here (2 states, 2 actions, 2 observations), will have 4 ways (2 actions, raised to the power of 2 observations) of assigning $|\mathcal{Z}|$ one-step trees as the subtrees of a two-step policy tree, (where $|\mathcal{Z}|$ is the number of observations).

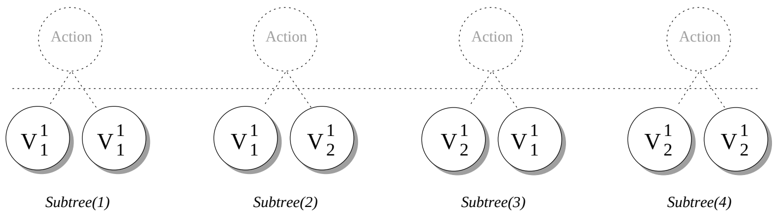


Figure 2.7. The set of 4 possible subtrees of this two-step policy tree, as given by the output of $Subtree(i)$.

Let us define an operator $subtree(i)$ that gives the i^{th} such possible assignment, depicted in figure 2.7. There are then 8 ($|\mathcal{A}|2^2$, where $|\mathcal{A}|$ is the number of actions) total distinct two-step policy trees as figure 2.8 shows. For the Γ^{t-1} policy trees at horizon $t-1$, there will be in general $|\Gamma^{t-1}|^{|\mathcal{Z}|}$ possible assignments of the α-vectors at $t-1$ to subtrees at horizon t. The operator $subtree(i)$ does not appear in equation (2.11), but is implicit in constructing V_p^2, and dictates which value function corresponds to V_{p,z_j}^1, which policy tree at $t-1$ gets assigned to be a subtree at time t.

We are computing the expected value over each of the i assignments of subtrees to observations, because we do not know which observation we will receive after the action. Therefore, for a given arrangement of policy subtrees, we back up the values of all the subtrees weighted by the associated observation probability. This has an additional effect of preserving the linearity of the value function of the policy tree. Backing up a single policy subtree induces a non-linearity, as the observation causes some states to be more probable than others. This effect is cancelled out by the observation probability at each state, causing the expected value function to remain linear.

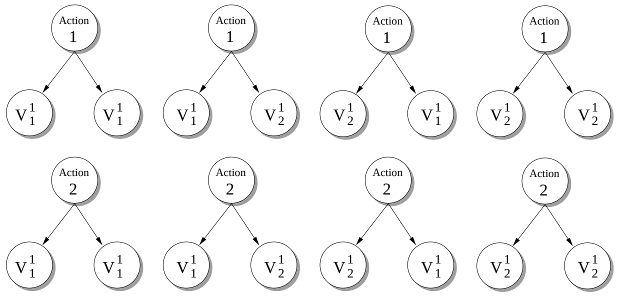


Figure 2.8. The set of all possible 2 step policy trees for our problem.

```
1.  $\Gamma^0 = \{\langle R(\cdot, a_1), a_1\rangle, \ldots, \langle R(\cdot, a_{|\mathcal{A}|}), a_{|\mathcal{A}|}\rangle\}$
2.  t = 1
3.  do
4.      $\Gamma^t = \emptyset$
5.      $\forall a \in \mathcal{A}$
6.      for $p = 1$ to $|\Gamma^{t-1}|^{|}\mathcal{Z}|$
7.          Get $\{\alpha^{t-1}_{z(1)} \ldots \alpha^{t-1}_{z(|\mathcal{Z}|)}\} = subvectors(p, \Gamma^{t-1})$
8.          $\forall s \in \mathcal{S}$
9.              $\alpha^t(s) = R(s,a) + \gamma \sum_{i=1}^{|\mathcal{S}|} T(s,a,s_i) \sum_{j=1}^{|\mathcal{Z}|} O(s_i,a,z_j)\alpha^{t-1}_{z(j)}(s_i)$
10.             $\Gamma^t = \Gamma^t \cup \langle \alpha^t, a\rangle$
11.         change = $max_b(\max_{\alpha_t} \alpha^t \cdot b) - \max_{\alpha_{t-1}}(\alpha^{t-1} \cdot b)$
12.     t = t + 1
13. while not converged
```

Table 2.2. An algorithm for POMDP value iteration.

Horizon t Planning. Instead of explicitly building deeper policy trees as we increase t, we can plan in terms of the value function for each policy tree V_i^{t-1}, using $V_i^{t-1} = \alpha_i \in \Gamma^{t-1}$. It may be counter-intuitive to think in terms of the α-vectors, but a vector α^t is nothing more than the plane that defines the value function for a specific policy tree of depth t, which we termed V_i^t in equation (2.11). Let us define $subvectors(i, \Gamma)$, which returns the i^{th} possible assignment of α-vectors in Γ to observations $\mathcal{Z}$, just as $subtree(i)$ gave the i^{th} possible assignment of policy subtrees. We can then construct the recursive value iteration procedure in table 2.2. The algorithm essentially consists of recursively computing new sets of α-vectors from the previous set of α-vectors until a convergence criterion is met.

For clarity, a few details of the algorithm should be explained.

1. $\Gamma^0 = \{\langle R(\cdot, a_1), a_1\rangle, \ldots, \langle R(\cdot, a_{|\mathcal{A}|}), a_{|\mathcal{A}|}\rangle\}$

   Here, we start with the initial set of α-vectors as the immediate reward vectors. $R(\cdot, a_i)$ is just the vector of immediate rewards for action a_i at each state. Typically, Γ consists only of α-vectors. We will store in Γ the pair $\langle \alpha^t, a\rangle$, so that at execution time, we know what action to use for each α.

6. for $p = 1$ to $|\mathcal{Z}|^{\Gamma^{t-1}}$

   This step enumerates the different ways of choosing subtree α-vectors for the new α-vector

7. Get $\{\alpha_1^{t-1} \ldots \alpha_{|\mathcal{Z}|}^{t-1}\} = subvectors(p, \Gamma^{t-1})$

   This step does the actual assignment of α-vectors to observations, according to the current value of p, for use in step 9. This step causes $\{\alpha_1^{t-1} \dots \alpha_{|\mathcal{Z}|}^{t-1}\}$ change as p increments in step 6.

9. The α_j^{t-1} in this step is not the j^{th} vector in Γ^t, but the α-vector assigned to observation z_j.

12. change = $max_b(\max_{\alpha_t} \alpha^t \cdot b) - \max_{\alpha_{t-1}}(\alpha^{t-1} \cdot b)$

    This step allows us to test for convergence in step 13 by finding the belief with the largest difference in value for the previous set of α vectors (Γ^{t-1}) and the just-created set of α-vectors. If the new α-vectors do not change the value function at any belief (note: belief, not state), then the algorithm has converged and can be stopped.

Pruning. Each iteration generates a new set of α vectors which represent the value for different regions of the belief space. As described previously, the value function is the supremum of these vectors. However, some vectors may not be part of the value function anywhere in the belief space. Figure 2.9 depicts this scenario; there are three vectors, two of which (together) dominate the third vector (labelled "$\alpha3$"). This vector plays no part in the value function at this horizon, and furthermore, will have no influence in any longer horizons (Sondik, 1971; Littman, 1996; Cassandra, 1998). This can be seen from the fact that the value function at horizon V^t (equation (2.11)) depends only on the value function at V^{t-1}. Recall that we compute V^t exclusively from the model parameters O and T, and the parameters of V^{t-1}. If an α-vector at $t-1$ has no influence on the value function, it can have no influence on the value function V^t. Therefore, during value iteration, if an α^t-vector is dominated, it can be removed from Γ^t.

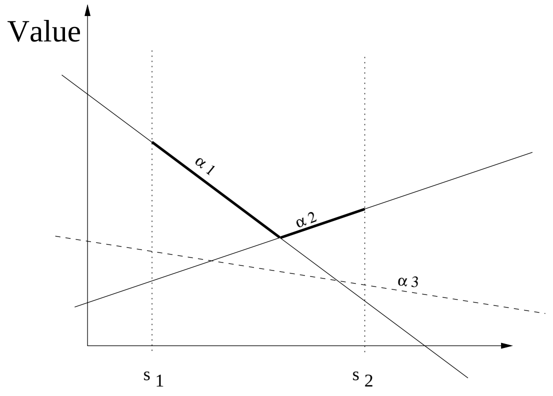


Figure 2.9. Three example α-vectors. The third vector ("$\alpha3$") is dominated by the union of the other two vectors, and can be safely eliminated.

Most exact value iteration algorithms rely on this fact (Sondik, 1971; Monahan, 1982; White III, 1991; Littman, 1996; Cassandra et al., 1997), namely that only *some* α-vectors should be propagated from one iteration to the next. A trivial, but "hopelessly computationally intractable" (Littman (1996), pp. 149) approach proposed by Sondik (1971) would add an additional step to value iteration in table 2.2, to test every α-vector for dominance. More efficient filtering algorithms (White III, 1991; Cassandra et al., 1997) have since been proposed.

In contrast, a large number of value-iteration techniques, such as Sondik's original one-pass algorithm (Sondik, 1971) and Witness (Littman, 1996), one of the most popular value iteration techniques, have been used to solve successfully increasingly larger POMDPs. These algorithms operate by backing up the α-vectors and then finding a region of the belief space where α-vector continues to dominate. The approach is to iteratively generate α-vectors that will survive.

Choosing Actions at Execution. The parameters of the α-vectors give the value at corners of the belief state, whereas we would like to know the value function at arbitrary beliefs, which can be computed from the value function using

$$V(b) = \max_{\alpha \in \Gamma} \sum_{i=1}^{|\mathcal{S}|} b(s)\alpha(s), \tag{2.12}$$

and the optimal policy is the action associated with that α-vector, which is why we store the action a along with the α-vector in Γ, in step 10.

Complexity

Since our complete policy will consist of the supremum of α-vectors that correspond to policy trees, the complexity of the value function (and the policy) can be determined from the complexity of the policy trees. Figure 2.10 shows a larger policy tree.

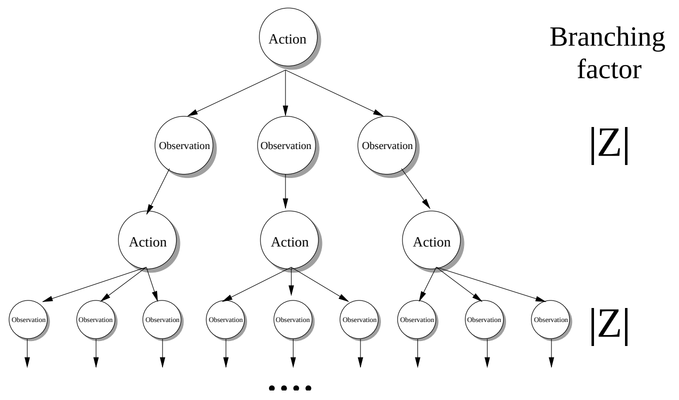


Figure 2.10. A policy tree.

Assuming a depth of t, each policy tree has a total number of nodes:

$$\begin{aligned} 1 + |\mathcal{Z}| + |\mathcal{Z}|^2 + \ldots + |\mathcal{Z}|^{t-1} &= \sum_{i=0}^{t-1} |\mathcal{Z}|^i && (2.13) \\ &= \frac{|\mathcal{Z}|^t - 1}{|\mathcal{Z}| - 1} && (2.14) \end{aligned}$$

Each node in the tree corresponds to an action choice, and therefore we have $|\mathcal{A}|$ ways of labelling each of the $\frac{|\mathcal{Z}|^t-1}{|\mathcal{Z}|-1}$ nodes. If Γ is the number of α-vectors, we have

$$|\Gamma| = \mathcal{A}^{\frac{|\mathcal{Z}|^t-1}{|\mathcal{Z}|-1}} \tag{2.15}$$

α-vectors and possible policy trees.

The bound on the complexity of the α-vectors in equation (2.15) is a somewhat loose bound, in that it is rare that all possible policy trees (and α-vectors) will be required to express the full optimal policy. Typically, some α-vectors will be dominated, and therefore removed from the policy. This improves the complexity of equation (2.15) to a recurrence of the form

$$|\Gamma_t| = O(|\mathcal{A}||\Gamma_{t-1}|^{|\mathcal{Z}|}) \tag{2.16}$$

where $|\Gamma_{t-1}|$ is the number of α-vectors at time $t-1$, and $|\Gamma_t|$ is the number at time t. Different forms of exact POMDP value iteration may have different recurrences from equation (2.16), but none are polynomial.

There is one additional subtlety; these bounds assume that the policy does not know the initial belief. If the initial belief is known, then the size of the set of policy trees is only singly exponential in the horizon length. Furthermore, few problems exist where the initial belief is not specified; consequently, while the true complexity of POMDPs may be bounded by a double-exponential in the horizon length, this may be a pessimistic estimate.

Littman (1996) and Cassandra (1998) showed a number of complexity bounds for POMDPs; these bounds demonstrate the intractability of finding an exact solution for POMDPs where a solution is known to exist. The general case of an infinite-horizon, stochastic POMDP is EXPTIME-hard for boolean rewards, and is known to be undecidable for general (but bounded) rewards, even in the discounted case (Madani et al., 2003). Stochastic POMDP problems with a finite horizon are PSPACE-hard (PSPACE-complete for boolean rewards), and a deterministic POMDP with a finite horizon is NP-complete. For discounted, infinite horizon problems, the optimal policy may consist of infinitely many vectors.

2.4. Approximation Methods for POMDPs

The doubly-exponential nature of equation (2.16) indicates the computational intractability of solving POMDPs exactly. Although the numerous algorithms for pruning Γ^t mentioned above (Sondik, 1971; Monahan, 1982; White III, 1991; Littman, 1996; Cassandra et al., 1997) help control the proliferation of α-vectors to some degree, the largest POMDPs solved tend to have tens of states (Hauskrecht, 2000). There are a number of techniques and heuristics for instead finding approximations to the value function or policies that perform well in a number of situations.

Value Function Approximators

SPOVA. The Smooth Partially Observable Value Approximation (SPOVA) algorithm (Parr & Russell, 1995) is one of the earliest value-function approximation algorithms. SPOVA represents the value function as a soft-max of the α-vectors, as in

$$V(b) = \sqrt[k]{\sum_{\alpha \in \Gamma} (b \cdot \alpha)^k}, \tag{2.17}$$

where k, the smoothing factor, is a free parameter, as is the (possibly fixed) number of vectors in Γ. The authors derive an expression for computing the gradient of the Bellman residual with respect to equation (2.17) at specific belief points. This gives a procedure for updating the α-vectors using gradient descent, gradually increasing k until some termination condition is met (e.g., number of iterations, change in Bellman error, etc.).

The major disadvantage to SPOVA is the large number of free parameters. The number of α-vectors must be chosen, as well as the belief points for sampling the gradient, and the smoothing factor k. The authors describe some procedures for adjusting these parameters as necessary, but it seems likely that attempts to solve real-world problems will become mired in local maxima of the gradient descent, where the maxima are largely a result of poorly chosen parameterisations of the value function. As a result, SPOVA has only been reported on small abstract problems from the literature.

Point-Based Approximations. Possibly the most promising approaches for finding approximate value-functions are the point-based methods, which instead of optimising the value function over the entire belief space, do so only for specific beliefs. Cheng (1988) described a method for backing up the value function at specific belief points in a procedure called "point-based dynamic programming" (PB-DP). These PB-DP steps are interleaved with standard backups as in equation (2.11). Zhang & Zhang (2001) improved this method by choosing the Witness points as the backup belief points, iteratively increasing the number of such points. The essential idea is that point-based backups are significantly cheaper than full backup steps. Indeed, the algorithm described by Zhang & Zhang (2001) outperforms Hansen's exact policy-search method by an order of magnitude for small problems. However, the need for periodic backups across the full belief space still limits the applicability of these algorithms to small abstract problems.

More recently, Pineau et al. (2003a) have abandoned full value function backups in favour of only point-based backups in their "point-based value iteration" (PBVI) algorithm. By backing up *only* at discrete belief points, the backup operator is polynomial instead of exponential (as in value iteration), and, even more importantly, the size of Γ remains constant. The PBVI has been applied successfully to problems at least an order of magnitude larger than its predecessors. It is difficult to measure success in that no provably optimal solution has been found for problems of this size; however, PBVI outperforms all other competitors substantially.

PBVI still remains constrained by two factors; first of all, the choice of beliefs is currently governed by a heuristic exploratory strategy. Secondly, having at least partially defeated the curse of history, PBVI becomes prey to the curse of dimensionality. Regardless, PBVI is one of the most promising value-function based approaches; it seems likely that dimensionality-reduction techniques in combination with PBVI could be very powerful.

Monte Carlo POMDPs. The Monte Carlo POMDP (Thrun, 1999) is an approximation algorithm similar to the point-based approximations in that the value function is maintained at specific belief samples. However, the value function is no longer represented using α-vectors. The MC-POMDP performs a number of one-step roll-outs on samples of the belief space, and computes the expected reward of taking an action from each sample. The value for an arbitrary belief is then found by linear averaging from the k-nearest neighbours. One advantage of the MC-POMDP is that it uses a non-parametric representation, and can therefore be applied to continuous state spaces, but the reported results suggest that learning a policy for complicated problems can be computationally difficult.

Grid-Based Approximations. Lovejoy (1991) proposed the first value-function approximation based on a regular grid, which is very similar to the AMDP algorithm that we describe in chapter 3 in the full belief space as opposed to a low-dimensional representation of the belief space. Hauskrecht (2000) describes this algorithm as having the advantage of "the simplicity with which we can locate grid points", (Hauskrecht (2000), pg. 68) and the marked disadvantage of the curse of dimensionality in the state space: "any increase in grid resolution is paid for in an exponential increase in the grid size" (Hauskrecht (2000), pg. 68).

A variable resolution approach was suggested by Lovejoy (1991) as a possible solution to the exponential nature of the regular grid, and specific algorithms have been developed by Brafman (1997), Hauskrecht (2000) and Zhou & Hansen (2001). The variable resolution algorithms are still constrained by the size of the belief space; these algorithms compute the value of a belief using interpolation rules, such as the Coxeter-Freudenthal-Kuhn triangulation (Moore, 1992; Davies, 1996). As Hansen notes, "interpolation algorithms for non-regular grids are much less efficient" (Zhou & Hansen (2001), pg. 1). The solution of Zhou & Hansen (2001) is to build a "regular" non-regular grid; while the value function of the grid is computed only at non-regular points, the interpolation is performed over a regular grid. Grid points in the regular grid that are absent from the non-regular grid are called "virtual points" and have their value also computed through interpolation.

Many of the grid-based algorithms vary on the choice of splitting criterion, that is, where to increase the resolution of the grid. Zhou & Hansen (2001) use the error-bound given by Lovejoy (1991), which in principle should lead to the optimal variable-resolution representation. However, the complexity of this bound is such that in practice is "intractable for problems with more than nine or ten states" (Zhou & Hansen, 2001). The majority of algorithms use an approximation to this bound (Brafman, 1997; Hauskrecht, 2000; Zhou & Hansen, 2001). In chapter 6 we will describe a different criterion based on predicted reward as in McCallum (1995b). In all grid-based methods, "the size of the state space is the primary factor that limits the scalability" (Zhou & Hansen (2001), pg. 7), and these algorithms have not proven to scale to the size of problems that we wish to address, or have been addressed by other methods such as the policy search algorithms (Meuleau et al., 1999).

Grid-based methods can also be viewed as clustering the belief space. Ivanov et al. (2000) described a *perceptual* clustering algorithm using Expectation-Maximisation (Dempster et al., 1977) for discovering "intrinsic" states, although the algorithm implicitly choose actions for the found state space. The problem solved by this algorithm is more general than the problem addressed in this thesis, in that we assume an *a priori* model. The algorithm could be used to find a small set of states intrinsic to good performance by simulating data from the model. However, this approach is unlikely to scale to complex policies and large state spaces.

Policy Heuristics

There is a family of policies that operate in partially observable worlds, but use greedy heuristics to ignore, or postpone the issues of partial observability. The first and simplest heuristic is the “maximum-likelihood” heuristic (Nourbakhsh et al., 1995). During planning, only the underlying MDP is solved using standard dynamic programming for MDPs (see section 2.1). A full belief is maintained during execution, but the agent uses the maximum-likelihood state for determining the next action from the MDP. This heuristic has been used successfully in robot navigation, but it depends on the fact that the policy of the most likely state in the belief space is a good approximation of the policy of the true state. This is usually the case in problems where the uncertainty is due to bounded noise, as opposed to unknown state features. This heuristic ignores competing hypotheses represented in the belief, which can lead to sub-optimal performance when the policy dictated by the most likely state is inappropriate for the actual current state.

The policy described by the “maximum-likelihood” heuristic for a particular belief b can be represented as:

$$\pi(b) = \underset{a}{\operatorname{argmax}}(Q(\underset{s}{\operatorname{argmax}}(b(s)), a)) \tag{2.18}$$

where $Q(s, a)$ is the expected reward one action later after taking action a, and is given by:

$$Q(s, a) = R(s, a) + \gamma \sum_{j=1}^{|\mathcal{S}|} p(s_j|s, a) \cdot V(s_j) \tag{2.19}$$

where $V(s_j)$ is the converged value for the MDP at state s_j given by equation (2.1). Although the policy is defined for a belief state b, the equation for finding the policy is the same as for the MDP using the maximum likelihood state $\operatorname{argmax}_{s_i} b(s_i)$.

A second heuristic is the voting heuristic (Simmons & Koenig, 1995), which can be viewed as a smoother version of the maximum-likelihood heuristic. Again, during planning only the underlying MDP is solved but a full belief is maintained during execution. At every step, each state votes for the action dictated by the MDP at that state, where each vote is weighted by the probability of that state. The weighted votes for each action are totalled and the action with the largest weight is taken. The policy for this heuristic is:

$$\pi(b) = \underset{a}{\operatorname{argmax}} \sum_{i=1}^{|\mathcal{S}|} b(s_i) \cdot \delta(\pi_{MDP}(s), a) \tag{2.20}$$

where

$$\begin{aligned} \pi_{MDP}(s) &= \operatorname*{argmax}_a Q(s,a) && (2.21) \\ \delta(a_i, a_j) &= \begin{cases} 1 & a_i = a_j \\ 0 & a_i \neq a_j \end{cases}, && (2.22) \end{aligned}$$

and $Q(s,a)$ is defined as in equation (2.19). The assumption made by this heuristic is that although the belief state can represent competing hypotheses, the policy at many of the relevant competing hypotheses is likely to be the same. For example, for a mobile robot travelling down a corridor the optimal policy may be the same for all states—continue travelling down the corridor—even if the mobile robot does not know where it is in the corridor. So long as the uncertainty in the state resolves itself before two high-probability states in the belief require *different* actions then this heuristic will approximate the optimal policy, and indeed has been used successfully for corridor-based mobile robot navigation. However, if the probability distribution is very diffuse, then this heuristic is likely to fail.

Information Gathering Heuristics. Recall from section 2.2 that the true value function of the POMDP is piece-wise linear and convex. More intuitively, if the true state of the system is known, it is easy to choose an action that maximises the immediate reward. If the true state of the system is not known, then it is much harder to maximise reliably the immediate reward. Some heuristics make use of this fact and treat beliefs that are close to the edge of the belief simplex as "better" beliefs than those close to the middle. A number of heuristics, largely proposed by Littman and Cassandra (Littman et al., 1995; Cassandra et al., 1996; Hauskrecht, 2000), attempt to model the effect of actions that reduce uncertainty, or *information gathering* actions.

The first of these is the Q-MDP heuristic (Littman et al., 1995), sometimes called the "Fully-observable after one step" heuristic, which computes one step of POMDP value iteration and then uses the MDP Q values for future expected reward. The policy for a particular belief b can be written as:

$$\pi(b) = \operatorname*{argmax}_a \sum_{i=1}^{|S|} b(s_i) Q(s_i, a), \qquad (2.23)$$

where $Q(s_i, a)$ is the optimal MDP value function. The effect of this policy is to act as if the true state of the problem will become observable after a single step, and as Cassandra notes, "if an action is available which is fairly neutral in terms of rewards" (Cassandra et al. (1996), pp. 262.), the policy will probably choose that action. If, however, the belief does not resolve itself into a single state after that action then the policy will continue to choose the same action with no forward progress.

Hauskrecht (2000) recognised that the major source of computational complexity of value iteration stemmed from the history of actions and observations (cf. equation (2.11)). Fast-Informed Bound (FIB) overcomes this by considering a fixed amount of history, maintaining a fixed set of α-vectors. The original update equation (cf. equation (2.11)) in step 9 of the value iteration procedure given table 2.2 is

$$\alpha^t(s) = R(s,a) + \gamma \sum_{i=1}^{|\mathcal{S}|} T(s,a,s_i) \sum_{j=1}^{|\mathcal{Z}|} O(s_i,a,z_j)\alpha_j^{t-1}(s_i), \tag{2.24}$$

where $\alpha_j^{t-1}(s_i)$ is given by the $subvectors(i,\Gamma)$ function. FIB dispenses with the $subvectors()$ routine, using the same assignment of α-vectors to observations from iteration to iteration. This results in a rule similar to Q-MDP, weighting the future reward by observations:

$$\pi(b) = \operatorname*{argmax}_a \sum_{i=1}^{|\mathcal{S}|} b(s_i)R(s_i,a) + \sum_{j=1}^{|\mathcal{Z}|}\sum_{k=1}^{|\mathcal{S}|} b(s_i)T(s_i,a,s_k)O(s_k,a,z_j)Q(s_i,a). \tag{2.25}$$

The value function computed by this equation is bounded by the optimal POMDP value function below and the QMDP value function above. Hauskrecht (2000) demonstrated the use of FIB on a 20-state problem that was intractable using exact methods, but it fares badly on larger real world problems compared to more sophisticated methods (Pineau & Thrun, 2002).

Two more heuristics due to Cassandra et al. (1996) address the need to model explicit information-gathering actions and to trade off the information against reward. These are dual-mode controllers that alternate between pursuing expected reward and gathering information to refine the belief state. The entropy of the belief state is given by

$$\begin{aligned} H(b) &= -\sum_{i=1}^{|\mathcal{S}|} b(s_i)\log_2 b(s_i) && (2.26)\\ \overline{H}(b) &= \frac{H(b)}{H(u)}, && (2.27)\end{aligned}$$

where $H(u)$ is the entropy of the uniform distribution over all states, and $\overline{H}(b)$ is the entropy normalised to lie in the interval $[0,1]$. The first of the dual-mode controllers is the entropy control that gives the policy:

$$\pi(b) = \begin{cases} \operatorname{argmin}_a E_{\mathcal{Z}}\left[H(p(s|a,z))\right] & \text{if } \overline{H}(b) > \kappa \\ \pi_X(b) & \text{otherwise} \end{cases}, \tag{2.28}$$

where $\pi_X(b)$ is some other heuristic, such as the maximum-likelihood heuristic, and κ is a free parameter chosen by the user. This heuristic simply tries to prevent the entropy, or uncertainty about the belief state, from growing larger than κ, and can act sub-optimally when the uncertainty is irrelevant.

The improved version of this heuristic is the Weighted-Entropy heuristic (Cassandra et al., 1996). Given the optimal value function for the completely-unobservable MDP $V_{CU}(s)$ and the optimal value function for the completely-observable underlying MDP $V_{CO}(s)$ (as in equation (2.1)), the policy given by this heuristic is defined by:

$$\pi(b) = \underset{a}{\operatorname{argmax}} \left[\overline{H}(b) V_{CU}(b,a) + (1 - \overline{H}(b)) V_{CO}(b,a)\right]. \tag{2.29}$$

Essentially, this method arbitrates between the action dictated by a completely-observable model (V_{CO}) which chooses actions as if the system knows the state of the world exactly, and the action dictated by a completely-unobservable model (V_{CU}), which will choose actions as if the system has no idea what the state of the world is. The two methods vote for actions, and the voting is weighted by the current entropy. The higher the entropy, the higher the weight on acting as if the system has no knowledge of the state of the world. The advantage of this method is that if the same sequence of actions leads to the same expected reward regardless of the current belief state, the heuristic will take advantage of such a sequence. Such sequences might exist in worlds with funnelling actions or funnelling states. However, if no such sequences exist, then this heuristic still embodies at every step a choice between state identification, and pursuing reward. Furthermore, if funnelling actions or states exist for *some* belief states but not others, this heuristic will not always be able to take advantage of such actions. Finally, computing V_{CU} is $NP-hard$ (Littman, 1996) and can therefore be computationally intractable. Cassandra (1998) also proposes versions of both heuristics that compute a distribution over actions by summing weighted votes for each action, as in the voting heuristic discussed above, and then examining the entropy of the action space rather than the belief space.

The two policy heuristics (maximum-likelihood state and the voting methods) rely at heart on the value function of the underlying MDP. The convexity of the POMDP value function suggests that periodically, the POMDP policy should take an action that may have low immediate reward, but moves the belief state away from the middle of the belief space. Because the underlying MDP is by definition fully observable, it cannot represent the effect of information gathering actions that refine the belief state. To the MDP, and consequently the heuristics, an information gathering action is a wasted action unless this action also moves the agent closer to the goal. The Q-MDP heuristic is a step toward addressing the problem, by acknowledging that the true underlying state may not always be known, but makes the assumption that by waiting long enough, the true state will become known.

Conversely, the entropy and weighted-entropy heuristics explicitly model the problems with uncertainty about the true state. However, both heuristics over-compensate by modelling the fact that uncertainty above an arbitrary level is never acceptable or is acceptable only if the expected reward is the same for all currently plausible states.

The principal shortcoming of all of these methods is that they cannot make long-term decisions with respect to uncertainty—the planning algorithms in almost all cases assumes full observability (the entropy heuristic being the sole exception, which overcompensates by assuming no observability when full observability no longer applies), and the heuristics make adjustments based on the belief only at execution. The contribution in this work to develop representations of the state uncertainty that can be used by the planner to find good policies tractably.

Hierarchical Methods. Of some interest are hierarchical methods, particularly since these approaches (more so than many of the preceding approaches to POMDPs) have been demonstrated for real world tasks such as navigation and interaction. Theocharous (2002) developed a hierarchical model of POMDPs as an extension of Hierarchical Hidden Markov Models (Fine et al., 1998). This hierarchical model is in the same vein of abstraction as the MDP approximations such as MAXQ (Dietterich, 2000) or ALisp (Andre & Russell, 2002). The approach subdivides the model into a tree of smaller POMDPs—smaller in the sense that each node contains fewer states. The tree is polynomial in the state space of the problem, but since solving each node is exponential, the overall reduction in computation may be substantial. Theocharous (2002) applied this approach to mobile robot navigation, with some success. However, the major drawback to this form of Hierarchical POMDP is that each node is represented as a "macro action" to its parent POMDP, and associated with a task that has an entry state and an exit state. The assumption is that when a task is started the agent is known to be in the entry state, and when the task is completed the agent is known to be in the exit state. The nature of POMDPs suggest that this is at best a brittle assumption.

A different approach to POMDPs is the Policy-Contingent Abstraction (PolCA) for MDPs and its POMDP extension, PolCA+, developed by Pineau et al. (2003b). The principle difference between PolCA+ and other hierarchical abstractions is that the hierarchy is action-based; that is, every state may be present at every node of the tree, but not every action is. The decomposition is again task-based, and nodes in the tree can still be thought of as "macro actions", however the assumption is that not every action is relevant to every sub-task. The PolCA+ decomposition has been applied successfully to the problem of dialogue management, which has a very natural action-based decomposition for managing multiple tasks. The major disadvantage to PolCA+ is that few tasks have a natural action-based decomposition along tasks. The MDP form of PolCA is only *recursively* optimal as defined by Dietterich (2000), which implies a strong task-based structure to the reward function, and PolCA+ has no optimality guarantees.

One of the chief disadvantages of hierarchical methods is that the particular hierarchy must be provided by the user. None of the hierarchical methods have been extended to automatic discovery of the hierarchy, which is important in that the policy quality of these methods is sensitive to the particular hierarchical decomposition.

Policy Search

Policy search methods are based on optimising the policy directly, as an alternative to the value-function methods. One of the earliest such algorithms is REINFORCE (Williams, 1992), which used a gradient descent over the parameters of a neural network to improve the policy. More recently, Baird & Moore (1999) described "Value and Policy Search" (VAPS), an algorithm that combines both policy and value function approaches to reinforcement learning in a single update rule. With appropriate choices of parameters, the VAPS rule can be made equivalent to a number of planning approaches, including value iteration. The authors demonstrate VAPS applied to POMDP problems (where only the observation is available).

In general, these policy search methods have large sample complexities for problems where the correct sequence of actions takes the controller arbitrarily far from the location of the reward (such as problems that require information gathering far from the goal) (Baxter & Bartlett, 2000; Pineau et al., 2003a). The convergence speeds may in practice be improved using reward shaping (Ng et al., 1999), but this necessitates domain knowledge for good shaping functions, as no automatic methods exist for computing them. Additionally, Kakade (2001) demonstrated that the gradients used by many existing policy search methods can take arbitrarily long to converge to good policies.

A number of algorithms specific to graphical representations of policies have been developed using policy search. The U-tree algorithm developed by McCallum (1995a,b) represents the policy as a mapping from observation to actions in a tree of action and observation sequences. The policy is constructed by increasing the depth of the tree until each leaf of the tree can accurately predict the expected reward for the sequence of actions and observations that leads to that leaf. The algorithm searches for deeper trees until the termination condition of correctly predicted rewards is met. While this approach was used successfully to solve an extremely large POMDP ("over 21,000 states" McCallum (1995b), pg. 94), it is not clear how it will scale to problems with policies consisting of long sequences of actions and observations, such as navigation tasks. In particular, the optimal solution tree will share the same complexity as the optimal POMDP value function. However, the U-tree algorithm contains some extremely relevant ideas for approximating value functions, especially the discussion of variable resolution representations in chapter 6.

Meuleau et al. (1999) developed related algorithms for solving large POMDPs where the policy class can be expressed as a simple policy graph (as distinct from the policy trees described earlier.) These algorithms rely on the fact that the policy for any finite-horizon POMDP (and some infinite-horizon discounted POMDPs) can be expressed as a finite (although possibly large) policy graph. The approach is to express a new MDP as the cross-product of the original POMDP state space and a policy graph. The value of this MDP can be computed using value iteration, and the problem is then reduced to finding the policy graph that produces the cross-product MDP with the highest value. This approach is NP-hard in the number of states in the policy graph; consequently, the algorithm is most suited for problems where the size of the policy graph is known to be small. The authors also use stochastic policies for gradient descent, with some success (Meuleau et al., 1999). As in the work of McCallum (1995b), the dependence on simple policy classes suggest that this approach will not scale to the large navigation problems discussed in this thesis, however it constitutes a substantial contribution to algorithms whose complexity are not dominated by the model parameters.

The G-POMDP algorithm (Baxter & Bartlett, 2000) is a direct counter to the "folklore ... that gradient-based methods suffer from unacceptably large variance" (Baxter & Bartlett (2000), pg 7). The advantage of the policy-based approaches is that they may be more robust to errors due to approximation. Approximate value function approaches typically "do not minimise the maximum norm between [the true] and [approximate value functions], but typically some l_2 norm", (Baxter & Bartlett (2000), pg.1). This suggests that although the average error between the true value function and the error approximate value function may be bounded, there may be regions of the state (or belief) space where error can be arbitrarily large. Since real world value function methods are almost always approximations, these statements constitute a strong argument against value function methods.

The G-POMDP algorithm is restricted to the class of randomised policies, mapping observations to controls according to probability distributions. G-POMDP uses gradient descent to optimise the policy parameters, and provides a way to compute an approximation to the gradient of the policy from small sample sets. G-POMDP is the first algorithm to be able to do so from a from a single sample path of the POMDP, and Baxter & Bartlett (2000) provides a proof of convergence to the optimal policy.

Despite the argument against value function methods, G-POMDP contains two shortcomings: the restriction to the randomised policy class and the convergence speed. If the policy class is very simple, then the optimal policy may be a stochastic one. If the optimal policy for the policy class is deterministic, the parameters of the stochastic policy will converge to a deterministic policy, but may only reach a fully deterministic policy in the limit (e.g., the policy may require infinite weights to be represented exactly). More problematic is that the convergence time is bounded by the "mixing" time of the POMDP (Baxter & Bartlett (2000), pg. 2). The authors correctly claim this as an advantage over other policy search methods, which have convergence times bounded by the time to revisit states. Regardless, the time to converge to the stationary distribution of large POMDPs can be substantial.

Pegasus (Ng & Jordan, 2000) is a different policy search algorithm that operates by transforming a stochastic MDP or POMDP model M to a deterministic model M' that contains pre-computed "random numbers" from the stochastic components of M. In this way, different evaluations or samples from M' are "akin to Monte Carlo trajectories ... with the randomisation 'fixed' in advance and 're-used' for evaluating different [policies]." Kearns et al. (2000) provide a proof that re-using the "randomness" in this manner provides uniformly good estimates of the values of policies. Additionally, by transforming the stochastic model into a deterministic model, finding an optimal policy is dramatically simplified.

Pegasus is a potentially useful policy search algorithm for a number of reasons. Pegasus contains "no dependence on the size of the state space or on the 'complexity' of the POMDP's transitions and rewards." (Ng & Jordan (2000), pg. 5). Pegasus also has only a polynomial dependence on the VC dimension of the policy class in the deterministic model: "the crucial quantity is ... the composition of policies and the ... model" (Ng & Jordan (2000), pg.5). If a deterministic simulation of the stochastic model can be found that belongs to a "simple" (i.e., low VC dimension) policy class in the deterministic model, then the Pegasus approach should find the optimal policy quickly.

Unfortunately, it is not clear how to choose a good deterministic representation of a model, and in fact, the authors demonstrate that it is easy to affect the performance of Pegasus on simple problems by choosing a deterministic representation poorly. However, this policy search method does offer hope that if a simple policy exists, choosing the appropriate representation should allow finding the policy quickly, regardless of the explicit model complexity.

Unlike most value function methods, policy search algorithms have been demonstrated successfully on a number of real world applications, such as autonomous helicopter flight (Bagnell & Schneider, 2001), and gait control of a six-legged walker (Andrew Ng, unpublished). While many of the policy search algorithms discussed above owe their success to constraining the policy class, these algorithms have been the most successful in addressing POMDP problems with large state spaces and complex transition and observation models.

2.5. Conclusion

In this chapter we have discussed the problem of sequential decision making in the real world. We described decision-theoretic models for decision making, in particular the partially observable Markov decision process (POMDP). There are a wide variety of techniques for solving POMDPs, however, few of them are capable of solving the real-world problems that we are interested in. The most general exact algorithms do not scale to the size of problems we wish to address, and the approximation algorithms are not capable of modelling uncertainty to the degree that we need.

The problem with many of these techniques is that each tends to be "all-or-nothing" with respect to uncertainty. Either the space of all possible beliefs is to be considered, with the accompanying complexity, or the probabilistic approach is only used for state estimation and the uncertainty of beliefs is drastically simplified. For the kinds of problems that we will address, we need an intermediate approach, as in figure 1.5. In the next chapter we will describe an algorithm called the Augmented MDP, that attempts to plan in *some* representation of the belief space, chosen based on domain knowledge. We will use the insights from this algorithm to develop more techniques in later chapters.

CHAPTER 3

The Augmented MDP

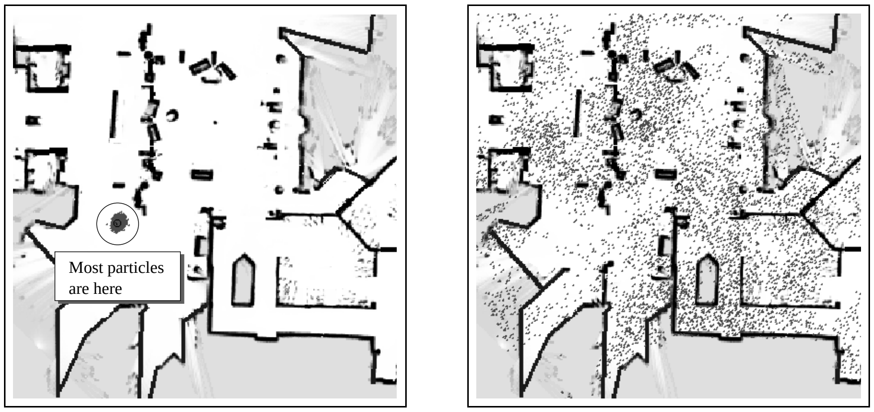
He had marked down certain islands in that sea; Martin Alonzo was of opinion that they were in their neighbourhood, and the Admiral replied that he thought the same, but as they had not met with them, it must have been owing to the currents which had carried them to the northeast and that they had not made such progress as the pilots stated. . . . At sunset Martin Alonzo called out with great joy from his vessel that he saw land, and demanded of the Admiral a reward for his intelligence.

– The log of Christopher Columbus
September 25th, 1492

CHAPTER 2 described a planning representation called the POMDP, for planning in the space of beliefs. The POMDP allows us to find a controller that maximises rewards even when the true state is unknown and the estimate of the state is quite uncertain. We also saw that finding an exact optimal POMDP solution incurs an unacceptably high computational cost.

In this chapter, we describe an intermediate approach that allows the planner to model the state uncertainty and the value of information, while maintaining computational tractability. This approach consists of using a representation that restricts the space of possible beliefs. By choosing a sufficiently low-dimensional representation, we are also able to discretise the belief space. This allows us to characterise a POMDP as a discrete MDP over the (new) belief space, and use the conventional MDP solution algorithms from section 2.1.

There are a number of different choices for representing beliefs using a small number of belief features or statistics. The two features we use in this chapter are the state of highest probability, and the degree of uncertainty, or entropy. These two features may be an appropriate choice for characterising beliefs because for many real world problems, the particular shape of the belief is irrelevant for the purposes of optimal control. That is, the action dictated by the optimal full POMDP policy would be the same for all beliefs with the same approximate location and level of uncertainty. Once we describe how to characterise



(a) An example high-certainty belief (b) An example low-certainty belief

Figure 3.1. Two example probability distributions over robot pose. The small black dots are particles drawn from the distribution over discrete grid positions. On the left is a distribution where the robot's location is relatively certain. On the right is a distribution where the robot's location is very uncertain. Section 4.1 describes in detail why particles are allowed in non-free areas of the map. Briefly, it is to accommodate small inconsistencies between the maps and the real world.

the belief space by a low-dimensional set of statistics, we describe how to convert the parameters of a full POMDP model to the corresponding low-dimensional model, and how to find solutions for this "compressed" representation. We call this algorithm the "Augmented MDP" (AMDP), in that the representation is the conventional MDP augmented with a dimension of uncertainty. In the following chapter, we will demonstrate the use of the Augmented MDP on example problems from robot navigation and speech dialogue management.

The original algorithm, on which the AMDP is based, for dealing with navigation problems in uncertain environments (such as the Smithsonian National Museum of American History) was called "Coastal Navigation" (Roy et al., 1999). This algorithm assigned a cost to being uncertain along the trajectory of the robot from point to point as well as the travel cost of passing through that point, and then it attempted to minimise the overall cost of the trajectory. However, the coastal navigation algorithm proved to be overly limited. By modelling uncertainty as a cost associated to actions, the model only allowed uncertainty to increase monotonically. It was not possible to model arbitrary decreases in positional uncertainty due to periodic re-localisation. Coastal navigation forced the robot to stay as localised as possible as often as possible, which was not always the optimal behaviour.

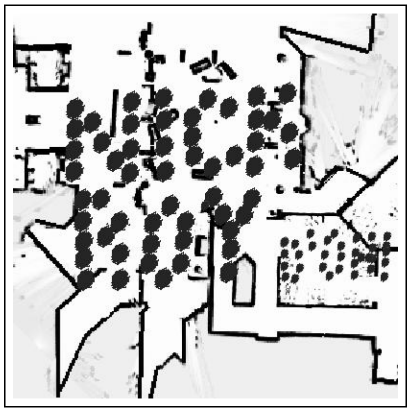


Figure 3.2. This is a belief that cannot be characterised as different from those in figure 3.1 by the statistics we are using, however, this belief is sufficiently unlikely as not to affect the quality of our plan.

3.1. The Augmented MDP

If we examine beliefs from a mobile robot navigating in different environments, certain kinds of regularity are apparent. This is exactly as we would hope; if the beliefs exhibited no regularity, then we would not be able to find a compact representation. The beliefs that we encounter span a range, and figures 3.1(a) and 3.1(b) shows examples from the extrema of this range. Figure 3.1(a) is an example belief with a high degree of certainty; the belief is depicted using a particle filter, where particles are drawn from the state space according to the probability of each state in the belief. In figure 3.1(a) the robot's position is localised to a specific area with high probability, depicted by the very dense concentration of particles in one region. The majority of beliefs that are equally compact can be characterised using the maximum-likelihood state (or possibly the mean). Figure 3.1(b) shows the opposite case; the particles are distributed across the entire map. The maximum-likelihood state is now essentially meaningless, however, the level of uncertainty distinguishes this and similar beliefs from the kinds of belief we see on the left.

Figure 3.2 shows an example of a belief in which the belief features (maximum-likelihood state and entropy) are *not* sufficient for making good control decisions. It is possible to construct such a checker-board belief that has the same low uncertainty as in figure 3.1(a), but the maximum-likelihood state is essentially meaningless (as in figure 3.1(b)) and the optimal action is likely to be substantially different. However, we can take advantage of the fact that this belief is implausible; a mobile robot is extremely unlikely to ever encounter such a distribution about its pose. The probability of encountering such a distribution is not provably 0 for all environments and all mobile robots, but the assumption of the AMDP is that in many real-world problems, these kinds of beliefs are unlikely. In some domains such as dialogue management where multi-modal beliefs seem likely, the AMDP also performs well because multi-modal beliefs can be considered equivalent to high-variance uni-modal beliefs for the purposes of control. We accept losing the ability to plan optimally specifically for beliefs such as in figure 3.2 in exchange for making the planning problem tractable.

A very oversimplified view of AMDP policies may give an insight into how our belief features capture the relevant aspects for planning. AMDP policies tend to choose actions that do not cause the belief to become uncertain. Intuitively, by choosing such actions that keep the belief very certain, the controller attempts to ensure that the actual shape of beliefs becomes less relevant. Furthermore, if the controller chooses actions that never lead to uncertainty, then the maximum likelihood state statistic is a good approximation for making control decisions.

The Augmented MDP Representation

There are different statistics that correspond to probability distribution certainty. Possibly the most general statistic is entropy (cf. chapter 2, equation (2.27)),

$$H(b) = -\sum_{i=1}^{|\mathcal{S}|} b(s_i) \log_2 b(s_i), \tag{3.1}$$

where $H(b)$ is the normalised entropy of the belief b (the distribution is defined over the state space $\mathcal{S}$), and we again normalise to get $\overline{H}(b)$ by dividing by the entropy of the uniform distribution $H(u)$.

We concatenate the maximum-likelihood state (MLS) with the entropy (this notation should not be confused with the inner product), to get the low-dimensional representation $\tilde{b}$ of the full belief b,

$$\tilde{b} = \langle \operatorname*{argmax}_{s \in \mathcal{S}} b(s); \overline{H}(b) \rangle. \tag{3.2}$$

The space of beliefs $\tilde{B}$ is then

$$\tilde{B} = \mathcal{S} \times [0, 1] \tag{3.3}$$

It may also be helpful to notice that entropy roughly corresponds to the distance the belief is from the edges of the belief space. Recall that the convexity of the value function dictates that the expected reward will, for most typical POMDPs, be lower toward the centre of the belief space. The higher the entropy, the closer to the middle of the belief space the system is in, and the lower the subsequent expected reward. By modelling this uncertainty with entropy, we allow the planner to model beliefs where the expected reward will be low, and take information gathering actions, pushing the belief state back toward the edges of the belief space when necessary.

If the belief is defined over a metric state space (e.g., a Cartesian grid), then the variance, or second moment of the distribution, is another way of measuring uncertainty. In the same circumstances, the mean, or first moment of the distribution, could be used instead of the maximum likelihood state. If the distribution is a rotationally-symmetric Gaussian then the sample mean and variance are jointly sufficient statistics for recovering the true belief (which will be discussed further in a subsequent section). In this scenario, the belief state can be represented using a two-dimensional vector regardless of the number of states in the underlying MDP, and any exact, optimal policy computed from this representation will be equivalent to an exact, optimal policy for the full POMDP.

The reason for preferring the entropy and mode is that they can be calculated for problems with non-metric state spaces that arise, for example, in dialogue management problems. However, the disadvantage is that the representation cannot distinguish between a distribution consisting of a single mode and high variance, and a distribution with two widely separated modes of moderate variance. We are strongly relying on the assumption that all distributions of the same uncertainty (entropy) are equivalent.

3.2. Planning with the Augmented MDP

In section 2.2, we described how to find the optimal value function and policy for a full POMDP, where the belief b is represented by a full probability distribution. We were able to represent the value function as the supremum of a set of hyperplanes over the belief space, which resulted in a value iteration procedure, as described by equation (2.11) and implemented in table 2.2.

Once we represent the belief as a set of statistics, however, this approach is no longer applicable. The space of beliefs $\tilde{B}$ is no longer a continuous simplex in $\mathbb{R}^{|\mathcal{S}|-1}$, but a set of continuous spaces (a continuous space along the entropy dimension, for each state in the set $\mathcal{S}$). The value function cannot be expressed as the supremum of a set of hyperplanes over this space; we do not know the functional form of the value function in this representation or how to compute it.

We therefore use a function approximator for the value function. Gordon (1995a) demonstrated that "fitted value iteration" can be proven to converge on function approximators for discounted MDPs if the function approximator is a non-expansion in the max-norm (the function approximator does not exaggerate distances). This result restricts the choice to function approximators such as local weighted averaging, k-nearest-neighbour, linear interpolation, grids, and other "averagers" (Gordon, 1995b). So long as our low-dimensional belief space accurately represents the full belief space, we are planning in a belief-space MDP, and this result still holds.

The simplest averager is a regular grid to discretise the belief space $\tilde{B}$. The advantage to this representation is that if we re-parameterise the POMDP for the new space, we can re-use the value iteration algorithm from section 2.1 with the new update equation

$$V^t(\tilde{b}_i) = \max_a \left(\tilde{R}(\tilde{b}_i, a) + \gamma \sum_{j=1}^{|\tilde{B}|} \tilde{T}(\tilde{b}_i, a, \tilde{b}_j) \cdot V^{t-1}(\tilde{b}_j) \right) \tag{3.4}$$

where $\tilde{R}$ and $\tilde{T}$ are the re-parameterised reward and transition function. In essence, we have turned the POMDP into a discrete belief-space MDP.

One open question is the appropriate discretisation for the entropies. This issue is a form of model selection and will be discussed in greater detail in chapter 6. In general, it is difficult to know *a priori* the right discretisation. One possible criterion for evaluating the discretisation is the policy quality. This suggests one possible approach: compute policies with increasing discretisation resolutions until the difference between successive value functions falls below some threshold. However, our experimental results (reported in the next chapter) suggest that there exist problems that are robust to different entropy discretisations.

Computing the Augmented MDP Model Parameters

In the next two sections, we describe how to generate the low-dimensional model parameters:

- $\tilde{R}(\tilde{b}, a) \to \mathbb{R}$: the immediate reward of the low-dimensional belief $\tilde{b}$.
- $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j) \to \mathbb{R}$: the transition probabilities from low-dimensional belief $\tilde{b}_i$ to $\tilde{b}_j$.

The algorithms for computing these parameters strongly depend on the ability to reconstruct a full belief b from the corresponding low-dimensional belief $\tilde{b}$. This may be an ill-posed problem (Hadamard, 1902) in the sense that multiple solutions may exist, but for the next two sections we will assume the existence of an invertible, deterministic projection to a unique full belief. The subsequent section discusses this assumption in further detail.

Computing the Reward Function

The original reward function $R(s, a)$ represents the immediate reward of the state. We cannot know, given either a low-dimensional or high-dimensional belief, what the immediate reward *will* be, but we can compute the expected reward. We therefore represent the

AMDP reward as the expected value of the immediate reward of the full model, under the current belief,

$$\tilde{R}(\tilde{b}, a) = \mathrm{E}_b(R(s,a)) \tag{3.5}$$

$$= \sum_{i=1}^{|\mathcal{S}|} R(s_i, a) b(s_i). \tag{3.6}$$

Equation (3.6) contains the assumption that b can be recovered from $\tilde{b}$. For many problems, this expected reward function will have the effect of reducing the immediate reward for belief states with high entropy (compared to belief states with low entropy), unless the maximum reward can be achieved across most states that are covered by a high entropy belief state. That is, for many problems the planner will be driven to find beliefs centred on high-reward states that have low uncertainty.

Equation (3.6) gives rise to the algorithm in table 3.1 for computing rewards. Step 2 of table 3.3 contains exactly this procedure.

```
1. For each belief $\tilde{b}_i$ and action $a_j$:
2.     Recover $b_i$
3.     Compute $\tilde{R}(\tilde{b}, a) = \sum_{i=1}^{|\mathcal{S}|} R(s_i, a)b(s_i)$.
```

Table 3.1. The algorithm for computing the low-dimensional reward function $\tilde{R}$.

Computing the Transition Function

Computing the low-dimensional transition function $\tilde{T}$ is not as simple as computing the low-dimensional reward function $\tilde{R}$, in that we need to consider not only the transition function from the full model but also the observation function. Figure 3.3 depicts the process of transitions in the full belief space for a specific initial belief and action.

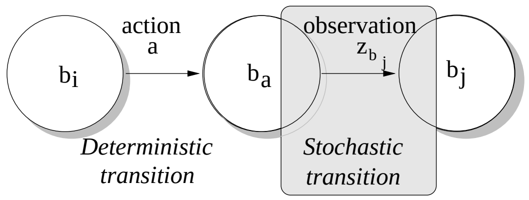


Figure 3.3. Belief transitions are a function of an action and an observation. The action is deterministic in the belief space, therefore the probability of a belief transition is a function of the probability of the observation.

This figure shows that we can separate the Bayes' filter equation (2.8) into a deterministic transition to b_a, the belief after acting but before sensing, and the stochastic transition to b_j, the full posterior.

$$b_a(s) = \sum_{j=1}^{|\mathcal{S}|} T(s, a, s_j) b_i(s_j) \tag{3.7}$$

$$b_j(s) = \alpha\, O(s, a, z) b_a(s), \tag{3.8}$$

where again $T(\cdot)$ gives the transition probability and $O(\cdot)$ gives the observation probability. Equation (3.7) is deterministic because all terms in this expression are specified.[1] In equation (3.8) the observation is the only term *not* specified; therefore, the transition from b_i to b_j occurs only when a specific z is generated, and the probability of this transition is the probability of that observation z.

For each (b_i, a), let us call $\mathcal{Z}_{b_j} \subseteq \mathcal{Z}$ the set of observations that generate the transition $(b_i, a) \rightarrow b_j$. The probability of the transition from b_i to b_j is then a function of the total probability of generating a member from the set of observations $\mathcal{Z}_{b_j}$. This gives the total probability of the transition as a sum over all such $z_k \in \mathcal{Z}_{b_j}$,

$$p(b_j | b_i, a) = \sum_{k=1}^{|\mathcal{Z}_{b_j}|} p(z_k | b_a) \tag{3.9}$$

$$= \sum_{k=1}^{|\mathcal{Z}_{b_j}|} \sum_{l=1}^{|\mathcal{S}|} p(z_k | s_l) b_a(s_l) \tag{3.10}$$

$$= \sum_{k=1}^{|\mathcal{Z}_{b_j}|} \sum_{l=1}^{|\mathcal{S}|} O(s_l, a, z_k) \sum_{m=1}^{|\mathcal{S}|} T(s_m, a, s_l) b_j(s_m). \tag{3.11}$$

Recall that we have assumed an invertible, deterministic mapping from $\tilde{b}_i$ to b_i, so $p(\tilde{b}_j | \tilde{b}_i, a) = p(b_j | b_i, a)$. Figure 3.4 depicts the process of computing the posterior $\tilde{b}_j$.

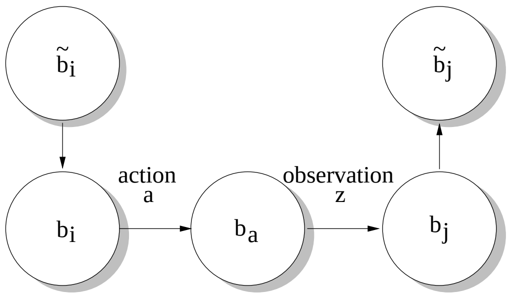


Figure 3.4. The process of computing a single transition probability.

While equation (3.11) tells us how to compute $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$, it does not tell us how to identify $\mathcal{Z}_{b_j}$. A naïve algorithm could iterate through all triplets $(\tilde{b}_i, a, \tilde{b}_j)$, compute the

[1]Equation (3.7) is deterministic for beliefs. This equation is very similar to the stochastic equation for predicting posterior states in MDPs. The difference here is that our posterior state is not known deterministically, but our posterior belief is.

posterior $\tilde{b}'$ for each $(\tilde{b}_i, a, z)$, and set $T(\tilde{b}_i, a, \tilde{b}_j)$ equal to the sum of $p(z|\tilde{b}, a)$ for all z such that $\tilde{b}' = \tilde{b}_j$. A more efficient algorithm can be found if we notice that we want to complete the entire transition table, not just a specific triplet $T(\tilde{b}_i, a, \tilde{b}_j)$. This is the algorithm in table 3.2. Figure 3.4 depicts steps 3, 5 and 6 for a specific z.

```
1. For each belief $\tilde{b}_i$ and action $a$:
2.     For each $\tilde{b}_j$ : $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = 0$
3.     Recover $b_i$
4.     For each observation $z$
5.         Compute $b_j$ from the Bayes' filter, equation (2.8).
6.         Compute $\tilde{b}_j$ from $b_j$
7.         Add $p(\tilde{b}_j|\tilde{b}_i, a)$ from equation (3.11) to $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$
```

Table 3.2. The algorithm for computing the low-dimensional transition function $\tilde{T}$.

Multiple observations may lead to the same posterior $\tilde{b}_j$ in step 6 of table 3.2, so we do not set $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = p(\tilde{b}_j|\tilde{b}_i, a)$, but instead accumulate the probability mass.

Computing the Value Function

Given the algorithm for computing the rewards, we can use equation (3.4) to compute the value function over our discrete belief space. This equation is almost identical to equation (2.1), with the replacement of the immediate reward and transition functions computed in tables 3.1 and 3.2. The full Augmented MDP algorithm is given in table 3.3.

Recovering Full Beliefs

Recall that the AMDP assumes that for a given belief the high-dimensional probability distribution can be recovered. If the low-dimensional representation consists of jointly-sufficient statistics for the beliefs, then recovering the high-dimensional belief is a well-posed problem and an algorithm for doing so usually exists. If this is not the case, one approximating assumption is to define a restricted class of distributions for which the statistics *are* jointly-sufficient.

For example, the statistics of maximum-likelihood state and entropy are jointly sufficient for Gaussians with a single variance parameter. That is, our beliefs can be described by

$$b(s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(s-\mu)^2}{2\sigma^2}}. \tag{3.12}$$

```
1. Generate the discrete low-dimensional belief space $\tilde{B}$ from the full model
2. Compute low-the dimensional reward function $\tilde{R}$
   For each $\tilde{b} \in \tilde{B}, a \in \mathcal{A}$
   (a) Recover $b$ from $\tilde{b}$
   (b) Compute $\tilde{R}(\tilde{b}, a) = \sum_{i=1}^{|\mathcal{S}|} R(s_i, a) b(s_i)$.
3. Compute the low-dimensional transition function $\tilde{T}$
   For each $\tilde{b} \in \tilde{B}, a \in \mathcal{A}$
   (a) For each $\tilde{b}_j : \tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = 0$
   (b) Recover $b_i$
   (c) For each observation $z$
   (d)     Compute $b_j$ from the Bayes' filter equation (2.8).
   (e)     Compute $\tilde{b}_j$ from $b_j$
   (f)     Add $p(\tilde{b}_j|\tilde{b}_i, a)$ from equation (3.11) to $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$
4. Compute the value function for $\tilde{B}$
   (a) t = 0
   (b) For each $\tilde{b}_i \in \tilde{B} : V^0(\tilde{b}_i) = 0$
   (c) do
   (d)     change = 0
   (e)     For each $\tilde{b}_i \in \tilde{B}$:
             $V^t(\tilde{b}_i) = \max_a \left(\tilde{R}(\tilde{b}_i, a) + \gamma \sum_{j=1}^{|\tilde{B}|} \tilde{T}(\tilde{b}_i, a, \tilde{b}_j) \cdot V^{t-1}(\tilde{b}_j)\right)$
             change = change + $V^t(\tilde{b}_i) - V^{t-1}(\tilde{b}_i)$
   (f) while change $> 0$
```

Table 3.3. The Augmented MDP algorithm

The mean is identically the maximum-likelihood state, and the variance can be computed from the entropy using:

$$\sigma^2 = \int_{-\infty}^{\infty} b(\mu, \sigma^2)(s-\mu)^2 \tag{3.13}$$

$$= -2\sigma^2 \int_{-\infty}^{\infty} b(\mu, \sigma^2) \left(-\frac{(s-\mu)^2}{2\sigma^2}\right) \tag{3.14}$$

$$= -2\sigma^2 \int_{-\infty}^{\infty} b(\mu, \sigma^2) \left(\log e^{-\frac{(s-\mu)^2}{2\sigma^2}}\right) \tag{3.15}$$

$$= -2\sigma^2 \left(\int_{-\infty}^{\infty} b(\mu, \sigma^2) \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(s-\mu)^2}{2\sigma^2}}\right) - 2\sigma^2 \left(\int_{-\infty}^{\infty} b(\mu, \sigma^2) \log \sqrt{2\pi\sigma^2}\right) \tag{3.16}$$

$$= -2\sigma^2 \left(\int_{-\infty}^{\infty} b(\mu, \sigma^2) \log b(\mu, \sigma^2)\right) - 2\sigma^2 \log \sqrt{2\pi\sigma^2} \left(\int_{-\infty}^{\infty} b(\mu, \sigma^2)\right) \tag{3.17}$$

We can simplify both integrals using the definition of entropy and the law of total probability:

$$\sigma^2 = 2\sigma^2 H(b(\mu, \sigma^2)) - 2\sigma^2 \log \sqrt{2\pi\sigma^2} \tag{3.18}$$

$$2 = H(b(\mu, \sigma^2)) - \log \sqrt{2\pi\sigma^2} \tag{3.19}$$

$$\sigma^2 = \frac{1}{2\pi} e^{2(H(b(\mu,\sigma^2))-2)}. \tag{3.20}$$

A similar computation can be carried out for multi-dimensional state spaces and covariance matrices, under the assumption that the covariance matrix can be written as $\sigma^2 I_n$, where I_n is the n-dimensional identity matrix.

By definition, the mean and variance are jointly-sufficient statistics for representing Gaussian distributions. These statistics can be computed directly from our statistics of maximum-likelihood state and entropy, which must consequently also be sufficient. Therefore, if our class of beliefs is restricted to the Gaussians with a single variance parameter, we can recover a unique high-dimensional belief from any low-dimensional representation. This property ensures that any error in our policy is directly attributable to errors in our value function (e.g., the function approximation) and *not* due to errors in the representation. We will see in the following chapter that although our beliefs are not exact Gaussians, they are well-approximated by Gaussians, and we can in fact compute good policies by assuming that they are in fact true Gaussians.

It may not be possible (or reasonable) to assume the beliefs are drawn from some identifiable restricted class of distributions. An alternate approach to estimating the model parameters is is to sample full beliefs and true states (i.e., from simulating the model), and use these samples to estimate the low-dimensional model parameters.

Let us assume a set of t sample full beliefs $B = \{b_1, b_2, ..., b_t\}$, the corresponding true states $\{s(b_1), s(b_2), ..., s(b_t)\}$ actions $\{a(b_1), a(b_2), ..., a(b_t)\}$ at each step. $\tilde{R}(\tilde{b})$ can then be computed using

```
1. Initialise all $\tilde{R}(\tilde{b}, a) = 0$
2. $\forall \tilde{b} \in \tilde{B}$:
3. For each $b_i \in B$ where the low-dimensional $\tilde{b}_i = \tilde{b}$
4. Add to $\tilde{R}(\tilde{b}_i, a(b_i))$ the actual immediate reward $R(s(b_i), a(b_i))$.
5. Normalise each $\tilde{R}(\tilde{b}, a)$ by the number of samples added.
```

We can also compute $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$ using the following algorithm:

```
1. Initialise all $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = 0$
2. for $k = 1 \ldots t-1$
3.         Compute $\tilde{b}_i$ from $b_k$ and $\tilde{b}_j$ from $b_{k+1}$
4.         Increment $\tilde{T}(\tilde{b}_i, a(b_k), \tilde{b}_j)$ by 1
5. Set all $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = 1/\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$
```

These two algorithms are equivalent to the certainty-equivalent model learning algorithm from reinforcement learning (Kumar & Varaiya, 1986; Kaelbling et al., 1996). The model parameters computed in this manner will have a strong dependence on the policy used to sample the beliefs. If domain knowledge of the problem suggests a reasonable restricted class of belief distributions for estimating the model parameters, and the size of the belief space $\tilde{B}$ is not prohibitively large for the algorithms in table 3.1 and table 3.2, then in practice these algorithms are more reliable estimates of the model parameters. If, however, these algorithms cannot be used, then sampling may be an acceptable alternate approach.

3.3. Conclusion

This chapter describes the Augmented MDP, a new algorithm for solving POMDP problems. The approach represents beliefs not as full distributions over states, but as a set of statistics over states, specifically the maximum likelihood state and the entropy. By discretising the new belief space, we are able to convert a POMDP into a belief state MDP.

In the next chapter, we will demonstrate the use of the AMDP in two domains, mobile robot navigation and speech dialogue management. We extend the mobile robot navigation problem to handle not only robust navigation in sparse environments but also environments with dynamic obstacles. Similarly, we will show that the planner can generate a dialogue management strategy that automatically compensates for differing levels of speech recognition.

CHAPTER 4

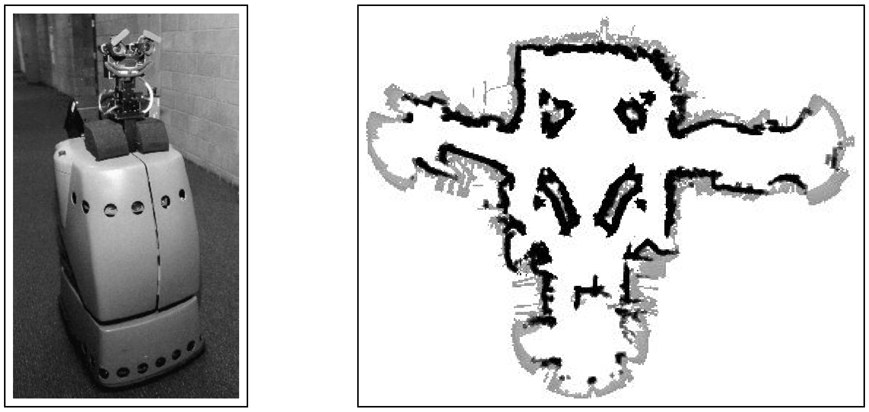
Experimental Results for The Augmented MDP

Directness there must be and singleness of aim: it is all aim, all trajectory.
– Wyndham Lewis
"One-Way Song"

In this chapter we will examine the performance of the Augmented MDP (AMDP) on two example problems in the real world, mobile robot navigation and spoken dialogue management. We will first describe how each problem can be phrased as a POMDP. We will further motivate this by demonstrating how, even with robust probabilistic state estimation, a controller that assumes full observability, for example by extracting a single state estimate from the full distribution over states, will perform poorly. We will then solve each problem using the AMDP planner to compute a new policy that is demonstrably superior than the MDP solution in terms of reward.

4.1. Mobile Robot Navigation

The first example of the AMDP is motivated by Minerva, a museum tour-guide robot that operated in the Smithsonian for two weeks in the summer of 1998 (Thrun et al., 2000). Minerva operated autonomously for hours on end, giving tours to thousands of people; Minerva needed to be able to navigate reliably throughout this time. Although the robot used Markov localisation (Burgard et al., 1996), a form of probabilistic state estimation that had been previously demonstrated to be robust in real world environments (Burgard et al., 1999), the state estimate was highly uncertain (if not incorrect) during many times of the day. This uncertainty was a result of two features of the operating environment. The first was the sparseness of the environment. Figure 4.1 shows an overhead map of the operational area in the museum, which consisted of large areas of free space with very little environmental structure. Furthermore, the environment was extremely crowded, with groups of people surrounding the robot and blocking the laser range sensor. The crowds prevented the robot from seeing the true structure of the environment which was contained in the map. They also introduced false structure into the sensor perception.



(a) Minerva

(b) A map of the Smithsonian

Figure 4.1. In panel (a), Minerva, the museum tour-guide robot that operated in the Smithsonian National Museum of American History for 2 weeks in the summer of 1998. In panel (b), an overhead map of the robot's operational area in the museum. The dimensions of the area shown in the map are $53m$ by $67m$.

In the context of navigation, we shall refer to the problem of state estimation as "localisation", and assume that we are using a probabilistic localisation algorithm such as Markov localisation (Burgard et al., 1996) or Monte Carlo localisation (Thrun et al., 2000). The high expected uncertainty of the estimate of Minerva's localisation in the Smithsonian museum meant that the mean state estimate used by the motion planner (Burgard et al., 1999) was often likely to be wrong. However, the very large state space (22,226 states)[1] meant that an exact POMDP solution using the full probability distribution from the localiser was intractable. The same problem exists for another environment that we have tested in simulation, the Longwood retirement facility shown in figure 4.2(a) which has 50,652 grid cells.

We therefore applied the AMDP to the navigation problem, in order to model the uncertainty of the localisation process. The AMDP model took into account both the local structure of the environment for localisation and the probability of the structure being blocked by people. By modelling the effect of the local environmental structure and the local crowds on the sensing, and therefore localisation process, the AMDP generated more

[1]The map in figure 4.1 is approximately $53m$ by $67m$, with each grid cell $0.2m$ on a side. There are $335 \times 263 = 88105$ total grid cells, but many of these are occupied, or outside the operational area. The total number of usable cells is 22,226. We assume a holonomic robot, which allows us to ignore orientation for planning, although not for state estimation.

robust trajectories. These trajectories minimised where appropriate the uncertainty of localisation (due to either crowds or uninformative sensor data), which in turn minimised the probability of erroneously choosing a very low-reward action (e.g., moving outside the operational area) or becoming lost .

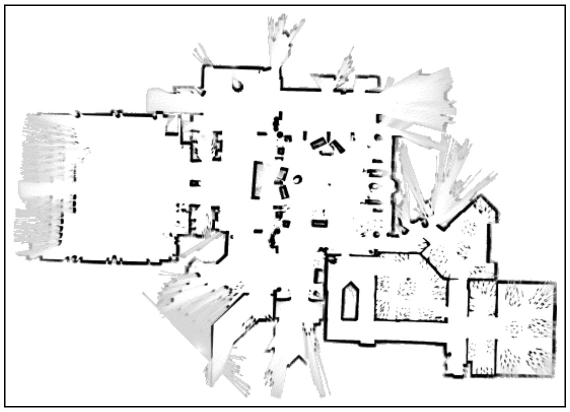
Similar to the AMDP, the approach of measuring uncertainty with entropy and then acting to improve the localisation estimate can be found in the active localisation literature (Fox et al., 1998). However, active localisation does not plan trajectories that improve localisation while simultaneously achieving some goal, but instead dictates the optimal trajectory only for discovering the true location of the robot. A POMDP planner computes a trade-off between active localisation when necessary and conventional distance-optimal planning for maximising reward.

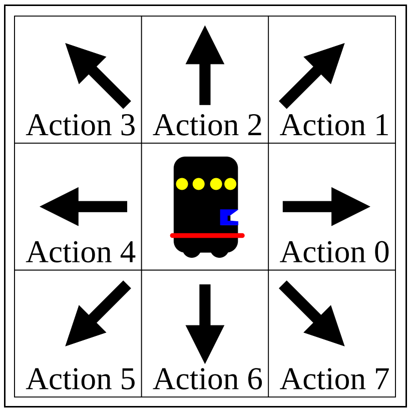
Many of the heuristics we discussed in chapter 2 have been used to recover from localisation failure (Takeda et al., 1994; Nourbakhsh et al., 1995; Koenig & Simmons, 1996), but we will show the failure of one of these heuristics later in this chapter; in general these heuristics do not allow the global solutions we require for good performance in many problems. In topological environments, full POMDP planners have been used with some success (Mahadevan, 1998) by modelling physical spaces topologically (representing space as a connected graph) instead of metrically (representing space as a grid). However, these approaches rapidly approach the complexity limit inherent in exact POMDP planning and are not tractable for the finer-grained grid-based representation presented here.

We begin with a description of navigation as a Markov decision process, and then describe probabilistic state estimation for navigation (localisation). We show how to implement navigation as an AMDP, and conclude with experimental results comparing a conventional motion planner (MDP) and the AMDP.

Navigation as an MDP

As described in section 2.1, we must specify the state and action spaces, and the reward and transition functions for the navigation problem. There exist many motion planning algorithms (Latombe, 1991; Kavraki et al., 1996; LaValle & Kuffner, 2001) for planning in fully-observable continuous domains; conventional discrete-state MDP planning has also been extended to continuous space motion planning problems (Munos & Moore, 1999; Roy & Thrun, 2002). However, in order to allow us to later extend the navigation problem to POMDP-style planning (which assumes discrete states and actions), we will assume heavily discretized state and action spaces from the outset.


(a) Sample state space



(b) Action space

Figure 4.2. (a) A sample map for navigation, of the Longwood at Oakmont retirement facility. The white region is free space, the black regions are obstacles, and the light grey regions are unknown. This map is $125m \times 35m$, with a resolution $0.01m^2$ per pixel. (b) The action space of robot navigation.

We assume the existence of an *a priori* grid map of the environment, such as in figure 4.2(a). The grid map describes a discrete state space

$$\mathcal{S} = X \times Y, \tag{4.1}$$

In figure 4.2(a), the grey-black areas are obstacles and the white area is free space. The action space consists of the 8 actions shown in figure 4.2(b). Again, we assume a holonomic robot that allows us to ignore orientation for planning, although we shall require it for state estimation described in the next section.

We define a simple transition model that takes the robot with 90% probability to the state indicated by the action, and with 5% probability each to the two neighbouring states (around the perimeter of figure 4.2(b)). This transition function assumes that the robot is holonomic, and only applies to free grid cells. Actions from or into occupied grid cells are assumed to self-transition with 100% probability, regardless of action. This transition model does not properly correspond to the motion model used for localisation as described in the next section; the MDP is incorporating a strong abstraction of the transition model that allows us to plan quickly. Planning with the localisation motion model would require a much larger space of actions.

For the reward function, we assign the goal state $g = (x_g, y_g)$ a large positive reward 1000.0. Additionally, we assign a small negative cost -1 of being in a free cell and a large negative cost -1000.0 of being in an occupied grid cell. The cost in all cases is independent of the specific motion action, which results in

$$R(s,a) = R(s) = \begin{cases} -1000 & s \text{ is occupied} \\ -1 & s \neq g \text{ and } s \text{ is free} \\ 1000 & s = g \end{cases} . \tag{4.2}$$

Notice that the reward function depends only on the state. We are making a simplification, that all actions have the same relative cost; the only relevant issue is the cells that are visited along the trajectory. However, this reward function may not be sufficient for all navigation tasks. For example, if the state space also included orientation, then the reward function could be extended to capture the fact that many robots can move in the current direction more readily than changing directions.

Finally, the initial state is assumed to be wherever the robot is initially.

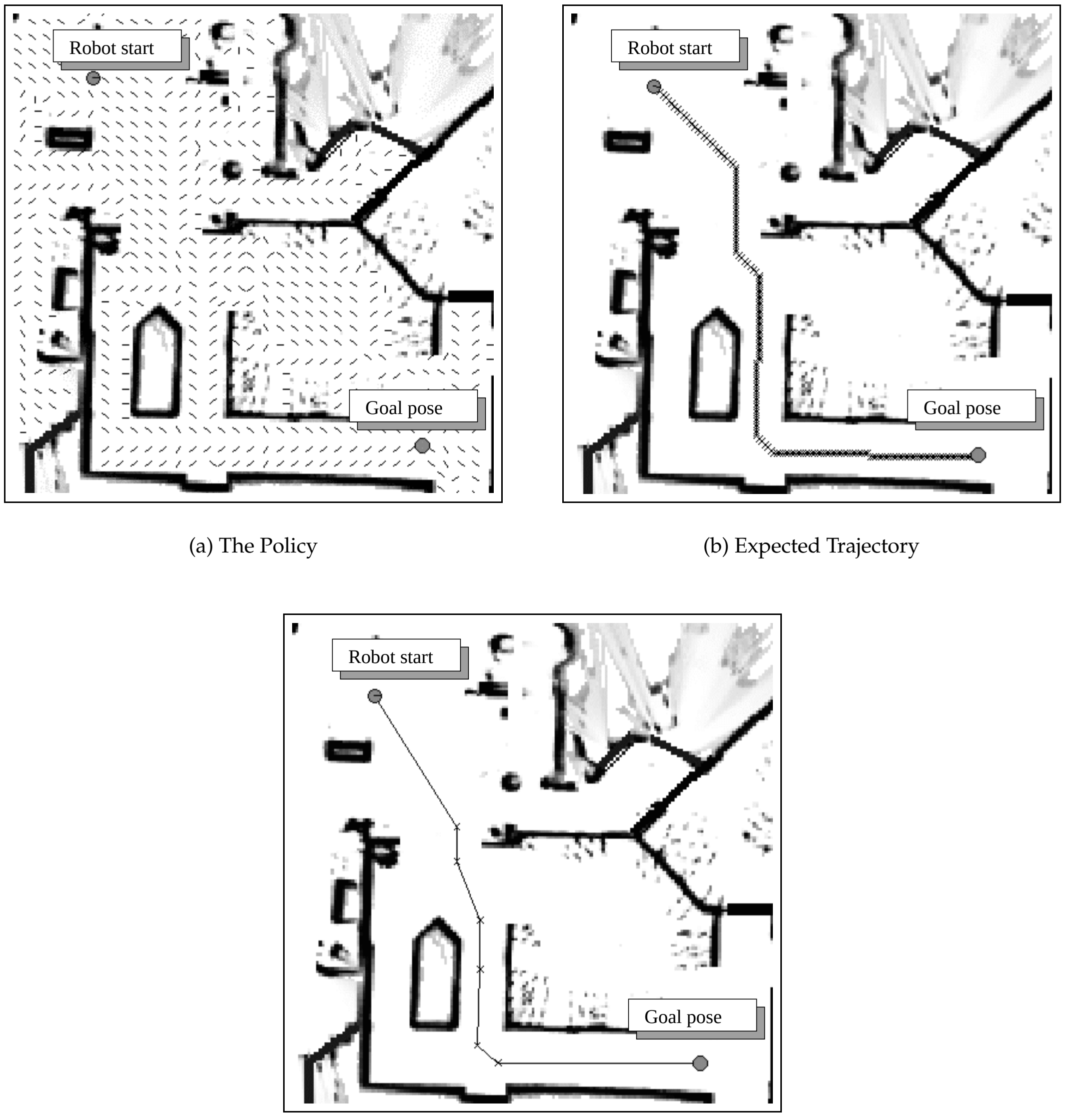
We apply the MDP value iteration algorithm described in section 2.1, table 2.1 to generate a value function $V(s)$. For the state space shown in figure 4.2(a) and the goal shown in figure 4.3, we compute the policy shown in figure 4.3(a). In this figure, each small vector line corresponds to the action for the state at that location. (The action for only every 25th state is shown for clarity.)

The details of implementing this policy are beyond the scope of this thesis; however, a policy is typically executed on a mobile robot by computing a trajectory of the maximum-likelihood states from the value function. This trajectory is then smoothed into a sequence of waypoints, and a local controller is used to convert the robot's displacement from the nearest waypoint into wheel velocities. Further details can be found in such work as Konolige (2000) or Roy & Thrun (2002).

Localisation

Early motion planners assumed that a robot would never be lost—that a robot could always know its position via dead reckoning without error (Latombe, 1991). This assumption proved to be untenable due to the small and inevitable inconsistencies in actual robot motion; robots that rely solely on dead reckoning for their position estimates lose their position quickly. There are wide variety of techniques for implementing probabilistic localisation, including but not limited to Kalman filters (Kalman, 1960; Leonard & Durrant-Whyte, 1991), grid-based approaches (Burgard et al., 1996; Fox et al., 1998), and particle-filter approaches (Thrun et al., 2000; Fox, 2001).

In order to allow the extension of navigation to POMDP planning, we assume that the probability distributions provided by localisation are defined over a state space similar



(a) The Policy

(b) Expected Trajectory

(c) Smoothed Trajectory

Figure 4.3. Robot navigation as a Markov decision process. (a) The policy computed using MDP value iteration. Each small vector line depicts the direction of the optimal action at each cell; only the actions for every 5th grid cell along the x and y axis are shown. (b) The maximum-likelihood trajectory, extracted from the policy. (c) The smoothed maximum-likelihood trajectory used for control.

to equation (4.1), from an *a priori* map such as in figure 4.2(a). However, accurate position estimation is substantially more robust when the three-dimensional pose of the robot is estimated. Consequently, although we may only be planning over the state space in equation (4.1), the state space for localisation, $\mathcal{S}_l$, is

$$\mathcal{S}_l = X \times Y \times \Theta. \tag{4.3}$$

When the belief $b(s)$ is provided to the planner, we can compute a distribution over $\mathcal{S}$ from the distribution over $\mathcal{S}_l$ by marginalising over the Θ dimension.

The environmental models (i.e., maps) we use assume that the robot is equipped with a range sensor. The majority of localisation algorithms assume range sensors such as laser sensors (Burgard et al., 1999; Thrun et al., 2000) or sonar sensors (Leonard & Durrant-Whyte, 1991; MacKenzie & Dudek, 1994; Borenstein et al., 1996; Burgard et al., 1996; Fox et al., 1999), although localisation has been developed using other sensors such as vision (Cox & Wilfong, 1990; Dudek & Zhang, 1996; Sim & Dudek, 1998).

The particular sensor we use is a SICK laser range finder (Sick Optics, 2003) that projects a beam of infrared light parallel to the floor and about 30cm above it. This sensor returns 180 range measurements where each measurement corresponds to the nearest obstacle along a particular direction, for 180 directions around one half of the circumference of the robot, in 1° increments. Our robots have two such sensors back to back, which provide a 360° field of view. An example laser measurement is shown in figure 4.4. The robot is the black square approximately in the middle, the black line segments are the endpoints of the laser data, and the white regions are the known empty space. The light grey regions are regions for which this measurement contains no information. The front and rear laser sensors are slightly offset, leaving a small blind spot on either side.

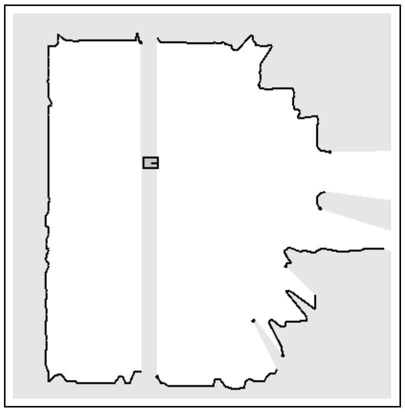


Figure 4.4. An example laser range measurement. The robot is the black square approximately in the middle, the black line segments are the endpoints of the laser data, and the white regions are the known empty space. The light grey regions are regions for which this measurement contains no information.

We use the Bayes' filter of equation (2.8) to maintain the current belief over the robot's pose. We typically implement the Bayes' filter for localisation as a two-step process, with a prediction step where a^t is provided by an odometry reading:

$$p(s^t|a^t, s^{t-1}) = \sum_{j=1}^{|\mathcal{S}|} p(s^t|s_j^{t-1}, a^t)p(s_j^{t-1}) \tag{4.4}$$

and a measurement step where z^t is given by the laser range finder

$$p(s^t|\, z^t) = \alpha p(z^t|\, s^t)p(s^t), \tag{4.5}$$

and α is a normalisation constant.

The terms $p(s^t|s_j^{t-1}, a^t)$ and $p(z^t|\, s^t)$ are referred to as the motion model and the sensor model respectively. The motion model is generated from the physical characteristics of the robot and the particular action; the further s^t deviates from the intended destination of a^t at s_j^{t-1}, the lower the probability typically is. The sensor model is generated from the physical characteristics of the sensor and the map of the environment, and typically assigns high probability to readings that correspond to grid cells at the boundary of occupied and free space, and low probability to readings that correspond to other grid cells (i.e., free space or inside obstacles). It is beyond the scope of this thesis to discuss how these models are constructed. Further details can be found in many discussions of localisation, such as Leonard & Durrant-Whyte (1991); Burgard et al. (1996); Thrun et al. (1998).

One subtlety for implementations of localisation is that we assume that the state space consists of the entire grid world, including grid cells that are labelled as containing obstacles. These occupied grid cells are not prevented from having probability mass, although such cells that receive probability from the motion model typically converge quickly to zero probability by the perception model. If the map corresponds exactly to the true world, then some computation maybe be saved by disallowing the motion model to increase the probability of occupied cells. However, even the best maps rarely correspond exactly to the real world. Consequently, the true robot position may in fact correspond to an grid cell modelled as occupied. One way to handle this unfortunate case is to allow sensor data to change the map (Burgard et al., 1999). Instead, for simplicity, we do allow probability mass in occupied cells specifically during state estimation.

Navigation Failures

The preceding description of planning and localisation leads to the algorithm in table 4.1 for executing an MDP plan using probabilistic state estimation.

In figure 4.5, we see a situation where this heuristic fails. In this example, the map is of the Longwood at Oakmont retirement facility from figure 4.2(a). The perception range has been artificially limited to $2m$; the robot cannot observe any environmental feature

```
Given a goal state $g \in \mathcal{S}$, and a current position estimate $b$
1.  Update $R(g)$
2.  Compute $V(s)$ using value iteration, table 2.1
3.  While $\operatorname{argmax}_s b(s) \neq g$
4.      Execute action $a = \pi(\operatorname{argmax}_s b(s))$ from equation (2.2)
5.      Update $b$ according to odometry $a$
6.      Update $b$ according to observation $z$
```

Table 4.1. The algorithm for planning and executing a motion strategy using an conventional planner (i.e., an MDP) and probabilistic state estimation.

further than $2m$. The action transition function variance has been set at a high level, and the model parameters adjusted accordingly. These two features of the problem cause the robot to have difficulty tracking its position across expanses where environmental features are distant and sparse.

Unless the robot stays close to highly distinct perceptual features in the environment, the probability is high that the robot will become essentially lost, although the position estimate should correctly reflect this fact. We see exactly this scenario in figure 4.5.

In each panel of figure 4.5, the small grey dots are particles drawn from the distribution of poses (the orientation of each particle is not drawn). The open black circle is the mean of the distribution being used for control, the closed grey circle at the top of each panel is the goal, and the closed black circle is the true (unobservable) position of the robot. We see in figure 4.5(a) that the robot is initially well-localised, and stays well-localised until it moves out of sensor range of the environmental features, in figure 4.5(d). By figure 4.5(e) the probability distribution has started to spread, and in the final frame the robot is essentially lost. The distribution reflects the fact that the robot could be up against any wall, when in fact it is up against the left wall.

This example shows the failure of maximum-likelihood heuristic described in chapter 2. However, none of the other heuristics perform substantially better. The dual-mode entropy heuristic, with careful choice of the threshold parameter, may eventually achieve the goal; however, this depends on whether the second, “unobservable” controller is able to recover from the high-level of uncertainty. Certainly, in terms of total reward achieved, this heuristic will not perform as well as the AMDP controller we describe in the following section. This simulated experiment assumed that the sensing capabilities of the robot were substantially impoverished compared to the sensors used in the real world; however, robots operating in very unstructured environments such as antarctic or desert rovers will

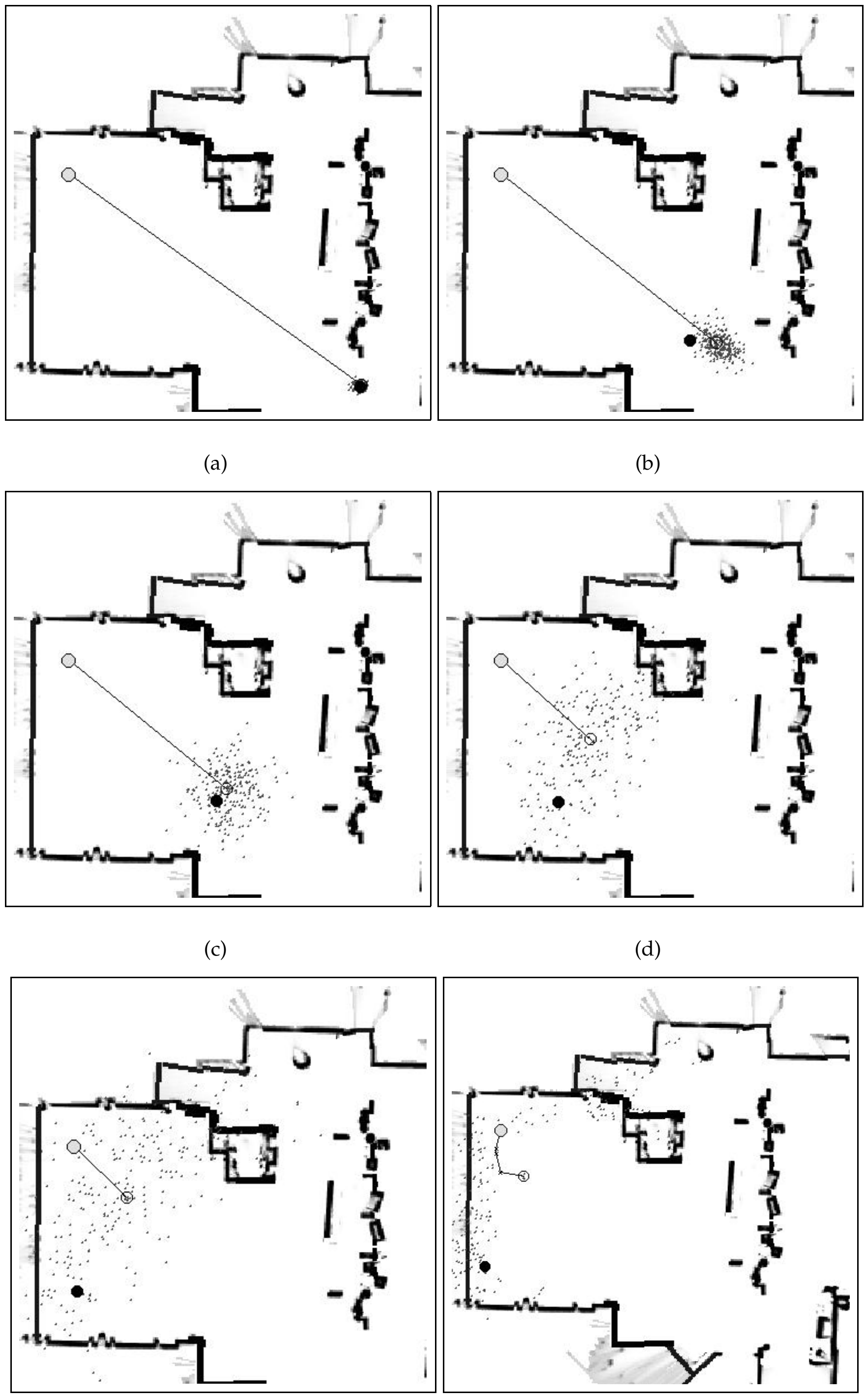

(a) (b) (c) (d) (e) (f)

Figure 4.5. An example of MDP-style navigation failing. Top left (a) is the initial, start distribution, where the robot is the solid black circle near the bottom right of the panel and the goal in the top left. The open circle is the state estimate used for control, and the solid line is the maximum-likelihood trajectory. Bottom right (f) is the final position. The probability distribution has become extremely uncertain, and the robot (the solid black circle) is up against the left wall.

have very similar properties. The example shown here illustrates a kind of failure that can occur when the controller does not model perceptual uncertainty.

4.2. Augmented MDP

The failure of the navigation algorithm described in table 4.1 is an example of the failure of the maximum-likelihood heuristic described in section 2.4. The optimal exact POMDP controller, in contrast, would be able to model the possibility of such localisation failures and the cost of potentially being very unlocalised. If a policy exists that can get the robot to the goal more reliably by avoiding being uncertain, a POMDP controller would be able to take advantage of this.

The sizes of the state spaces of the environments we wish to navigate in (the Smithsonian museum is 22,226 grid cells and the Longwood facility is 50,652 grid cells) are orders of magnitude larger than we could possibly hope to solve using exact POMDP solvers. We therefore use the AMDP algorithm to find motion plans that do model uncertainty, but do not suffer the intractability of exact POMDP planning.

Navigation Using the AMDP

We follow the steps of the algorithm in table 3.3 to generate the AMDP solution.

1. Generate the low-dimensional belief space $\tilde{B}$. Our belief space will be defined over the same state space as in equation (4.1). The current belief will be provided by the localisation algorithm, marginalised over Θ. As in equation (3.2), we represent the belief as

$$\tilde{b} = \langle \operatorname*{argmax}_{s \in \mathcal{S}} b(s); \overline{H}(b) \rangle \tag{4.6}$$

We assume a discretisation of the entropy $\overline{H}(b)$ into 100 discrete cells.

Recall that the AMDP algorithm requires a mechanism for recovering a full belief b from the low-dimensional representation $\tilde{b}$. For the navigation problem, we restrict the class of beliefs b to the class of the discrete approximation to a Gaussian with mean at the maximum-likelihood state from $\tilde{b}$, and a single variance such that the entropy of b matches the entropy of $\tilde{b}$. Recall also from the discussion in chapter 3 that the mean and variance using equation (3.20) can be directly computed from $\tilde{b} = \langle \operatorname{argmax}_{s \in \mathcal{S}} b(s); \overline{H}(b) \rangle$, and therefore a unique, high-dimensional b can be represented using $\tilde{b}$.

We will continue to use the action space described in figure 4.2(b).

2. Compute the low-dimensional reward function $\tilde{R}$. We use the reward function defined in equation (4.2). Using the algorithm in table 3.1, we generate $\tilde{R}$ from:

For each $s_i \in \mathcal{S}$, and $h_j \in [0, .01, \ldots, 1]$

1. Recover a Gaussian belief b with $(\mu = s_i, \sigma^2)$ such that $\overline{H}(b) = h_j$
2. Compute $\tilde{R}(\langle s_i, h_j \rangle) = \sum_{i=1}^{|\mathcal{S}|} R(s_i) b(s_i)$

Step 2 has no dependence on actions because the original reward function R has no dependence on actions.

3. Compute the low-dimensional transition function $\tilde{T}$. The algorithm given in table 3.2 for computing the low-dimensional transition function $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$ requires computing, for each $(\tilde{b}_j, a)$, the following quantities:

1. The full-dimensional b_i
2. For each observation z
3. $\quad$ b_j from the Bayes' filter equation (2.8).
4. $\quad$ $\tilde{b}_j$ from b_j

We can compute how the beliefs change with actions and observations from the localisation model; recall the prediction step from equation (4.4),

$$p(s^t|a^t, s^{t-1}) = \sum_{j=1}^{|\mathcal{S}|} p(s^t|s_j^{t-1}, a^t) p(s_j^{t-1}) \tag{4.7}$$

and a measurement step where z^t is given by the laser range finder

$$p(s^t|\, z^t) = \alpha p(z^t|\, s^t) p(s^t), \tag{4.8}$$

where α is a normalisation constant. $p(s_i|a^t, s^j)$ is again the motion model and $p(z^t|s^t)$ is the sensor model at s^t.

However, there are 5000^{360} possible sensor measurements.[2] Computing every b_j from (b_i, a) for each z leads to some computational intractability. Recall that

$$\begin{aligned} \tilde{T}(\tilde{b}_i, a, \tilde{b}_j) &= p(\tilde{b}_j|\tilde{b}_i, a) && (4.9) \\ &= \sum_{k=1}^{|\mathcal{Z}_{b_j}|} p(z_k|b_a) \text{ for } z_k \in \mathcal{Z}_{b_j}. && (4.10) \end{aligned}$$

Recall that b_a is the belief after acting, as defined in equation (3.7), and $\mathcal{Z}_{b_j}$ is the set of observations that cause the transition to b_j.

Since the majority of the transitions $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$ are extremely unlikely, most will have probability close to 0. We approximate $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$ by sampling from the observation likelihood function, $p(z|b_a)$:

$$p(z|b_a) = \sum_{i=1}^{|\mathcal{S}|} p(z|s_i) b_a(s_i). \tag{4.11}$$

[2]The sensor takes 360 such measurements in 1° increments around the robot, and each range measurement is 1 cm resolution from 1m to 50m, giving 5000 possible such range measurements.

This factored model allows us to first sample a specific s_i from the belief b_a and then sample an observation from sensor model $p(z|s_i)$.

In practice, and for results reported here, we use only 9 such observations, using the maximum likelihood observation from the each grid cell in the 3×3 grid centred around the maximum likelihood position of the robot: $\operatorname{argmax}_s b_(s)$. Sampling from different poses allows us to model the different posteriors that can result when the observation changes dramatically, such as occlusions changing, etc. Other sampling schemes are possible, such as sampling larger sets of observations for beliefs with higher entropy, however averaging over the 3×3 grid performed reasonably at capturing the different posteriors that can result of small displacements in the robot position. For each of the 9 sample observations, we compute a posterior belief $b_j = p(s|z, a)$, and use this to generate the transition probabilities, as in table 3.2. Nine samples may seem insufficient but this scheme works in practice. This algorithm is described in table 4.2.

```
 1. For each belief $\tilde{b}_i$ and action $a$:
 2.     For each $\tilde{b}_j : \tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = 0$
 3.     Recover $b_i$
 4.     Computer $b_a$ from equation (3.7)
 5.     For $k = 1 \ldots 9$
 6.         Sample a state $s_k$ from $b_a$.
 7.         Sample an observation $z_k$ from $p(z|s_k)$
 8.         Compute $b_j$ from the Bayes' filter, equation (2.8).
 9.         Compute $\tilde{b}_j$ from $b_j$
10.         Add $p(\tilde{b}_j|\tilde{b}_i, a)$ from equation (3.11) to $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$
```

Table 4.2. The modified algorithm for computing the low-dimensional transition function $\tilde{T}$ via sampling the observations.

4. Compute the value function for $\tilde{B}$. Once $\tilde{R}$ and $\tilde{T}$ have been computed, a value function can be computed as in 2.1.

Modelling Dynamic Obstacles

Recall that the AMDP for navigation was motivated by two problems: sparse environmental structure and crowdedness. The sensor model used by localisation (equation (4.5)) models the effect of sparse environmental structure; if the environment does not contain enough features for accurate localisation, the output of the Bayes' filter will be a probability distribution of high uncertainty. However, localisation as described in the previous section does not model dynamic obstacles or people: unexpectedly short range measurements that correspond to free space in the map receive low probability. We can modify this sensor model to account for the probability that a free-space range measurement is the result of an unmapped dynamic obstacle, such as a person.

Let us introduce a new parameter $\psi \in \Psi$ that describes a configuration of p people over the state space $\mathcal{S}$. We can compute the probability of an observation as

$$p(z|s) = \int_{\Psi} p(z|\psi, s)p(\psi|s)d\psi, \tag{4.12}$$

where $p(z|\psi, s)$ is the sensor model for a specific robot pose and configuration of people. $p(\psi|s)$ is uniform probability for all configurations ψ where no person is in the same state as the robot, and 0 probability for all configurations ψ where one of the p people is in the same state as the robot.

Unfortunately, we do not have an easy way of computing equation (4.12). Monte Carlo integration could be used by sampling from Ψ, but would potentially require an unacceptable number of samples. Instead, we compute an approximation to equation (4.12) by modifying our observation model.

Recall that each observation $z \in \mathcal{Z}$ is a set of range measurements; for a single laser range finder (Sick Optics, 2003), this is a set of 180 range measurement around half the circumference of the robot. Let us define m_i to be a single such range measurement, and z is then a total of $|m|$ measurements. At time t, the specific observation is $z^t = \{m_1^t, m_2^t, \ldots m_{|m|}^t\}$. The probability of z^t is the joint probability of all such measurements,

$$p(z^t|s^t) = p(m_1^t, m_2^t, \ldots m_{|m|}^t|s^t). \tag{4.13}$$

The approximation that we introduce in order to compute equation (4.12) is that the probability of each measurement m_i is independent of all other measurements,

$$p(z^t|s^t) = \prod_{i=1}^{|m|} p(m_i^t|s^t), \tag{4.14}$$

where $p(m_i^t|s^t)$ is again the sensor model from the physical characteristics of the sensor and the environment, just as in equation (4.5). The approximation we have introduced here is a strong one; consecutive range measurements are unlikely to be independent. Two consecutive range measurements are much more likely to be similar than they are to be different. However, given this approximation, we can write equation (4.12) as

$$
\begin{aligned}
p(z|s) &= \prod_{i=1}^{|m|} \int_{\Psi} p(m_i|\psi, s)p(\psi|s)d\psi && (4.15)\\
&= \prod_{i=1}^{|m|} \int_{\Psi} p(\psi|m_i, s)p(m_i|s)p(\psi|s)d\psi && (4.16)\\
&= \prod_{i=1}^{|m|} p(m_i|s) \int_{\Psi} p(\psi|m_i, s)p(\psi|s)d\psi. && (4.17)
\end{aligned}
$$

$p(m_i|s)$ is the sensor model without people (computed only from the map and robot pose s, as before). The integral $\int_{\Psi} p(\psi|m_i, s)p(\psi|s)d\psi$ gives the total probability of the configurations of people consistent with measurement m_i and robot pose s. When there are few configurations of people that are consistent with this observation then this term will have a value close to 0.

We can compute a solution for the integral by noticing that $p(\psi|m_i, s)p(\psi|s)$ is 0 for all configurations ψ such that there is a person "blocking" the range measurement m_i from s, as in figure 4.6.

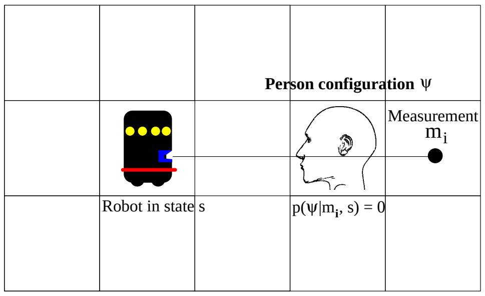


Figure 4.6. An example of a configuration ψ where $p(\psi|m_i, s) = 0$.

The integral in equation (4.17) will be 0 for all such configurations,[3] and reduces to a problem of estimating the total probability that no person blocks the measurement m_i. If we assume a fixed number of people p in the environment, then probability is $\frac{p}{|\mathcal{S}|}$ that any single grid cell (out of $|\mathcal{S}|$) is blocked by a person. If a measurement m passes through l free-space grid cells, then the total probability that none of these l grid cells is blocked is

$$\left(1 - \frac{p}{|\mathcal{S}|}\right)^l, \tag{4.18}$$

[3]The integral in equation (4.17) will be 0 for any configuration where the person blocks the observation m_i is assumes that the sensor never fails to observe a person. This is clearly an approximation that we use to make equation (4.15) tractable. The contribution to equation (4.17) will be very small for any realistic sensor and can be ignored.

which gives us the solution to the integral in equation (4.17),

$$\Rightarrow \int_{\Psi} p(\psi|m_i = l, s)p(\psi|s)d\psi \quad = \quad \left(1 - \frac{p}{|\mathcal{S}|}\right)^l \tag{4.19}$$

for p, the number of people in the environment, $|\mathcal{S}|$, the size of the state space, and l, the number of grid cells covered by measurement m_i. Figure 4.7 depicts equation (4.19) as l increases.

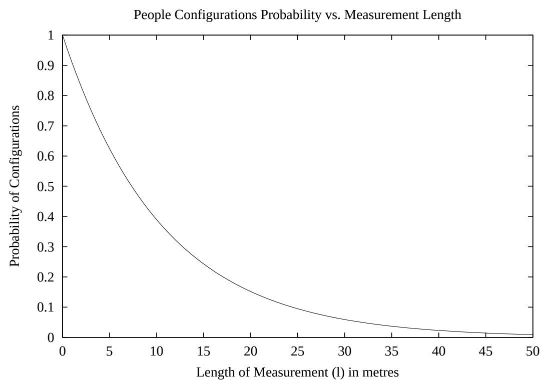


Figure 4.7. This graph shows the probability of configurations of people $\psi \in \Psi \backslash \Psi_{blocking}$, given measurement $m_i = l$, with increasing l. In this instance $p = 2000$ people and $|\mathcal{S}| = 22,226$.

We can substitute equation (4.19) back into equation (4.14) to compute a new sensor model,

$$p(z^t|s^t) = \prod_{i=1}^{|m|} p(m_i^t|s^t) \left(1 - \frac{p}{|\mathcal{S}|}\right)^{m_i^t}. \tag{4.20}$$

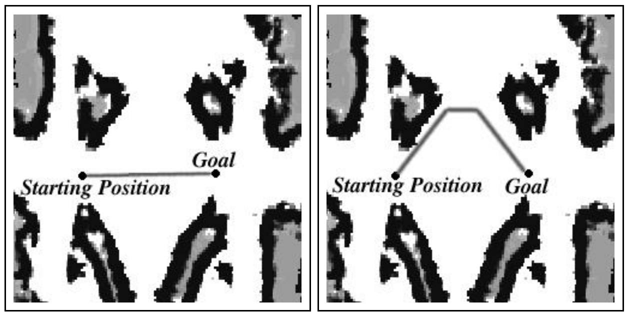
This leads to a further modified algorithm for computing $\tilde{T}$, given in table 4.3.

```
1.  For each belief $\tilde{b}_i$ and action $a$:
2.      For each $\tilde{b}_j : \tilde{T}(\tilde{b}_i, a, \tilde{b}_j) = 0$
3.      Recover $b_i$
4.      Computer $b_a$ from equation (3.7)
5.      For $k = 1 \ldots 9$
6.          Sample a state $s_k$ from $b_a$.
7.          Sample an observation $z_k$ from $p(z|s_k)$
8.          Compute $b_j$ from the Bayes' filter equation (2.8) using
                the sensor model $p(z_k|s_k)$ in equation (4.20)
9.          Compute $\tilde{b}_j$ from $b_j$
10.         Add $p(\tilde{b}_j|\tilde{b}_i, a)$ from equation (3.11) to $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$
```

Table 4.3. The modified algorithm for computing the low-dimensional transition function $\tilde{T}$ via sampling the observations and the unmapped dynamic obstacle sensor model.

Experimental Results

Smithsonian National Museum of American History. We first tested the Augmented MDP navigation in a simulated model of the Smithsonian National Museum of American History (Roy & Thrun, 1999). Figure 4.8 shows the effects of two different planners in the sample environment. Panel (a) shows the trajectory of a conventional, shortest path planner. Note that the robot moves directly toward the goal. Panel (b) shows the trajectory given by the AMDP planner. The black areas show obstacles and walls, and the dark grey areas show where no information is available to the sensors, because all environmental features are outside the range of the sensors. In this example the robot sacrifices a shorter path in return for maintaining higher positional knowledge throughout the trajectory. The higher positional knowledge comes from being closer to the walls of the display cases, increasing the ability to localise because more environmental structure is in view, and also because the likelihood that the information from the sensors is corrupted by crowds is reduced.

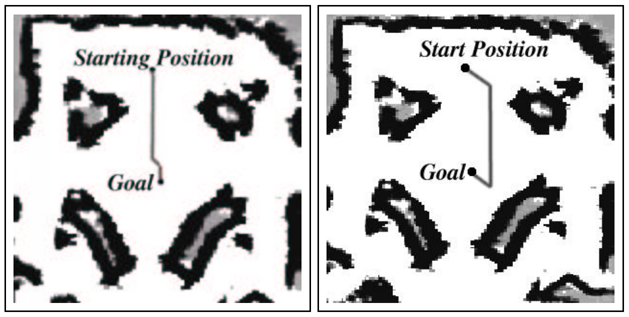


(a) Conventional trajectory (b) AMDP trajectory

Figure 4.8. An example of a conventional trajectory (a) and the AMDP trajectory (b) for the same start and goal point.

Figure 4.9 shows a different example in the Smithsonian museum model. Again in panel (a) is the conventional trajectory and panel (b) is the trajectory from the AMDP. This example illustrates the ability of the planner to take advantage of information gathering actions. The robot moves toward the goal, goes past it, relocalises once it is in sensor range of the obstacle, and then returns to the goal. These periodic relocalisations are essential for the robot to arrive at the goal with maximum reliability.

We performed quantitative analysis of the performance of the AMDP compared to the conventional planner, in simulation. In order to show that the AMDP really does improve navigation reliability, we decreased the maximum sensor range to different values, reducing the ability of the robot to gather information and localise accurately. As a measure of



(a) Conventional trajectory (b) AMDP trajectory

Figure 4.9. Another example of (a) a conventional trajectory and (b) the AMDP trajectory for the same start and goal point.

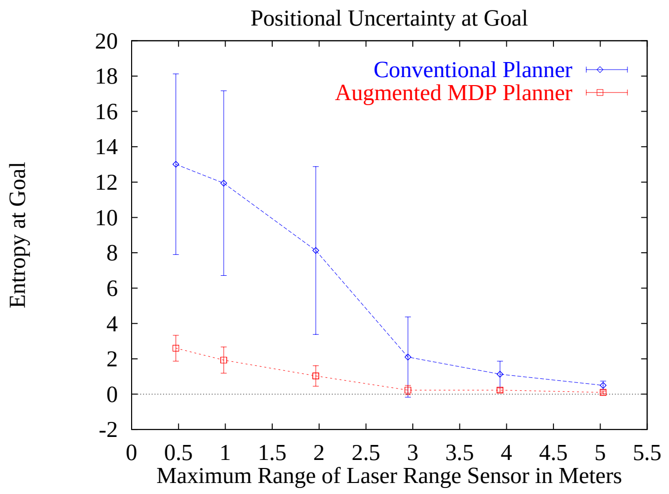
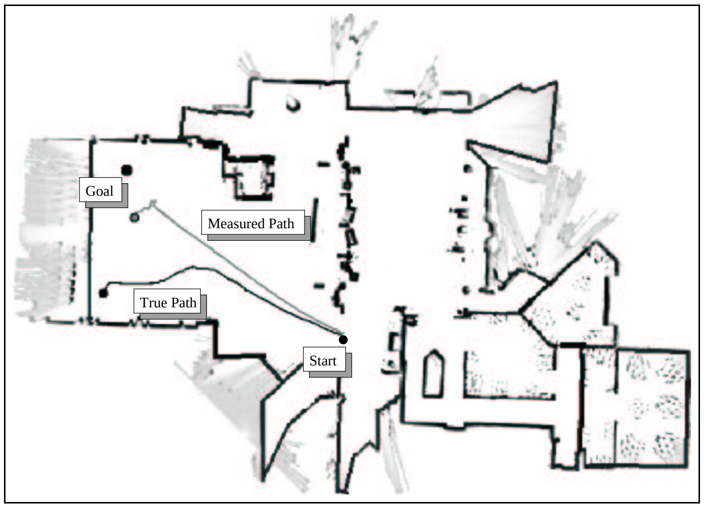


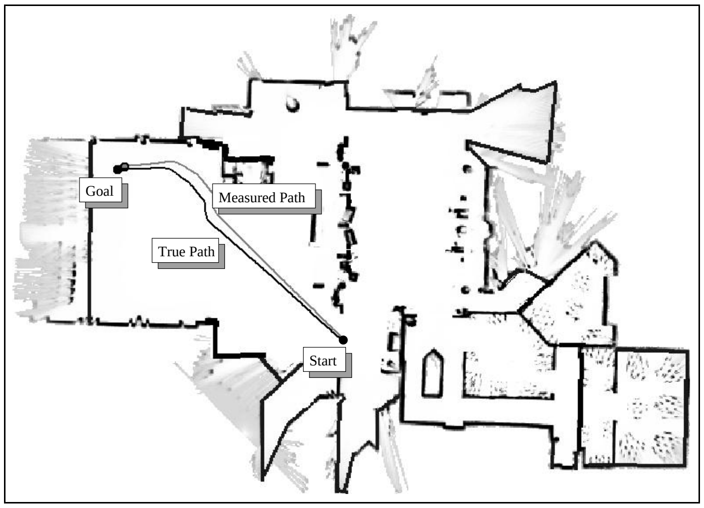
Figure 4.10. The positional certainty of the AMDP compared to the conventional motion planner, in the face of decreasing sensor ability. As the sensor becomes worse and worse (the left-hand side of the graph), the AMDP's performance does not degrade nearly as quickly as the conventional planner.

navigation reliability, we examined the entropy of the final belief state at the goal, over a series of simulated trajectories in the museum environment. A lower entropy of the belief state implies better positional knowledge and consequently more reliable goal attainment. Figure 4.10 depicts the average positional certainty at the goal of the AMDP compared to the conventional planner, as a function of maximum laser range, from 0.5m to 5m. The upper line is the performance of a conventional shortest-distance path planner, and the lower line is the AMDP planner. The AMDP planner has a lower uncertainty for all ranges of the laser sensor, and is substantially lower at shorter ranges, confirming that the AMDP has the most effect when the localisation is worst. Even better, the AMDP does not appear to degrade substantially when the localisation is at its worst.

Longwood at Oakmont retirement facility. In a different environment, we also examined the actual distance from the true robot position to the goal. Figure 4.11 depicts measured and actual trajectories for a robot in the Longwood environment. The conventional trajectory on the left clearly demonstrates the robot becoming lost; not shown is the probability distribution that reflects that the robot is lost.



(a) Conventional trajectory



(b) AMDP trajectory

Figure 4.11. An example of a conventional trajectory (a) and the AMDP trajectory (b) for the Longwood at Oakmont environment. The sensor range in this simulated experiment is $2m$, as in the experiment in figure 4.5.

Figure 4.12 shows the improved ability of the robot to arrive at the goal. When the laser range finder had a maximum range of $1m$, the conventional motion planner chose trajectories that left the true position of the robot an average distance of $7.6m$ from the goal. By comparison, the AMDP planner chose trajectories that left the robot on average $0.3m$ from the goal. When the maximum laser range measurement was increased to $10m$, both planners achieved the goal to within approximately $0.15cm$. In both cases, the motion planner was attempting to arrive at the goal with $0.3m$ accuracy. The AMDP planner was also able to reach the goal much more consistently, reflected by the much smaller variance of the goal accuracy.

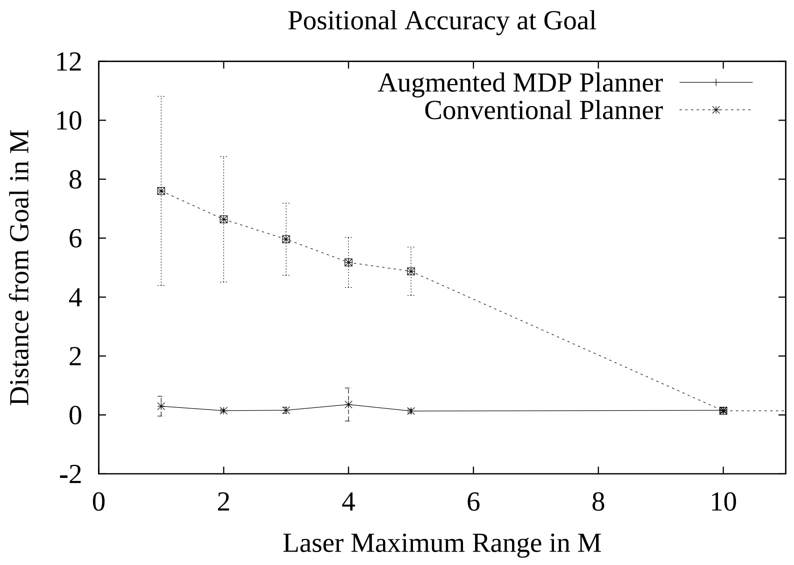


Figure 4.12. The performance of the AMDP compared to the conventional motion planner, in the face of decreasing sensor ability. As the sensor became worse and worse (the left-hand side of the graph), the ability of the AMDP to reach the goal accurately did not degrade nearly as quickly as the conventional planner.

4.3. Speech Dialogue Management

The development of automatic speech recognition has made possible more natural human-computer interaction, and more recently, rich interaction between humans and robots (Torrance, 1994; Westphal & Waibel, 1999; Denecke & Waibel, 1997). Speech recognition and speech understanding, however, are not yet at the point where a computer can reliably extract the intended meaning from every human utterance. Recent research in dialogue management has shown that Markov decision processes can be useful for generating policies (Young, 1990; Levin et al., 1998; Singh et al., 1999; Litman et al., 2000); the actions are the speech productions from the system, the state represents the progress of the dialogue as a whole, and the goal is to maximise some reward, such as fulfilling some request of the user.

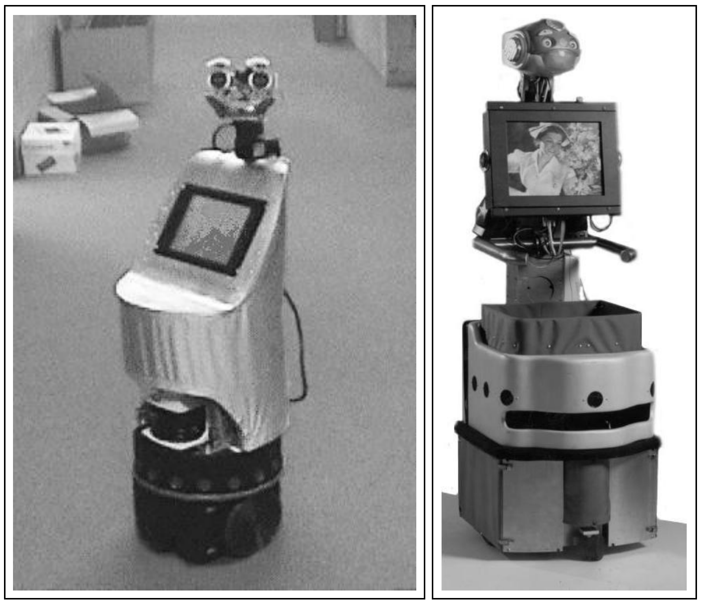


Figure 4.13. Florence Nightingale and Pearl, the nursebots used as robot platforms for the experiments in speech dialogue management.

The correct way to represent the state of the dialogue is still an open problem (Singh et al., 1999). One solution is to restrict the system to a single goal, for example booking a flight in an automated travel agent system; the system state is described in terms of how close the agent is to being able to book the flight. Such systems suffer from three principal problems. First, most existing dialogue systems are built for a single purpose over a single task domain, for example retrieving e-mail or making travel arrangements (Levin et al., 1998). While it is not impossible to extend these systems to multiple domains, most existing dialogue systems do not model confidences on the human utterances, and therefore do not account for the reliability of utterances while performing a strategy. Some systems do use the log-likelihood values for speech utterances, however, these values are only thresholded as to whether the utterance needs confirmation or not (Niimi & Kobayashi, 1996; Singh et al., 1999). Secondly, while system-initiated strategies perform best with these approaches, mixed-initiative strategies are a more natural model for human-robot interaction.

The real problem that lies at the heart of these three issues is that of observability. The ultimate goal of a dialogue system is to satisfy a user request; however, what the user wants is only partially observable at best. A conventional MDP demands to know the current state at all times, therefore the state has to be wholly contained in the system. System-initiated dialogues are successful because there is no uncertainty in determining when the user wants something, and a single task-domain system works well, because the relevant goals and actions are easily specified.

Dialogue Systems and POMDPs. We consider a POMDP to be a natural way of modelling dialogue processes when the state of the system is viewed as the state of the user (e.g., the user is asking a question, telling the robot to go somewhere, or is not talking to the system). This model inverts the conventional notion of state in a dialogue, in which the state of the dialogue measures the progress of the dialogue. The uncertainty model inherent in a POMDP policy allows the dialogue planner to recover from noisy or ambiguous utterances in a natural and autonomous way. At no time does the machine interpreter have any direct knowledge of the state of the user, i.e, what the user wants. The machine interpreter can only infer this state from the (noisy) speech of the user.

The particular domain that we use in the following discussion is based on a mobile robot, Florence Nightingale (Flo), developed as a prototype nursing home assistant. Flo uses the Sphinx II speech recognition system (Ravishankar, 1996), and the Festival speech synthesis system (Black et al., 1999). Figure 4.13 shows a picture of the robot.

Since the robot is a nursing home assistant, we use task domains that are relevant to everyday life. Table 4.4 shows a list of the task domains the user can ask about: the time, the weather, what is on different TV stations, and can also ask to deliver a video message (called *facemail*) to another person. These abilities have all been implemented on Flo, and the information about the weather and TV schedules is downloaded on request from the web.

Time
Weather (Current and for tomorrow)
TV Schedules for different channels (ABC, NBC, CBS)
Delivering video messages (facemail)

Table 4.4. The task domains for Flo.

From the domains of conversation, we extracted 17 states that correspond to possible intentions of the user. The state representation was compiled by hand, and does not necessarily correspond to the true state representation of all possible user intentions. A future experiment is to validate the state representation with a set of user experiments.

no_request	want_tv	want_facemail
request_begun	want_nbc	recording_facemail
request_done	want_abc	want_facemail_sent
want_time	want_cbs	want_facemail_mike_m
want_weather	want_current_weather	want_facemail_mike_v
	want_tomorrow_weather	want_facemail_sebastian

Table 4.5. The list of states used in the full decision problem in these experiments.

Again from the domains of conversation, we extracted 17 actions that correspond to the next action for each user state.

`do_nothing`	`ask_which_station`	`say_being_recording`
`say_hello`	`say_nbc`	`record_facemail`
`repeat`	`say_abc`	`ask_which_facemail`
`say_time`	`say_cbs`	`deliver_facemail_mike_m`
`ask_which_day`	`say_current_weather`	`deliver_facemail_mike_v`
	`say_tomorrow_weather`	`deliver_facemail_sebastian`

Table 4.6. The list of actions used in the full decision problem in these experiments.

If we translate these tasks into the MDP framework that we have described, the decision problem has 17 states, and the state transition graph is given in figure 4.14. The different tasks have varying levels of complexity, from simply saying the time, to delivering facemail, in which the user moves through four different states before the task is completed.

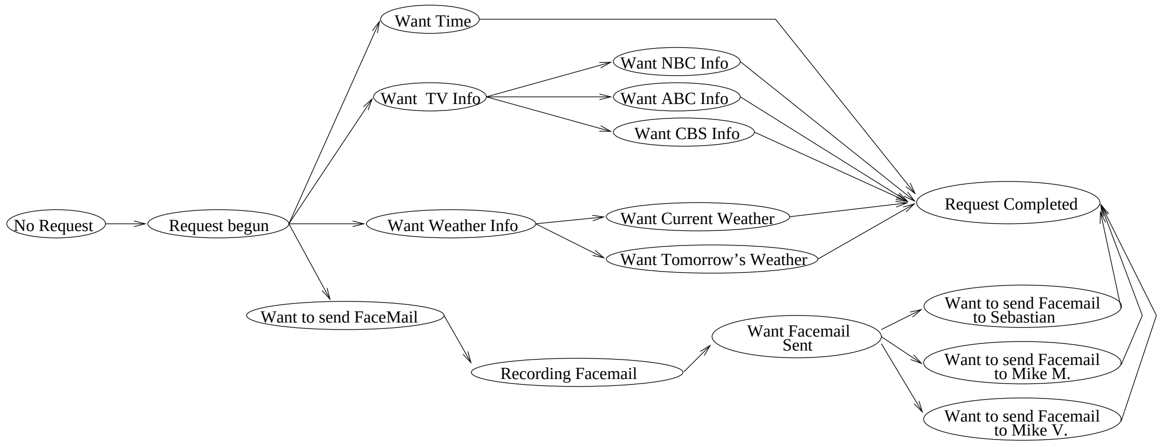


Figure 4.14. The 17-state MDP model, representing the kinds of requests Flo can handle, showing the high-probability transitions.

We compiled this into a POMDP model with 17 states, 17 actions and 18 keyword observations. We converted this POMDP into an AMDP using the equations given in section 3.1. Once again, we represent the belief $\tilde{b}$ as:

$$\tilde{b} = \langle \underset{s}{\operatorname{argmax}}\, b(s); \overline{H}(b) \rangle. \tag{4.21}$$

Belief updating is performed according to transition and emission functions, just as in equation (4.4) and equation (4.5). For robot navigation, the action and sensor models were learned empirically from actual data. In the spoken dialogue case, these models were hand-coded to reflect possible non-deterministic changes in the user's intentions; for example, even if the user asked initially for the time, the user may change their mind and ask for the weather even if the system complies with the initial request. The action model should in fact be learned or at least verified by a user study, but this is beyond the scope of this thesis. Similarly, we hand-coded the observation emission model that reflected the likely errors in the speech recognition system.

Finally, the reward is structured so that taking an incorrect action (such as `say_hello` when no greeting has been given, or `deliver_mail` when no mail has been given) had a reward of -10, but the reward for taking the correct action is +20. Some special cases exist: the reward for transitioning to the `request_done` state is +200, the reward for asking a question or confirming a response is -1, and the reward for trying to take a message inappropriately or delivering a message to the wrong person is -200. The reason for the additional negative reward in delivering to the wrong person is to highlight the difference between a costly action that is merely irritating (giving an inappropriate response) and an action that can be much more serious (travelling to the wrong destination).

Planning and Executing in the Speech Domain. Given the model as specified, we convert the POMDP into the AMDP as in table 3.3. We will not go through the details of converting the model here; the AMDP algorithm described in chapter 3, table 3.3 was implemented exactly as written. Because the set of observations is small, it is not necessary to sample from a distribution of observations as we did in the experiments using robot navigation. We compute the transition probabilities for all destination beliefs.

We made one change to the policy execution algorithm for dialogue management in the implementation of the Bayes' filter explained in chapter 2 during execution. We used the model's observation probability table $O(s, a, z)$ for planning, however we used a different probability model for observations during execution, in order to better reflect the recognition output. The speech recognition system Sphinx provided an estimate of the probability of each possible observation, $p_x^t(z)$, at each time t. We therefore modified the Bayes' filter equation (2.8) to be

$$b^t(s_i) = \alpha \sum_{j=1}^{|\mathcal{Z}|} p_x(z_j)\, O(s_i, a^t, z^t) \sum_{j=1}^{|\mathcal{S}|} T(s_j, a^t, s_i) p(s^{t-1} = s_j) \tag{4.22}$$

The estimate $p_x(z)$ does not reflect the model parameters, but is used to update the posterior belief of the system more accurately than the model parameters alone would. As a consequence, it may be that the policy computed from the model may only be optimal over the course of several dialogues (assuming that the planning model parameters $O(s, a, z)$ are the maximum-likelihood parameters over the course of several dialogues), but the policy may be sub-optimal over an arbitrary single dialogue. The best way to model changing observation probabilities is not obvious, but our experimental results support our approach.

Table 4.7 shows an example dialogue from the AMDP system. The left hand columns are the emitted observation and the entropy of the system that resulted. The most interesting lines are the fourth and fifth lines of the dialogue: although the true state of the user is `want_facemail`, the observation made is that the user is in state `want_time`. The conventional MDP would take the observation as given, but the observation from the speech recognition system was highly uncertain, so the entropy of the belief becomes quite high ($H(s) = 20$). The entropy stays high before getting a more certain observation. The system confirms the desired state, before moving on. We see a similar action a little later; here, because the cost of mis-delivering a facemail is very high, the system takes confirmation action even when the entropy is relatively low.

Observation	Entropy	Action	Reward	True State
`<silence>`	0	`do_nothing`	20	`none`
`flo hello`	2	`ask_repeat`	-1	`request_begun`
`flo hello`	0	`say_hello`	20	`request_begun`
`was on a`	20	`ask_repeat`	-1	`want_facemail`
`flo time is`	20	`ask_repeat`	-1	`want_facemail`
`flo i want to deliver please`	6	`confirm_facemail`	-1	`want_facemail`
`flo i want to deliver facemail`	0	`say_begin_recording`	20	`want_facemail`
`<recording_facemail>`	0	`record_facemail`	20	`recording_facemail`
`flo send it`	1	`ask_repeat`	-1	`want_facemail`
`flo sent it`	0	`ask_which_facemail`	20	`want_facemail`
`flo i want to send it to mike`	3	`ask_repeat`	-1	`want_facemail_mike_m`
`flo i want to send it to mike m`	0	`deliver_facemail_mike_m`	200	`want_facemail_mike_m`
`thanks flo goodbye`	0	`do_nothing`	20	`request_done`

Table 4.7. An example dialogue, from the Augmented MDP.

Simulated Experimental Results

We examined the performance of the AMDP system as compared to a full POMDP solution and an MDP dialogue manager. The full POMDP solver we used was provided by Cassandra and used the Incremental Improvement algorithm (Cassandra et al., 1997). The full POMDP solver was unable to find a policy for more than 7 states, so the first result compares the dialogue managers on a restricted domain of only 7 states. The experiments used synthetically generated observations based on the model parameters. The observations were generated according to the prior observation model, and fed to the dialogue manager.

Figure 4.15 demonstrates the need for a faster solution method for POMDP problems. The graph depicts number of states compared to solution time, for the Incremental Improvement full POMDP algorithm, the AMDP and the conventional MDP. The complexity of finding the optimal POMDP policy is not only a function of the number of states, but

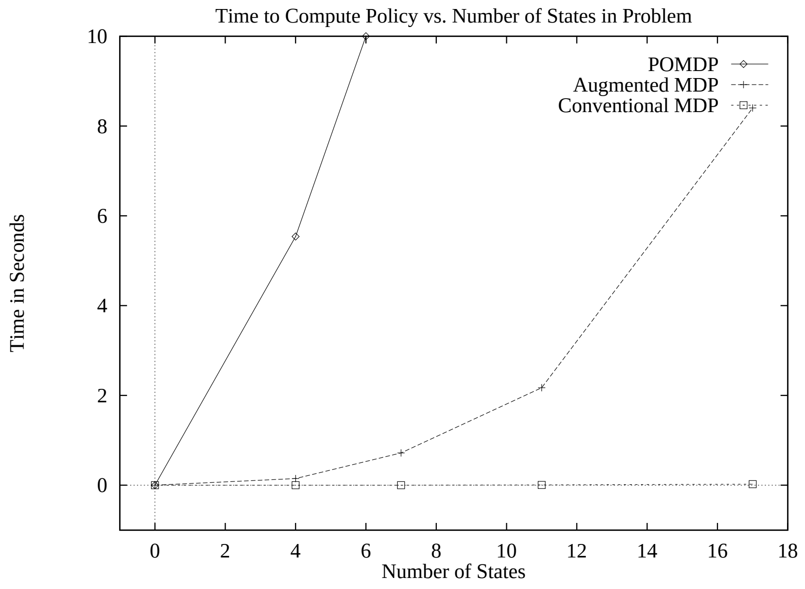


Figure 4.15. A comparison of the solution times for the full POMDP, AMDP and conventional MDP. The full POMDP is in fact exponential in growth as a function of the number of states.

this graph provides some rough intuition of the scalability of the three algorithms. The full POMDP could only be found for 7 states, taking 729 seconds on a 400 MHz Pentium II.

Figure 4.16 shows the performance of the three algorithms, POMDP, AMDP and conventional MDP over 100 dialogues in the 7 state problem. The graph depicts total accumulated reward against number of actions, and the salient result is the slope of each line; the faster the policy accumulates reward, the better the policy. This graph demonstrates that the full POMDP accumulated the reward fastest. This is to be expected, because the POMDP policy is the optimal policy.

The AMDP outperformed the MDP, but spent time asking confirmation questions in situations that the POMDP deemed unnecessary. Furthermore, the AMDP, while outperforming in total reward compared to the MDP solution, acquired the reward slower. The MDP solution appears competitive for two reasons; firstly, the problem domain is sufficiently small that the MDP did reasonably well (i.e., with few states, there is little room for confusion). Secondly, while the AMDP was not heavily penalised for asking confirmation questions, such actions did delay the accumulation of positive reward.

If we examine the accumulation of reward per dialogue as shown in figure 4.17 (as compared to reward per action as shown in figure 4.16), we see that the AMDP was closer to the optimal POMDP policy. The problem domain is still too constrained to demonstrate the failure of the MDP policy.

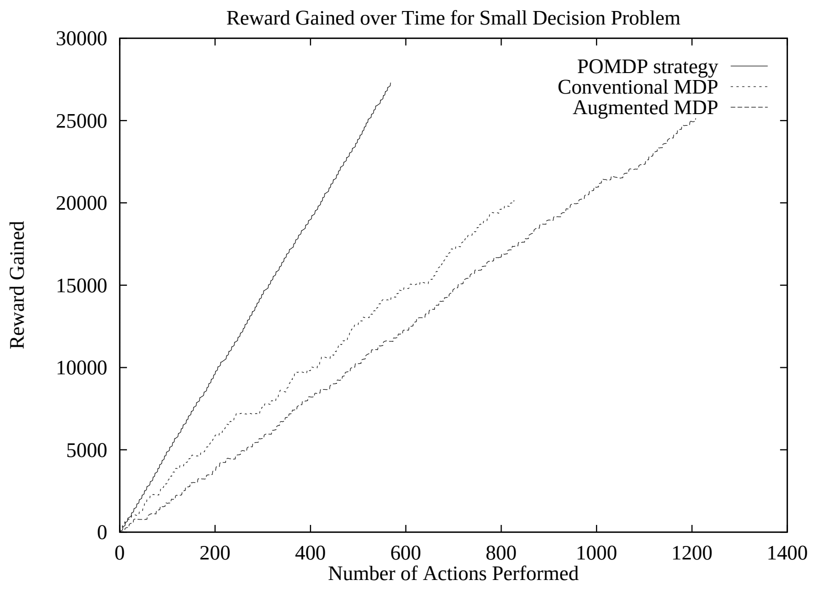


Figure 4.16. A comparison of the policies from the full POMDP solution, the AMDP and a conventional MDP solution, over a 7-state toy dialogue problem. The graph shows reward gained with each action. Note that while the full POMDP acquires the most reward fastest, the Augment MDP policy still acquires more reward than the conventional MDP.

Figure 4.18 shows the performance of the AMDP compared to the conventional MDP dialogue manager over the full 17 state problem. Because of the number of states, no POMDP solution could be computed for this problem. The AMDP clearly outperformed the conventional MDP strategy, as it more than tripled the total accumulated reward over the lifetime of the strategies, at the cost of taking longer to reach the goal state in each dialogue. Table 4.8 shows these results in more detail. The average reward for the AMDP is 18.6 per action, whereas the maximum reward for most actions is 20, suggesting that the AMDP took the right action about 95% of the time. Furthermore, the average reward per dialogue for the AMDP was 230 compared to 49.7 for the conventional MDP, which suggests that the conventional MDP was making a large number of mistakes in each dialogue.

Augmented MDP	
Average Reward	18.6 +/- 57.1
Average Dialogue Reward	230.7 +/- 77.4
Conventional MDP	
Average Reward	3.8 +/- 67.2
Average Dialogue Reward	49.7 +/- 193.7

Table 4.8. A comparison of the rewards accumulated for the two algorithms using the full model in simulation.

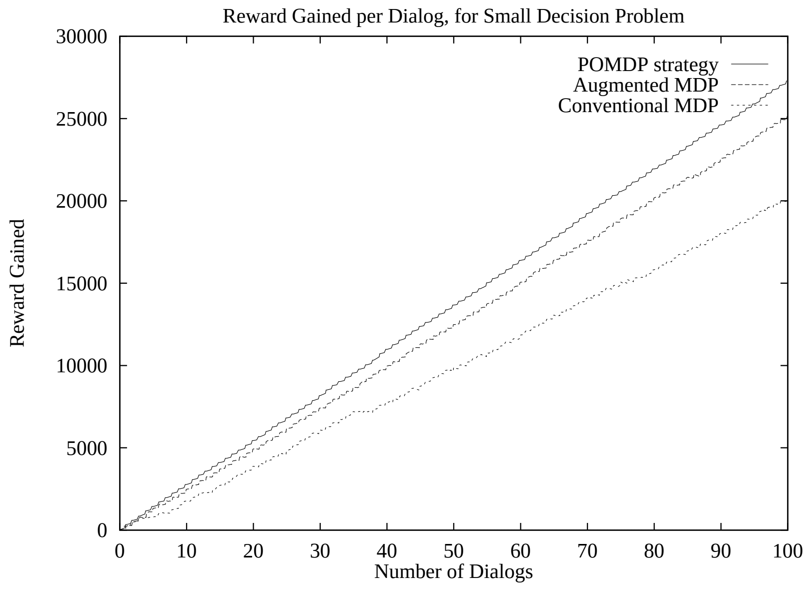


Figure 4.17. A comparison of the policies from the POMDP, AMDP and conventional MDP solutions over the 7-state toy problem. In this graph, the reward is per dialogue, to avoid penalising policies with confirmatory actions that delay the eventual reward.

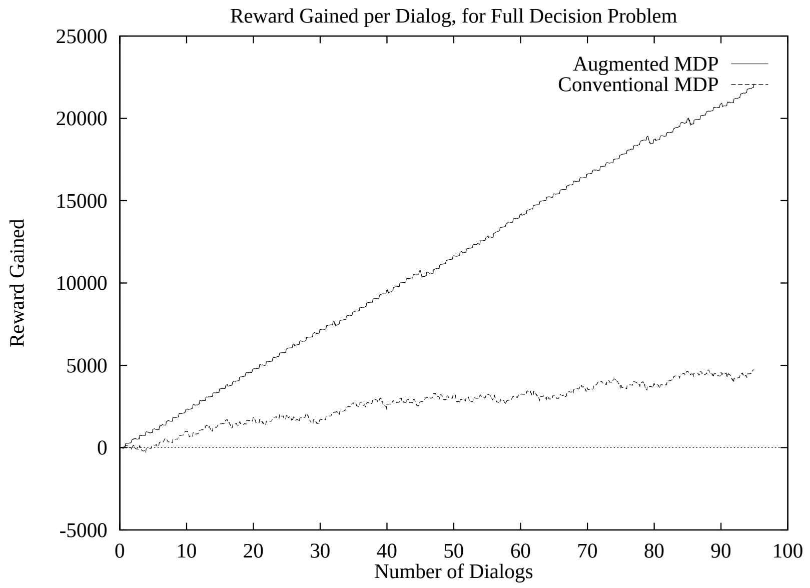


Figure 4.18. A comparison of the policies from the AMDP and conventional MDP solutions, over the full 17-state problem. Again, the graph depicts reward accumulated per dialogue, over the course of several dialogues.

Finally, the standard deviation for the AMDP was much narrower, suggesting that this algorithm was getting its rewards much more consistently than the conventional MDP.

Verification of Model on Users

We verified the utility of the POMDP by testing the model on actual human users. The experiment consisted of having users interact with the mobile robot under a variety of conditions. The users tested both the AMDP and an implementation of a conventional MDP dialogue manager. Both planners used exactly the same model. The users were presented first with one manager, and then the other, although they were not told which manager was first and the order varied from user to user randomly. The user labelled each action from the system as "Correct" (+100 reward), "OK" (-1 reward) and "Wrong" (-100 reward). The "OK" label was used for responses by the robot that were questions (i.e., did not satisfy the user request) but were relevant to the request, e.g., a confirmation of TV channel when a TV channel was requested.

		Augmented MDP	Conventional MDP
User 1	Reward Per Action	52.2	24.8
	Errors per request	0.1 +/- 0.09	0.55 +/- 0.44
	Time to fill request	1.9 +/- 0.47	2.0 +/- 1.51
User 2	Reward Per Action	36.95	6.19
	Errors per request	0.1 +/- 0.09	0.825 +/- 1.56
	Time to fill request	2.5 +/- 1.22	1.86 +/- 1.47
User 3	Reward Per Action	49.72	44.95
	Errors per request	0.18 +/- 0.15	0.36 +/- 0.37
	Time to fill request	1.63 +/- 1.15	1.42 +/- 0.63

Table 4.9. A comparison of the rewards accumulated for the two algorithms using the full model on real users, with results given as mean +/- std. dev.

For three different users, the system performed differently, compensating for the speech recognition accuracy which varied significantly between users. Notice that in user #2's case, the AMDP manager took longer to satisfy the requests, but in general gained more reward per action. This is because the speech recognition system generally had lower word-accuracy for this user, either because the user had unusual speech patterns, or because the acoustic signal was corrupted by background noise.

By comparison, user #3's results show that in the limit of good sensing, the AMDP policy approached the MDP policy. This user had a much higher recognition accuracy from the speech recogniser, and consequently both the augmented and conventional MDP acquire rewards at equivalent rates, and satisfied requests at similar rates.

4.4. Conclusion

In this chapter, we described two real world problems: mobile robot navigation, and spoken dialogue management. We showed how these problems can be solved using the

Augmented MDP algorithm of the previous chapter, and demonstrated measurably superior performance compared to existing algorithms for achieving reward in the face of perceptual uncertainty.

The Augmented MDP algorithm typically works well in domains where the optimal policy involves reducing uncertainty throughout the policy and as much as possible at the goal. Additionally, the AMDP representation is well suited for tasks where the reward corresponds to achieving a single goal. The AMDP representation however, will not perform well on tasks that involve trading off costs for resolving uncertainty for different state features. For example, in the dialogue domain, if the robot were attempting to satisfy user requests for multiple users simultaneously, this might involve asking clarification questions of only some users, or taking actions that could disambiguate the requests of multiple users simultaneously. The AMDP representation is not likely to perform well in such tasks that involve prioritising goals. However, the AMDP representation is still suprisingly effective at a number of problems that do not raise these issues.

We will see in the following chapters, however, that the underlying representation of the AMDP of $\langle \mathrm{argmax}_{s \in \mathcal{S}} b(s); \overline{H}(b) \rangle$ is still not sufficient for some problems. The rest of this thesis is concerned with how to find better belief representations. Chapter 5 discusses how to automatically discover good low-dimensional representations of beliefs, not restricted just to the statistics of maximum likelihood state and entropy. Chapter 6 describes better function approximation methods for solving the belief state MDPs.

CHAPTER 5

Dimensionality Reduction on Belief Spaces

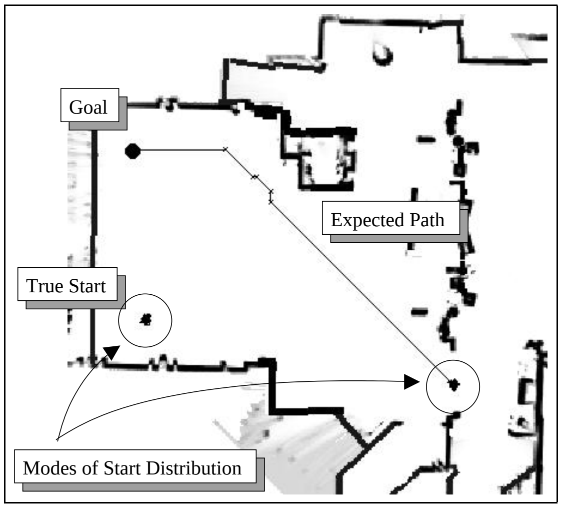
Have you ever really thought about your beliefs? Are your beliefs working for you? If you install the belief system of someone who is confused about life, you'll produce similar results in your life.

– Marnie Pehrson
"10 Steps to Fulfilling Your Divine Destiny"

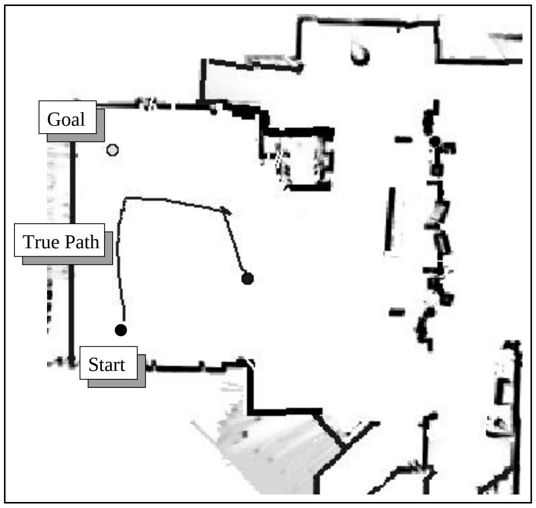
THE question addressed in this chapter is how to automatically find a more appropriate low-dimensional representation $\tilde{B}$ of the full belief space B for problems where the Augmented MDP statistics are not sufficient. If we can find different low-dimensional representations, we should be able to compute value functions over these different belief spaces as we did for the AMDP. This chapter will be concerned exclusively with the problem of finding appropriate representations; we will describe how to plan with these representations in subsequent chapters.

In chapter 4, we demonstrated the AMDP on real-world problems. In these examples, the AMDP out-performed conventional algorithms because the controller was able to predict and avoid situations where the state estimate can become too uncertain for good control. However, figure 5.1 demonstrates that there are real world problems where this representation is not sufficient for finding good policies.

In figure 5.1(a) we see a simulated navigation problem where the robot is to arrive at the goal in the top left. The true initial pose of the robot is not known, and unlike the examples we saw in chapter 4, the initial distribution of the robot is bi-modal: the robot could be in one of two widely-separated locations. If we compare this image with figure 4.9 in chapter 4, the expected policy in figure 5.1(a) matches the executed policy from figure 4.9(b), because the initial belief in both cases has the same maximum-likelihood state and entropy. The simulated laser range finder in this example had a maximum range of $2m$.



(a) The expected trajectory from the AMDP



(b) The actual (failed) trajectory

Figure 5.1. On the left is the expected trajectory from the Augmented MDP, where the initial distribution has the same statistics as in figure 4.11, but is bi-modal with widely separated modes. On the right is the actual, failed trajectory. The solid black line is the true (unobserved) trajectory of the robot.

The executed policy in figure 5.1(b) performs badly; we see that the true path of the robot (even in the simulation) does not match the expected trajectory and is clearly sub-optimal. The maximum-likelihood state was very unstable, essentially "flickering" between the two modes as the belief changed with each action and observation.

The principal shortcoming lies in the choice of low-dimensional belief features. The AMDP relies on the fact that, for many real-world problems, the particular shape of the belief is not relevant for control. The AMDP will find good policies only where the low-dimensional representation of belief can distinguish between two beliefs that have different actions in the optimal POMDP.

5.1. Dimensionality Reduction

"Dimensionality reduction" is the search for a projection from a high-dimensional space containing data, to some low-dimensional compact representation of the data space (Cox & Cox, 1994). Ideally, dimensionality reduction involves no information loss—all aspects of the data can be recovered equally from the low-dimensional representation as the high-dimensional one. However, dimensionality reduction is usually performed with respect to some loss function (Collins et al., 2002), which penalises the loss of some information in the data (e.g., some features of the data) more than the loss of other information (or features). We will see that finding a low-dimensional representation of probability distributions will require a specific loss function to preserve the important aspects of the data using as few dimensions as possible.

If the data are assumed to lie on some low-dimensional manifold embedded in the high-dimensional space, then dimensionality reduction can be viewed as the search for that low-dimensional manifold (Roweis & Saul, 2000; Tenenbaum et al., 2000). If we consider the evolution of beliefs from a POMDP as a trajectory inside the belief space, then our assumption is that trajectories for most large, real world POMDPs lie on low-dimensional manifolds embedded in the belief spaces.

Our goal now is to use dimensionality reduction to find the low-dimensional manifolds that represent, in some sense, the "true" belief spaces of real world POMDPs, and plan just on those implicit manifolds instead of over the explicit belief spaces. If the compact manifolds are sufficiently low-dimensional, then it should be computationally tractable to find value functions over the space of beliefs that we care about.

The general class of dimensionality reduction algorithms is a search for a low-dimensional manifold representing the data. Different algorithms search for a manifold that preserves some property, such as variance (Principal Components Analysis, Roweis & Saul (2000)), local neighbourhood structure (Locally-Linear Embedding, Roweis & Saul (2000)), or global distance metrics (Isomap, Tenenbaum et al. (2000)). We would ideally like to find a low-dimensional manifold that automatically discovers the features of the beliefs that are sufficient for good control. In this chapter, we will make the approximation that belief reconstruction is the property we wish to preserve, instead of *control* accuracy. We want to differentiate between all beliefs, not just between those with different actions under the optimal policy, such as in the work Poupart & Boutilier (2002). Ours is an approximation in that if we choose too few dimensions, we may not choose the best dimensions for good control. However, our approach finds good representations with many fewer dimensions than the existing approaches based on control accuracy, while preserving the ability to make good control decisions.

POMDP Dimensionality Reduction. We will assume that we have a sample set of data from the high-dimensional belief space, instead of compressing the model parameters directly. While there are techniques to compress models (Poupart & Boutilier, 2002) using linear projections, we will realise better compression using non-linear approaches based on data. One additional advantage to using sampled belief sets is that belief tracking is substantially easier than planning. We may wish to dispense with the high-dimensional model completely for planning, and only use it to collect beliefs. This also raises the question of *how* to collect these beliefs—beliefs are generated as a consequence of actions, and the space they describe will be dependent on the actions used to generate the beliefs. If we are particularly naïve and unlucky, we might generate beliefs that describe a manifold that would never be encountered during execution of the optimal policy. We will ignore this issue temporarily, and revisit it in later chapters.

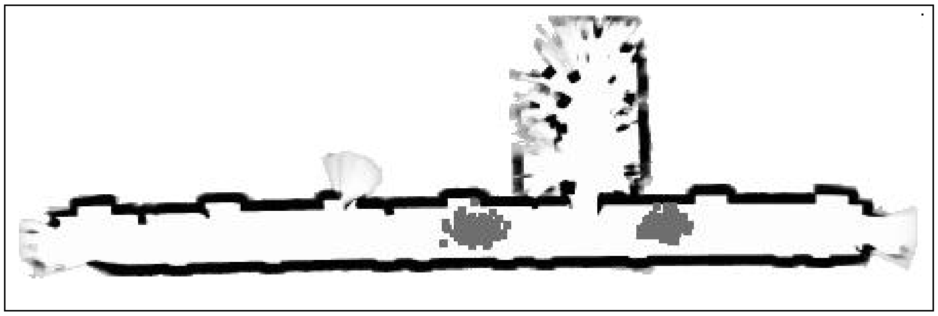


Figure 5.2. An example of a sparse belief in mobile robot navigation. Although the distribution is multi-modal, it is still relatively compact: the majority of the states contain no particles and therefore have zero probability.

Additionally, many of the real POMDPs that we will discuss have the property that the belief probability distributions themselves are sparse; that is, the probability of most states in the world is near zero for most beliefs. Intuitively, mobile robots and other real world systems have local uncertainty (which can often be multi-modal), but rarely encounter global uncertainty with any kind of regular structure, e.g., figure 3.2). Figure 5.2 depicts a mobile robot travelling down a corridor, and illustrates the sparsity of the belief space. Although the distribution is multi-modal, each mode is compact and most positions of the robot have zero probability.

Exponential Family PCA. We will take advantage of these characteristics of POMDP beliefs by using a variant of a common dimensionality reduction technique, Principal Components Analysis (PCA) (Joliffe, 1986). PCA is well-suited to dimensionality reduction where the data lie near a linear manifold in the higher-dimensional space (Joliffe, 1986; Roweis & Saul, 2000; Tenenbaum et al., 2000). We will see that POMDP belief manifolds are often poorly approximated by linear manifolds; in particular, sparse beliefs are usually very non-linear. We will show how to modify PCA by employing a different loss function; we choose from a class of loss functions that allow us to rephrase the dimensionality reduction problem as a convex optimisation problem in terms of "generalised Bregman divergences" (Bregman, 1967). This algorithm can also be viewed as transforming the data into a space where it does lie near a linear manifold; the algorithm which does so (while also correctly handling the transformed residual errors) is called Exponential family PCA (E-PCA) (Collins et al., 2002). E-PCA will allow us to find the implicit manifolds with only a handful of dimensions, even for belief spaces with thousands of explicit dimensions.

Our algorithm begins with a sampled set of beliefs from a POMDP. It uses these beliefs to find a representation of belief space using a small number of belief features. Finally, it plans over the low-dimensional space by discretising the features and using standard value iteration to find a policy over the discrete beliefs as in chapter 3. Again, in this chapter we will only discuss how to find the belief features; we will describe how to plan with these features in the next chapter.

Although the specific form of dimensionality reduction presented in this chapter was developed elsewhere (Collins et al., 2002), the use of sampled beliefs for POMDP compression and the use of the Exponential family PCA are novel techniques, developed first in Roy & Gordon (2003).

Notation. In this chapter, we will add one piece of notation: throughout the process of dimensionality reduction, we will refer to a set of sample beliefs as $X = \{b_1, b_2, \dots, b_{|X|}\}$. We will slightly abuse the notation in that X is also the matrix of column vectors $X = \left[b_1 \mid \dots \mid b_{|X|}\right]$. The low-dimensional representation of this set of samples will therefore be $\tilde{X}$. $\tilde{X}$ will also be the matrix of column vectors $\left[\tilde{b}_1 \mid \dots \mid \tilde{b}_{|X|}\right]$. These terms should not be confused with the full high-dimensional belief space B or the low-dimensional manifold $\tilde{B}$ that we would like to find.

5.2. Principal Components Analysis

One of the most common forms of dimensionality reduction is Principal Components Analysis (Joliffe, 1986). Given a set of data, PCA finds the most compact representation of the data such that the variance of the reconstructed data is mostly preserved. A transformation that preserves variance is appealing because it will also preserve our ability to distinguish between beliefs that are far apart in Euclidean norm, in order to avoid mistakes in choosing control actions.

The PCA transformation can be represented as a factorisation of the belief samples X into component matrices U and $\tilde{X}$,

$$X = U\tilde{X}^T. \tag{5.1}$$

In equation (5.2), X is a matrix of column vectors $b_1, \dots, b_{|X|}$, where each column vector b_i of length $|\mathcal{S}|$ is the ith sample of data. U corresponds to the set of bases that span the projection space of l dimensions, where $l < |\mathcal{S}|$. U is a matrix of l column vectors $U = [u_1, \dots, u_l]$, where each column is a vector of length $|\mathcal{S}|$. $\tilde{X}$ represents the parameterisation of the data in the low-dimensional space, and is a matrix of $|X|$ column vectors $\tilde{X} = \left[\tilde{b}_1, \dots, \tilde{b}_{|X|}\right]$ of length l. The size of l represents the size of the low-dimensional space, which is less than $|\mathcal{S}|$, the size of the original space of the data.

From a geometric perspective, U comprises a set of bases that span a hyperplane in the high-dimensional space of X; $\tilde{X}$ are the co-ordinates of the data on that hyperplane. If no hyperplane exists that contains the data exactly, PCA will find the surface of a given dimensionality l that best approximates the variance of the data. Minimising the error in variance between the original data X and the low-dimensional representation $\tilde{X}$ is equivalent to minimising the sum of squared error loss:

$$L(X, U, \tilde{X}) = ||X - U\tilde{X}^T||^2. \tag{5.2}$$

PCA typically finds an orthonormal basis matrix U; we are ignoring this feature here. We can therefore collect a set of sample beliefs $X = \{b_1, \dots, b_{|X|}\}$, assemble them into the data matrix X, apply PCA to compute $\tilde{X}$ and use U as our low-dimensional belief space $\tilde{B}$. If PCA can find a low-dimensional representation of the data with little loss, then we can readily use value iteration over this surface and should have no difficulty in distinguishing between beliefs for making good control decisions.

Problems with PCA

Unfortunately, Principal Components Analysis performs poorly at representing probability distributions. In particular, many of the real world problems we wish to solve are sparse. That is, the beliefs are characterised by large regions of low or zero probability, with compact modes of probability moving around in these regions. The beliefs will not vary principally along a single direction in the belief space, but will vary a small amount at a time along a few axes at a time. PCA will therefore require many bases to capture the variance well enough to make control decisions.

Figure 5.3 shows a toy problem that we can use to evaluate the success of PCA at finding low-dimensional representations. The abstract model has a two-dimensional state space: one dimension of position along a circular corridor, and one binary orientation. States $s_1 \dots s_{100}$ inclusive correspond to one orientation, and states $s_{101} \dots s_{200}$ correspond to the other. The reward is at a known position that is different in each corridor; therefore, the agent needs to discover its orientation, move to the appropriate position, and declare it has arrived at the goal. When the goal is declared the system resets (regardless of whether the agent is actually at the goal). The agent has 4 actions: `left`, `right`, `sense_orientation`, and `declare_goal`. The observation and transition probabilities are given by von Mises distributions (Mardia & Jupp, 2000), an exponential family distribution defined over $[-\pi : \pi)$. The von Mises distribution is a "wrapped" analog of a Gaussian; it accounts for the fact that the two ends of the corridor are connected, and because the sum of two von Mises variates is another von Mises variate, we can guarantees that the true belief distribution is always a von Mises distribution over the corridor for each orientation.

This instance of the problem consists of 200 states, with 4 actions and 102 observations. Actions 1 and 2 move the controller left and right (with some von Mises noise) and action 3 returns an observation that uniquely and correctly identifies which half of the maze the agent is in (the top half or the bottom half). Observations returned after actions 1 and 2 identify the current state modulo 100. The probability of each observation is a von Mises distribution with mean of the true state (modulo 100). That is, these observations indicate (approximately) where the agent is horizontally.

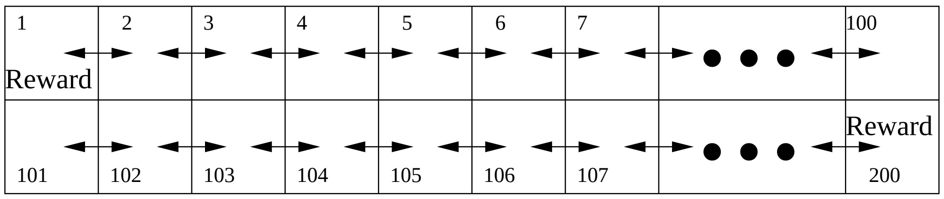


Figure 5.3. The toy maze of 200 states.

This maze is interesting because it is relatively large by POMDP standards (200 states) and contains a particular kind of uncertainty—the agent *must* use action 3 at some point to uniquely identify which half of the maze it is in (recall that the state observations are modulo 100—the state observations contain no information about which corridor the agent is in). The problem is too large to be solved by conventional POMDP value iteration, but structured such that heuristic policies will perform poorly, including the AMDP.

Figure 5.4 shows 4 sample beliefs from this problem, from a total data set of 500. Notice that each belief is essentially two von Mises distributions, a distribution over each half of the maze.

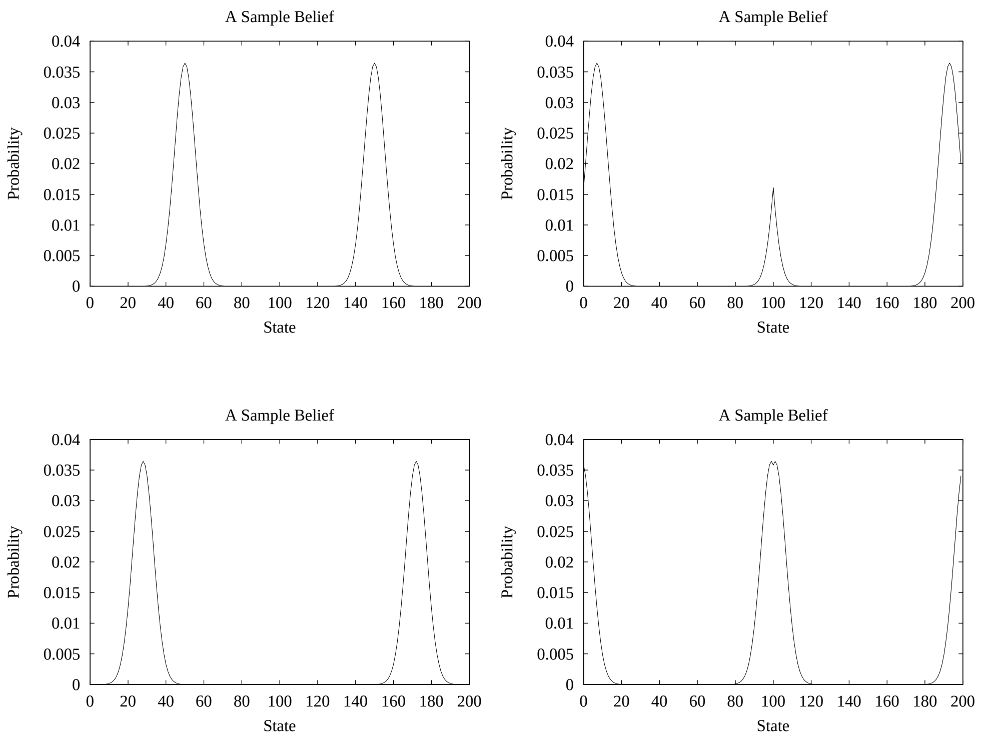


Figure 5.4. Sample beliefs for the toy problem, from a sample set of 500.

Figure 5.5 examines the performance of PCA on representing these beliefs by computing the average error between the original beliefs X and their reconstructions $U\tilde{X}$.[1] In figure 5.5(a) on the left we see the average squared error (L_2) compared to the number of bases, and in figure 5.5(b) on the right, we see the average Kullbach-Leibler (KL) divergence. The KL divergence between a belief b and its reconstruction $r = U\tilde{b}$ from the low-dimensional representation $\tilde{b}$ is given by

$$KL(b \,\|\, r) = \sum_{i=1}^{|\mathcal{S}|} b(s_i) \ln \left(\frac{b(s_i)}{r(s_i)} \right) \tag{5.3}$$

Minimising the L_2 error is the explicit objective of PCA, but comparing it to the KL divergence can be informative, in that the KL divergence will be more sensitive to errors in the high-probability events, and more sensitive to under-predictions than over-predictions.

Despite the fact that probability distributions in the collected data set have only 3 degrees of freedom, the reconstruction error remains relatively high until somewhere between 10 and 15 degrees. If we examine the reconstruction of a sample belief, we can see

[1]We use a popular implementation of PCA based on the Golub-Reinsche algorithm (Golub & Reinsch, 1970) available through the GNU Scientific Library (Galassi et al., 2002).

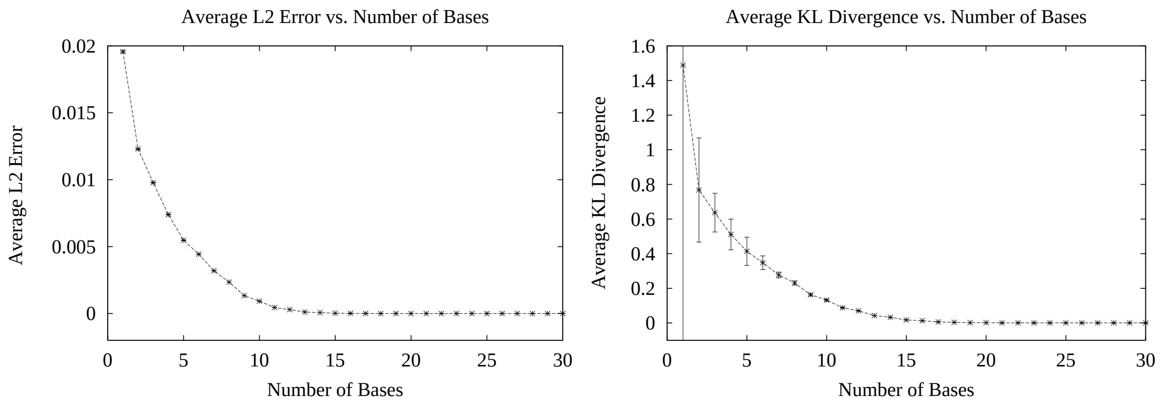


Figure 5.5. The average error between the original sample set X and the reconstructions $U\tilde{X}$. (a) L_2 error, explicitly minimised by PCA, and (b) the KL divergence.

the kinds of errors that PCA is making. Figure 5.6(a) shows the original sample belief (the solid line) and its reconstruction (the dotted line). Notice that the reconstructed belief has some strange artifacts: it contains ringing (multiple small modes), and also is negative in some regions. PCA is a purely geometric process; it has no notion of the original data as probability distributions, and is therefore free to generate representations of the data that are non-negative. PCA representations are not constrained to be positive, or sum to 1. Figure 5.6(b) shows the same example belief and reconstruction, where the reconstruction is corrected to be a true probability distribution by shifting the distribution to be above 0 and then re-normalising.

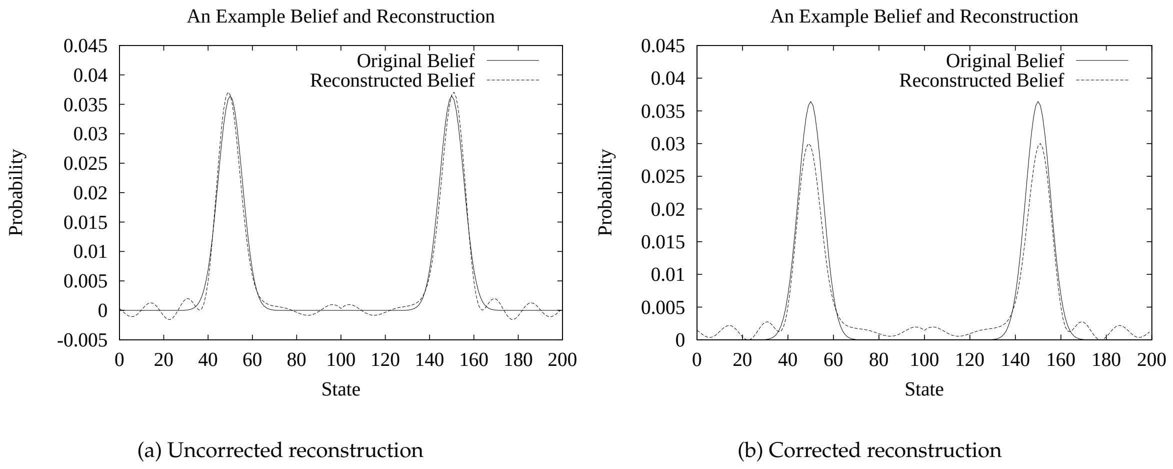


Figure 5.6. An example belief and its reconstruction, using 10 bases. (a) On the left, the uncorrected reconstruction. (b) On the right, the reconstruction is scaled and re-normalised to be a true probability distribution.

Comparing figures 5.6(a) and 5.6(b), we see that the correction introduces substantial errors. This illustrates the fact that PCA will give invalid probability distributions, and furthermore that the correction process introduces representational errors.

Notice also that the PCA process is making the most significant errors in the low-probability regions. This is particularly unfortunate because, as discussed earlier, real-world probability distributions tend to be characterised by compact masses of probability, surrounded by large regions of zero probability. What we would therefore like to do is to find a form of PCA that constrains the data to be non-negative, and is also better at representing sparse probability distributions—improving the existing errors made on low-probability events.

5.3. Exponential family PCA

The question to be answered is what loss functions are available instead of the sum of squared errors in order to generalise PCA. We would like to choose a loss function that better reflects the need to represent probability distributions. Unfortunately, we cannot just choose arbitrary loss functions because only certain loss functions will continue to allow us to factor the data.

Collins et al. (2002) provide the following alternate description of Principal Components Analysis:

> *PCA also has another convenient interpretation that is perhaps less well known. In this probabilistic interpretation, each point $\mathbf{x}_i$ is thought of as a random draw from some unknown distribution P_{θ_i}, where P_θ denotes a unit Gaussian with mean $\theta \in \mathbb{R}^D$. The purpose then of PCA is to find the set of parameters $\theta_1, \ldots, \theta_n$ that maximises the likelihood of the data, subject to the condition that these parameters all lie in a low-dimensional subspace. In other words, $\mathbf{x}_1, \ldots \mathbf{x}_n$ are considered to be noise-corrupted versions of some true points $\theta_1, \ldots, \theta_n$ which lie in a subspace; the goal is to find these true points, and the main assumption is that the noise is Gaussian.*

From this interpretation of PCA, we can see that minimising squared error is equivalent to minimising the negative log-likelihood of the data under the Gaussian distribution. The derivation of this due to Collins et al. (2002) is given in appendix A.1. Collins et al. (2002) provide the insight that the log-likelihood of *any* exponential family distribution (such as the normal distribution) parameterised by $\theta = U\tilde{b}$ can also be written as generalised Bregman divergence (defined below),

$$\begin{aligned} \underset{\tilde{b}}{\operatorname{argmin}} -\log p(b \mid U\tilde{b}) &= \underset{\tilde{b}}{\operatorname{argmin}}\, B_F(g(U\tilde{b}) \,\|\, \tilde{b}) && (5.4) \\ &= \underset{\tilde{b}}{\operatorname{argmin}} \left(F(U\tilde{b}) - b \cdot U\tilde{b}\right), && (5.5) \end{aligned}$$

where g and F will be explained in the next section. Equation (5.5) follows from the definition of the generalised Bregman divergence (Gordon, 1999), eliminating terms that do not depend on $U\tilde{b}$. The full details of how to arrive at equation (5.5) are provided in appendix A.2.

Without yet understanding what g and F are, we can see that by writing the dimensionality reduction problem as the minimisation of generalised Bregman divergence, we have turned it into a convex optimisation problem, of the type well studied in the optimisation literature (Rockafellar, 1970). Collins et al. (2002) call this generalisation of PCA "Exponential family PCA" (E-PCA).

We next define the "generalised Bregman divergence," and discuss why this quantity represents an appropriate way to view the dimensionality reduction problem.

Generalised Bregman Divergences

The Bregman divergence (Bregman, 1967)

$$B_F(p \,\|\, q) = F(p) - pq + F^*(q) \tag{5.6}$$

is a distance between two points (p, q) on a convex function F in terms of tangents to that convex function, as shown in figure 5.7 (Gordon, 1999). The generalised Bregman divergence uses the tangent given by the convex dual of F, $G = F^*$ (Gordon, 1999), as opposed to the first-order Taylor expansion. We can also think of the Bregman divergences as measuring a distance between projections into some space, and our parameters $U\tilde{b}$ representing the belief b projected into that space. $G' = g(U\tilde{b})$ is then the back-projection from the low-dimensional parameters into the original data space. We therefore call the derivative f of the convex function F the "link function," as in a non-linear link between spaces.

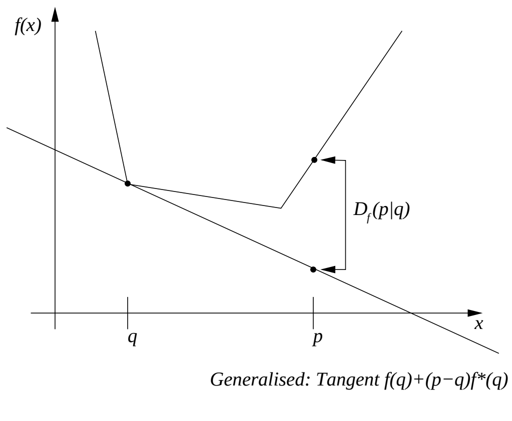


Figure 5.7. The Bregman divergence $B_F(p \,\|\, q)$ between p and q with respect to function F. The generalised Bregman divergence uses the convex dual F^* of F to compute the tangent.

The generalised Bregman divergence measures how close b and $g(U\tilde{b})$ are with respect to how fast the link function f changes according to f^*. Where the link function f changes rapidly, then b and $g(U\tilde{b})$ must be very close together for the approximation at b to be good.

The advantage of writing the loss between b and $U\tilde{b}$ as a generalised Bregman divergence is that the loss is therefore convex and the gradient in U or $\tilde{b}$ separately is well-defined. We can take advantage of convex optimisation techniques to find globally unique solutions that minimise the loss between b and $U\tilde{b}$, the expected value of b under the data likelihood function. Note that we can do this for arbitrary data likelihood functions, including the Gaussian model in conventional PCA. We can re-write PCA as the minimisation of a generalised Bregman divergence using

$$F(b) = \frac{\|b^2\|}{2} \tag{5.7}$$

$$f = b \tag{5.8}$$

$$G = F \tag{5.9}$$

$$g(U\tilde{b}) \quad = \quad U\tilde{b} \tag{5.10}$$

$$B_F(g(U\tilde{b}) \,\|\, b) \quad = \quad \frac{\left(U\tilde{b}\right)^2}{2} - b \cdot (U\tilde{b}). \tag{5.11}$$

E-PCA allows us to use a range of different loss functions, each of which leads to a different link function in analogy to generalised linear models (Gordon, 2003). By casting our loss minimisation problem in terms of maximising the log likelihood of the data, the link function is implicitly dictated by our choice of loss function. As long as we choose the link and loss functions appropriately, there will exist efficient algorithms for finding U and $\tilde{X}$ given X. The loss functions themselves are useful for analysis, whereas the dimensionality reduction algorithm uses only the link functions and their derivatives. We can pick the loss functions and compute the matching link functions; or, we can pick the link function directly and accept the corresponding loss functions (Gordon, 2003). For the problem of representing POMDP beliefs, we will use our knowledge of probability distributions to choose the loss function explicitly, and compute the matching link function.

The Loss Function for POMDPs

Each choice of link and loss functions results in a different dimensionality-reduction error model, and therefore a potentially different decomposition of X. This choice is where we can inject our knowledge about acceptable errors in representing X. Recall from section 5.2 that we prefer an error model that is constrained to non-negative errors. Also recall that we wished to increase the loss for errors close to 0. And finally, from equation (5.4) we need an error model that is an exponential family distribution. These three requirements taken together suggest the Poisson error function; a Poisson distribution is characterised by being constrained to be greater than 0, and by a variance that decreases as the mean goes to 0:[2]

$$\begin{aligned} -\log P(b|\mu) &= -\log\left(\frac{e^{-\mu}\mu^b}{b!}\right) && (5.12)\\ &= \mu - x\log\mu + \log b! && (5.13)\end{aligned}$$

Note that this loss includes a term $b!$, which is not defined for continuous data. However, a similar term can be computed for continuous b; we are ignoring this term as it depends only on b, and we are minimising the loss over U and $\tilde{b}$. The data likelihood function is no longer strictly Poisson, however it is still a well-defined exponential family distribution and shares the relevant properties.

If we set $\mu = e^{U\tilde{b}}$ in equation (5.13) and compute the loss for all b_i, then

$$L(X, U, \tilde{X}) = \sum_{i=1}^{|X|}\left(e^{U\tilde{b}_i} - b_i \cdot U\tilde{b}_i - \log b_i!\right), \tag{5.14}$$

We cannot directly minimise this loss function, but instead take advantage of equation (5.4) to minimise the generalised Bregman divergence that corresponds to this loss function and differs only in functions of b_i:

$$\begin{aligned} B_F(g(U\tilde{b}_i)\,\|\,b_i) &= B_F(e^{U\tilde{b}_i}\,\|\,b_i) && (5.15)\\ &= e^{U\tilde{b}_i} - b_iU\tilde{b}_i + b_i\log b_i - b_i. && (5.16)\end{aligned}$$

We eliminate the term that depends only on b_i, and write it over all beliefs X in matrix notation

$$B_F(U\tilde{X}\,\|\,X) = F(U\tilde{X}) - X \circ U\tilde{X}, \tag{5.17}$$

where the "matrix dot product" $A \circ B$ is the sum of products of corresponding elements in A and B.

The link function is the component-wise function

$$\bar{X} = F(U\tilde{X}) = \exp(U\tilde{X}) \tag{5.18}$$

and its associated loss function is

$$L(X, U, \tilde{X}) = \sum_{i=1}^{|X|}\exp(U\tilde{x}_i) - x_i \circ U\tilde{x}_i. \tag{5.19}$$

[2]It may also be that for many real world problems, our set of beliefs X is generated by a particle filter. In this case the entries of X are the number of particles from a large sample which fell into each small bin, so a Poisson loss function is again most appropriate.

It is worth noting that using the Poisson loss for dimensionality reduction is similar to Lee and Seung's non-negative matrix factorisation (Lee & Seung, 1999), although their algorithm minimises a loss function (squared error or KL divergence) under the constraint of non-negative data.

Equation (5.19) gives us a well-defined convex optimisation problem for finding the factors U or $\tilde{X}$.

Finding the E-PCA Parameters

Gordon (2003) describes a fast, Newton's Method approach for finding U and $\tilde{X}$ for an arbitrary link function F, which we summarise here. This algorithm uses Iteratively Reweighted Least Squares, similar to the algorithm for generalised linear regression (McCullagh & Nelder, 1983). In order to use Newton's Method to minimise equation (5.17), we need the derivative of the generalised Bregman divergence for U and $\tilde{X}$:

$$\begin{aligned} \frac{\partial}{\partial U} B_F(U\tilde{X}||X) &= \frac{\partial}{\partial U} F(U\tilde{X}) - X \circ U\tilde{X} && (5.20)\\ &= F'(U\tilde{X})\tilde{X}^T - X\tilde{X}^T && (5.21)\\ &= (f(U\tilde{X}) - X)\tilde{X}^T && (5.22) \end{aligned}$$

and

$$\begin{aligned} \frac{\partial}{\partial \tilde{X}} B_F(U\tilde{X}||X) &= \frac{\partial}{\partial \tilde{X}} F(U\tilde{X}) - X \circ U\tilde{X} && (5.23)\\ &= U^T F'(U\tilde{X}) - U^T X && (5.24)\\ &= U^T(f(U\tilde{X}) - X), && (5.25) \end{aligned}$$

where f is the derivative of the link function, $f = F'$. A full explanation of these derivatives is given in appendix A.3.

If we apply Newton's Method to equation (5.25), we can iteratively compute the j^{th} column of $\tilde{X}$: $\tilde{X}_{\cdot j}$. Let us set $q(\tilde{X}_{\cdot j}) = U^T(f(U\tilde{X}_{\cdot j}) - X_{\cdot j})$, and linearise about $\tilde{X}_{\cdot j}$ to find roots of $q()$. This gives

$$\begin{aligned} \tilde{X}^{new}_{\cdot j} &= \tilde{X}_{\cdot j} - \frac{q(\tilde{X}_{\cdot j})}{q'(\tilde{X}_{\cdot j})} && (5.26)\\ \tilde{X}^{new}_{\cdot j} - \tilde{X}_{\cdot j} &= -\frac{U^T(f(U\tilde{X}_{\cdot j}) - X_{\cdot j})}{q'(\tilde{X}_{\cdot j})} && (5.27) \end{aligned}$$

We need an expression for q':

$$\begin{aligned} \frac{\partial q}{\partial \tilde{X}_{\cdot j}} &= \frac{\partial}{\partial \tilde{X}_{\cdot j}} U^T(f(U\tilde{X}_{\cdot j}) - X_{\cdot j}) && (5.28)\\ &= \frac{\partial}{\partial \tilde{X}_{\cdot j}} U^T f(U\tilde{X}_{\cdot j}) && (5.29)\\ &= U^T D_j U && (5.30) \end{aligned}$$

We define D_j in terms of the diag operator that returns a diagonal matrix:

$$D_j = \mathrm{diag}(f'(U\tilde{X}_{\cdot j})), \tag{5.31}$$

where

$$\mathrm{diag}(b) = \begin{bmatrix} b(s_0) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & b(s_{|\mathcal{S}|}) \end{bmatrix}. \tag{5.32}$$

Combining equation (5.27) and equation (5.30), we get

$$\begin{aligned} (U^T D_j U)(\tilde{X}^{new}_{\cdot j} - \tilde{X}_{\cdot j}) &= U^T(X_{\cdot j} - f(U\tilde{X}_{\cdot j})) & (5.33) \\ \Rightarrow U^T D_j U \tilde{X}^{new}_{\cdot j} &= (U^T D_j U)\tilde{X}_{\cdot j} + U^T D_j D_j^{-1}(X_{\cdot j} - f(U\tilde{X}_{\cdot j})) & (5.34) \\ &= U^T D_j (U\tilde{X}_{\cdot j} + D_j^{-1}(X_{\cdot j} - f(U\tilde{X}_{\cdot j})), & (5.35) \end{aligned}$$

which is a weighted least-squares problem that can be solved with standard linear algebra techniques. In order to ensure that the solution is numerically well-conditioned, we typically add a regulariser to the divisor, as in

$$\tilde{X}^{new}_{\cdot j} = \frac{U^T D_j (U\tilde{X}_{\cdot j} + D_j^{-1}(X_{\cdot j} - f(U\tilde{X}_{\cdot j}))}{(U^T D_j U + 10^{-5} I_l)}. \tag{5.36}$$

Similarly, we can compute a new U by computing the i^{th} row of U: $U_{i\cdot}$, as

$$U^{new}_{i\cdot} = \frac{(U_{i\cdot}\tilde{X} + D_i^{-1}(X_{i\cdot} - f(U_{i\cdot}\tilde{X}))D_i\tilde{X}^T}{(\tilde{X}D_i\tilde{X}^T + 10^{-5} I_l)}. \tag{5.37}$$

where I_l is the identity matrix.

5.4. The E-PCA Algorithm

We can now construct an algorithm for automatically finding a good low-dimensional representation $\tilde{b}$ for the high-dimensional belief set b, which we have been referring to as $\tilde{X} = \{\tilde{b}_1, \dots, \tilde{b}_n\}$ and $X = \{b_1, \dots, b_n\}$. This algorithm is given in table 5.1; the optimisation for U and $\tilde{X}$ is iterated until some termination condition is reached, such as a finite number of iterations, or when some minimum error ϵ is achieved.

Our definitions of PCA and E-PCA do not explicitly factor the data into U, S and $\tilde{X}$, where S contains the eigenvalues of the decomposition and U is orthonormal. However, being able to recover the eigenvalues can often be useful for such tasks as identifying the dimensionality of the underlying manifold.

The steps 9 and 11 from Gordon's Newton's Method (2003) raise a second issue. Although solving for each matrix U or $\tilde{X}$ separately is a convex optimisation, solving for the two simultaneously is not. We are therefore subject to potential local minima, although in practice we have not found this to be a problem, and once the bases are found (and fixed), computing the low-dimensional representation of a high-dimensional belief is still convex.

```
1. Collect a set of sample beliefs from the high-dimensional belief space
2. Assemble the samples into the data matrix $X = [b_1| \ldots |b_{|X|}]$
3. Choose an appropriate loss function, $L(X, U, \tilde{X})$
4. Compute $B_F(U\tilde{X} \parallel X)$ from equation (5.5).
5. Differentiate to find expressions for $\tilde{X}_{\cdot j}^{new}$ and $U_{i\cdot}^{new}$ as in equation (5.36) and
   equation (5.37)
6. Fix an initial estimate for $\tilde{X}$ and $U$ randomly
7. do
8.       For each column $\tilde{X}_{\cdot j} \in \tilde{X}$,
9.             Compute $\tilde{X}_{\cdot j}^{new}$ using current $U$ estimate
10.      For each row $U_{i\cdot} \in U$,
11.            Compute $U_{i\cdot}^{new}$ using new $\tilde{X}$ estimate
12. while $B_F(U\tilde{X} \parallel X) < \epsilon$
13. Orthonormalise $U$.
```

Table 5.1. The E-PCA Algorithm for finding a low-dimensional representation of a POMDP, including Gordon's Newton's method (2003).

Once U and $\tilde{X}$ have been found for a given data set X, we can also use step 9 in table 5.1 for computing the low-dimensional parameterisation $\tilde{b}$ for some new belief b. This is now exclusively a convex optimisation and we can find the global extremum in very few iterations.

Recovering a full-dimensional belief b from the low-dimensional representation $\tilde{b}$ is even easier:

$$x = F(U\tilde{b}), \tag{5.38}$$

which for our loss function is $\exp(U\tilde{b})$.

5.5. Experimental Results

Having used the loss function from equation (5.19) with the iterative optimisation procedure described by equation (5.36) and equation (5.37) to find the low-dimensional factorisation, we can look at how well this dimensionality-reduction procedure performs on some POMDP examples.

Toy Problem

Recall figure 5.5, where we were unable to find good representations of the data with fewer than 10 or 15 bases, even though our domain knowledge indicated that the data had 3 degrees of freedom. Examining one of the sample beliefs in figure 5.6, we saw that the representation was failing badly in the low-probability regions.

We can now take the same data set from the toy example, compress the data using E-PCA and compare the performance of PCA and E-PCA. Figure 5.8 shows the E-PCA is substantially more efficient at representing the data, as we see both the L_2 error and KL divergence falling very close to 0 after 4 bases. The L_2 error at 4 bases is 4.64×10^{-4}.

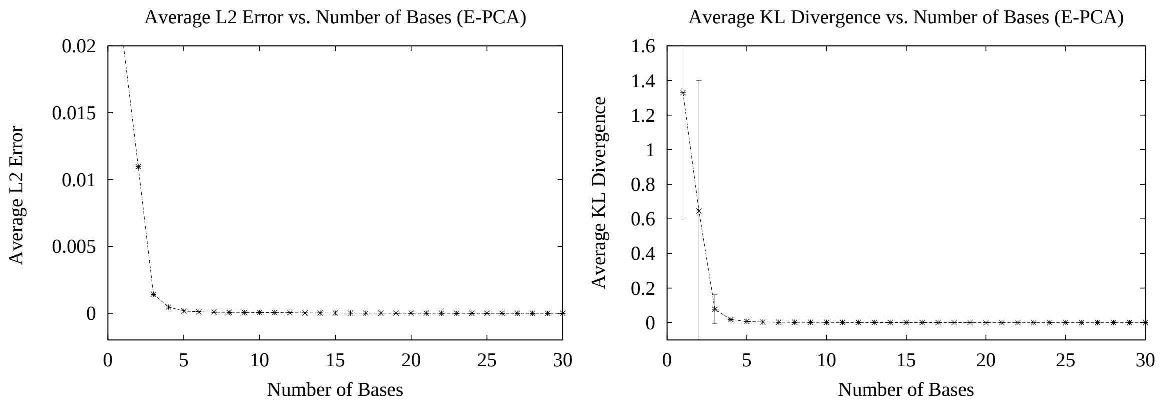
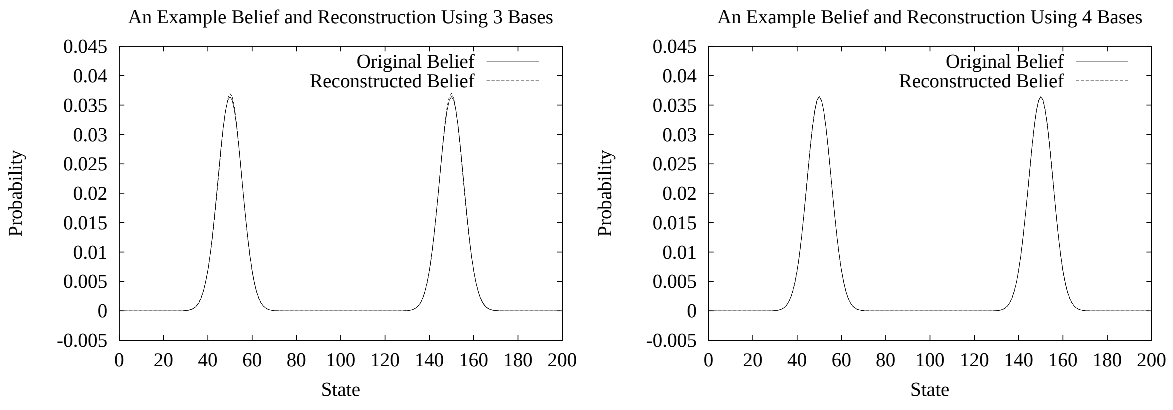


Figure 5.8. The average error between the original sample set X and the reconstructions $U\tilde{X}$. (a) The L_2 error, explicitly minimised by PCA (but not E-PCA), is 4.64×10^{-4} after 4 bases. The error bars are present but very small. (b) The KL divergence is 0.018 after 4 bases.



(a) E-PCA using 3 bases

(b) E-PCA using 4 bases

Figure 5.9. The same example belief from figure 5.6 and its reconstruction, using 3 and 4 bases. (a) On the left, the reconstruction using 3 bases, shows small errors at the peak of each mode. (b) On the right, the reconstruction using 4 bases, where the original belief and its reconstruction are indistinguishable to the naked eye.

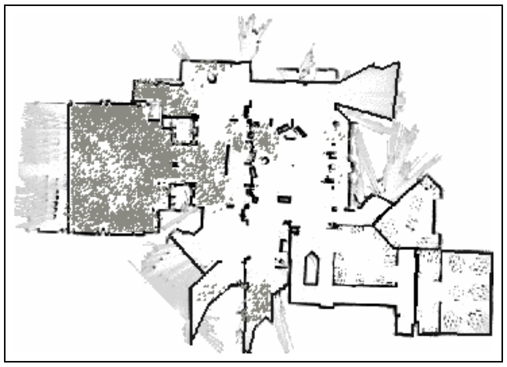
Figure 5.9 shows the E-PCA reconstruction of the same example belief as in figure 5.6. We see that many of the artifacts present in the PCA reconstruction are absent. Using only 3 bases, we see that the E-PCA reconstruction is already substantially better than PCA using 10 bases, although there are some small errors at the peaks of the two modes. Using 4 bases, the E-PCA reconstruction is indistinguishable to the naked eye from the original belief. This belief was chosen at random, and this kind of accuracy for both 3 and 4 bases is typical for this data set.

Robot Beliefs

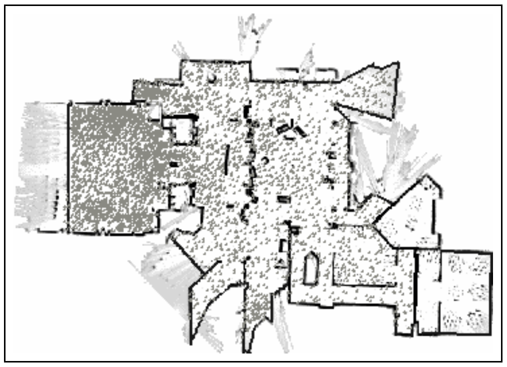
The previous results were for a relatively simple belief space. Figure 5.10 shows similar results for a more realistic environment. This data set of beliefs does not correspond exactly to beliefs from the pure navigation problem as in chapter 4, but to the "People finding" problem which will be described in more detail in chapter 6. However, we can see in figure 5.10 that this problem generates spatial distributions of some complexity that are not easily captured using the features from the original Augmented MDP representation. We will see in chapter 6 that the AMDP features do a poor job of characterising beliefs for control, similar to the example of navigation failure earlier in this chapter.

The state space in this example is constrained such that grid cells that received zero probability mass in all samples of the data set were eliminated from the state space, resulting in a state space of $|\mathcal{S}| = 11,132$. We see an example distribution in figure 5.10(a), and the PCA reconstruction using 40 bases in figure 5.10(b). Because the state space has been constrained, we do not have a problem of probability mass showing up in impossible locations. PCA still scatters substantial probability mass throughout the rest of free space, but the gross structure is roughly correct. In contrast, figure 5.10(c) shows the E-PCA representation which appears to be a substantially better reconstruction of figure 5.10(a).

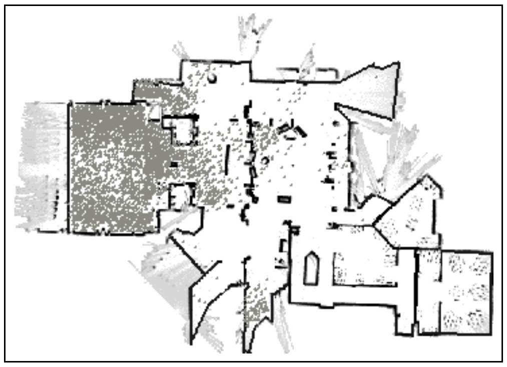
Figures 5.11(a) and (b) provides a quantitative description of the relative performance of the two algorithms on this data set, showing the average KL divergence between the original and reconstructed beliefs. In the interests of experimental speed, the state space was reduced to 705 states (53×37 grid cells with a resolution of $1m$, with the empty cells removed). This figure shows that (just as in the toy example) the KL divergence in the E-PCA representation falls rapidly and smoothly as the number of bases increases, and there is strong consistency in the representation quality. In contrast, the PCA representation performs poorly, has wide variance in the representational quality, and does not improve significantly. The average L_2 error of the E-PCA falls to less than 10^{-3} after 6 bases. Figures 5.11(c) and (d) show the same results with the original resolution up to 12 bases for the E-PCA.



(a) Original Belief



(b) Reconstruction with PCA

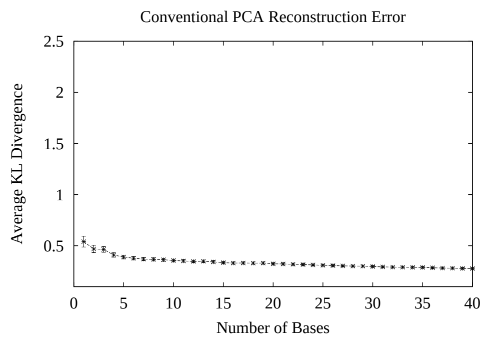


(c) Reconstruction with E-PCA

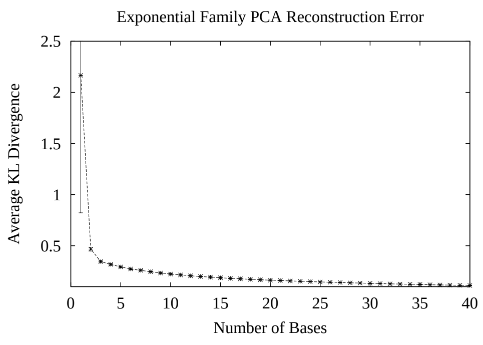
Figure 5.10. The performance of PCA and E-PCA on a sample belief, from a data set of 500 beliefs gathered from the problem of people finding in the Longwood retirement resort. (a) A sample belief. (b) The PCA reconstruction, using 40 bases. (c) The E-PCA reconstruction, using 6 bases.

5.6. Time Complexity

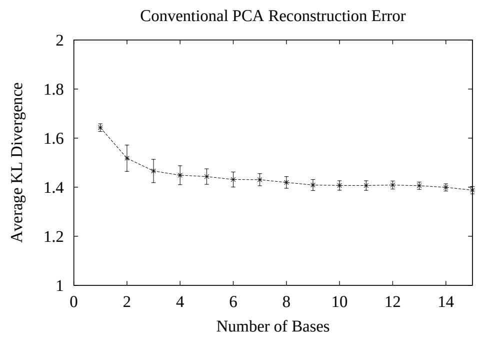
These experiments demonstrate that the E-PCA algorithm can scale to finding low-dimensional surfaces of very high-dimensional spaces. The algorithm is iterative and therefore no recurrence for the total running time is available. However, for a data set of $|X|$ samples of dimensionality n, computing a manifold of size l, each iteration of the algorithm is $O(|X|nl^2 + |X|l^3 + nl^3)$. Each step of the Newton's algorithm is dominated by a set of matrix multiplies and the final step of inverting an $l \times l$ matrix, which is $O(l^3)$. The U step consists of $|X|$ iterations, where each iteration has $O(nl)$ multiplies and the $O(l^3)$ inversion. The V step consists of n iterations, where each iteration has $O(|X|l)$ multiplies and the $O(l^3)$ inversion, leading to the total complexity given above.



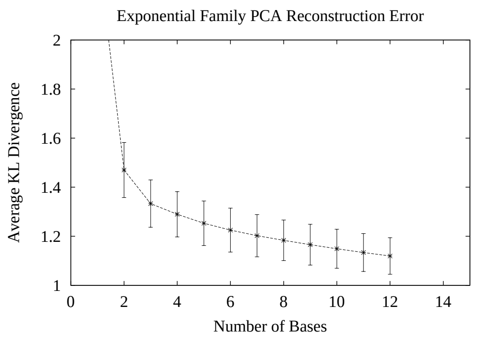
(a) Low Resolution PCA KL Divergence



(b) Low Resolution E-PCA KL Divergence



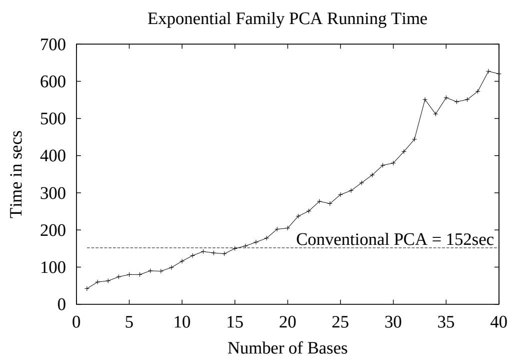
(c) High Resolution PCA KL Divergence



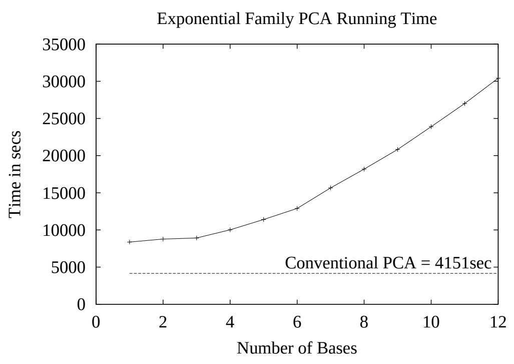
(d) High Resolution E-PCA KL Divergence

Figure 5.11. The performance of PCA and E-PCA over a data set of 500 sample beliefs, as measured by average KL divergence. Figures (a) and (b) are for a map with $1m^2$ per state and figures (c) and (d) are with $0.01m^2$ per state. (a),(c) The PCA performance is generally poor, has wide variance in the representational quality, and does not improve significantly. (b),(d) The E-PCA reconstruction error falls rapidly and smoothly as the number of bases increases, and there is strong consistency in the representation quality.

Figure 5.12 shows the time to compute the E-PCA bases for 500 sample beliefs, using both the low-resolution and high-resolution state spaces described in the previous sections, on a 1 GHz Athlon CPU with 900M of RAM; this implementation used Java 1.4.0 and Colt 1.0.2. Also shown are the computation times for conventional PCA decomposition. For the smaller state space the E-PCA decomposition is faster than PCA for fewer than approximately 15 bases; this is misleading because the PCA algorithm always computes the full decomposition ($l = n$, where l is the reduced dimensionality and n is the full dimensionality, i.e., the state space).



(a) Low Resolution E-PCA Execution Time



(b) High Resolution E-PCA Execution Time

Figure 5.12. The time to compute the E-PCA representations for the low-resolution and high-resolution representations. As in figure 5.11, figure (a) is for a state space of $1m^2$ per state discretisation (705 states) and figures (b) is for $.01m^2$ per state discretisation (11,132 states).

5.7. Model Selection

One of the open questions we have not addressed in this chapter is that of choosing the appropriate number of bases for our representation. Unless we have problem-specific information, such as the true number of degrees of freedom in the belief space (as in the toy example of section 5.2), it is difficult to identify the appropriate dimensionality of the underlying manifold for control. It is possible to identify the *exact* dimensionality of the surface on which the data lie. One common approach is to examine the eigenvalues of the decomposition, which can be recovered using the orthonormalisation step of the algorithm in table 5.1. (This assumes our particular link function is capable of expressing the manifold that our data lies on.) The eigenvalues from conventional PCA are often used to determine the appropriate dimensionality of the underlying manifold; certainly the reconstruction will be lossless if we use as many bases as there are non-zero eigenvalues.

However, we do not necessarily want a lossless representation; we minimise the loss function in order to represent the data more accurately, but a zero-loss representation may be of significantly higher dimensionality than a ϵ-loss representation. That is, there may be many features which contribute only small corrections to the overall representation, with correspondingly small eigenvalues. In the PCA of the robot data set, the 168th to the 173rd eigenvalues are all below 10^{-2} (1% of the largest eigenvalue, which is 1.420), but the eigenvalues only fall to zero at the 174th eigenvalue.[3] This kind of decision is essentially the model selection problem (Akaike, 1974; Schwarz, 1978): how to choose a description that is a good fit ("describes the important features evident in the data"), and yet is concise ("one that is easy to describe") (Hansen & Yu, 2001).

A bigger problem is that the relative sizes of the eigenvalues in E-PCA do not directly correspond to the contribution of each basis to the reconstruction error. The non-linear projection introduced by the link function causes the relation between the sizes of eigenvalues and the error contribution to be non-linear and possibly non-monotone.

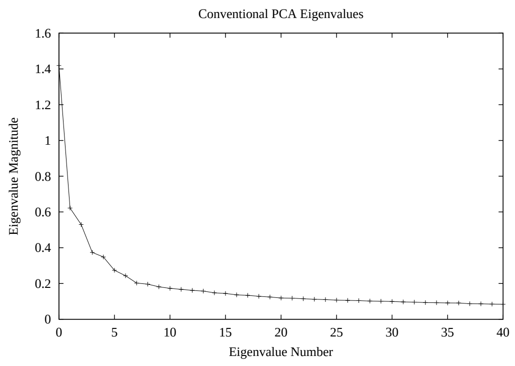
Figure 5.13 illustrates this problem. We know from figure 5.11 that the KL divergence (and L_2 error) fall very close to zero roughly around the 6th bases using E-PCA, but the error using PCA stays large for many more bases. Unfortunately, the eigenvalues for both PCA and E-PCA do not reflect this fact. The eigenvalues for both representations have approximately the same shape, and additionally, the PCA representation appears to level off around 25 eigenvalues, whereas the E-PCA appears to continue improving. Referring back to figure 5.11, it is true that the E-PCA representation continues to improve using more than 6 bases, but the level of improvement is definitely minimal after that point.

Instead of using eigenvalues to choose the appropriate manifold dimensionality, we can use reconstruction quality, as in figure 5.8 or figure 5.11. Using reconstruction quality to estimate the appropriate dimensionality is a common choice for both PCA and other dimensionality reduction techniques (Tenenbaum et al., 2000). One alternate choice would be to evaluate the reward for policies computed for different dimensionalities and choose the most compact representation that achieves the highest reward, essentially using control error rather than reconstruction quality to determine dimensionality.

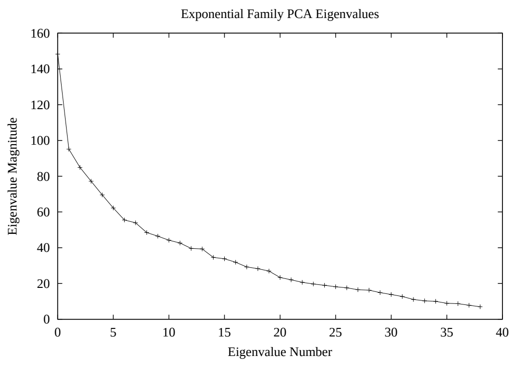
5.8. Related Work

A number of other approaches to the problem of dimensionality reduction have been applied to a wide variety of applications. The approaches described in this section all differ in that they are optimised to preserve different features of the data set in the low-dimensional representation. For example, Principal Components Analysis can be viewed as a variance-preserving projection, whereas Locally-Linear Embedding explicitly tries to preserve local geometry.

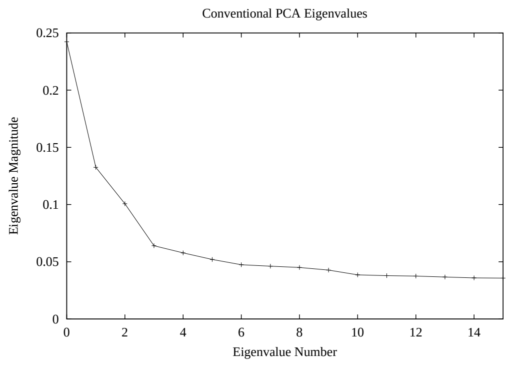
[3]The implementation of PCA that was used for this experiment generated eigenvalues that were actually non-zero for the 174th and higher order bases, but this is due to numerical imprecision, as these are all on the order of the machine precision.



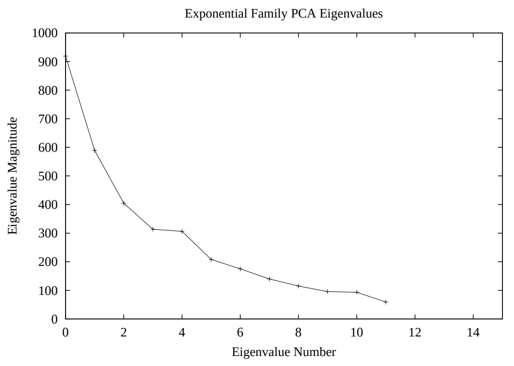
(a) Low Resolution PCA Eigenvalues



(b) Low Resolution E-PCA Eigenvalues



(c) High Resolution PCA Eigenvalues



(d) High Resolution E-PCA Eigenvalues

Figure 5.13. The eigenvalues for PCA and E-PCA for the robot navigation data set. The PCA and E-PCA eigenvalues fall at roughly the same rate, but the E-PCA eigenvalues are much larger (notice the different range of each y-axis) even after the 6th basis, the point at which the reconstruction error falls very close to 0. As in figure 5.11, figures (a) and (b) are for a state space of $1m^2$ per state discretisation and figures (c) and (d) are for $.01m^2$ per state discretisation.

Reachability Analysis

A number of attempts have been made to use reachability analysis to constrain the set of beliefs for planning (Washington, 1997; Hauskrecht, 2000; Zhou & Hansen, 2001; Pineau et al., 2003a). If the reachable set of beliefs is finite (and relatively small), then forward search to find this set is a perfectly reasonable approach, and the policy computed over these beliefs is of course optimal. However, this situation is relatively rare in real world problems. The premise of this thesis is that the set of reachable beliefs is infinite, but the space of reachable beliefs is representable using many fewer dimensions than is indicated by the model.

Reachability analysis has been used with some success as a heuristic in guiding search methods, especially for focusing computation on finding function approximators (Washington, 1997; Hansen, 1998; Hauskrecht, 2000; Zhou & Hansen, 2001). We use this idea as a heuristic to guide the collection of representative beliefs for finding the low-dimensional manifold. However, the problem still remains of how to compute the low-dimensional representation given the finite set of representative beliefs.

POMDP Compression

The value-directed POMDP compression algorithm of Poupart & Boutilier (2002) is the dimensionality-reduction technique that is possibly closest in spirit to ours, if not in technique. The authors develop an algorithm for computing a low-dimensional representation of a POMDP directly from the model parameters R, T, and O by finding the "Krylov" subspace for the reward function under belief propagation. The Krylov subspace is the smallest subspace that contains some point and is closed under multiplication. The Krylov subspace computed here is the smallest subspace that contains the immediate reward vector and is closed under a set of linear functions defined by state transitions and observation model.

The major advantage of this approach is that it optimises the correct criterion: the value-directed compression will only distinguish between beliefs that have different value. The major disadvantage of this approach is that the Krylov subspace is constrained to be linear. The compression can be viewed as PCA of the optimal value function over the reachable beliefs. As we have seen in this chapter, beliefs are unlikely to lie on a low-dimensional hyperplane, and the reported results for this technique indicate that it will not scale to the size of problems we wish to address.

However, a value-directed compression algorithm using E-PCA, or a non-linear form of the Krylov subspace, would be a major contribution to solving large POMDPs.

Multidimensional Scaling

Multidimensional scaling (MDS) is one of the oldest approaches for dimensionality reduction (Schoenberg, 1935; Young & Householder, 1938; Torgerson, 1952; Cox & Cox, 1994). MDS explicitly preserves differences between data samples, although conventional MDS, like PCA, assumes that the data lie on some linear manifold. When the difference metric is Euclidean distance, then PCA and MDS are exactly equivalent (Tenenbaum et al., 2000).

A more recent contribution by Tenenbaum et al. (2000) extends MDS-like methods to non-linear manifolds in an algorithm called "Isomap". The key insight of Isomap is to use geodesic distances (the shortest distance through the connected graph) as the distance metric between data samples. Consequently, Isomap involves an additional preprocessing step in addition to MDS, where a connected graph of the data is constructed.

Constructing a connected graph assumes the existence of a function that can determine neighbour relations. This is typically done using a distance threshold: two points are assumed to be neighbours if the distance is below some threshold. This presupposes a good way to choose this threshold, which may be considered a shortcoming of Isomap (Balasubramanian et al., 2002)[letter]. However, the authors suggest a reasonable, domain-independent way of automatically finding a good distance threshold for neighbours (Balasubramanian et al., 2002)[response]. Once the graph of the data is built, it remains to compute the geodesic distance matrix between all pairs of samples and apply classical MDS (i.e., PCA) to this matrix.

The advantage of MDS-like algorithms is that they operate over the global data set, and thus preserve global distances, whether these distances are metric, or some "non-linear squashing of distances or shortest graph paths" (Hinton & Roweis, 2003). Additionally, for data that lie on a simple manifold of Euclidean space, Isomap is guaranteed to converge to the true dimensionality and geometric structure of the manifold.

The major disadvantage of MDS, however, is that each step quickly becomes more computationally demanding as the size and dimensionality of the data set increases. Computing the connected graph for n samples of dimensionality l will be[4] $\theta(ln^2)$, computing the distance matrix using Dijkstra's algorithm will be $O(mn + n^2 \log n)$ (Dijkstra, 1959) for m edges in the graph, and computing the eigenvalues for the distance matrix will be either $O(n^3)$ (using Golub-Reinsche) or $O(nl)$ for large l (Chandrasekaran et al. (1997), pg. 322). For a model with thousands of samples and thousands of dimensions, these complexities are unacceptably high.

Local Methods

In contrast to the global, MDS-like approaches, there have been dimensionality reduction techniques that depend on purely local properties. Kohonen (1982, 1995) proposed the "Self-Organising Map" (SOM) as an unsupervised learning model, and also a model of neural computation. The representation consists of a graph of nodes, where each node

[4]This is true from inspection. The graph is constructed by computing the Euclidean distance of all pairs. Computing the Euclidean distance is O(l) and there are n^2 pairs.

represents a particular high-dimensional value. The correspondences between nodes and specific high-dimensional values is initially random; the graph is trained by presenting a data sample to the graph. The node with the value that is “closest” (for some distance metric) to this data point is updated to be closer to the new value, as are the immediate neighbours of this node.

The SOM has the following useful properties: in an unsupervised way, the nodes “learn” the high-dimensional data, and nodes that represent similar data tend to be neighbours in the graph. Therefore, if the nodes are arranged in some low-dimensional manner, then the graph can be viewed as a projection from the high-dimensional space of the data to the low-dimensional manifold represented by the layout of the graph.

Disadvantages of the SOM include the fact that there are many free parameters. A kernel must chosen; the correct way of updating the nodes is left open—at the very least, a learning rate must be set. The SOM also lacks any proofs for convergence. Finally, and most serious for our purposes, it is not clear how best to organise the nodes in anything except a grid (referred to by Roweis & Saul (2000) as “architectural specifications”).

Bishop et al. (1998) attempted to overcome these shortcomings by adding the notion of a well-defined probability density to the SOM model in the “Generative Topographic Mapping” (GTM). The GTM is a close relative of the E-PCA in that it assumes a non-Gaussian generative model in the projection from low-dimensional to high-dimensional spaces. However, the probabilistic model of GTM places Gaussians on a regular grid in the low-dimensional space, and then uses Expectation-Maximisation (Dempster et al., 1977) to learn the projections. In a sense, the GTM lies at the intersection of the SOM literature, the E-PCA and the EM algorithm for Sensible PCA due to Roweis (1997). However, the GTM still contains the problem of architectural specification.

Roweis & Saul (2000) describe an alternate algorithm called “Locally Linear Embedding” (LLE), which can be considered a local alternative to the global reduction of Isomap. LLE represents each point as the weighted combination of its neighbours and operates in two phases: computing the weights of the k nearest neighbours for each high-dimensional point, and then reconstructing the data in the low-dimensional co-ordinate frame from the weights. Just as Isomap requires a single free parameter for the appropriate threshold, LLE contains a free parameter specifying the number of neighbours for each local “patch”.

The advantage of LLE is that it can be much more computationally tractable than the global methods. Computing the weights for n data points takes n small linear regressions, and recovering the low-dimensional representation is a sparse $N \times N$ PCA for N neighbours, which can be done efficiently. However, because the LLE is local, global structure can be distorted (Tenenbaum et al., 2000), and it is unlikely that the LLE would in general preserve the overall shape of the manifold; distances in one part of the manifold are unlikely to correspond well with distances in other parts of the manifold. Unlike the SOM-based methods, the LLE has well-defined convergence properties, no learning rules to be tuned, and automatically discovers the appropriate architecture of the underlying manifold.

Non-simple Manifolds

All of the previous methods are optimal for simple manifolds, i.e., manifolds that do not intersect or contain holes, usually in a Euclidean space. These methods will provide *some* approximation for manifolds that are non-simple, but in general these properties will be lost.

Hinton & Roweis (2003) proposed a technique called "Stochastic Neighbour Embedding" to model explicitly data with a "many-to-one" relation between the high and low dimensional representations, that is, there may be high dimensional samples that correspond to multiple locations on the low-dimensional manifold. The authors have some interesting preliminary work, but have not (yet) settled on a good strategy for solving the optimisation problem.

Limitations

All of these methods, however, have serious shortcomings that currently make them inappropriate for this work. MDS is obviously subject to the failings of PCA (e.g., the assumption of linearity). None of the methods describe how to project new high-dimensional data points easily into the existing data set. Isomap's global distance metric in particular can make adding new data points computationally intractable. LLE also suffers in that new data points modify the representation, which would be disastrous for planning purposes as a new plan would need to be constructed after each step. The authors suggest one way to escape this problem is to find a function approximator such as a supervised neural network, trained on the local manifold found by LLE; if such a function approximator can be found, then LLE may become extremely relevant to the problem of belief representation for POMDPs. In general, extending these methods to the problem of belief reconstruction may be an interesting line of research.

5.9. Conclusion

In this chapter, we have discussed the Exponential family PCA (E-PCA), an algorithm for automatically discovering structure in the belief space of real-world POMDPs. Conventional dimensionality reduction approaches such as PCA perform poorly on the tasks at hand; however, we demonstrated how to improve PCA by explicitly using the knowledge that the data set consists of probability distributions. We modified the error function of PCA to ensure non-negative reconstruction, and to improve performance in the low-probability events in each sample.

In comparison on real-world data sets, we saw that the E-PCA algorithm was able to represent probability distributions with very low error using a very small number of bases (typically ≈ 6). Of course, the number of bases will vary with the conditions, and the kind of environment. In the next chapter, we will see how to use this low-dimensional representation for planning purposes. We will describe how to compute a low-dimensional belief-state MDP from high-dimensional data, and how to find good policies.

CHAPTER 6

Finding POMDP Policies with the E-PCA

But, Mousie, thou art no thy lane,
In proving foresight may be vain;
The best-laid schemes o' mice an' men
Gang aft agley,
An'lea'e us nought but grief an' pain,
For promis'd joy!

– Robert Burns
"To A Mouse, On Turning Her Up
In Her Nest With The Plough"

IN chapter 5, we described Exponential family PCA (E-PCA), an algorithm for computing appropriate low-dimensional representations of belief spaces. In this chapter, we discuss how to plan over the low-dimensional representation of the belief space, similar to the AMDP algorithm in chapter 3. For some small problems, we will be able to use a regular discretisation of the belief features found by E-PCA. For some different problems, however, the dimensionality of the reduced belief space is too large to allow a regular discretisation. We will therefore use a variable resolution discretisation approach in order to find policies in these larger belief spaces. We describe an algorithm for finding good variable resolution discretisations, and how to plan with these models.

6.1. Planning Under the E-PCA

The original Augmented MDP algorithm describes how a POMDP can be solved by converting the PODMP model into a belief space MDP that allows good policies to be found tractably. This algorithm consists of 3 principal steps:

1. Computing the low-dimensional belief space $\tilde{B}$
2. Computing the model parameters $\tilde{R}$ and $\tilde{T}$
3. Computing the value function $\tilde{V}$ for $\tilde{B}$

In chapter 3, we chose a specific low-dimensional representation of the belief space; we saw in chapter 5 that this representation was not sufficient for computing good policies, and we described the E-PCA algorithm for computing more appropriate representations.

However, the E-PCA only provides a mechanism for finding a small set of features for representing high-dimensional beliefs, rather than describing a complete MDP model in terms of beliefs. We would ideally like to use the same procedures as in chapter 3 to compute the model parameters for the low-dimensional belief space $\tilde{B}$. In the following sections, we will describe an algorithm for choosing a good discretisation of $\tilde{B}$ and how to compute the model parameters for this representation.

Collecting Sample Beliefs. In order to compute the low-dimensional space $\tilde{B}$, we first collect a set of sample beliefs, X. The algorithm we use for collecting these beliefs is given in table 6.1. We must use a heuristic for making control decisions; for some problems a good heuristic is the fully observable MDP policy, since the simulator also gives access to the (simulated) true state. This is a useful controller for sampling beliefs in some problems because it tends to generate beliefs with high probability of high-reward states. In the example problem of people-finding (discussed below), we also collected beliefs using an exploration heuristic.

```
1.  Initialise robot belief $b^t$
2.  Choose true initial state $s$ from $b^t$
3.  Compute policy $\pi(s)$ according to heuristic
    E.g., Using MDP value iteration (table 2.1).
4.  t = 0
5.  do
6.      Execute $a$ according to $V(s)$ and update true state $s$
7.      Compute $b^{t+1}$ from equation (2.8)
8.      $X = X \cup \{b^{t+1}\}$
9.      $t = t + 1$
10. while $|X| <$ desired size
```

Table 6.1. The algorithm for collecting sample beliefs. This algorithm must be conducted in simulation, because it requires knowledge of the true state.

In step 1, the belief is initialised in a problem-dependent manner. For example, in the navigation problem, we initialised the belief by choosing randomly from a set of possible initial distributions which included the uniform distribution and a set of hand-chosen distributions.

Variable Resolution

The belief space $\tilde{B}$ in chapter 3 consists of only two components: the maximum-likelihood state and normalised entropy. We were able discretise this space using a regular grid because of the fixed-sized, low dimensionality. Using the E-PCA to find better representations leads to a higher dimensional space for some different models. For example, we will see that the navigation problem at the beginning of chapter 5, figure 5.1, results in a belief space with 5 dimensions. Furthermore, each dimension spans a significant range. The need for relatively high precision in some regions of the manifold in order to distinguish different beliefs causes an explosion in the size of the belief space: "any increase in grid resolution is paid for in an exponential increase in the grid size" (Hauskrecht (2000), pg. 68). Even if we discretise relatively coarsely at 100 discrete levels per dimension, this generates a belief state space $\tilde{B}$ of 100^5 possible states. Exact value iteration does not scale well to state spaces of this size (Gordon, 1999).

In order to overcome the exponential growth associated with regular grids, we use a variable resolution approach. The discretisation of $\tilde{B}$ into the set $\tilde{B}^* = \{\tilde{b}^*_0, \tilde{b}^*_1, \ldots, \tilde{b}^*_{|\{\tilde{b}^*\}|}\}$ uses each $\tilde{b}^*_i \in \tilde{B}^*$ to represent a region of belief space. When the $\tilde{b}^* \in \tilde{B}^*$ are placed in a regular grid (as in the AMDP), each cell of $\tilde{B}$ is the same size ("uniform resolution"); when the $\{\tilde{b}^*\}$ are at differing intervals, we partition $\tilde{B}$ into cells of different sizes ("variable resolution"). The question then is how to choose the appropriate partitioning $\tilde{B}^*$. Recall that in order to compute the E-PCA belief space $\tilde{B}$, we begin by collecting a set of sample beliefs X; we use the low-dimensional projections of this sample set as a starting point for the variable resolution partitioning $\tilde{B}^*$.

We approximate the value of all beliefs $\tilde{b} \in \tilde{B}$ from the values of the beliefs in the discrete set $\tilde{B}^*$; for example, the k-nearest-neighbours method computes the value of a belief by interpolating the k neighbouring values. We will use a 1-nearest-neighbour representation. The nearest-neighbour representation raises the question of a distance metric; we assume a strict Euclidean distance over the low-dimensional belief space,

$$\tilde{b}_{nn}(\tilde{b}) = argmin_{\tilde{b}^*} ||\tilde{b} - \tilde{b}^*||, \qquad (6.1)$$

where $\tilde{b}_{nn}(\tilde{b})$ is the nearest belief in the set $\{\tilde{b}^*\}$. This distance metric can be viewed as generating cells that are the Voronoi partition induced by $\tilde{b}^*$.

Model Parameters

We compute the model parameters of our belief space MDP by computing the reward function $\tilde{R}$ and transition function $\tilde{T}$ for each discrete grid cell $\tilde{b}_i^* \in \tilde{B}^*$. The E-PCA representation automatically provides a way to recover an estimate of the high-dimensional belief b from each $\tilde{b}_i^* \in \tilde{B}^*$. The reward function $\tilde{R}(\tilde{b}, a)$ can then be computed according to equation (3.6) as in table 3.1.

Computing the transition function $\tilde{T}$ is very similar to the algorithm in the AMDP representation, with a slight difference. For a transition $\tilde{T}(\tilde{b}_i, a, \tilde{b}_j)$, we generate a posterior $p(b'|b_i, a, z)$ for the initial full belief b_i, action a and some observation z. We find the nearest-neighbour $\tilde{b}_j^*$, instead of the low-dimensional grid-cell $\tilde{b}_j$, which requires an additional step of search for the nearest neighbour. Table 6.2 gives the modified algorithm for computing the transition probabilities in this representation.

```
1. Initialise $\tilde{T}(\tilde{b}_i^*, a, \tilde{b}_j^*) = 0$ for all $(\tilde{b}_i^*, a, \tilde{b}_j^*)$.
2. For each $b_i^* \in \tilde{B}^*, a \in \mathcal{A}$
   (a) Compute $b_a$ from equation (3.7)
   (b) For each observation $z$
   (c)     Compute $b'$ from the Bayes' filter equation (2.8).
   (d)     Compute low-dimensional $\tilde{b}'$ from $b'$.
   (e)     Find nearest-neighbour $\tilde{b}_j^* \in \{\tilde{b}^*\}$ from $\tilde{b}'$
   (f)     Add $p(\tilde{b}_j|\tilde{b}_i, a)$ from equation (3.11) to $\tilde{T}(\tilde{b}_i^*, a, \tilde{b}_j^*)$
```

Table 6.2. The algorithm for computing the transition function for the sampled belief space. This algorithm is essentially identical to that in table 3.2, but contains the additional step of finding the nearest neighbour.

The Splitting Criterion

Ideally, we would like to assign a partitioning such that all beliefs in each belief space cell $\tilde{b}_i^*$ have approximately the same value under the optimal value function. It is very unlikely that the discretisation provided by the initial set of samples is optimal; in practice, the discretisation is usually unnecessarily fine in some regions of $\tilde{B}$ and insufficiently fine in other regions. The policy is typically sub-optimal in regions of $\tilde{B}$ where our discretisation is not fine enough; we can no longer distinguish between beliefs that have different actions under the optimal policy, even if the low-dimensional representation supports such a distinction.

The question is how to choose the correct partitioning $\tilde{B}^*$. We cannot know the optimal partitioning with respect to the optimal value function directly, as computing this will be as difficult as finding the optimal policy directly. We instead use an iterative algorithm that increases the discretisation in places where it is too coarse, refining the discretisation where beliefs that should apply different actions have been assigned to the same low-dimensional cell.

A heuristic can be used for determining when and where a discrete state $\tilde{b}_i^*$ should be refined further. An initial value function $\tilde{V}$ is estimated for the current partitioning $\tilde{b}^*$ (computing the value function requires model parameters, described in the previous section). A set of belief samples X_{new} are generated from the current policy and partitioning, and for each $\tilde{b}_{new} \in X_{new}$, we consider a refined partitioning $\{\tilde{b}_0^*, \tilde{b}_1^*, \ldots, \tilde{b}_i^*\} \cup \{\tilde{b}_{new}\}$. The model parameters are adjusted for the new representation and a new value function computed. If the change in the value function is greater than some threshold ϵ (a free parameter, to prevent refinement with little gain), then the sample $\tilde{b}_{new}$ is added to the partitioning. This is summarised by the algorithm in table 6.3.

```
1. Collect a sample set $X$
2. For each $b_i \in X$
3.     Compute $\tilde{b}_i$ from $b_i$, and find nearest neighbour $\tilde{b}_{nn} \in \tilde{B}$
4.     Compute new $\tilde{T}(\tilde{b}_i, a, \cdot)$, $\tilde{T}(\cdot, a, \tilde{b}_i)$ (described in the next section).
5.     Compute $\tilde{V}'$ using the algorithm in table 2.1.
6.     If $|\tilde{V}'(\tilde{b}_i) - \tilde{V}'(\tilde{b}_{nn})| > \epsilon$
7.         $\tilde{B}^{*t+1} = \tilde{B}^{*t+1} \cup \{\tilde{b}'\}$
```

Table 6.3. The algorithm for one step of refining the variable resolution representation.

Combining the model-building procedures from table 6.3 and table 6.2 give the complete algorithm in table 6.4 that we use for solving POMDPs with the E-PCA representation.

In the remaining sections, we describe experimental results using this algorithm on the toy problem and robot navigation problem from chapter 5, and a more complicated problem of finding people.

6.2. The Toy Problem

We first tested the E-PCA belief features using a regular discretisation representation. As shown in figure 6.1, we used a smaller version (40 states) of the toy problem described in chapter 5 in order to ensure that regular discretisation was possible at a sufficiently fine resolution. The regular discretisation allowed us to use the AMDP algorithm in chapter 3, table 3.3 unchanged.

```
1.  Collect a set of sample beliefs $X = \{b_1, b_2, \ldots, b_{|X|}\}$ from $B$.
2.  Compute the low-dimensional representation of $X$ using E-PCA, as in table 5.1
3.  Compute the initial discrete space $B^0 = \{\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_{|X|}\}$ from $\{b_i \in X\}$.
4.  Compute $\tilde{R}$ using the algorithm in table 3.1.
5.  Compute $\tilde{T}$ using the algorithm in table 6.2.
6.  t = 0
7.  do
8.      $\tilde{B}^{*t+1} = \tilde{B}^{*t}$
9.      Compute $\tilde{V}$ for $\tilde{B}^{*t}$ using the algorithm in table 2.1.
10.     Execute $a$ according to current belief $b$ to get posterior $b_i$
11.     Compute $\tilde{b}_i$ from $b_i$, and find nearest neighbour $\tilde{b}_{nn} \in \tilde{B}^{*t}$
12.     Compute new $\tilde{T}(\tilde{b}_i, a, \cdot)$, $\tilde{T}(\cdot, a, \tilde{b}_i)$ using $\tilde{B}^{*t} \cup \tilde{b}_i$ and table 6.2.
13.     Compute $\tilde{V}'$ using the algorithm in table 2.1.
14.     If $|\tilde{V}'(\tilde{b}_i) - \tilde{V}'(\tilde{b}_{nn})| < \epsilon$
15.         $\tilde{B}^{*t+1} = \tilde{B}^{*t+1} \cup \tilde{b}'$
16.     t = t+1
17. while $(|\tilde{B}^{*t}| > |\tilde{B}^{*t-1}|)$
```

Table 6.4. The Variable Resolution Augmented MDP algorithm.

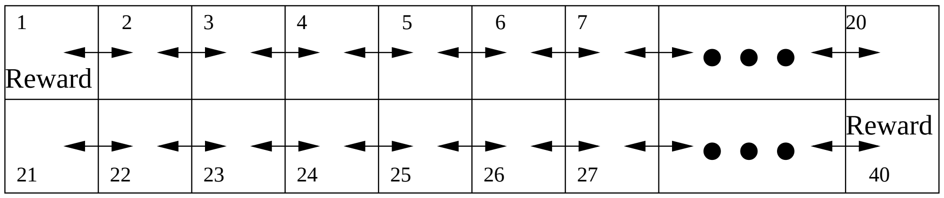


Figure 6.1. The toy maze of 40 states. Except for the smaller state space, the model is identical to figure 5.3.

Figure 6.2 shows sample beliefs from this smaller model (cf. figure 5.4). Recall from the model description in section 5.3 that each full distribution consists of 1 or 2 von Mises distributions over the two corridors. When the `sense_orientation` action is taken, one of the two modes is eliminated from the distribution.

Figure 6.3 shows the reconstruction performance of the E-PCA features as measured by the average KL-divergence between the sample beliefs and their reconstructions. The error of the E-PCA falls to zero after 3 bases (since an l-basis E-PCA can fit an $(l-1)$-parameter exponential family exactly), which is exactly as predicted from the degrees of freedom of the beliefs described in chapter 5 (noticing that in this 40 state example we get better compression compared to the 200 state example in chapter 5). We should therefore be able to use this decomposition to generate good policies using only 3 dimensions. The KL divergence is over 25 trials; error bars are shown in figure 6.3 but are negligible.

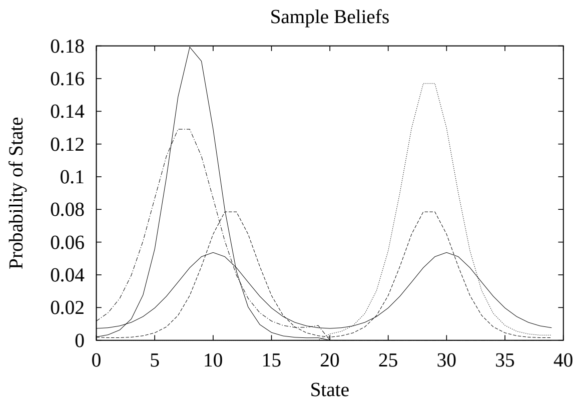


Figure 6.2. Some sample beliefs from the two-dimensional problem, generated from roll-outs of the model. Notice that some beliefs are bimodal, whereas others are unimodal in one half or the other of the state space.

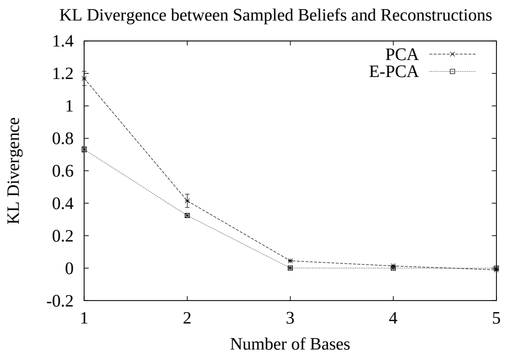


Figure 6.3. A comparison of the average KL divergence between the sample beliefs and their reconstructions, against the number of bases used, for 500 samples beliefs.

Figure 6.4 shows a comparison of the policies from different algorithms. The E-PCA does approximately twice as well as the Maximum-Likelihood heuristic (described in chapter 2); this heuristic guesses its orientation, and is correct only about half the time. The original Augmented MDP representation does very poorly because it is unable to distinguish between a unimodal belief that knows its orientation but not its position, and a bimodal belief that knows its position but not its orientation. These results are averaged over 10000 trials.

It should be noted that this problem is sufficiently small that using conventional PCA fares reasonably well. In the next sections, we will see problems where the PCA representation does poorly compared to E-PCA; these problems are also too large to use a regular discretisation and we must use the variable resolution representation developed in previous sections.

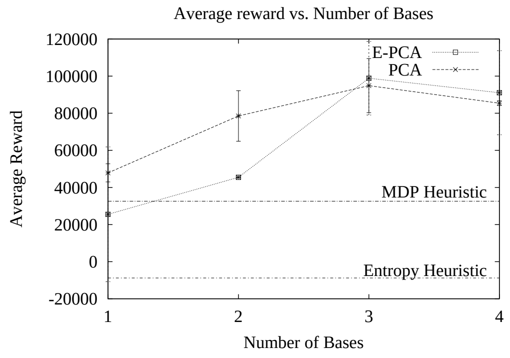


Figure 6.4. A comparison of policy performance using different numbers of bases, for 10000 trials, with regular grid discretisation. Policy performance was given by total reward accumulated over trials.

6.3. Robot Navigation

We next tested the E-PCA approach on simulated robot navigation problems in two example environments, the Wean Hall 5300 corridor shown in figure 6.5 and the Longwood retirement facility shown in figure 4.2(a).

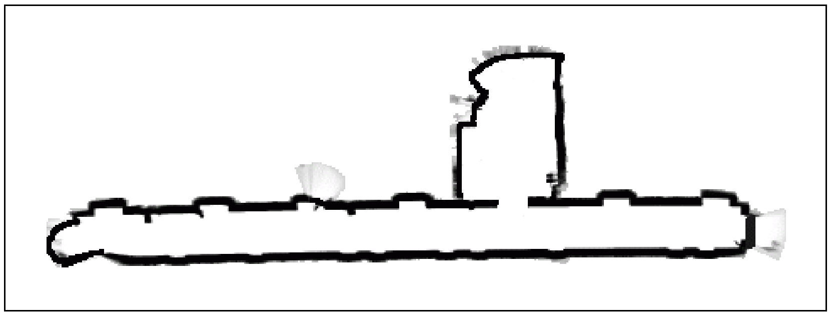


Figure 6.5. A map of the Wean Hall 5300 corridor.

We collected a sample set using the algorithm given in table 6.1. With this sample set we computed the low-dimensional belief space $\tilde{B}$ according to the algorithm in table 5.1. Figure 6.6 shows the reconstruction performance of the E-PCA approach, measured as average KL-divergence between the sample belief and its reconstruction.

The full E-PCA model was built using the variable resolution algorithm in table 6.4. In this case, the environment was sufficiently simple that no additional beliefs were added. We evaluated the policy for the relatively simple problem depicted in figure 6.7. We set the robot's initial belief such that it may have been at one of two locations in the corridor, with the objective to get to within $0.1m$ of the goal state (each grid cell is $0.1m \times 0.1m$). The controller received a reward of $+1000$ for arriving at the goal state and declaring so; a reward of -1000 for declaring itself to be at the goal incorrectly, and a smaller reward of -1

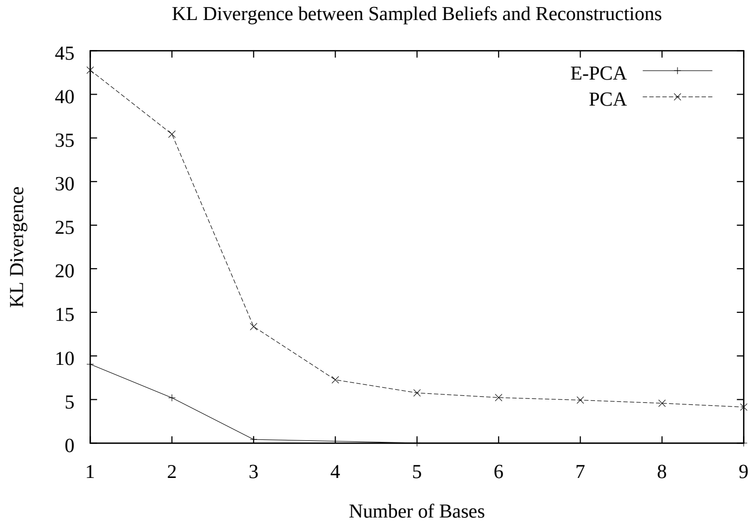


Figure 6.6. The average KL divergence between the sample beliefs and their reconstructions against the number of bases used, for 500 samples beliefs for a navigating mobile robot. Notice that the E-PCA falls close to 0 for 5 bases, whereas conventional PCA has much worse reconstruction error even for 9 bases, and is not improving rapidly.

for each motion. The total number states used for planning in this example were the 500 states along the corridor, and the set of actions was constrained to forward and backward motion.

Figure 6.7 shows a sample robot trajectory using the E-PCA policy and 5 basis functions. Notice that the robot drives past the goal to the lab door in order to verify its orientation before returning to the goal; the robot does not know its true position, and cannot know that it is in fact passing the goal. If the robot had started at the other end of the corridor, its orientation would have become apparent on its way to the goal.

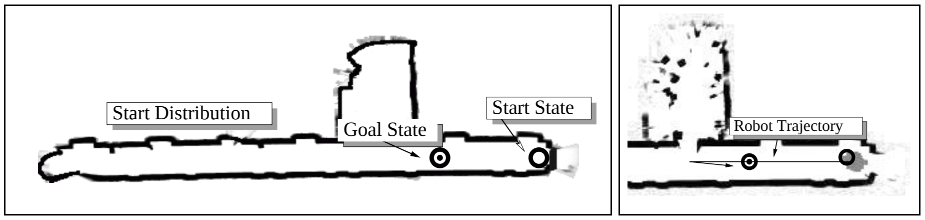


Figure 6.7. An example robot trajectory, using the policy learned using 5 basis functions. On the left are the start conditions and the goal. On the right is the robot trajectory. Notice that the robot drives past the goal to the lab door to localise itself, before returning to the goal.

Figure 6.8 shows the average policy performance for three different techniques. The Maximum-Likelihood heuristic could not distinguish orientations, and therefore approximate 50% of the time declared the goal in the wrong place. We also evaluated the AMDP using the best 5 bases computed using conventional PCA, which performed substantially better than the maximum-likelihood heuristic in that the controller did not incorrectly declare that the robot had arrived at the goal. However, this representation could not detect when the robot *was* at the goal, and also chose sub-optimal (with respect to the E-PCA policy) motion actions regularly. The E-PCA outperformed the other techniques because it was able to model its belief accurately.

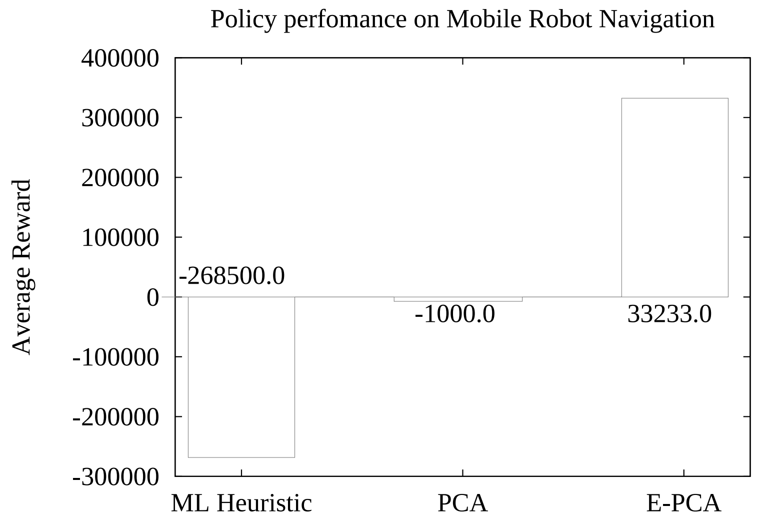


Figure 6.8. A comparison of policy performance using E-PCA, conventional PCA and the Maximum Likelihood heuristic, for 1,000 trials.

Figure 6.9 shows a second example of navigation in simulation; this is the problem in figure 5.1, where we saw the original AMDP representation of the belief space fail. Using the algorithm of table 6.1 to collect a sample set of 500 beliefs, we computed the low-dimensional belief space $\tilde{B}$.

Figure 6.10 shows the average KL divergence between the original and reconstructed beliefs. The improvement in the KL divergence error measure slowed down substantially around 6 bases; we therefore used 6 bases to represent the belief space.

Figure 6.11 shows an example execution of the policy computed using the E-PCA, with the complete trajectory shown in figure 6.12(a). The reward parameters were the same as in the previous navigation example. The robot parameters were the same as we reported in chapter 5: maximum laser range of $2m$, and high motion model variance. The first action

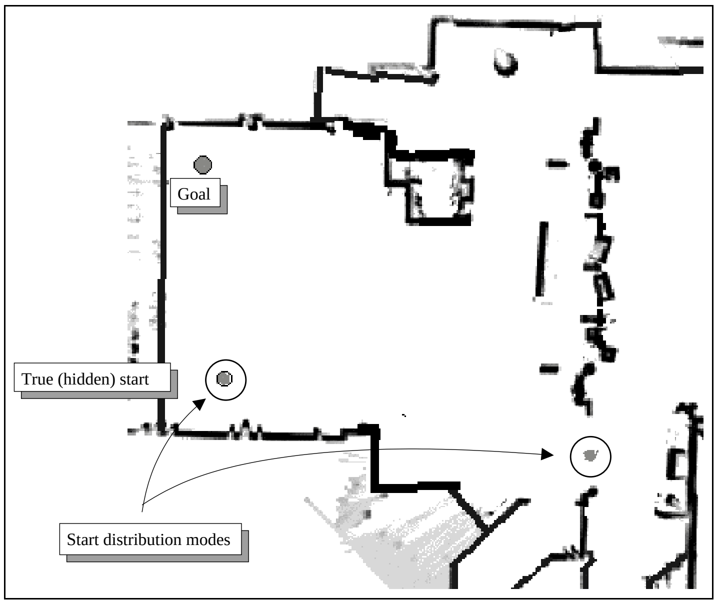


Figure 6.9. The sample navigation problem in Longwood, cf. figure 5.1.

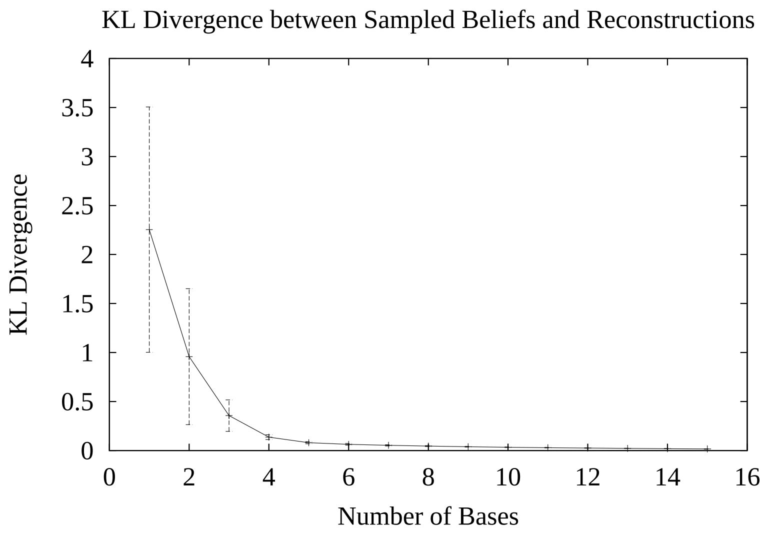


Figure 6.10. The average KL divergence between the sample beliefs and their reconstructions against the number of bases used, for 500 samples beliefs for a navigating mobile robot in Longwood.

the policy chose was to turn the robot around and move it closer to the nearest wall (panel (a)). This had the effect of eliminating the second distribution mode on the right of panel (a). The robot then followed essentially a "coastal" trajectory up the left-hand wall in order to stay localised, although the uncertainty in the y direction became relatively pronounced

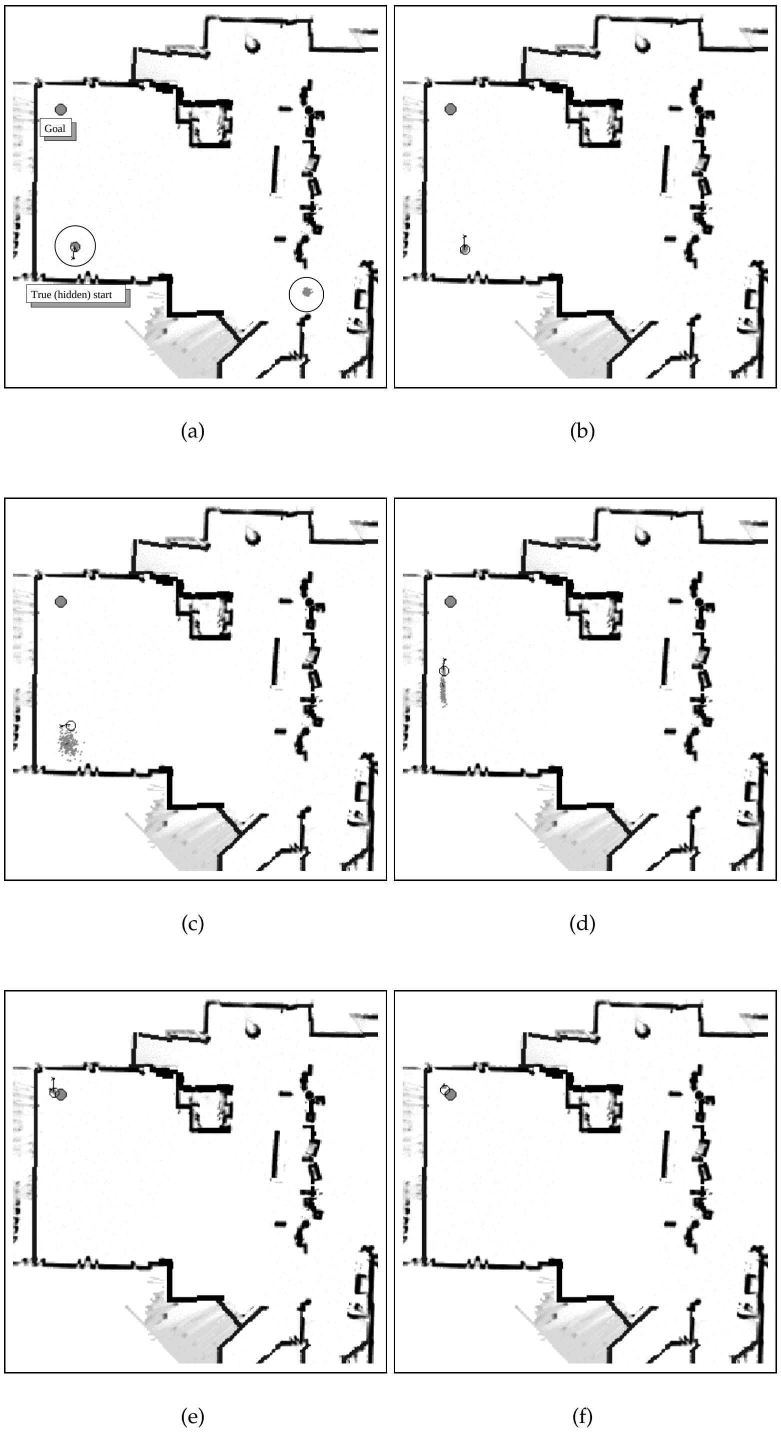
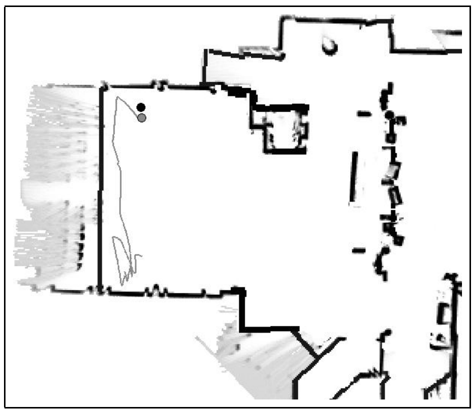


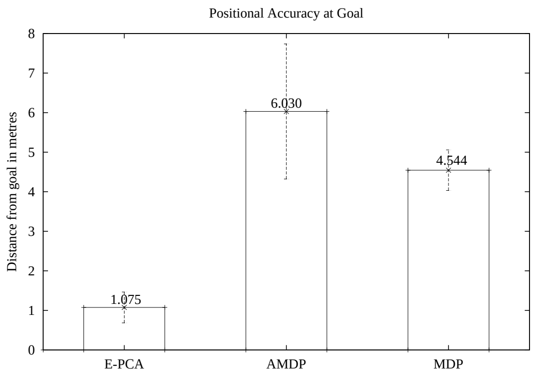
Figure 6.11. The robot trajectory using the policy learned with the E-PCA representation over 6 bases. Notice the strong "coastal" resemblance between this policy and those in chapter 4. The open circle is the true (hidden) state of the robot, and the small bar shows the direction of the next action. The goal is the grey circle at the top left corner of each panel, and the small grey dots are particles drawn from the current belief at each time step. Notice that there are two modes (cf. figure 6.9) in panel (a), but not in the remaining panels.

by panel (d). In panel (e) we see that the uncertainty resolved itself, and the robot moved to the goal.

It is interesting to note that this policy contains the same "coastal" attribute as the trajectories we saw from the original AMDP in chapter 4. The E-PCA representation was able to reach the goal more accurately (that is, get closer to the goal) not because a qualitatively different policy was required; this representation was successful because it was able more accurately to represent the beliefs and the effects of actions on the beliefs.



(a) The Complete Trajectory



(b) Performance

Figure 6.12. (a) The complete trajectory from figure 6.11. (b) A comparison of policy performance using E-PCA, conventional MDP and coastal navigation.

Finally, figure 6.12(b) shows a quantitative comparison of the policy using the E-PCA representation, the AMDP policy using the original two-dimensional representation, and the conventional MDP policy. The vertical axis shows the average distance from the goal, over 25 trajectories; a smaller distance indicates higher accuracy. The policy computed using the E-PCA representation is able to arrive at the goal with a significantly higher accuracy.

6.4. Person Finding

The next example problem is that of person finding. This problem is motivated from the Nursebot domain, where residents experiencing cognitive decline can sometimes become disoriented and start to wander. In order to make better use of the health-care providers' time, we would like to use a robot to find the residents quickly. We assume that the person is not actively avoiding the robot, but is also not stationary.

This is a qualitatively different problem in many respects from the problems we have seen so far; the uncertainty is the result of not knowing the true value of some features of the problem space (the location of the person). In the navigation and dialogue management domains the uncertainty was caused more by noisy sensors than by lack of knowledge of state features. In those domains, the controller could on occasion take advantage of actions that kept the uncertainty bounded and small, and the original Augmented MDP (maximum-likelihood state and entropy) representation was effective. The "person finding" domain has the opposite property: we assume that the start distribution is always uniform and therefore the uncertainty is maximal. The person finding domain is similar to the navigation problem where the initial distribution is multi-modal; the person finding problem can be viewed as an extreme case. Just as we saw in the navigation problem with a bi-modal initial distribution, the original AMDP representation fares badly.

Person Finding as a POMDP. The state space is the cross product of the person's position (using the grid-map representation as before) and the robot's position. The action space is identical to the navigation action space (as in figure 4.2(b)). The transitions of the person state features are modelled by Brownian motion with a fixed, known velocity, which models the essential randomness of the person's motion. (If the person was moving to avoid being "captured" by the robot, a different transition model would be required.) The distribution over person position is represented using a particle filter.

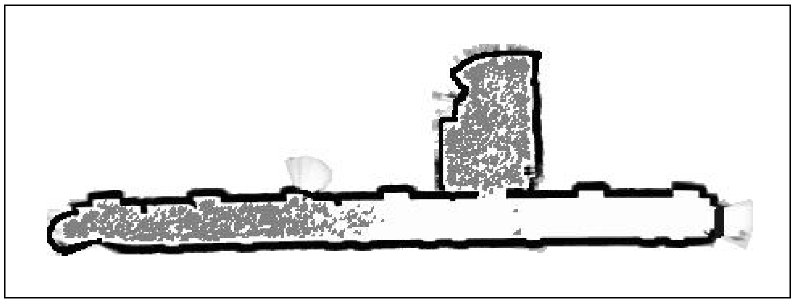
We assume that the position of the person is unobservable until the robot is close enough to see the person (when the robot has line-of-sight to the person, up to some maximum range, usually 3 metres); the observation model has 1% false negatives and no false positives. For the sake of simplicity, we also assume that the robot's position is always fully observable, although it *could* be modelled as unobservable, as in Montemerlo et al. (2002). There is no practical reason for assuming full observability in the robot's pose in solving the real problem, but this example allows us to separate the issue of sensor noise (as in robot localisation) from the issue of missing information (as in finding the person).

The reward function is maximal when the person and the robot are in the same location. We assume that for a state s, $s_{person} = person(s)$ gives the location of the person and $s_{robot} = robot(s)$ gives the location of the robot. The reward of s is

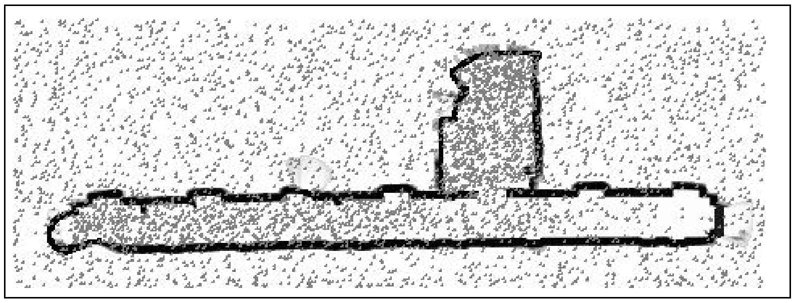
$$R(s) = \begin{cases} -1000 & robot(s) \text{ is an occupied map cell} \\ -1 & person(s) \neq robot(s) \text{ and } robot(s) \text{ is a free-space map cell} \\ 1000 & person(s) = robot(s) \end{cases}, \tag{6.2}$$

where, as before, the reward function of this problem has no dependence on the particular action. We set the initial state to the robot's known initial pose.

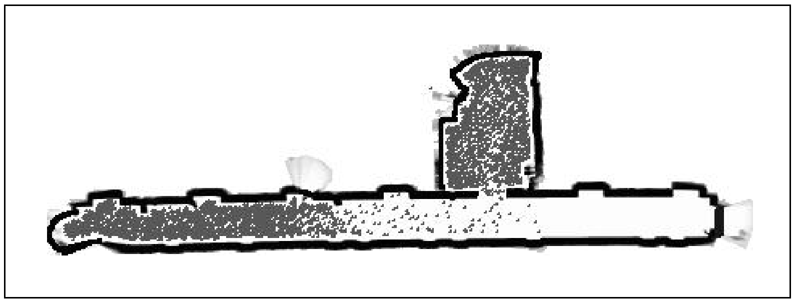
We collected the beliefs using the algorithm in table 6.1, using the MDP controller and the true state. We also added some beliefs gathered with an exploration controller and



(a) Original Belief



(b) Reconstruction with PCA



(c) Reconstruction with E-PCA

Figure 6.13. The performance of PCA and E-PCA on a sample belief. The map is 238×85 grid cells, at a $0.2m$ resolution. (a) A sample belief. (b) The PCA reconstruction, using 40 bases. (c) The E-PCA reconstruction, using 6 bases.

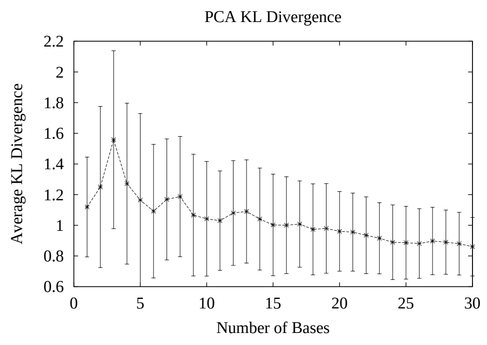
the Maximum-Likelihood controller, in an attempt to generate reasonable sampling of the belief space. The fact that our initial attempt to sample from the low-dimensional space did not do a satisfactory job indicates one problem that we will discuss in a later section; poor sampling of the beliefs could cause problems in generating a good low-dimensional representation of the POMDP as a whole; the beliefs obtained using a heuristic controller may not be representative of the beliefs for the optimal controller. Furthermore, in this problem we still needed to refine the variable resolution representation, as in the terminal loop of table 6.4.

Figure 6.13(a) shows an example distribution from a person tracking problem in a relatively simple environment, from the data set of 500 sample beliefs. There was a wide range of distributions in the data set, from uniform across the entire map, to a multi-modal distribution such as in figure 6.13(a), to single compact modes. The map is 238×85 grid cells, with a resolution of $.2m$.

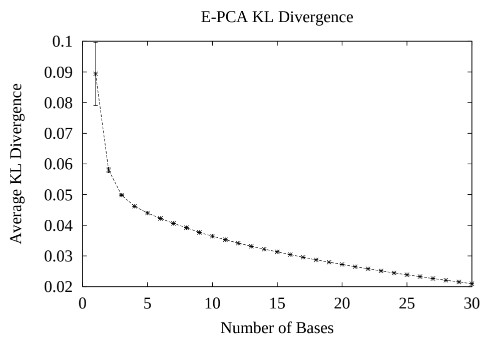
Figure 6.13(b) shows the PCA reconstruction using 40 bases, and we see that the performance is very poor. Even with 40 bases, the PCA representation is assigning substantial probability mass to regions of the map that should be empty. Recall that in figure 5.10, the state space was constrained such that grid cells that received zero probability mass in all samples of the data set were eliminated from the state space. Figure 6.13(b) should make clear that in this example we did not constrain the beliefs in the same manner. Removing unneeded states is therefore purely an efficiency consideration; the E-PCA is able to correctly model these zero-probability states, as shown in figure 6.13(c). The E-PCA reconstruction in figure 6.13(c) shows good reconstruction using only 6 bases.

Figure 6.14 provides a quantitative description of the relative performance of the two algorithms, showing the average KL divergence between the original and reconstructed beliefs. In the interests of expediency in generating the reconstruction comparisons for multiple dimensionalities, the state space was reduced to 799 states (47×17 grid cells with a resolution of $1m$) in figure 6.14(a) and figure 6.14(b). The KL divergence in the E-PCA representation falls rapidly and smoothly as the number of bases increases, and there is strong consistency in the representation quality. In contrast, the PCA representation performs poorly, has wide variance in the representational quality, and does not improve significantly. Figure 6.14(c) and figure 6.14(d) show the same phenomenon in the high-resolution state space ($20,230$ states, 238×85), up to 10 bases. We see that as the state space increases conventional PCA performs even worse.

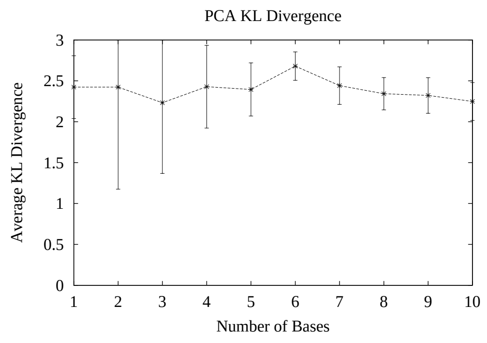
Recall that it is only the person position that is unknown; we assume that the robot pose is known throughout. When we compute a policy, it is therefore a mapping from robot pose s_{robot} and a low-dimensional belief $\tilde{b}$ over the position of the person. This raises the question of how to compute the nearest-neighbour $\tilde{b}_{nn}$ in step 11 of the algorithm in table 6.4. The optimal action in figure 6.15(a) is clearly different from the optimal action in figure 6.15(b); these two beliefs should not be considered neighbours for the purposes of control. We therefore find the nearest neighbour to $\tilde{b}$ only over the beliefs that occurred at the same robot pose as $\tilde{b}$.



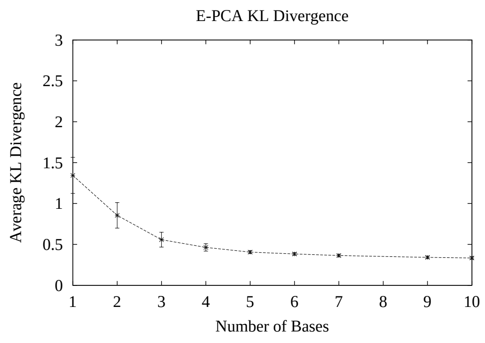
(a) PCA KL Divergence



(b) E-PCA KL Divergence



(c) PCA KL Divergence



(d) E-PCA KL Divergence

Figure 6.14. The average reconstruction (KL divergence) performance of PCA and E-PCA. (a) The PCA performance is generally poor and high variance. (b) The E-PCA reconstruction error falls rapidly and smoothly as the number of bases increases, and there is strong consistency in the representation quality. Notice the very different scales on the y-axes of the two graphs. (c) and (d) The PCA and E-PCA reconstruction errors for the full (238×85) resolution grid, up to 10 bases.

Person Finding Using the Maximum-Likelihood Heuristic. We initially tested the Maximum-Likelihood heuristic (as in section 2.4) and the fully-observable MDP policy. Figure 6.16 shows a sample execution trace of this policy, in the very simple world of the Wean 5300 corridor. The robot travelled down to the far end of the corridor, then returned to look for the person in the lab. Notice that the probability mass (and the person) “leaked” into locations already explored by the robot; the person was always just beyond the sensor range of the robot. This initially appears to be unmodelled “adversarial” behaviour on the part of the person. However, the probability distribution (which does not model the

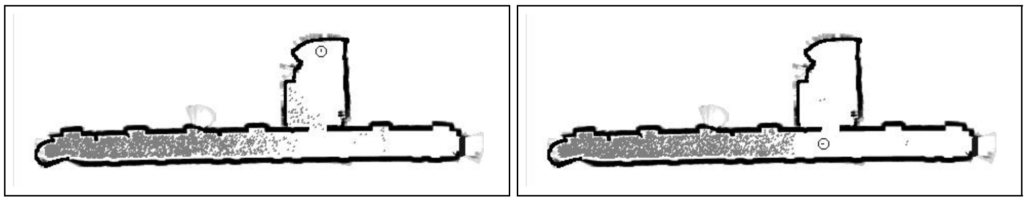


Figure 6.15. Two similar beliefs with different actions under the optimal policy. The open circle is the current robot position, the grey particles are drawn from the distribution of person poses.

person as adversarial) correctly tracks the probability of the person's true location as non-zero at all times. The person's true trajectory was a random, low-probability trajectory that happens to take advantage of the robot's policy.

Although the performance of this heuristic in this simple world is moderately good, this sample execution trace shows how the Maximum-Likelihood heuristic performs sub-optimally. Additionally, the beliefs cannot be represented using the original AMDP statistics. The belief space representation must be able to handle a wide variety of multi-modal distributions (as in figure 6.16(d) or figure 6.16(f)).

We computed the policy for the E-PCA representation using the algorithm in table 6.4. A sample of the execution trace is shown in figure 6.17, which shows a policy that is not easily captured using a simple heuristic. The robot travels part-way down the corridor, then returns to clear out the lab space before finishing the end of the corridor. The transition function built by the algorithm in table 6.2 encodes how the beliefs change as a function of actions, and we see how the policy is able to exploit this to increase the probability of finding the person.

In order to show the necessity of good discretisation of the low-dimensional belief surface, we chose a very small number of initial samples (72) for representing the surface. We then refined the discretisation using steps 7 to 17 in table 6.4. The policy built from the initial set of sample beliefs appeared in practice to be a poor approximation of the optimal discretisation; the policy improved substantially with an additional 260 samples.

In figure 6.18 we show a comparison of the performance of various heuristics with the E-PCA policy and the optimal "Fully Observable" policy. The evaluation metric is the average number of actions required to find the person, over 10,000 actions. (The problem resets every time the person is found.) The 6 policies shown are:

- Fully observable policy: this policy was for a fully-observable variant of the problem, where the person's position is always known. In this case, the cost incurred is always the travel time to the person's position. This evaluation is shown as the

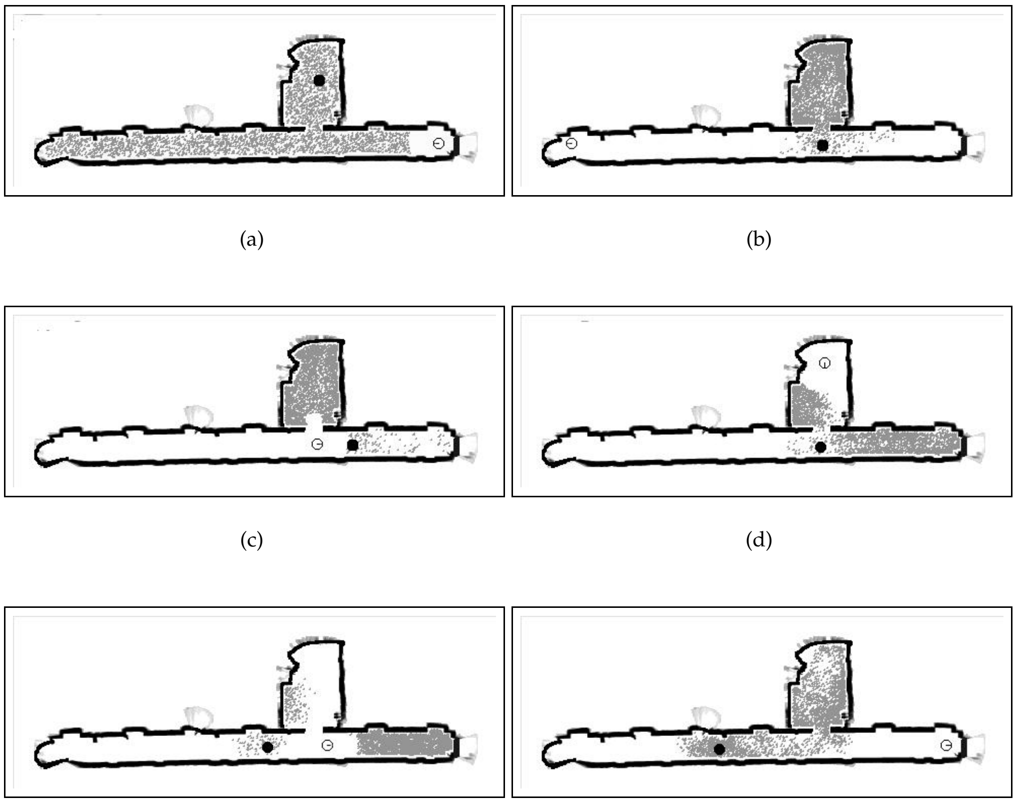

(a) (b) (c) (d) (e) (f)

Figure 6.16. An example of a suboptimal person finding policy. The grey particles are drawn from the distribution of where the person might be, initially uniformly distributed in (a). The black dot is the true (unobservable) position of the person. The open circle is the observable position of the robot.

solid line across the graph, and serves as a baseline, in that no policy could hope to be better than this policy.

- Closest: this heuristic policy took the robot to the nearest state of non-zero probability.
- Densest: this heuristic policy took the robot to the state that had the most probability mass in the robot's field of view. It was expected that this heuristic would outperform the Maximum-Likelihood heuristic; however, the policy tended to allow probability mass (and the person) to "leak".
- MDP: this heuristic policy is the Maximum-Likelihood heuristic shown in figure 6.16.

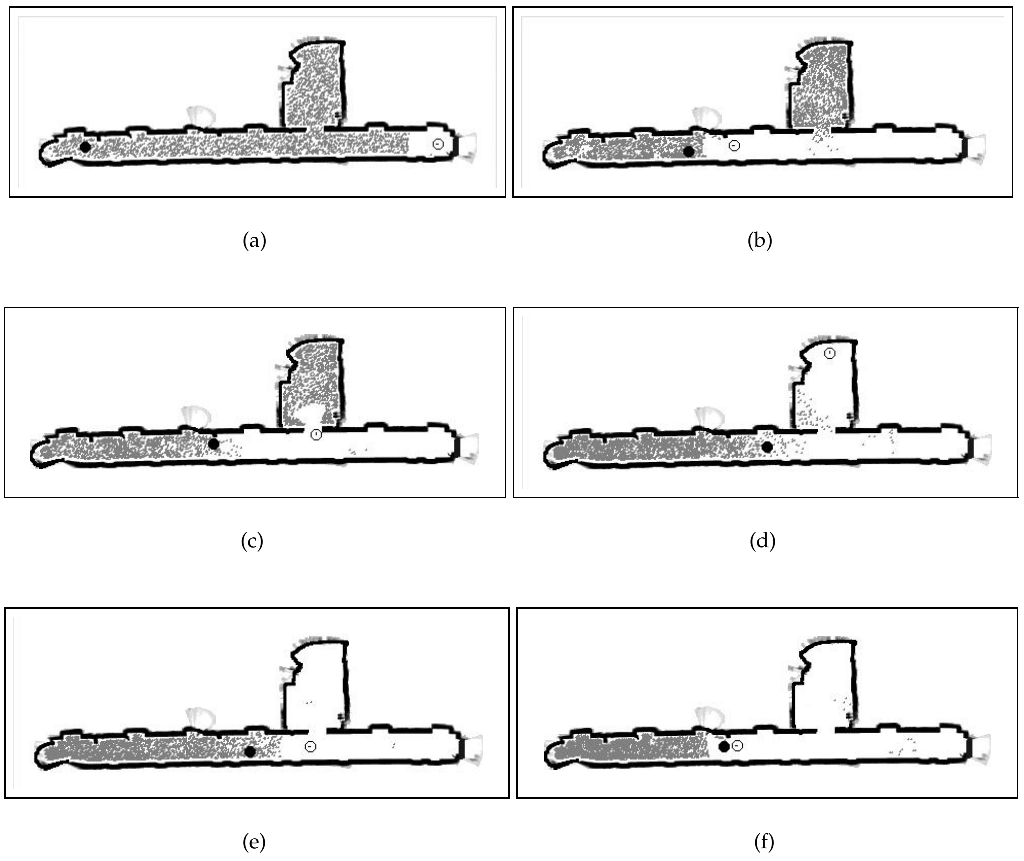


Figure 6.17. The policy computed using the E-PCA representation. The initial conditions in panel (a) are the same as in figure 6.16.

- E-PCA: this policy used the E-PCA representation and the initial set of beliefs, using 6 bases. This policy corresponds to the algorithm in table 6.4 with an inappropriately small number of initial samples.
- Var. Res. E-PCA: this policy used the E-PCA representation with a larger set of beliefs. This policy corresponds to the algorithm in table 6.4 adding 188 additional samples (after the initial sample size of 72) in steps 7 to 17.

The performance in figure 6.18 shows that the E-PCA representation is performing close to the theoretical bound of the fully-observable case, and is doing so consistently (as reflected by the small error bars). Performance this good is an artifact of the problem; in general, the optimal POMDP policy will not approach the theoretical bound to this degree. Not shown is an evaluation for the original Augmented MDP; this representation fared sufficiently badly as never to find the person for some trials. Figure 6.18 also makes clear

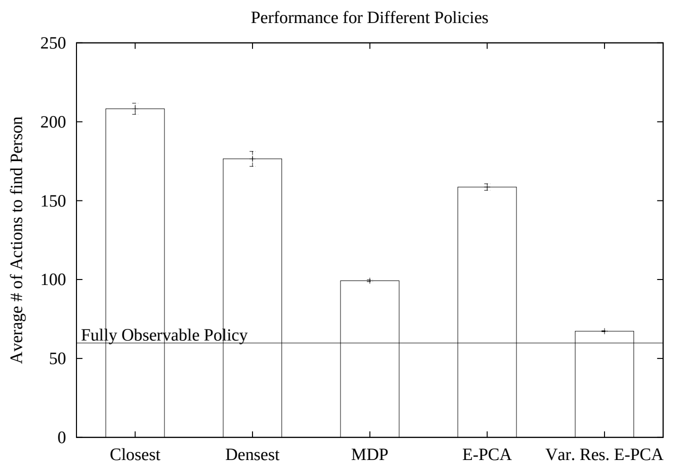


Figure 6.18. A comparison of 6 policies for person finding in a simple environment. The baseline is the solid line and is the fully-observable (cheating) solution. The E-PCA policy is for a fixed (variable resolution) discretisation. The Variable Resolution E-PCA is for a discretisation where additional belief samples have been added.

the need for an appropriate variable resolution representation; the initial discretisation performed worse than the Maximum-Likelihood heuristic.

Figure 6.19 shows an evaluation of the various algorithms in the Longwood environment of figure 4.2(a). We see in this environment similar results to that of figure 6.18, although the E-PCA result is much higher variance. This experiment was performed using 6 bases; the policy should improve with additional bases.

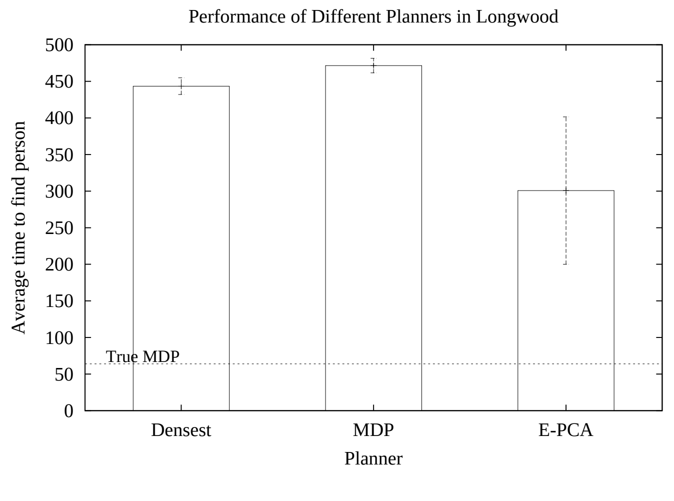


Figure 6.19. A comparison of 3 policies for person finding in a more complicated environment. The E-PCA used 6 bases and 5,712 samples in the variable resolution discretisation.

Additional Refinements

Variable Resolution in High-Dimensional Spaces. We motivated the compression of POMDPs from that fact that high-dimensional belief spaces lead to computational intractability. This is most clear in the discrete grid algorithms, such as Lovejoy (1991); Brafman (1997). However, variable resolution algorithms such as Zhou & Hansen (2001) are an attempt to overcome this limitation, and are very similar in spirit to the variable resolution algorithm described in this chapter. In fact, assuming that the beliefs in a real-world POMDP are indeed drawn from some low-dimensional surface embedded in the high-dimensional space, the optimal variable resolution representation in the high-dimensional space would have the finest discretisation close to this surface, and much coarser discretisation away from the surface where no actual beliefs are ever encountered.

One natural approach might therefore be to try to solve all POMDPs using these variable resolution algorithms directly in the high-dimensional space, and dispense with the low-dimensional space. Unfortunately, there are two difficulties with this. First of all, as we have already discussed, the limiting factor of finding the correct set of samples or fine-resolution grid cells is the dimensionality of the full space (see chapter 2, or Zhou & Hansen (2001)). Secondly, and more crucially, distance metrics in the high-dimensional space will make mistakes from the perspective of control as small variations in low-probability events can dominate the distance measurement, even though they are irrelevant for making control decisions.

For example, in the first toy problem presented in this chapter, we compared the L_2 distance metric on the low-dimensional manifold with KL divergence in the high-dimensional space. Sampling beliefs randomly directly on the manifold, the high-dimensional KL divergence and the low-dimensional L_2 nearest-neighbour relationships agreed only 45.4% of the time (over 100,000 samples). Furthermore, of these nearest-neighbour errors, 2% resulted in control errors, where the action of the KL nearest-neighbour disagreed with *all* L_2 neighbours on the low-dimensional manifold. This is a small number of control errors, but for a very small abstract problem. We expect to see a much larger number of control errors using a high-dimensional distance metric in real-world problems.

Belief Space Sampling. Recall from the description of the person-finding problem, that our initial set of samples for computing the low-dimensional representation was gathered using a mixture of controllers. Some beliefs were gathered using the fully observable MDP controller, a hand-coded controller for exploring beliefs, and also the maximum-likelihood heuristic controller. We used a mix of controllers because the fully observable MDP did a poor job of sampling some representative beliefs.

Two possible problems can result from a poor sample set. The E-PCA may not see enough beliefs to compute the necessary set of features for good control. Secondly, the parameterisation of the model may not capture the reward or the transition function correctly. Experimentally, the E-PCA algorithm appears to be robust to a number of different sampling schemes. Additionally, the algorithm described in table 6.4 compensates for modelling errors by adding new samples and recomputing the model whenever the model disagrees with new data. However, one possibility for eliminating the need for adding heuristic controllers to gather the initial sample set would be to make use of a stochastic controller initially, and use a gradient descent policy search algorithm such as G-POMDP to learn a "reasonable" (but not necessarily optimal) stochastic controller. Once a good stochastic controller is learned, it could be used to gather beliefs as a seed for learning a better value function representation.

Distance Metrics. The algorithm in table 6.4 contains a strong assumption that the nearest-neighbour metric is correct. Because the projection to $\tilde{b}$ is a convex optimisation, we know that we will not fall victim to local maxima; however, the low-dimensional surface may contain long narrow troughs and other regions of equal optimality under the loss function. We can therefore use of a Mahalanobis-like distance, where the weight term is given by the Hessian of the gradient of each projection, as in

$$\text{Hess}(\tilde{b}) = U^T D_j U \tag{6.3}$$

where the b is the full belief, and $D_j = diag(f(UV_{\cdot j}))$. By weighting each projection by its Hessian (the gradient of the projection), we compensate for a point lying in a "trough" on the loss function surface. If two projections lie in such a trough, the distance measure will receive almost no weight in the direction of the trough, and should therefore reflect a strong neighbour relationship. Preliminary experiments using this distance metric have been inconclusive.

Interpolation

Many variable resolution techniques use interpolation for computing smoother values for points that are not in the discrete representation. One of the most common such schemes is the Coxeter-Freudenthal-Kuhn (or sometimes just Kuhn) triangulation (Moore, 1992; Davies, 1996; Munos & Moore, 1999; Hauskrecht, 2000; Zhou & Hansen, 2001), which interpolates each value over $d + 1$ neighbour points in a d-dimensional space. In comparison, rectangular representations interpolate over 2^d points (Munos & Moore, 1999). Interpolated representations such as the Kuhn triangulation also provide well-defined transition probabilities over the interpolation points using "Barycentric" co-ordinates; further details can be found in the such work as (Davies, 1996) or (Munos & Moore, 1999).

Interpolation can be added to the AMDP algorithm easily by modifying step 2e and 2f of the algorithm for computing transition probabilities, given in table 6.2:

(2e) Compute $\{\tilde{b}'_0, \dots, \tilde{b}'_{d+1}\}$ neighbours from $\tilde{b}'$ and the barycentric co-ordinates $\{\lambda_0, \dots \lambda_d\}$

(2f) Add weighted $p(\tilde{b}'_j|\tilde{b}_i, a)\lambda_j$ from equation (3.11) to $\tilde{T}(\tilde{b}_i, a, \tilde{b}'_j)$ for each neighbour $\tilde{b}'_j$

6.5. Related Work

There exist a number of alternate approaches to using variable resolution, discrete representations for solving continuous space MDPs. One of the most well-known is the Parti-game (Moore & Atkeson, 1995). This algorithm assumes subdivides the state space into rectangular regions, splitting each state s_i whenever an action from s_i is discovered to fail. The original formulation of the algorithm contains several strong assumptions: the underlying dynamics are deterministic, the objective is a specific goal region (as opposed to maximising an arbitrary reward function), and any feasible solution to the goal is judged acceptable (as opposed to the optimal solution).

The Parti-game style of variable resolution for continuous MDPs was developed further by Munos & Moore (1999, 2002). This variable resolution algorithm relaxes most of the assumptions of the Parti-game algorithm, except for that of deterministic dynamics. The authors claim this assumption can also be relaxed to allow application to stochastic models by pre-processing the model to remove the intrinsic stochasticity of the transitions from the model; however, they provide no details on how to do this, and it seems likely that the specific splitting criteria that they describe do not easily lend themselves to stochastic domains.

Munos & Moore (2002) use the Kuhn (Moore, 1992; Davies, 1996) triangulation for interpolating the value function, and develop different splitting criteria for refining the triangular grid. One of the most useful concepts in this work is that of "influence"; the influence of state s_i on s_j is the probability that a diffuse process from s_j will eventually encounter s_i:

$$I(s_i|s_j) = \sum_{t=0}^{\infty} p^t(s_i|s_j) \tag{6.4}$$

where $p^t(s_i|s_j)$ is the probability of being in state s_i at time t given that the process started from s_j. Regions of the belief space that have high influence at the start need to have more reliable estimates of their value function, and therefore are more likely to need higher resolution representation. The best splitting criterion evaluated by Munos & Moore (2002) is the influence-weighted variance, where the criterion based on the variance is the estimate of the variance of the value function at state s_i.

Our algorithm described in table 6.4 is closely related to this quantity, however it should capture stochastic worlds much more appropriately. It can be thought of as a version of the algorithm of Munos & Moore (2002) where the *change* in influence-weighted variance is computed for a potential split, and a cell is only split where this change is an improvement.

Two related approaches to finding the appropriate discretisation can be found in the G-algorithm (Chapman & Kaelbling, 1991) and the U-tree algorithm (McCallum, 1995b) as described in chapter 2. The G-algorithm builds a decision tree using distinguishing perceptual bits (in the binary sense), and uses the Kolmogorov-Smirnov statistical test to determine when the tree should be further refined, just as the U-tree algorithm does for an action-observation policy tree as described in chapter 2.

Each belief in our low-dimensional discrete set $\tilde{B}_i^*$ can be viewed as a particular set of histories of sequences and actions, exactly as the U-tree explicitly encodes such histories. The question of when to grow a tree further is therefore similar to asking when to split a grid cell. Unlike the variable resolution algorithm (Munos & Moore, 2002) described above, the U-tree computes the correct quantity for stochastic worlds by maintaining a "fringe" beyond the leaves of the U-tree; the fringe contains the expected reward of a tree one layer deeper. The tree is extended to include the fringe along branches where doing so improves the predicted reward distribution. The Kolmogorov-Smirnov criterion is the correct splitting criterion in that the it is trying to optimise its ability to predict (and therefore maximise) the problem objective function; this is similar to the value-directed compression of Poupart & Boutilier (2002), which is the correct objective for compression in that the compression is trying to optimise the representation of the objective function. The influence-weighted heuristic of Munos & Moore (2002) is an approximation for estimating these quantities without using data.

6.6. Conclusion

In this chapter, we have described how to use the Exponential family PCA (E-PCA) from chapter 5 for solving POMDPs that were not amenable to the original Augmented MDP representation. Some models required a low-dimensional belief space $\tilde{B}$ of too high dimensionality to be solved using a regular discretisation of the E-PCA belief features. To compensate we developed a variable-resolution algorithm for refining the discretisation only in regions where fine discretisation proved necessary for good control.

We demonstrated the use of this algorithm on a small synthetic model, and also on the navigation problem from chapter 5 that motivated the development of the E-PCA representation. We also demonstrated this algorithm on the problem of finding people. This problem involved a richer representation (mixed observable and unobservable features) and a more complicated variable resolution representation. In the robot navigation examples, the initial sample set provided a good discretisation of the low-dimensional surface. In the person-finding problem, we demonstrated that the initial sample set was not sufficient and led to a measurably sub-optimal policy. By using refinement with additional samples from the belief surface, we were able to find compute substantially better policies.

In the robot navigation problem we were able to compute trajectories that arrived at the goal with significantly higher accuracy than standard navigation techniques, and in the person finding problem, the policies found the person measurably faster than a number of heuristics.

CHAPTER 7

CONCLUSION

There two things which I am confident I can do very well: one is an introduction to any literary work, stating what it is to contain, and how it should be executed in the most perfect manner; the other is a conclusion, showing from various causes why the execution has not been equal to what the author promised to himself and to the public.

– Samuel Johnson

The central thesis of this work is that probabilistic state estimation can be integrated into decision making for real world problems tractably while still producing good policies if compact representations of the probabilistic estimates are used.

The partially observable Markov decision process is a control formalism where each decision is based on the full probability distribution over states. In chapter 2 we described how to find an optimal POMDP controller exactly and a number of algorithms for finding approximate POMDP controllers.

The majority of the existing algorithms can be characterised with one of two shortcomings: either the algorithm does not scale to problems that are encountered in the real world (e.g., any value function technique), or the algorithm produces poor solutions for the types of uncertainty that are common in real world problems (e.g., policy heuristics). It is worth emphasising that the size of the state space is not the only possible limitation; some algorithms (e.g., policy search techniques) have successfully solved large models but which happen to have relatively simple policies.

In this thesis, we described a general framework for choosing better representations of the probabilistic state estimates, or "beliefs." Much of the complexity of computing exact POMDP solutions comes from the very high-dimensional nature of the space of possible distributions. We described possible representations of the belief space with two issues in mind:

- the representation is sufficiently low-dimensional for efficient value function approximation using k-nearest-neighbour; and
- the representation allows good control.

We began by describing one such representation based on only two features of the belief space: the maximum-likelihood state and entropy. Even with this impoverished representation, we were able to find good policies for some real-world POMDPs with large state spaces ($|\mathcal{S}| = 22,226$), very large observation spaces ($|\mathcal{Z}| = 5000^{360}$) and policies involving long sequences of actions.

Unfortunately, some real-world problems encounter beliefs that the AMDP representation does not model well, especially for making control decisions. The most important issue with any low-dimensional representation is whether it can distinguish between beliefs that require different actions. The AMDP representation was not able to distinguish between the different actions required of a low-entropy, bi-modal belief and a high-entropy, uni-modal belief.

Consequently, we devised an algorithm for automatically computing low-dimensional representations of belief spaces, using a dimensionality reduction approach on data sampled from the simulated model. We introduced an approximation by computing the best factorisation for reconstructing the data, rather than the best factorisation for the purposes of control. This algorithm is related to Principal Components Analysis, in that it factors the high-dimensional set of beliefs into a low-dimensional set of bases and parameters. The algorithm differs from standard Principal Components Analysis in that the error function explicitly models the fact that the data are sparse probability distributions. The loss function is based on exponential family distributions, hence the name E-PCA.

We used E-PCA to find low-dimensional representations of the belief spaces of a number of real world problems. By planning directly over the low-dimensional representations, we were able to find good policies for large POMDPs in which the simpler representation of the AMDP had produced poor policies. However, these representations were still sufficiently high-dimensional that the discrete function approximator we had used for the AMDP suffered from exponential growth. We therefore also devised an algorithm for finding good variable-resolution discretisations of the low-dimensional representation, in order to allow tractable function approximation of the belief space.

The three contributions of this work are:

- Demonstrating that POMDPs with large state spaces and complex policies can successfully be solved using appropriate representations of the belief space (chapter 3, chapter 4)

- Describing an algorithm for automatically finding good representations (chapter 5)
- Describing an algorithm for planning in the low-dimensional representations, in particular how to convert full POMDPs into the new representation and how to find good value function approximations. (chapter 6)

7.1. Future Extensions

There are a number of interesting possibilities for extending the algorithms in this work, in order to improve their efficiency or increase their domains of applicability.

Value-Function-Based Dimensionality Reduction

The heuristic that we chose for dimensionality reduction was simply one of reconstruction error, as in

$$L(X, U, V) = F(UV) - X \circ UV, \tag{7.1}$$

(cf. equation 5.19). Minimising the reconstruction error should allow near-optimal policies to be learned. However, we would ideally like to find the most compact representation that minimises control errors. This could possibly be better approximated by taking advantage of transition probability structure. For example, dimensionality reduction that minimises prediction errors would correspond to the loss function:

$$L(X, U, V, T) = F(UV) - X \circ UV + ||V_{\cdot,2\ldots n} - TV_{\cdot,1\ldots n-1}||^2 \tag{7.2}$$

where $V_{\cdot,1\ldots n-1}$ is the $l \times n-1$ matrix of the first $n-1$ column vectors in V, and $V_{\cdot,2\ldots n}$ is the $l \times n-1$ matrix of the $n-1$ column vectors in V starting from the second vector. This has the effect of finding a representation that allows v^{t+1} to be predicted from Tv^t, with the caveat that the V must be arranged all for the same action.

A way to phrase the optimal dimensionality reduction problem is to find the reduction that clusters beliefs that have similar values under the optimal value function. We obviously do not have access to this value function, but we could consider using an approximation such as from a previous iteration of dimensionality reduction, or the MDP fully-observable value function. This would lead to a reduction that maximises ability to differentiate states with different values.

Unfortunately, these objective functions do not lend themselves to the same Newton's Method minimisation as in chapter 5. Some preliminary experiments using gradient descent for the minimisation of equation (7.2) failed to converge to a useful representation, however, it may be possible to use alternate optimisation techniques.

Finding Multiple Manifolds

One shortcoming of the algorithms described in this work is that they contain the assumption that all beliefs can be described using the same low-dimensional representation. However, it is relatively easy to construct an example problem which generates beliefs that lie on two distinct low-dimensional manifolds. When using the E-PCA algorithm to find a low-dimensional representation of beliefs sampled from both manifolds, the apparent dimensionality of the beliefs may appear much higher than a set of beliefs sampled from one surface alone.

There are a number of possible solutions to this problem. One approach would be to use an Expectation-Maximisation algorithm, where the missing data is the assignment of samples to a particular set of basis vectors. (Given that our exponential family loss function allows us to compute a well-defined probability for a sample given a set of bases, we should be able to use EM exactly.) An alternate approach would be to use a local distance metric, such as one described in the Related Work section of chapter 5. For example, the global co-ordination algorithm of Roweis et al. (2002) finds a global co-ordinate frame for a collection of independent Factor Analysers. While Factor Analysers themselves are inappropriate for representing beliefs (as they contain the same linearity assumption of PCA), the global co-ordination algorithm may be applicable to finding a good low-dimensional representation from a collection of E-PCA bases.

Combining Compression with Policy Search

While this work has largely been motivated by finding better representations of beliefs, it is not the only approach to solving large POMDPs. Policy search methods (Meuleau et al., 1999) and hierarchical methods (Pineau et al., 2003b) have also been able to solve large POMDPs. It is interesting to note that the Augmented MDP with E-PCA is essentially independent of policy complexity but strongly dependent on belief complexity, whereas the policy search and hierarchical methods are strongly dependent on policy complexity but largely independent of belief space complexity. It seems likely that progress in solving large POMDPs in general will lie in the union of both approaches, such as using belief space reduction to reduce the policy search space.

Finding good decompositions in hierarchical methods can also be viewed as policy search; belief space reduction may be a good heuristic in this search. For example, the PolCA algorithm relies on an *a priori* model of the action space hierarchy. The optimal partitioning of actions at each node of the hierarchy could be the partitioning that generates the lowest-dimensional representations in each of the children. Such an algorithm raises additional interesting model-selection questions, for example, how many partitions should be generated at each node, and when to stop generating the tree.

Finding $\tilde{B}$ from the Model

The E-PCA algorithm finds a low-dimensional representation $\tilde{B}$ of the full belief space B from sampled data. We demonstrated that the reliance on sampled data is not an obstacle for some real world problems. Furthermore, using only sampled beliefs could be an asset for large problems where generating and tracking beliefs can be considerably easier than planning.

When we do have a model, however, it may be preferable to try to compute a low-dimensional representation directly from the model parameters. Poupart & Boutilier (2002) use the notion of a Krylov subspace to do this. The subspace computed by their algorithm may correspond exactly with a conventional PCA (taken with respect to the value function rather than beliefs), and we have seen a number of instances where PCA does a poor job of finding low-dimensional representations. The most likely explanation is that beliefs do not lie on low-dimensional planes for most problems, but instead on curved manifolds. An extremely useful algorithm would be one that finds a subspace closed under the transition and observation function, but is not constrained to a plane.

Better Function Approximation

One disadvantage to the non-linear dimensionality reduction is that exact POMDP value iteration (as described in section 2.3 of chapter 2) can no longer be applied. Any non-linear projection will prevent value function backups of the form (cf. equation (2.11))

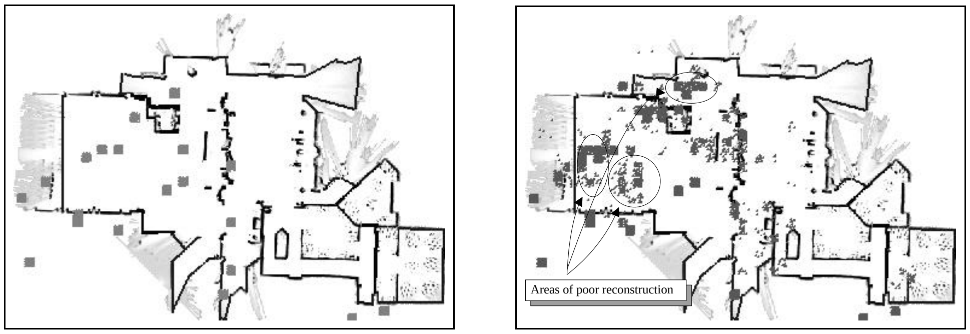
$$V_p^t(s) = R(s, a_p) + \gamma \sum_{i=1}^{|\mathcal{S}|} T(s, a_p, s_i) \sum_{j=1}^{|\mathcal{Z}|} O(s_i, a_p, z_j) V_{p,z_j}^{t-1}(s_i). \tag{7.3}$$

However, the computational complexity of computing the value function using this backup equation is not dominated by the size of the state space. Projecting the state space to some lower-dimensional representation will be of limited help in reducing the complexity of exact POMDP solutions.

One possible approach to better approximation of the value function is to use point-based estimates, as in the PBVI algorithm of Pineau et al. (2003a). One of the current limitations of PBVI is the size of the state space; using dimensionality reduction could allow PBVI to solve even larger problems. The principal obstacle to integrating non-linear dimensionality reduction and PBVI is the loss of the convexity assumption. However, if a non-linear backup operator (as in equation (2.11)) can be found for the low-dimensional manifold, then PBVI may be used as a practical way to overcome the exponential effect of history, while the dimensionality reduction overcomes the effect of the state space size.

Different Belief Structure

If the model happens to generate distributions that contain substantial structure, such as the checker-board pattern of figure 1.6(b), but are not particularly sparse, then the Poisson loss function appears to require a large number of bases in order to represent the space of beliefs. The fact that beliefs are no longer dominated by regions of low or zero probability allows the loss function too much latitude in reconstructing the particular structure in the belief.



(a) A checker-board belief

(b) The reconstruction of the checker-board

Figure 7.1. (a) A probability distribution that our representation reconstructs poorly. (b) The reconstruction of this belief using 5 bases. The collection of beliefs was constructed to have exactly 2 degrees of freedom.

Figure 7.1 shows an example belief and its reconstruction. The bases were computed for a collection of 72 beliefs, where each belief sample had identical structure but was translated vertically and horizontal. By constructing a set of sample beliefs in this manner, we know that the low-dimensional manifold containing the beliefs has exactly 2 degrees of freedom. It is therefore possible to measure how well the E-PCA representation performs as the belief complexity increases; the checker-board-like pattern in figure 7.1 achieves good performance around 8 or 9 bases. A a similar set of beliefs with much simpler structure requires only 3 bases, over-estimating the dimensionality of the manifold by only one basis.

Even if the loss function that best captures this structure is identified, it is not clear that there exists a corresponding exponential family distribution. That is, a linear combination of bases using a non-linear projection may not be a good model. Extending the dimensionality reduction to allow non-linear combinations of non-linear functions could allow even better reduction.

7.2. Summary

We have demonstrated an algorithm for planning for partially observable Markov decision processes by taking advantage of particular kinds of belief space structure that are prevalent in real world domains. Not all POMDPs will be solved using the exact form of the algorithms in this work. However, we have shown how the appropriate choice of belief representation allows us to find controllers that act appropriately in the face of real world uncertainty.

BIBLIOGRAPHY

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *AC-19*, 667–674.

Andre, D. and Russell, S. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 119–125, Edmonton, Canada. MIT Press.

Bagnell, J. A. and Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1615–1620, Seoul, South Korea. IEEE Press.

Baird, L. C. and Moore, A. W. (1999). Gradient descent for general reinforcement learning. In Kearns, M. S., Solla, S. A., and Cohn, D. A. (Eds.), *Advances in Neural Processing Systems 11 (NIPS)*, Denver, CO. MIT Press.

Balasubramanian, M., Schwartz, E., Tenenbaum, J., de Silva, V., and Langford, J. (2002). The isomap algorithm and topological stability. *Science*, *295*(5552), 7a–.

Baxter, J. and Bartlett, P. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In Langley, P. (Ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 41–48, Stanford, CA. Morgan Kaufmann.

Bellman, R. (1957). *Dynamic Programming*. NJ: Princeton University Press.

Bertsekas, D. (1976). *Dynamic Programming and Stochastic Control*. Academic Press.

Bishop, C., Svensén, M., and Williams, C. (1998). GTM: the generative topographic mapping. *Neural Computation*, *10*(1), 215–234.

Black, A., Taylor, P., and Caley, R. (1999). *The Festival Speech Synthesis System* (1.4 ed.).

Borenstein, J., Everett, H., and Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques*. Wellesley, MA: A. K. Peters, Ltd.

Brafman, R. I. (1997). A heuristic variable grid solution method for POMDPs. In Kuipers, B. K. and Webber, B. (Eds.), *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, pages 727–733, Providence, RI.

Bregman, L. M. (1967). The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Physics*, *7*(200-217).

Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, *114*(1-2), 3–55.

Burgard, W., Fox, D., Hennig, D., and Schmidt, T. (1996). Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 896–901, Portland, OR. AAAI Press.

Carreras, M. (1964). The Curse of the Mummy's Tomb. Hammer Films.

Cassandra, A., Littman, M. L., and Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 54–61, San Francisco, CA.

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

Cassandra, A. R., Kaelbling, L., and Kurien, J. A. (1996). Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Chandrasekaran, S., Manjunath, B. S., Wang, Y. F., Winkeler, J., and Zhang, H. (1997). An eigenspace update algorithm for image analysis. *CVGIP: Graphic Models and Image Processing*, *59*(5), 321–332.

Chapman, D. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In Mylopolous, J. and Reiter, R. (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 726–731, Sydney, Australia. Morgan Kaufman.

Cheng, H.-T. (1988). *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, Canada.

Collins, M., Dasgupta, S., and Schapire, R. (2002). A generalization of principal components analysis to the exponential family. In Dietterich, T. G., Becker, S., and Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems 14 (NIPS)*, Cambridge, MA. MIT Press.

Cox, I. and Wilfong, G. (Eds.). (1990). *Autonomous Robot Vehicles*. Springer Verlag.

Cox, T. and Cox, M. (1994). *Multidimensional Scaling*. London: Chapman & Hall.

Davies, S. (1996). Multidimensional triangulation and interpolation for reinforcement learning. In Mozer, M., Jordan, M. I., and Petsche, T. (Eds.), *Advances in Neural Information Processing Systems 9 (NIPS)*, pages 1005–1011, Denver, CO. MIT Press.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, *39*(1), 1–38.

Denecke, M. and Waibel, A. (1997). Dialogue strategies guiding users to their communicative goals. In Kokkinakis, G., Fakotakis, N., and Dermatas, E. (Eds.), *Eurospeech 97: ESCA 5th European Conference on Speech Communication and Technology*, pages 1339–1342, Rhodes, Greece.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematika*, *1*, 269–271.

Dudek, G. and Zhang, C. (1996). Vision-based robot localization without explicit object models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 76–82, Minneapolis, MN. IEEE Press.

Fine, S., Singer, Y., and Tishby, N. (1998). The Hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, *32*(1), 41–62.

Fox, D. (2001). Kld-sampling: Adaptive particle filters. In Dietterich, T. G., Becker, S., and Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems 14 (NIPS)*, Vancouver, Canada. MIT Press.

Fox, D., Burgard, W., and Thrun, S. (1998). Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, *25*(3-4), 195–207.

Fox, D., Burgard, W., and Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, *11*, 391–427.

Fox, D., Burgard, W., Thrun, S., and Cremers, A. (1998). Position estimation for mobile robots in dynamic environments. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 983–988, Madison, WI. AAAI Press.

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., and Rossi, F. (2002). *GNU Scientific Library Reference Manual* (3rd Edition ed.). http://www.gnu.org/software/gsl/.

Golub, G. and Reinsch, C. (1970). Singular value decomposition and least squares solutions. *Numerische Mathematik*, pages 403–420.

Gordon, G. (1995a). Stable function approximation in dynamic programming. In Prieditis, A. and Russell, S. (Eds.), *Proceedings of the 12 International Conference on Machine Learning*

(ICML), pages 261–268, San Francisco, CA. Morgan Kaufmann.

Gordon, G. (1995b). Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Carnegie Mellon University, Pittsburgh, PA.

Gordon, G. (2003). Generalized^2 linear^2 models. In Becker, S., Thrun, S., and Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15 (NIPS)*. MIT Press.

Gordon, G. J. (1999). *Approximate solutions to Markov decision processes*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

Hadamard, J. (1902). Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, 49–52.

Hansen, E. (1998). Solving POMDPs by searching in policy sepace. In *Proceedings of the 14th Conference on Uncertainty in Artifical Intelligence (UAI)*, pages 211–219, Madison, WI.

Hansen, E. and Zhou, R. (2003). Synthesis of hierarchical finite-state controllers for POMDPs. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, Trento, italy.

Hansen, M. H. and Yu, B. (2001). Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, *96*(454), 746–774.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *13*, 33–94.

Hinton, G. and Roweis, S. (2003). Stochastic neighbor embedding. In Becker, S., Thrun, S., and Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15 (NIPS)*. MIT Press.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT.

Ivanov, Y., Blumberg, B., and Pentland, A. (2000). EM for perceptual coding and reinforcement learning tasks. In *Proceedings of the 8th International Symposium on Intelligent Robotic Systems*, pages 93–100, Reading, UK.

Joliffe, I. T. (1986). *Principal Component Analysis*. Springer-Verlag.

Kaelbling, L. P., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

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

Kakade, S. (2001). A natural policy gradient. In Dietterich, T. G., Becker, S., and Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems 14 (NIPS)*, pages 531–1538, Vancouver, Canada. MIT Press.

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

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

Kearns, M., Mansour, Y., and Ng, A. (2000). Approximate planning in large POMDPs via reusable trajectories. In Solla, S. A., Leen, T. K., and Müller, K. R. (Eds.), *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 1001–1007, Denver, CO. MIT Press.

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

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *48*, 59–69.

Kohonen, T. (1995). *Self-Organizing Maps*. Berlin: Springer-Verlag.

Konolige, K. (2000). A gradient method for realtime robot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 639–646, Takamatsu, Japan.

Kumar, P. R. and Varaiya, P. P. (1986). *Stochastic Systems: Estimation, Identification and Adaptive Control*. Englewood Cliffs, New Jersey: Prentice Hall.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

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

Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*, 788–791.

Leonard, J. and Durrant-Whyte, H. (1991). Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, *7*(3), 376–382.

Levin, E., Pieraccini, R., and Eckert, W. (1998). Using Markov decision process for learning dialogue strategies. In *Proceedings of the IEEE Interational Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Litman, D. J., Kearns, M. S., Singh, S., and Walker, M. A. (2000). Automatic optimization of dialogue management. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, Saarbrucken, Germany.

Littman, M., Cassandra, A., and Kaelbling, L. (1995). Learning policies for partially observable environments: scaling up. In *Proceedings of the 12th International Conference on Machine Learning*, pages 362–370, San Francisco, CA.

Littman, M. L. (1996). *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, Providence, RI.

Lovejoy, W. S. (1991). Computationally feasible bounds for partially observable Markov decison processes. *Operations Research*, *39*, 192–175.

MacKenzie, P. and Dudek, G. (1994). Precise positioning using model-based maps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Diego, CA. IEEE Press.

Madani, O., Hanks, S., and Condon, A. (2003). On the undecidability of probabilistc planning and related stochastic optimization problems. *Artificial Intelligence*, *147*(1-2), 5–34.

Mahadevan, S. (1998). Partially observable Semi-Markov decision processes: Theory and application to engineering and cognitive sciences. In *AAAI Fall Symposium on Planning with Partially Observable Markov Decision Processes*, Orlando, FL.

Mardia, K. V. and Jupp, P. E. (2000). *Directional Statistics* (2nd ed.). Chichester, NY: Wiley.

McCallum, A. R. (1995a). Instance-based utile distinctions for reinforcement learning. In *Proceedings of the Twelfth International Machine Learning Conference (ICML)*, Lake Tahoe, CA.

McCallum, A. R. (1995b). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, New York.

McCullagh, P. and Nelder, J. A. (1983). *Generalized Linear Models* (2nd ed.). London: Chapman and Hall.

Meuleau, N., Kim, K.-E., Kaelbling, L. P., and Cassandra, A. R. (1999). Solving POMDPs by searching the space of finite policies. In Laskey, K. B. and Prade, H. (Eds.), *Proceedings of the Fifteenth International Conference on Uncertainty in Artificial Intelligence*, pages 417–426, Stockholm, Sweden. Morgan Kaufmann.

Meuleau, N., Peshkin, L., Kim, K.-E., and Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. In Laskey, K. B. and Prade, H. (Eds.), *Proceedings of the Fifteenth International Conference on Uncertainty in Artificial Intelligence*, pages 427–436, Stockholm, Sweden. Morgan Kaufmann.

Miller, J. R., Amidi, O., and Delouis, M. (1999). Arctic test flights of the cmu autonomous helicopter. In *Proceedings of the Association for Unmanned Vehicle Systems International 1999, 26th Annual Symposium*.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes. *Management Science*, *28*(1), 1–16.

Montemerlo, M., Whittaker, W., and Thrun., S. (2002). Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proceedings of the IEEE*

International Conference on Robotics and Automation (ICRA), pages 695–701, Washington, DC. IEEE Press.

Moore, A. and Atkeson, C. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, *21*(3), 199–233.

Moore, D. W. (1992). *Simplical Mesh Generation with Applications*. PhD thesis, Cornell University, Ithaca, NY.

Munos, R. and Moore, A. (1999). Variable resolution discretization for high-accuracy solutions of optimal control problems. In Dean, T. (Ed.), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1348–1355, Stockholm Sweden. Morgan Kaufmann.

Munos, R. and Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning*, *49*(2-3), 291–323.

Ng, A. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In Boutilier, C. and Goldszmidt, M. (Eds.), *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 405–415, San Francisco, CA. Morgan Kaufman.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In Bratko, I. and Dzeroski, S. (Eds.), *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pages 278–287, Bled, Slovenia. Morgan Kaufmann.

Niimi, Y. and Kobayashi, Y. (1996). Dialog control strategy based on the reliability of speech recognition. In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, pages 534–537.

Nourbakhsh, I., Powers, R., and Birchfield, S. (1995). DERVISH an office-navigating robot. *AI Magazine*, *16*(2), 53–60.

Parr, R. and Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

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

Pineau, J., Gordon, G., and Thrun, S. (2003b). Policy-contingent abstraction for robust robot control. In Meek, C. and Kjælruff, U. (Eds.), *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Acapulco, Mexico.

Pineau, J. and Thrun, S. (2002). An integrated approach to hierarchy and abstraction for POMDPs. Technical Report CMU-RI-TR-02-21, Carnegie Mellon University, Pittsburgh, PA.

Poupart, P. and Boutilier, C. (2002). Value-directed compression of POMDPs. In Becker, S., Thrun, S., and Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15 (NIPS)*, Vancouver, Canada. MIT Press.

Rabiner, L. R. (1990). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 257–286.

Ravishankar, M. (1996). *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon.

Rockafellar, R. T. (1970). *Convex Analysis*. New Jersey: Princeton University Press.

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press.

Roweis, S. (1997). EM algorithms for PCA and SPCA. In Jordan, M., Kearns, M., and Solla, S. (Eds.), *Advances in Neural Processing Systems 10 (NIPS)*. MIT Press.

Roweis, S. and Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500), 2323–2326.

Roweis, S. T., Saul, L. K., and Hinton, G. E. (2002). Global coordination of local linear models. In Dietterich, T. G., Becker, S., and Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems*, volume 14, Cambridge, MA. MIT Press.

Roy, N., Burgard, W., Fox, D., and Thrun, S. (1999). Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Detroit, MI.

Roy, N. and Gordon, G. (2003). Exponential family PCA for belief compression in POMDPs. In Becker, S., Thrun, S., and Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15 (NIPS)*, pages 1043–1049, Vancouver, BC. MIT Press.

Roy, N. and Thrun, S. (1999). Coastal navigation with mobile robots. In Solla, S. A., todd K. Leen, and Müller, K. R. (Eds.), *Advances in Neural Processing Systems 12 (NIPS)*, pages 1043–1049, Denver, CO. MIT Press.

Roy, N. and Thrun, S. (2002). Motion planning through policy search. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2419–2424, Lausanne, Switzerland.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Schoenberg, I. J. (1935). Remarks to Maurice Fréchet's article "sur la définition axiomatique d'une classe d'espace vectoriels distanciés applicables vectoriellement sur le'space de Hilbert". *Annals of Mathematics*, *36*, 724–732.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In McDermott, J. (Ed.), *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1039–1046, Milan, Italy. Morgan Kaufmann.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, *6*(2), 461–464.

Sick Optics (2003). *LMS 200/291 Technical Information*. http://www.sickoptic.com/Publish/docroot/Technical Information Sheet(s)/LMS 200-291 Tech Info.pdf.

Sim, R. and Dudek, G. (1998). Mobile robot localization from learned landmarks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Simmons, R. and Koenig, S. (1995). Probabilistic navigation in partially observable viroments. In *Proceedings of the 14th International Joint Conference on Artifiical Intelligence (IJCAI)*, pages 1080–1087, Montreal, Canada.

Singh, S., Kearns, M., Litman, D., and Walker, M. (1999). Reinforcement learning for spoken dialog systems. In Solla, S. A., Leen, T. K., and Müller, K. R. (Eds.), *Advances in Neural Information Processing Systems 12 (NIPS)*, Denver, CO. MIT Press.

Sondik, E. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University, Stanford, California.

Takeda, H., Facchinetti, C., and Latombe, J.-C. (1994). Planning the motions of mobile robot in a sensory uncertainty field. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *16*(10), 1002–1017.

Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*(5500), 2319–2323.

Theocharous, G. (2002). *Hierarchical Learning and Planning in Partially observable Markov Decision Processes*. PhD thesis, Michigan State University, Lansing, MI.

Thrun, S. (1999). Monte Carlo POMDPs. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12 (NIPS)*, Denver, CO. MIT Press.

Thrun, S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N., Schulte, J., and Schulz, D. (2000). Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*, *19*(11), 972–999.

Thrun, S., Fox, D., and Burgard, W. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, *431*.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2000). Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, *128*(1-2), 99–141.

Torgerson, W. S. (1952). Multi-dimensional scaling: 1. theory and method. *Psychometrika*, *17*, 401–419.

Torrance, M. C. (1994). Natural communication with robots. Master's thesis, MIT Dept of E.E.and C.S.

Washington, R. (1997). BI-POMDP: Bounded, incremental partially-observable Markov-model planning. In *Proceedings of the 4th European Conference on Planning (ECP)*.

Westphal, M. and Waibel, A. (1999). Towards spontaneous speech recognition for on-board car navigation and information systems. In *Eurospeech 99: Proceedings of the 6TH European Conference on Speech, Communication and Technology*, Budapest, Hungary.

White III, C. C. (1991). Partially observed Markov decision processes: A survey. *Annals of Operations Research*, *32*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3-4), 229–256.

Young, G. and Householder, A. S. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika*, *3*, 19–22.

Young, S. (1990). Use of dialogue, pragmatics and semantics to enhance speech recognition. *Speech Communication*, *9*(5-6), 551–564.

Zhang, N. L. and Zhang, W. (2001). Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *14*, 1–28.

Zhou, R. and Hansen, E. (2001). An improved grid-based approximation algorithm for POMDPs. In Nebel, B. (Ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 707–716, Seattle, Washington. Morgan Kaufmann.

APPENDIX A

Derivations

A.1. Equivalence of PCA Loss to Gaussian Likelihood

In chapter 5, section 5.3, we stated that the minimising the conventional loss function of Principal Components Analysis (PCA) was equivalent to maximising the likelihood of the data subject to a Gaussian likelihood function.

The PCA loss function is the sum of squared error, which we can write as

$$\begin{aligned} L(X,U,V) &= \sum_{i=1}^{|X|}(x_i - Uv_i)^2 && \text{(A.1)} \\ &= \sum_{i=1}^{|X|} -2\left(-\frac{(x_i - Uv_i)^2}{2}\right) && \text{(A.2)} \\ &= \sum_{i=1}^{|X|} -2\left(-\frac{(x_i - Uv_i)^2}{2} - \log\sqrt{2\pi}\right) - 2\log\sqrt{2\pi} && \text{(A.3)} \\ &= \sum_{i=1}^{|X|} 2\left(-\log\frac{1}{\sqrt{2\pi}}e^{-\frac{(x_i - Uv_i)^2}{2}}\right) - 2\log\sqrt{2\pi} && \text{(A.4)} \end{aligned}$$

The central term

$$-\log\frac{1}{\sqrt{2\pi}}e^{-\frac{(x_i - Uv_i)^2}{2}} \qquad \text{(A.5)}$$

is exactly the negative log-likelihood of the data x_i under a Gaussian distribution $\mathcal{N}(Uv_i, 1)$. Therefore, minimising the PCA loss function is the same thing as minimising this negative log-likelihood, which is equivalent to maximising the likelihood.

A.2. Converting PCA to a Generalised Bregman Divergence

We have seen that the loss function of PCA can be phrased as a negative log-likelihood of a specific exponential family distribution, the Gaussian. Any exponential family distribution can be written in the following manner. Assuming the distribution is parameterised using θ, then by definition (Collins et al., 2002),

$$-\log P(x|\theta) = F(\theta) - x\theta - \log P_0(x). \tag{A.6}$$

$F(\theta)$ is the cumulant-generating function, and it is therefore the log of the normalisation constant that enforces the law of total probability,

$$F(\theta) = \ln \int e^{\theta \cdot x} P_0(x) dx. \tag{A.7}$$

We can define a function $G(x)$ as the "convex dual" of G:

$$\inf_x (F(\theta) + G(x) - x\theta) = 0. \tag{A.8}$$

Putting equation (A.6) and equation (A.8) together,

$$-\log P(x|\theta) = (F(\theta) - x\theta + G(x)) - G(x) - \log P_0(x). \tag{A.9}$$

Substituting the definition of the Bregman divergence from equation (5.6), we see that

$$-\log P(x|\theta) = B_F(\theta \,\|\, x) + G(x) - \log P_0(x). \tag{A.10}$$

We can re-write our data as beliefs b, and parameters as bases U and low-dimensional parameterisations $\tilde{b}$. This is equivalent to finding bases that span the domain of F, and the appropriate parameterisation of b in this space. This gives us the final form of the optimisation,

$$\begin{aligned} -\operatorname*{argmin}_\theta \log P(x|\theta) &= \operatorname*{argmin}_\theta B_F(g(\theta)||x) && \text{(A.11)} \\ &= \operatorname*{argmin}_\theta F(U\tilde{b}) - b \cdot U\tilde{b}. && \text{(A.12)} \end{aligned}$$

A.3. Derivatives

The derivation of equation (5.22) and equation (5.25) are due to Gordon (2003), and reproduced here, with appropriate notation modifications.

We wish to compute the derivatives

$$\frac{\partial}{\partial \tilde{X}} B_F(U\tilde{X}||X) = \frac{\partial}{\partial \tilde{X}} F(U\tilde{X}) - X \circ U\tilde{X} \tag{A.13}$$

and

$$\frac{\partial}{\partial U} B_F(U\tilde{X}||X) = \frac{\partial}{\partial U} F(U\tilde{X}) - X \circ U\tilde{X} \tag{A.14}$$

Let us write $Z = U\tilde{X}$ and consider the two derivatives $\frac{\partial}{\partial \tilde{X}} F(U\tilde{X})$ and $\frac{\partial}{\partial \tilde{X}}(X \circ U\tilde{X})$. Consider a parameter $\tilde{X}_{ij}$. The derivative $\frac{d}{d\tilde{X}_{ij}} Z_{kl}$ is

$$\frac{d}{d\tilde{X}_i j} Z_{kl} = \frac{d}{d\tilde{X}_{ij}} U_{k\cdot} \tilde{X}_{\cdot l} = \begin{cases} U_{kl} & l = j \\ 0 & \text{otherwise} \end{cases} \tag{A.15}$$

so the derivative of $F(Z)$ is

$$\frac{d}{d\tilde{X}_{ij}}F(Z) = \sum_{kl} f_{kl}(Z)\frac{d}{d\tilde{X}_{ij}}Z_{kl} = \sum_{k} f_{kj}(Z)U_{ki} \tag{A.16}$$

and the derivative of $X \circ Z$ is

$$\sum_{kl} X_{kl}\frac{d}{d\tilde{X}_{ij}}Z_{kl} = \sum_{k} X_{kj}U_{ki} \tag{A.17}$$

Assembling equation (A.16) and equation (A.17) for each $\tilde{X}_{ij}$ gives

$$= U^T f(U\tilde{X}) - U^T X \tag{A.18}$$

$$= U^T(f(U\tilde{X}) - X) \tag{A.19}$$

We next consider the two derivatives $\frac{\partial}{\partial U}F(U\tilde{X})$ and $\frac{\partial}{\partial U}(X \circ U\tilde{X})$. Consider a parameter U_{ij}. The derivative $\frac{d}{dU_{ij}}Z_{kl}$ is

$$\frac{d}{dU_{ij}}Z_{kl} = \frac{d}{dU_{ij}}U_{k\cdot}\tilde{X}_{\cdot l} = \begin{cases} \tilde{X}_{kl} & k = i \\ 0 & \text{otherwise} \end{cases} \tag{A.20}$$

so the derivative of $F(Z)$ is

$$\frac{d}{dU_{ij}}F(Z) = \sum_{kl} f_{kl}(Z)\frac{d}{dU_{ij}}Z_{kl} = \sum_{l} f_{il}(Z)\tilde{X}_{jl} \tag{A.21}$$

and the derivative of $X \circ Z$ is

$$\sum_{kl} X_{kl}\frac{d}{dU_{ij}}Z_{kl} = \sum_{k} X_{kj}\tilde{X}_{ki} \tag{A.22}$$

Assembling equation (A.21) and equation (A.22) for each $\tilde{X}_{ij}$ gives

$$= X\tilde{X}^T - f(U\tilde{X})\tilde{X}^T \tag{A.23}$$

$$= (X - f(U\tilde{X}))\tilde{X}^T \tag{A.24}$$

Index

Document Log:

Manuscript Version 1 — January 19, 2003
Typeset by $\mathcal{AMS}$-LaTeX — 8 September 2003

NICHOLAS ROY

ROBOTICS INSTITUTE, CARNEGIE MELLON UNIVERSITY, 5000 FORBES AVE., PITTSBURGH, PA 15213, USA, *Tel.* : (412) 268-3733
E-mail address: `nickr@ri.cmu.edu`

Typeset by $\mathcal{AMS}$-LaTeX