Approximating Partially Observable Markov Decision Processes with Parametric Belief Distributions for Continuous State Spaces

A Dissertation Presented

by

Timothy Knapik

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

December 2012

Stony Brook University

The Graduate School

Timothy Knapik

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Wei Zhu – Dissertation Advisor
Professor, Department of Applied Mathematics and Statistics

Haipeng Xing – Chairperson of Defense
Associate Professor, Department of Applied Mathematics and Statistics

Jiaqiao Hu
Associate Professor, Department of Applied Mathematics and Statistics

Minghua Zhang
Professor, School of Marine and Atmospheric Sciences
Stony Brook University

This dissertation is accepted by the Graduate School

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Approximating Partially Observable Markov Decision Processes with Parametric Belief Distributions for Continuous State Spaces

by

Timothy Knapik

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2012

This dissertation focuses on training autonomous agents to plan and act under uncertainty, specifically for cases where the underlying state spaces are continuous in nature. Partially Observable Markov Decision Processes (POMDPs) are a class of models aimed at training agents to seek high rewards or low costs while navigating a state space without knowing their true location. Information regarding an agent's location is gathered in the form of possibly nonlinear and noisy measurements as a function of the true location. An exactly solved POMDP allows an agent to optimally balance seeking rewards and seeking information regarding its position in state space.

It is computationally intractable to solve POMDPs for state domains that are continuous, motivating the need for efficient approximate solutions. The algorithm considered in this thesis is the Parametric POMDP (PPOMDP) method. PPOMDP represents an agent's knowledge as a parameterised probability distribution and is able to infer the impact of future actions and observations.

The contribution of this thesis is in enhancing the PPOMDP algorithm making significant improvements in training and plan execution times. Several aspects of the original algorithm are generalized and the impact on training time, execution time, and performance are measured on a variety of classic robot navigation models in the literature today. In addition, a mathematically principled threefold adaptive sampling scheme is implemented. With an adaptive sampling scheme the algorithm automatically varies sampling according to the complexity of posterior distributions. Finally, a forward search algorithm is proposed to improve execution performance for sparse belief sets by searching several ply deeper than allowed by previous implementations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans are constantly making decisions that incur a cost or bestow a reward in the form of money, time, or a resource of some other form. In addition to the immediate costs/rewards, their decision will likely also have an impact on the future dynamics of the system. This requires one to not only think about immediate gain, but also consider how their current action will affect their ability to select beneficial actions in the future. With this in mind we consider replacing the human controller with an artificial intelligence or autonomous controller.

Markov Decision Processes (MPDs) provide a mathematical framework for balancing short-term versus long-term cost/reward, while also incorporating statistical uncertainty present in reality. Partially Observable Markov Decision Processes (POMDPs) are a logical generalization to the MDPs that adds an additional layer of uncertainty in the agent's knowledge of the true state of the system. While these models may describe how many problems in reality are perceived, the solutions in general are restrictive in terms of computation, memory, and most are considered completely intractable. Nonetheless, MDPs and POMDPs have been studied for decades and a rich mathematic foundation describing their applications, properties, variants, and solutions are well documented. These contributions, along with advances in computation power, have paved the way for a broad class of approximate algorithms to overcome the intractability of closed-form solutions. The remainder of this chapter will introduce in more detail: MDPs, POMDPs, and approximation algorithms.

1.1 Markov Decision Processes

A Markov Decision Process provides a mathematical description of how an agent may interact with a state space. The state space of the world may be discrete, or in many cases continuous. Similarly, the set of actions available to the agent may be defined as discrete or continuous. It is assumed that the agent is fully aware of its position in the state space, but state transitions due to actions are defined by a probability distribution. While the state transition is uncertain, the agent becomes fully aware of the updated state upon executing the action. In addition, a cost/reward is defined as a function of the agent's state and action in combination. For simplicity, it is assumed that the model evolves over discrete time intervals coinciding with the agent's actions.

The state dynamics governed by the actions may admit a large class of models, but must adhere to a Markovian restriction. That is, the future state is conditionally independent of all previous states and actions given the current state and action.

The goal is to train an agent to be fully autonomous and select intelligent actions to increase its long term reward. The simplest policy, or rather greediest policy, is to have the agent select the action with the highest immediate gain. It would be easy to construct a scenario where the action with largest reward leads the agent to a state where any future rewards are small or zero.

With this in mind, two standard optimality criterion have been defined to circumvent the issue. The first being the total expected discounted reward, where rewards earned in the future geometrically decrease with the passage of time. The second is the total expected average reward.

There are three standards ways of analyzing an MDP so that the agent may select the best action to maximize (or minimize in terms of cost) the optimality criteria. Value iteration is a dynamic programming algorithm that successively approximates the expected future rewards from each state. An agent with these values can implicity decide which actions are optimal from each state. Policy iteration assigns an action to each state that is, unless incredibly fortunate, suboptimal initially. Through a sequence of incremental improvements the actions taken from each state are modified until the algorithm converges to an optimal policy. Lastly, the solution may also be formulated as a Linear Program where the dual program's solution will yield the optimal policy. See [2] for Bellman's seminal work on MDPs, refer to [3] as an introductory textbook covering value and policy iteration, and for details on formulating policies via linear programs refer to [4] and [5] for details on policy iteration.

MDPs have been used as models in a wide variety of applications. In particular, they have been applied to solving problems in network flow, telecommunications, inventory stock management, finance, robot navigation, and many others [6, 7, 8].

1.2 Partially Observable Markov Decision Processes

Partially Observable Markov Decision Processes generalize MDPs in a natural way similar to how humans perceive reality. While an agent governed in a MDP world is uncertain where it will transition to while performing an action, the true state will be illuminated as the action is completed. The agent in a POMDP world has similar characteristics, with the exception that the true state of the world is never directly revealed. Instead, noisy observations defined by a known probability distribution as a function of the true state are made and received upon completion of each action. The agent must then use the observations to deduce its true state and to make informed decisions to maximize its reward.

With MDPs, the only necessary knowledge was the current state of the system which is always known. Past actions and rewards have no bearing on its future decisions. POMDPs still have state transitions according to the Markovian property, but without perfect knowledge of its current state the agent must consider the full history of actions and observations [9, 10]. The storage of information may only grow linearly, and this is a departure from the simple memory requirements of an MDP.

Many algorithms solve the problem of a growing memory requirement by calculating a belief distribution over the set of allowable states that serves as a sufficient statistic for the complete history. This distribution is then updated as actions are taken and observations are observed. Since the belief is serving as a sufficient statistic it conveys all the information gained from previous actions and observations and they then can be discarded. With this in mind the belief distribution is completely observable and perfectly known to the agent. With a small amount of work the POMDP can be transformed into an equivalent information state MDP where standard techniques can be applied. There are, however, two nontrivial aspects to the new MDP we must consider:

1. If the original state space is discrete with n states, the information state space will be continuous over an n-dimensional simplex.

2. If the original state space is continuous, then the information state space becomes infinite-dimensional over the set of all probability distributions over the original state space.

For the first case, we will only be able to use the results from studying continuous state MDPs which will are solvable, but naturally more complex in terms of computation than their discrete MDP counter parts. For the second case, an analytical solution is going to be completely intractable [11, 12]. Some concessions must be made to simplify the model in order to compute an approximation, but at the same time retain enough complexity so as to train a capable agent. Two common restrictions used in conjunction are:

1. Restrict the allowable belief distributions to a family that this is similar to what an agent may actually observe in practice.

2. Discretize the parameters of the chosen belief distribution rather than explicitly discretizing the continuous state space of the model.

Combining both of this methods, as outlined in [8**?**], results in a dramatic reduction in the dimensionality of the problem at the cost of an analytical solution. An explanation of this algorithm can be be explained with an example.

Consider giving an agent the task of navigating to a location in a two dimensional space. Suppose the state space is set of all allowable locations in this continuous space, and that there are potentially obstacles between the agent and the goal state. The set of actions will then take the form of movements in some direction. Recall that the converted MDP state space is the set of beliefs where the agent may be located in the state space. The simplest belief distribution to consider may be 100% certainty in the agent's location. This would be an ideal belief to have, but incredibly unlikely or unreachable without direct and perfect observations of the true state of the agent, even with any combination of actions. This type of belief, while desirable, falls into the category of being completely irrelevant. Without too much thought, we can imagine an infinite number of beliefs that are both incredibly unlikely and undesirable. Imagine an agent's belief that when visualized spelled your name across the state space. While this may appear comical (and is), these types of beliefs demand to be considered when solving for the optimal policy. We are, however, interested in an approximation. We have considered up until this point ideal but unreachable beliefs, and completely absurd beliefs. The question is then, *what types of beliefs is the agent likely to have in practice?*. This answer is certainly a function of the accuracy and nature of the observations made of the true state, but these specifics can be ignored for now.

Imagine yourself wandering around your apartment at night and due to a storm the power has gone out. You (carefully) make your way towards the location of a flashlight holding your arms in front of you sensing for nearby objects that can give you insight into your position. How would you then describe where you are at some point in time? A reasonable guess may be to answer with an exact location and an ellipse around this location indicating your uncertainty. If it has been some time since contact with any objects you may have a rather large ellipse, while if you (unfortunately) just walked into a wall this ellipse may be rather narrow. The two dimensional Gaussian distribution describes these set of beliefs quite well with only a few parameters, 2 for the mean, and 3 for the variances and covariance. Thus, the set of beliefs can restricted to a 5-dimensional continuous space rather than one that is infinite-dimensional.

While the reduction in state space is quite significant by only allowing a Gaussian belief, we can briefly discuss a potential shortcoming. Suppose through your travels with the flashlight you bump into a painting on the wall. At first you feel confident in the knowledge of your position until you realize that you have two paintings in two far away places in your apartment. Being restricted to a single Gaussian belief there are three likely options:

1. Believe you are located around painting #1.
2. Believe you are located around painting #2.
3. Create a Gaussian belief with a large ellipse encompassing both paintings.

Unfortunately, all three are these do not accurately represent what has been observed given the limits of a single Gaussian belief. This could be circumvented by allowing two (or more) Gaussian beliefs, but then we quickly lose the benefits of the dimension reduction. This toy example does not serve to discount the reduced belief space by using a single Gaussian, but more to illustrate potential pitfalls.

The transformation of the state space to a belief space of distributions will require us to reconsider two components as we transform this problem into an MDP:

1. What reward does an agent receive for having a belief distribution?
2. How does the agent transition between distributions and how are these probabilities calculated?

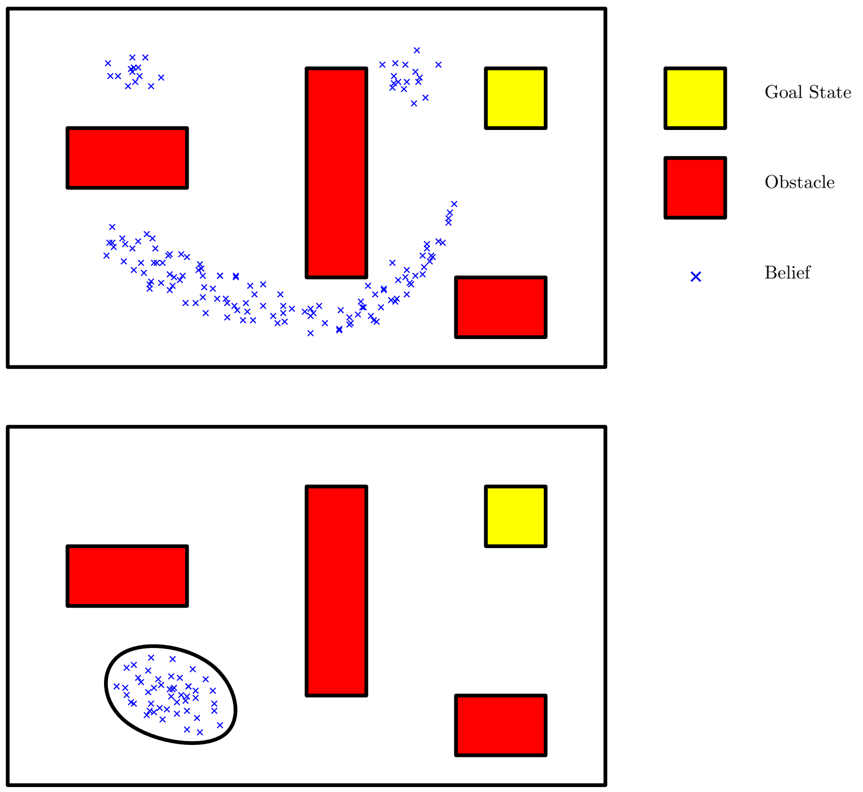


Figure 1.1: Both figures represent an environment with obstacles. The blue crosses represent possible locations an agent may believe to exist. The top figure represents clearly an unlikely set of beliefs to be held by an intelligent agent, while the bottom indicates a set of believes similar to that as samples from a two dimensional Gaussian distribution.

Both questions can be answered in the terms of sampling. The new rewards can be estimated by sampling from the belief distribution multiple times and taking the average reward according to the original reward function. The transitions will be slightly more complicated due to the possibly nonlinear nature of the state update equations. We will see how this can be overcome with the advances in filtering in the next section.

1.3 Particle Filters

Filtering is the process of approximating the unobserved state of a system given a history of noisy observations that are functions of the true system state. The evolution of the state typically follows a Markov process, that is given the current state, the future state follows a known probability distribution and it independent of the earlier state history. Similarly, the observation conforms to a known distribution given the current state and is independent of the past. Rather than explicitly estimate the value of the true system state, the conditional distribution of the current state given the full history of observations is preferred. This distribution is well known in analytical form due to the Chapman-Kolmogorov equation, but rarely admits a closed form solution with a few exceptions [13].

Filters in general sequentially estimate the conditional state distribution in two steps:

1. Predict the future state distribution using the system dynamics.
2. Update the distribution with the information gained from the observation.

The filter then proceeds in a recursive manner, following the above two steps as observations are received and then discarding them. This mitigates some of the computational burden be removing the need to store a long history of measurements and compute a conditional distribution as a function of all measurements [14].

The well known Kalman Filter was first introduced in 1960 and solved the filtering problem in closed form under the assumption of linear systems and Gaussian noise [15]. The Extended Kalman Filter (EKF) was developed later as an approximate solution for nonlinear dynamics and non-Gaussian noise. The EKF essentially linearizes the system by taking first order partial derivatives and then applies the Kalman Filter. While the EKF is computationally efficient, it does suffer from convergence issues and is limited in application. Another method applied to nonlinear dynamics is a grid-based approach, effectively discretizing the state space into cells and then approximating the model conditional density. The number of cells increases quickly as the dimensionality of the state space grows limiting the tractability of this approach. Arulampalam gives a brief account of these techniques in [16].

Particle filtering aims to solve the problems associated with nonlinear systems and non-Gaussian noise by representing the conditional density as a discrete set of weighted samples. The Particle Filter (PF) is based on results from sequential Monte Carlo (MC) methods. Typically the true density in question cannot be sampled from, and particles are instead generated according to an importance density, which ideally closely resembles the true posterior. The weights assigned to the particles are selected to represent the true distribution, and then reweighted as observations are processed. It can be shown that as the number of particles approaches infinity this method converges to the true conditional density. There are, however, a few shortcomings in this method. In particular it often suffers from sample impoverishment, that is a lack of diversity in the particles. Resampling and Monte Carlo Markov Chain methods are techniques designed to overcome this issue [13]. Particle Filters will play an important role in approximating transitions between belief distributions as well as our adaptive sampling scheme.

1.4 Contributions

The main focus of this thesis is improving techniques to approximate solutions to POMDPs with continuous state spaces. The technique advocated and improved in this thesis is the Parametric POMDP (PPOMDP) algorithm. Other competitor techniques include the discrete MDP method and Perseus which are both explained in section 4. The discrete MDP method partitions the continuous state space into a discrete number of cells. It then approximates transition probabilities and the effects of discrete measurements on the converted state space. A major deficiency in this method is not being able to incorporate continuous observations into its solution. Instead, continuous observations must be discretized into cells similar to the state space discretization. The Perseus algorithm selects a finite amount of candidate locations in the continuous state space. The POMDP is then converted to a continuous state MDP with a state vector of size equal to the number of candidate locations initially selected. This formulation allows for MDP techniques to be applied at the cost of a new state space that is likely of very high dimension. The PPOMDP technique advocated allows for continuous observation spaces and does not suffer from high dimensional state space conversions by representing an agent's belief with a parameterised distribution. It is, however, a relatively new technique with plenty of room for improvement. The contributions in this to this algorithm and in this thesis are as follows:

- The testing of efficiency and performance of the PPOMDP algorithm on a variety of practical models. (section 4.3).
- The generalization of the PPOMDP algorithm to allow non-square weighting matrices to allow a reduction in the number of future sampled belief states (section 4.3).

- The application of Kullback-Liebler-distance (KLD) sampling to allow adaptive sampling in across models and during the training of the agent. The KLD algorithm was also novelly adapted for PPOMDP and is merges the sampling of particles and future beliefs in a mathematically principled manner (section 4.4).

- The implementation of a novel forward search allowing for deeper searches in less computation time. This allows an agent to train on sparser belief sets while still collecting competitive rewards.

Chapter 2

Filtering

Filtering considers the problem of estimating the state of a system and its possibly hidden parameters with a set of noisy observations. It is assumed that the underlying mechanics governing the evolution of the system are known, but possibly stochastic in nature. Similar requirements are met for the observations as well. In general, if we let $x_t \in R^{n_x}$ be the state of the system at time t and $z_t \in R^{n_z}$ be the measurement obtained at time t, the transition probability distribution is given by $p(x_t|x_{t-1})$ and the measurement probability distribution $p(z_t|x_t)$. Note that the transition equation is Markovian, future states are only dependent on the most recent. In general we may write

$$\begin{aligned} x_t &= f(x_{t-1}, w_{t-1}) \\ z_t &= h(x_t, u_{t-1}, v_{t-1}) \end{aligned} \tag{2.1}$$

where $w_t \in R^{n_v}$ denotes a vector of process noise, $u_t \in R^{n_u}$ denotes input to the system, and $v_t \in R^{n_v}$ denotes measurement noise. The initial distribution of the system state, $p(x_0)$ is assumed given. If we let $x_{0:t} = \{x_0, x_1, \cdots, x_t\}$ and $z_{1:t} = \{z_1, z_2, \cdots, z_t\}$, then the aim of a filter is to estimate the distribution $p(x_{0:t}|z_{1:t})$, or the marginal distribution $p(x_t|z_{1:t})$. Ideally the marginal distributions may be computed recursively using the previous estimate to update the present estimate. This allows us to discard observations as they are processed.

Excluding very specific constraints, closed form solutions for calculating the filtered distributions do not exist. Kalman, in [15] was among the first to develop a closed form solution for systems with linear dynamics and Gaussian noise, aptly named the Kalman Filter (KF). Following this work, the Extended Kalman Filter (EKF) was derived as an approximation for nonlinear models. The EKF, at the time, was a breakthrough in nonlinear filtering, but suffered issues with convergence [17]. The particle filter (PF), and its many offspring, came later as a new filtering technique attempting to mitigate the problems exhibited in the EKF, and is able to be applied to any nonlinear dynamics and non-Gaussian distributions.

2.0.1 Kalman Filter

The Kalman Filter in [15] assumes a very specific model in the form of

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_{t-1} + w_{t-1} \\ z_t &= Hx_t + v_t \end{aligned}$$

where A, B, and H are appropriately sized and known matrices. The noise processes w_t and v_t are restricted to be multivariate Gaussian given by

$$\begin{aligned} p(w) &\sim N(\mathbf{0}, Q) \\ p(v) &\sim N(\mathbf{0}, R) \end{aligned}$$

where Q and R are known covariance matrices.

Using the KF can be broken down into two steps: prediction and update.

Let $\hat{x}_t^-$ be the state estimate before the observation z_t is considered, and $\hat{x}_t$ be the estimate after z_t is considered. Similarly, let P_t^- be the covariance matrix representing the uncertainty in the estimate $\hat{x}_t^-$, and let P_t be the covariance matrix representing the uncertainty in the estimate $\hat{x}_t$. Assume at time t the estimate $\hat{x}_{t-1}$ and covariance matrix P_{t-1} are known. Then the two step process to obtain $\hat{x}_t$ and P_t is given by:

- Predict

$$\begin{aligned}\hat{x}_t^- &= A\hat{x}_{t-1} + Bu_t \\ P_t^- &= AP_{t-1}A^T + Q\end{aligned}$$

- Update

$$\begin{aligned}\hat{y} &= z_t - H\hat{x}_t^- \\ S_t &= HP_t^-H^T + R \\ K_t &= P_t^-H^TS_t^{-1} \\ \hat{x}_t &= \hat{x}_t^- + K_t\hat{y} \\ P_t &= (I - K_tH)P_t^-\end{aligned}$$

While this was (and still is) an important result at the time, the class of problems it can be applied is quite limited.

2.0.2 Extended Kalman Filter

The Extended Kalman Filter was introduced shortly after the KF in 1966 and was applied to navigating and tracking objects in space [18]. The state estimate is found by effectively linearizing the transition and measurement dynamics via a Taylor series approximation. The models where the EKF may be applied are much less restrictive than previously explored, at the cost of an approximate solution over a closed form. Assume the transition and observation models take the form:

$$
\begin{aligned}
x_t &= f(x_{t-1}, u_{t-1}) + w_{t-1} \\
z_t &= h(x_t) + v_t
\end{aligned}
$$

where $f(x_{t-1}, u_{t-1})$ and $h(x_t)$ are potentially nonlinear functions, while w_t and v_t are still restricted to multivariate Gaussians. We may linearize $f(x_{t-1}, u_{t-1})$ and $h(x_t)$ by taking the Jacobians:

$$
\begin{aligned}
A_t &= \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1}, u_{k-1}} \\
H_t &= \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_t^-}
\end{aligned}
$$

The EKF then also proceeds in the recursive manner as before with:

- Predict

$$
\begin{aligned}
\hat{x}_t^- &= f(\hat{x}_{t-1}, u_{t-1}) \\
P_t^- &= A_{t-1} P_{t-1} A_{t-1}^T + Q
\end{aligned}
$$

- Update

$$
\begin{aligned}
\hat{y} &= z_t - H_t \hat{x}_t^- \\
S_t &= H_t P_t^- H_t^T + R \\
K_t &= P_t^- H_t^T S_t^{-1} \\
\hat{x}_t &= \hat{x}_t^- + K_t \hat{y} \\
P_t &= (I - K_t H_t) P_t^-
\end{aligned}
$$

This is nearly identical to the steps for the KF, except the transition and measurement equations have been linearized with the matrices A_t and H_t. Note, the prediction step uses the nonlinear function $f(x_{t-1}, u_{t-1})$ rather than using the approximation.

The KF and EKF both use Gaussian distributions to represent the state estimate. In same cases the EKF is known to fail, particularly if the underlying nonlinear state model results in severely skewed or multimodal distributions [19].

2.0.3 Particle Filter

Particle filters (PFs) are a class of algorithms following a sequential Monte Carlo (SMC) approach to estimating the the distribution $x_t|z_{1:t}$ for the general state and observation model defined by 2.1. The KF was limited to Gaussian disturbances and linear models, while the EKF expanded the filtering toolbox to nonlinear models, but also required Gaussian disturbances. PFs provide an approximate filtering solution for any transition and observation equation, and any noise disturbance assuming that 2.1 is known in probabilistic form [20]. Essentially, a PF provides a means to represent and recursively update $p(x_t|z_{1:t})$ as a cloud of weighted points over the state space. As the number of points tends towards infinity it becomes equivalent to an explicit representation of the posterior density we are interested in [21]. First, we decompose the prior pdf via the Chapman-Kolmogorav equation before the current observation has been considered:

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1} \tag{2.2}$$

Then the updated pdf may be written

$$p(x_t|z_{1:t}) = Cp(z_t|x_t)p(x_t|z_{1:t-1}) \tag{2.3}$$

where $C = p(z_t|z_{1:t-1}) = \int p(z_t|x_t)p(x_t|z_{1:t-1})dx_t$ is a normalizing constant. In general, except for restrictive conditions, a closed form solution for 2.3 is unknown, but this does serve as a starting point for developing a mathematically sound approximation.

Formally we may represent a particle set with N samples as $\{x_t^i : i = 1, \cdots, N\}$ with associated weights $\{w_t^i : i = 1, \cdots, N\}$ where x_t^i are points in the state space and the weights satisfy that $\sum_{i=1}^N w_t^i = 1$. Then a weighted approximation to the posterior distribution can be written

$$p(x_t|z_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta(x_t - x_t^i). \tag{2.4}$$

The accuracy of the approximation given by 2.4 is a function of the selection of sample points, their corresponding weights, and number of samples. Ideally, 2.4 will be most accurate if the samples are drawn according to $x_t^i \sim x_t|z_{1:t}$ and setting the weights to $w_t^i = \frac{1}{N}$. This distribution, however, it is typically challenging to sample from directly. Instead, samples will be drawn from an *importance density* $q(\cdot)$ where we hope this distribution matches the posterior closely and can be sampled from easily. The weighting assignment then takes the form

$$w_t^i \propto \frac{p(x_t^i|z_{1:t})}{q(x_t^i|z_{1:t})}. \tag{2.5}$$

Assuming mild conditions on the importance density, the weights can be recursively updated according to

$$w_t^i \propto w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, z_t)}. \tag{2.6}$$

For more details on this derivation see [16]. The recursive nature of 2.6 allows us to obtain and discard observations rather than retain a full trajectory. The question remains how is the important density selected. The most straightforward choice would be to set $q(x_t^i|x_{t-1}^i, z_t) = p(x_t^i|x_{t-1}^i)$, that is the transition equation given by 2.1. Doing so simplifies 2.6 to $w_t^i \propto w_{t-1}^i p(z_t|x_t^i)$. Essentially this weights the particles with states able to produce similar observations as the one received, z_t, highly. It does not, however, factor the current observation z_t into the sampling distribution. If the two distributions, $p(x_t|x_{t-1}, z_t)$ and $p(x_t|x_{t-1})$, differ greatly then a large number of samples will be needed to ensure a good approximation. Techniques to avoid this include applying a local linearization to the state space model, similar to the EKF to create a more accurate importance distribution able to incorporate the information of the observation z_t [22]. Particle filters updating sample sets according to 2.6 are refered to as sequential important sampling (SIS) algorithms and form a basis for more advanced techniques.

Algorithm 1 A basic Sequential Importance Sampling (SIS) algorithm. The algorithm recursively updates the particle weights and discards observations sequentially. The algorithm terminates after a user specified number of iterations.

```
Sample $x_0^1, x_0^2, \cdots, x_0^N$ according to an initial distribution $p_0(x)$
$t \leftarrow 0$
Set $w_t^i = \frac{1}{N}$ for $i = 1, \cdots, N$
while $t < MaxIterations$ do
    $t \leftarrow t + t$
    Receive observation $z_t$
    Sample $x_t^i \sim q(x_t | x_{t-1}^i, z_t)$, for $i = 1, \cdots, N$
    Set $w_t^i \leftarrow w_{t-1}^i \frac{p(z_t|x_t^i) p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, z_t)}$ for $i = 1, \cdots, N$
    Set $C = \sum_{i=1}^N w_t^i$
    Normalize $w_t^i \leftarrow \frac{w_t^i}{C}$
end while
```

The SIS algorithm is an important step towards solving approximating posterior densities of any functional form and under the assumption of noise variables, but it does suffer setbacks. In particular, there is the well documented *sampling impoverishment* or *sampling degeneracy* problem [16, 23, 24, 25]. The degeneracy problem occurs when the overwhelming majority of weight belongs to a single particle, and the remaining particles have essentially zero weight. A particle set in this extreme case is effectively no longer a cloud representation of the posterior distribution, but more of a point estimate. The number of particles in the set is still N, but effectively is 1. Liu introduced, in [26], the *effective sample size* , N_{eff}, to measure how many particles were effectively in the particle set as a function of the particle weights:

$$N_{eff} = \frac{1}{\sum_{i=1}^N (w^i)^2} \tag{2.7}$$

4.9 is is maximized to N when each of the weights are equal to $\frac{1}{N}$ and is reduced to 1 when all of the weight belongs to a single particle. A common technique to overcome sample impoverishment is to resample particles with replacement from the current particle set when N_{eff} falls below some threshold N_T. Some prefer not to set a threshold and resample each iteration no matter the effective sample size. The weights are then reset back to $\frac{1}{N}$. Particles with high likelihood will be sampled more frequently concentrating the cloud of points on these areas. Particle filtering algorithms resampling in this manner are referred to as Sequential Importance Sampling (SIR) algorithms.

The research of particle filters will play a pivotal role in our adaptation of the PPOMDP as we will use these technique when approximating transition distributions. Also, the notion of effective sample size will play a part in decreasing computation as part of our adaptive sampler.

Algorithm 2 A basic Sequential Importance Resampling algorithm. The particles are resampled with replacement from the current particle set if the effective sample size becomes too low.

```
Sample $x_0^1, x_0^2, \cdots, x_0^N$ according to an initial distribution $p_0(x)$
$t \leftarrow 0$
Set $w_t^i = \frac{1}{N}$ for $i = 1, \cdots, N$
**while** $t < MaxIterations$ **do**
    $t \leftarrow t + t$
    Receive observation $z_t$
    Sample $x_t^i \sim q(x_t | x_{t-1}^i, z_t)$, for $i = 1, \cdots, N$
    Set $w_t^i \leftarrow w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, z_t)}$ for $i = 1, \cdots, N$
    Set $C = \sum_{i=1}^N w_t^i$
    Normalize $w_t^i \leftarrow \frac{w_t^i}{C}$
    Calculate $N_eff = \frac{1}{\sum_{i=1}^N (w_t^i)^2}$
    **if** $N_{eff} < N_T$ **then**
        Sample $\hat{x}_t^i$ from $\{x_1, x_2, \cdots, x_N\}$ with $P(\hat{x}_t^i = x_j) = w_t^j$ for $i = 1, \cdots, N$

        Set $x_i \leftarrow \hat{x}_t^i$ for $i = 1, \cdots, N$
        Set $w_t^i = \frac{1}{N}$ for $i = 1, \cdots, N$
    **end if**
**end while**
```

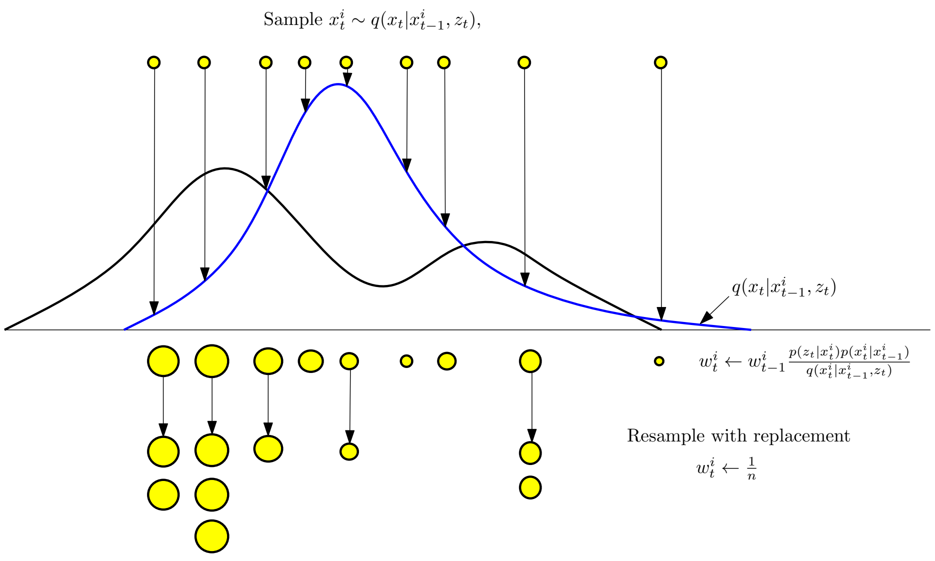

Sample $x_t^i \sim q(x_t|x_{t-1}^i, z_t)$,
$q(x_t|x_{t-1}^i, z_t)$
$w_t^i \leftarrow w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i,z_t)}$
Resample with replacement
$w_t^i \leftarrow \frac{1}{n}$

Figure 2.1: An illustration of the weighting process in a particle filter. The top yellow circles represent particles sampled from the importance distribution $q(x_t|x_{t-1}^i, z_t)$. The weights are assigned, with larger circles representing larger weights. Finally, the particles are resampled with replacement.

Chapter 3

Stochastic Control

A stochastic controller aims to make optimal decisions to minimize costs under uncertainty. Two powerful stochastic models are Markov decision processes and the generalization, partially observable Markov decision processes. This chapter will define both models and review the various solution methods.

3.1 Markov Decision Processes

A controller in a Markov decision process (MDP) at time k is assumed to be in state $x_k \in X$, select action $u_k \in U$, and transition stochastically to the future state x_{k+1} according to:

$$x_{k+1} = f(x_k, u_k, w_k)$$

where $w_k \in D$ is a random variable.

Formally we may define an MDP with the following:

1. The state space X

2. The action space U

3. The set of actions from available from state x_k as $U(x_k) \subseteq U$

4. $p(x_{k+1}|x_k, u_k)$, the next state transition probability given an action

5. $g(x_k, u_k, w_k)$, the immediate cost of taking action u_k while in state x_k

6. $p(w_k|x_k, u_k)$, the probability of w_k given the current state and action.

Starting from an initial state x_0, the goal of the controller is to form a policy $\pi = \{\mu_1, \mu_2, \cdots\}$ which describes a sequence of functions to select allowable actions from states at discrete times. That is, let $\mu_k : X \to U$ and assume that $\mu_k(x_k) \in U(x_k)$. The k subscript indicates time. Policies that are independent of time are called *stationary policies* and can be written $\pi = \{\mu, \mu, \cdots\}$, or referred to simply by μ. Let Π be the set of all policies.

The goal is to select optimal policies with respect to a cost criterion. We will consider the infinite-horizon discounted cost and the undiscounted average cost.

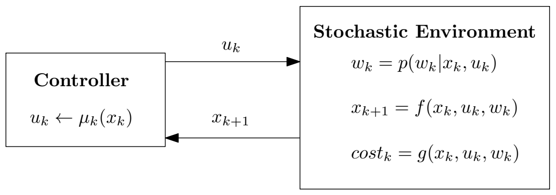

u_k
Controller
$u_k \leftarrow \mu_k(x_k)$
x_{k+1}
Stochastic Environment
$w_k = p(w_k|x_k, u_k)$
$x_{k+1} = f(x_k, u_k, w_k)$
$cost_k = g(x_k, u_k, w_k)$

Figure 3.1: This illustrates the MDP model. A policy function selects the next action based only upon the currently known state. The noisy variable w_k is generated and the state stochastically transitions to x_{k+1}

3.1.1 Discounted cost

Let $\gamma \in (0, 1)$ be a discount factor that lowers the value of future costs. Let $J_\pi(x_0)$ be the total expected discounted sum of costs starting from state x_0 and following policy π be

$$J_\pi(x_0) = \lim_{n\to\infty} E_{\{w_k\}_{k=0}^n} \left[\sum_{i=0}^{n-1} \gamma^n g\left(x_k, \pi(x_k), w_k\right) \right].$$

Let J^* represent the *optimal cost function* defined by

$$J^*(x) = \min_{\pi \in \Pi} J_\pi(x), x \in X$$

and let π_* be the *optimal policy*

$$\pi_*(x) = arg \min_{\pi \in \Pi} J_\pi(x), x \in X$$

Let $J : X \to \mathbb{R}$ and define the operator T such that:

$$(TJ)(x) = \min_{u \in U(x)} E_w \left[g(x, u, w) + \gamma J \left(f(x, u, w) \right) \right].$$

We can think of as J representing the cost of being in each state, and T being a function that calculates the optimal cost one step into the future due to the minimization over allowable actions. We can define a similar operator T_μ given a stationary policy μ such that

$$(T_\mu J)(x) = E_w \left[g(x, u, w) + \gamma J \left(f(x, \mu(x), w) \right) \right].$$

$T_\mu J$ is then the one stage cost of using policy μ if the initial costs are determined according to J. For notation convenience, we will define the composition of T and T_μ as

$$(T^k J)(x) = (T(T^{k-1} J))(x), x \in X$$
$$(T_\mu^k J)(x) = (T_\mu(T_\mu^{k-1} J))(x), x \in X$$

Assumption 1. *The cost function is bounded above and below*

$$|g(x, u, w)| \leq M, \forall x \in X, u \in U(x), w \in D.$$

We also restrict X, U, and D to be discrete sets. The following results have continuous analogs but require a much more complex mathematical treatment.

Proposition 1. *For any function J*

$$J^*(x) = \lim_{n\to\infty} (T^n J)(x), \forall x \in X.$$

This proposition affirms the convergence of any function J to the optimal cost function J^* after infinite operations by T. Using similar reasoning:

Corollary 1. *For any policy μ*

$$J_\mu(x) = \lim_{n\to\infty} (T_\mu^n J)(x), \forall x \in X$$

is the associated cost function for following policy μ

The following states that the optimal cost function can be written recursively rather than in limit form.

Proposition 2. *The optimal cost function satisfies*

$$J^*(x) = \min_{u\in U(x)} E_w \left[g(x, u, w) + \gamma J^* \left(f(x, u, w)\right)\right]$$

or more succinctly

$$J^* = TJ^*.$$

J^* is the unique solution to either of these equations.

Similarly, the costs from following some policy:

Corollary 2.

$$J_\mu(x) = E_w \left[g(x, u, w) + \gamma J_\mu \left(f(x, \mu(x), w)\right)\right]$$

or more succinctly

$$J_\mu = T_\mu J_\mu.$$

We see that the difference in form of J^* and J_μ lies in the absence of the minimization of actions (clearly as μ selects the actions). Calculating J_μ is much simpler and can be seen to be a system of linear equations with $|X|$ unknowns and $|X|$ equations. Thus, finding the infinite horizon costs for a given policy is relatively straightforward and solved with Gaussian elimination, but finding the infinite horizon cost of the optimal policy will be more of a challenge.

We can also make a statement about the requirements of an optimal policy.

Proposition 3. *A stationary policy μ is optimal if and only if $\mu(x) \forall x \in X$ the operator T_μ leaves J^* unchanged,*

$$TJ^*(x) = T_\mu J^*.$$

There are two widely accepted techniques to solve an MDP with discrete state and action space that attack the problem from two fronts:

1. Calculate the optimal cost function and infer the optimal policy.

2. Explore the space of policies until the optimal policy is found.

The following sections will describe these two techniques in more detail.

Value Iteration

Value Iteration (VI) is an algorithm that sequentially approximates J^* with dynamic programming. The idea is to begin with an initial function $J_0(x) = 0, \forall x \in X$. The algorithm then proceeds by successively calculating $TJ_0, T^2J_0, \cdots$.

Unfortunately, convergence is contingent upon an infinite number of iterations. It turns out, however, that the operator T is a contraction mapping and we can place a bound on the error between the optimal value function and $T^k J$ [7]. Define the norm on J as

$$||J||_\infty = \max_{x \in X} J(x),$$

then

$$||T^k J - J^*||_\infty \leq \gamma^k ||J - J^*||_\infty.$$

It can be seen that successive applications of the operator T geometrically decrease the error with respect to the maximum difference initially.

Algorithm 3 A simple implementation of the Value Iteration algorithm. The program continues until the maximum change of the value function is less than the user defined variable δ. The algorithm outputs the approximate value function as well as the corresponding policy.

```
$J(x) \leftarrow 0, \forall x \in X$
$\epsilon_{max} \leftarrow -\infty$
while $\epsilon > \delta$ do
    $J_{buff}(x) = J(x), \forall x \in X$
    $\epsilon_{max} \leftarrow -\infty$
    for $x \in X$ do
        for $u \in U$ do
            $J_{action}(u) = E_w\left[g(x,u,w) + \gamma J_{buff}\left(f(x,u,w)\right)\right]$
        end for
        $u_{min}(x) = \operatorname*{argmin}_{u \in U}\left[J_{action}(u)\right]$
        $J(x) \leftarrow J_{action}(u_{min}(x))$
        if $J_b(x) - J(x) > \epsilon_{max}$ then
            $\epsilon_{max} = J_b(x) - J(x)$
        end if
    end for
end while
```

It is important to note the VI requires a loop over the entire state space and action space. Finding optimal policies when one or both spaces are continuous is much more challenging and usually requires approximation. The PPOMDP algorithm in particular makes use of VI as it approximates continuous state POMDPs with discrete state MDPs where VI can be applied.

Policy Iteration

Recall that VI is guaranteed to converge to the true optimal value function at a geometric rate as a function of γ, but suffers from needing infinite applications of the operator T. This holds even for discrete state and action spaces. An alternative approach is policy iteration (PI), that is to sequentially modify an initial policy μ^0 and into a series $\mu^0, \mu^1, \mu^2, \cdots$ where each successive policy is an improvement over its predecessor. Essentially there are two steps that are repeated in succession, policy evaluation and then policy improvement. Given a discrete state and action space, the set of all policies is also discrete. Define a one step updated policy as

$$\mu^{k+1}(x) = arg \min_{u \in U(x)} E_w \left[g(x, \mu^k(x), w) + \gamma J_{\mu^k}(f(x, \mu^k(x), w) \right] .$$

Recall that J_μ for any policy can be solved via a system of linear equations. Essentially we are checking to see if any actions can replace those specified by μ^k that will reduce the long term cost. Formally it can be shown that by generating policies with the above:

Proposition 4. *For all k and x*

$$J_{\mu^{k+1}}(x) \leq J_{\mu^k}(x)$$

If the policy does not change after an iteration, that is $J_{\mu^{k+1}}(x) = J_{\mu^k}(x) \forall x \in X$, then the policy has converged to the optimal policy. Since the number of policies is discrete this algorithm also terminates after a discrete number of iterations.

Algorithm 4 A simple implementation of the Policy Iteration algorithm. The algorithm terminates after there is no change in policy and is then guaranteed to be optimal.

```
Generate a random policy $\mu$
**converged** $\leftarrow$ **false**
**repeat**
  Calculate $J_\mu$
  **for** $x \in X$ **do**
    $\mu_{updated} = arg\min_{u \in U(x)} E_w \left[ g(x, \mu(x), w) + \gamma J_\mu(f(x, \mu(x), w) \right]$
  **end for**
  **if** $\mu(x) = \mu_{updated}(x)\ \forall x \in X$ **then**
    **converged** $\leftarrow$ **true**
  **else**
    $\mu = \mu_{updated}$
  **end if**
**until converged**
```

Complexity of VI and PI

Both PI and VI suffer similar problems if the size of the state space is large. VI requires the calculation of an expectation inside a loop over all states and all actions results in a complexity on the order of $O(|X|^2|U|)$ operations per iteration, which is polynomial. The question is then how many iterations are necessary to find the optimal policy given a value of γ. This is a difficult question to answer, but it was shown by Littman in [27] that the number of iterations is proportional to $\frac{1}{1-\gamma} \log\left(\frac{1}{1-\gamma}\right)$. This term may grow quite large for values of γ near 1. Policy iteration also suffers from similar problems due to large state spaces. There is a similar loop over actions and states with an expectation resulting in a complexity of $O(|X|^2|U|)$ as well, but additionally the value function J_μ for the current policy μ must be calculated for each iteration. Recall that finding J_μ is the solution to a linear system with $|X|$

equations and $|X|$ unknowns. The standard time to find J_μ is then $O(|X|^3)$ resulting in a total complexity for a single iteration of the PI algorithm as $O(|X|^3 + |X|^2|U|)$. The total number of possible policies is $|U|^{|X|}$ and thus discrete but exponential as a function of state space size. We can be assured that PI will terminate in a finite amount of iterations, hopefully at a value far less than $|U|^{|X|}$.

Other Methods

The exact solutions to finding an optimal policy or value function serve as a starting point for developing reasonable approximations or computation speedup to the existing algorithms.

One area of research that has gathered considerable attention is reinforcement learning, and in particular Q-learning, introduced by Watkins in 1989 [28]. This algorithm attempts to tackle an issue with the need to fully enumerate all states and actions in single iterations of the VI and PI algorithms. The basic idea is, instead of considering all state action pairs, an agent is placed in the MDP world and given the freedom to explore, learning to select good actions that minimize costs over the long term. The Q in Q-learning refers to the Q-functions $Q_\mu(x, u)$ for each $x \in X$, $u \in U$, and some policy $\mu \in \Pi$. The value of $Q_\mu(x, u)$ is the cost of executing action u from state x and following policy μ thereafter. Algebraically this can be written

$$Q_\mu(x, u) = E_w \left[g(x, u, w) + \gamma J_\mu(f(x, u, w)) \right]. \tag{3.1}$$

The class of Q-function we are interested in are those following the optimal policy μ_*. The optimal value function can be written in terms of Q as

$$J^*(x) = \min_{u \in U(x)} Q_{\mu_*}(x,u).$$

$Q_{\mu_*}(x,u)$ are the functions that are being approximated in the Q-learning algorithm [29].

In order to find an accurate approximation, the agent must balance two opposing paradigms for searching the environment, that is the exploration versus exploitation dilemma. An agent with a high rate of exploration will continually try new actions that may be unlikely to be part of the optimal policy. An agent with a high rate of exploitation usually selects greedy actions it knows are beneficial with respect to actions already tested. The first method of exploring will exhaustively consider each action (as the agent explores indefinitely) guaranteeing an optimal solution at the cost of a very slow rate of convergence. The second method may rapidly select a "good" policy, but on that is suboptimal. Under the appropriate transition from exploration to exploitation it can be shown that Q-learning will converge to the optimal policy [30]. For a comprehensive review of reinforcement learning techniques applied to MDPs refer to [30].

Other techniques are interested in retaining an exact solution while greatly decreasing the running time of the solution method. In particular we briefly discuss Topological Value Iteration (TVI) and Focused Topological Value Iteration (FTVI) [31, 32]. As the names suggests, these techniques focus on improving the basic VI algorithm. The exploits to improve time come from recognizing that stochastic transitions from states given actions can be modeled as a directed graph. TVI decomposes the directed graph into its strongly connected components (SCCs) and applies VI sequentially to each component.

Under appropriate model dynamics this decomposition allows the algorithm to terminate up to orders of magnitude quicker than basic methods [31]. For some models, though, actions can be reversed creating a graph that is a single large strongly connected component. Dai and Weld resolve this issue by preprocessing the MDP initially. Similar to Q-learning, the initial phase of FTVI has an agent randomly explore the world in an effort discard provably suboptimal actions [32]. Once actions are eliminated the resulting graphical structure may typically be decomposed in SCCs where TVI may be applied. It was shown in [32] that this technique outperforms similar techniques up to two orders of magnitude on a variety of benchmark problems.

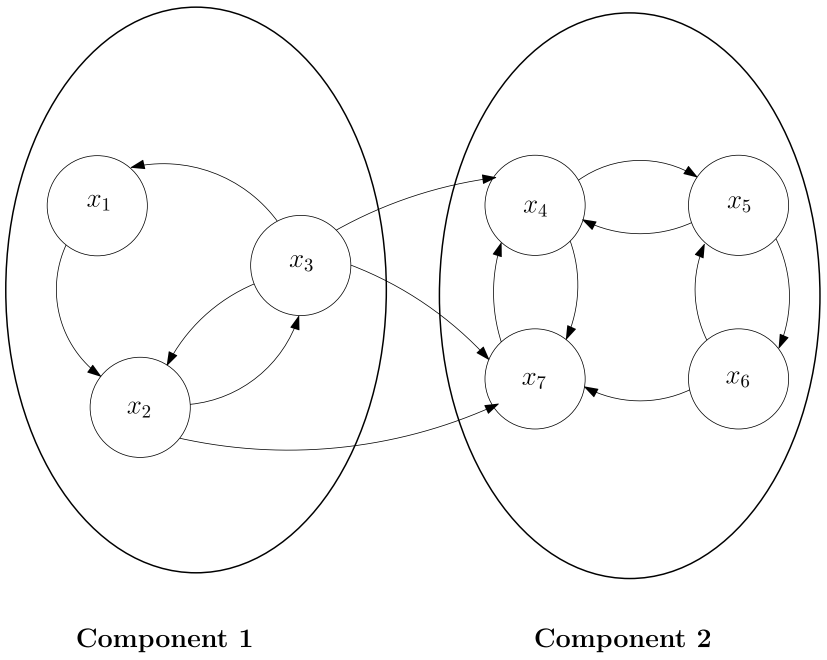


Figure 3.2: A graphical representation of an MDP decomposed into strongly connected components. Small circles indicate state. Arc represent transitions with positive probability from states assuming an action. The action labels have been omitted. The large ovals encapsulating the states indicate individual strong connected components.

3.2 Partially Observable Markov Decision Processes

An agent operating in an environment described by a POMDP adds an extra layer of uncertainty when compared with an MDP. The agent never directly observes its true state, but instead makes noisy observations a function of the unknown state. Formally we write:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, w_{k+1}) \\ z_k &= h(x_k, u_{k-1}, v_k) \end{aligned} \tag{3.2}$$

where f is the stochastic transition function as described for MDPs, h is the measurement function, and v_k is a random measurement disturbance with known distribution independent of transition disturbance w_k. Formally we may write:

1. The state space X
2. The action space U
3. The measurement space Z
4. $p(x_{k+1}|x_k, u_k)$, the next state transition probability given an action
5. $g(x, u)$, the immediate cost of taking action u while in state x
6. $p(z_{k+1}|x_{k+1}, u_k)$, the probability of observing z_{k+1} while in state x_{k+1} and having taken action u_k.
7. $p(w_{k+1}|x_k, u_k)$, the transition noise model.
8. $p(v_{k+1}|x_{k+1}, u_k)$, the measurement noise model.

With the noise dynamics for the measurement and transition defined we can write the transition probability as

$$p(x_{k+1}|x_k, u_k) = \int_{w_{k+1}} f(x_k, u_k, w_{k+1}) p(w_{k+1}|x_k, u_k) dw_{k+1} \tag{3.3}$$

and the measurement probability

$$p(z_{k+1}|x_{k+1}, u_k) = \int_{v_{k+1}} h(x_{k+1}, u_k, v_{k+1}) p(v_{k+1}|x_{k+1}, u_k) dv_{k+1}. \tag{3.4}$$

The complete information available to an agent at time k can be written as

$$I_k = \{z_0, u_0, z_1, u_1, \cdots, z_{k-1}, u_{k-1}, z_k\}, \tag{3.5}$$

that is a vector containing every action performed and observation observed. While the state itself is unknown at any time, the information vector I_k is always fully available to the agent. We may view this vector as an information-state or I-state. Let the space of information states be denoted with I.

Information known at time k stochastically transitions according the dynamics defined by the POMDP model. We denote the transition function f_I

$$I_{k+1} = f_I(I_k, u_k, z_{k+1}). \tag{3.6}$$

An agent with information I_k will never be able to view the true costs given actions without access to its actual state, therefore an approximation of the costs must be made. Let the approximated cost be g_I and the function G_I be

$$g_I = G_I(I_k, u_k). \tag{3.7}$$

The agent in a POMDP world needs a policy, $\pi_I : I \to U$, that maps information states to actions to minimize the expected cost. The expected cost given some policy π_I can be written

$$J_{\pi_I}(I_k) = \lim_{n\to\infty} E_I \left[\sum_{i=0}^{n-1} \gamma^n R_I \left(I_{k+i}, \pi(I_{k+i}) \right) \right], \tag{3.8}$$

with optimal policy π_I^*

$$\pi_I^*(I_k) = arg \min_{\pi_I} J_{\pi_I}(I_k). \tag{3.9}$$

Given an information space I, an information transition function f_I, and cost estimate R_I, the POMDP model is converted into what appears to be identical to an MDP. One pitfall, however, prevents us from finding optimal policies via previous discussed VI and PI techniques. The complete history I_k grows linearly with time as observations and actions are assimilated. Applying VI or PI to a state space that contains vectors growing in length indefinitely will be problematic. This does not erase the relationship between POMDPs and MDPs, but rather asks us to reconsider how information is stored over time to create a more manageable information state space.

3.2.1 Belief States

Since the true state of the system is unobserved in the POMDP model the agent must reason about its location purely from the recorded history of actions and observations, and its understanding of the system's dynamics. A standard technique is create a probability distribution over allowable states. Let $b_k(x)$ be the probability density function conditioned on the history of actions and observations up until time k for the location of the agent in state space. We

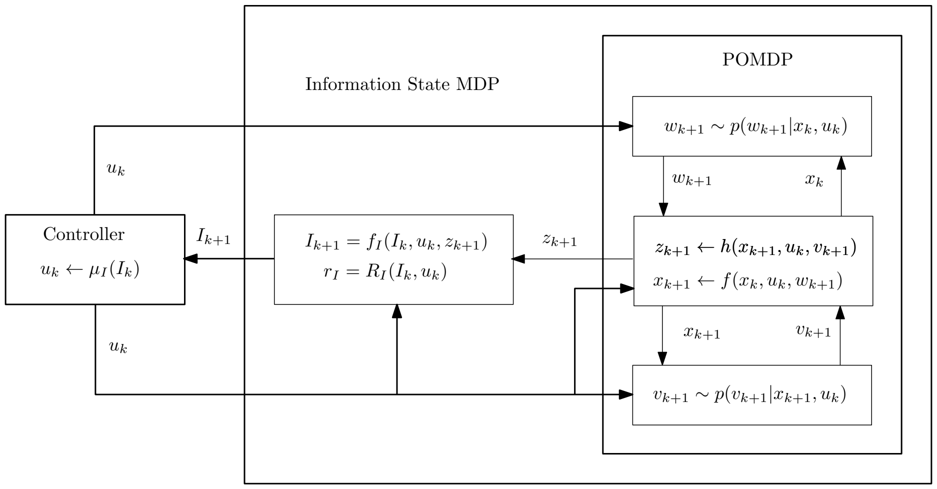

Information State MDP
POMDP
$w_{k+1} \sim p(w_{k+1}|x_k, u_k)$
w_{k+1}
x_k
u_k
Controller
$u_k \leftarrow \mu_I(I_k)$
I_{k+1}
$I_{k+1} = f_I(I_k, u_k, z_{k+1})$
$r_I = R_I(I_k, u_k)$
z_{k+1}
$z_{k+1} \leftarrow h(x_{k+1}, u_k, v_{k+1})$
$x_{k+1} \leftarrow f(x_k, u_k, w_{k+1})$
x_{k+1}
v_{k+1}
u_k
$v_{k+1} \sim p(v_{k+1}|x_{k+1}, u_k)$

Figure 3.3: This illustrates the POMDP model converted to the corresponding fully observable information state MDP model

may denote this with

$$b_k(x) = p(x_k = x|I_k).$$

We may decompose the information vector I_k

$$\begin{aligned} I_k &= \{z_0, u_0, z_1, u_1, \cdots, z_{k-2}, u_{k-2}, z_{k-1}, u_{k-1}, z_k\} \\ &= \{z_0, u_0, z_1, u_1, \cdots, z_{k-2}, u_{k-2}, z_{k-1}\} \cup \{u_{k-1}, z_k\} \\ &= \{I_{k-1}, u_{k-1}, z_k\} \end{aligned}$$

Using Baye's Theorem,

$$\begin{aligned}
b_k(x) &= p(x_k = x|I_k) = p(x_k = x|I_{k-1}, u_{k-1}, z_k) \\
&\propto p(z_k|x_k = x, u_{k-1})p(x_k = x|I_{k-1}, u_{k-1}) \\
&= p(z_k|x_k = x, u_{k-1}) \int_X p(x_k = x|I_{k-1}, u_{k-1}, x_{k-1})p(x_{k-1}|I_{k-1}, u_{k-1})dx_{k-1} \\
&= p(z_k|x_k = x, u_{k-1}) \int_X p(x_k = x|u_{k-1}, x_{k-1})p(x_{x-1}|I_{k-1})dx_{k-1} \\
&= p(z_k|x_k = x, u_{k-1}) \int_X p(x_k = x|u_{k-1}, x_{k-1})b_{k-1}(x_{k-1})dx_{k-1},
\end{aligned} \tag{3.10}$$

where $\propto$ stands for proportional to. The fourth line follows from the Markovian property of transitioning between states. This result is important in that it lets us recursively update the agent belief distribution with only u_{k-1}, z_k, and b_{k-1}. As observations and actions are made they can be discarded alleviating the issue of a constantly growing information state. We can characterize the way an agent's belief transitions from b_{k-1} to b_k with the function

$$b_k = \psi(b_{k-1}, u_{k-1}, z_k). \tag{3.11}$$

The corresponding reward for taking an action before b_k can be written as an expectation over states

$$\begin{aligned}
\hat{g}(b_k, u_k) &= E_{x_k}\left[g(x_k, u_k)|I_k\right] \\
&= \int_X g(x_k, u_k)b_k(x_k)dx_k \\
&= \langle g(\cdot, u_k), b_k\rangle.
\end{aligned}$$

With a fully observable state defined with its corresponding transition function defined along with a meaningful reward the POMDP has been converted to a belief MDP. There are, however, still a few shortcomings that need to be addressed before we can find good policies over the belief states. The previous equations have been written to allow a continuous state space X, but can easily be rewritten with summations if X turns out to be discrete. The next sections will discuss the differences in belief MDPs for when X is either discrete or continuous.

3.2.2 Discrete State Spaces

To illustrate the POMDP with a discrete state space we will introduce the so called Tiger Problem. The Tiger Problem considers a man who is forced to repeatedly open one of two doors. Behind one of the doors lies a great reward, while behind the alternative is a rather hungry tiger. The man does not have to immediately open a door, but is allowed to listen, perhaps with multiple attempts, to gain information about the whereabouts of the tiger. Once the man selects a door, he either receives the reward or is (to be slightly less severe as other descriptions of this problem) frightened by the tiger. The door is then shut, the tiger randomly relocated, and the process of selecting a door resumes. We can write the state, action, and observation spaces as

1. X = {Tiger-Left, Tiger-Right}
2. U = {Listen, Open-Left, Open-Right}
3. Z = {Hear-Tiger-Left,Hear-Tiger-Right}.

The state of the system (the location of the tiger) does not change when the action chosen is listening. Then the transition probabilities can be written:

If the man decides to listen for sounds from the tiger, it will be favored that he hears sounds coming from the door that the tiger is truly behind. The observation probabilities are then:

Table 3.1: Transition Probabilities

u = Listen	Tiger-Left	Tiger-Right
Tiger-Left	1	0
Tiger-Right	0	1
u = Open-Left		
Tiger-Left	.5	.5
Tiger-Right	.5	.5
u = Open-Right		
Tiger-Left	.5	.5
Tiger-Right	.5	.5

Table 3.2: Observation Probabilities

u = Listen	Tiger-Left	Tiger-Right
Hear-Left	.85	.15
Hear-Right	.15	.85

Finally we must consider a reward. We assume a small cost for listening, a large reward for selecting the door with the treasure, and a severe penalty for selecting the door with the tiger. This is summarized in the table:

Table 3.3: Rewards

	Tiger-Left	Tiger-Right
Listen	-1	-1
Open-Left	-100	10
Open-Right	10	-100

With the transition, observation, and reward equations written the POMDP agent can convert this problem into the corresponding belief-MDP. The state space in this belief-MDP is the set of all probability distributions over the two discrete states, Tiger-Left and Tiger-Right. We may let p represent the belief that the tiger is behind the left door, then the belief the tiger is behind the right door is just $1 - p$. This means we need only a vector (in this case of scalar) of length one to represent the state space of beliefs. In general, if there are n discrete states in the POMDP model you will need a vector of length $n-1$ to represent the states. The reduction in dimension by one comes from the constraint that the sum probability is 1. Let the vector describing the distributions be b where $b(x)$ is the probability the agent is in state $x \in X$.

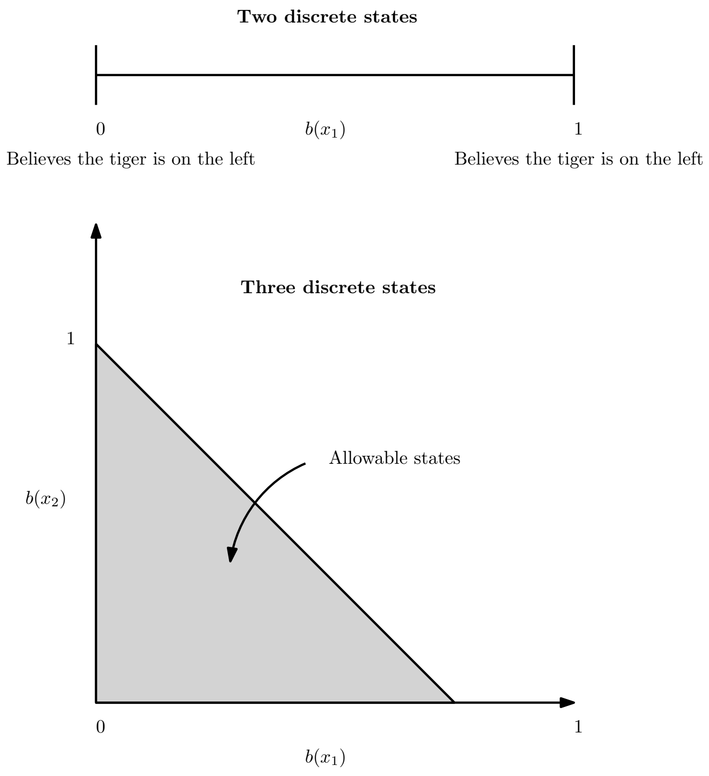


Figure 3.4: The belief simplexes for a two and three state POMDP. The top axis corresponds to the tiger problem. Only a single line is necessary to represent the agent's belief that the tiger is behind the left or right door. The bottom axis represent the belief space for 1 more dimension. Any point in the shaded triangle imply a belief in the third state and assuredly sum to 1.

Any POMDP with discrete state space can be converted into a belief-MDP whose belief's can be represented with discrete length vectors representing probability distributions. Unfortunately, this new state space is continuous in nature as the distributions are multinomial. This makes the use of PI and VI algorithms infeasible as both require summations over the state space. While the previous solution methods are impractical, the equations for the optimal value function and policy are insensitive to continuous versus discrete state spaces. Recall for the standard discounted MDP the backup operator T and its relation to the optimal value function:

$$\begin{aligned}(TJ)(x) &= \min_{u \in U(x)} E_w \left[g(x,u,w) + \gamma J\left(f(x,u,w)\right)\right] \\ J^*(x) &= \lim_{n\to\infty} (T^n J)(x), \forall x \in X,\end{aligned}$$

where the w is the source of error due to transitioning between states.

We can now replace parts of this equation with the belief MDPs counterparts, first we attack the expectation. The expectation is taken with respect to w, a noisy variable affecting the transitions between states. The transition equation for a belief MDP has been written $b_k = \psi(b_{k-1}, u_{k-1}, z_k)$. The noise in switching belief states at time $k-1$ is in the uncertainty in the future measurement z_k. Thus, the expectation will be changed to one with respect to z_k. The reward function $g(x,u,w)$ is simply replaced with $\hat{g}(b,u) = \langle g(\cdot,u), b\rangle = \sum_{x\in X} g(x,u)b(x)$. Finally, we also switch the min operator to the max operator since in this particular example we are trying to maximum reward. The new operator becomes

$$\begin{aligned}(TJ)(b) &= \max_{u\in U} E_Z\left[\hat{g}(b,u) + \gamma J(\psi(b,u,Z))\right] \\ &= \max_{u\in U}\left[\sum_{x\in X} g(x,u)b(x) + \gamma \sum_{z\in Z} p(z|b,u) J(\psi(b,u,z))\right] \\ &= \max_{u\in U}\left[\sum_{x\in X} g(x,u)b(x) + \gamma \sum_{z\in Z} p(z|b,u) J(b_u^z)\right]\end{aligned}$$

where $b_u^z = \psi(b, u, z)$, the future belief state when the action and observation are given. Since we have discrete states, the future belief may be written

$$\begin{aligned} b_u^z(x') &= p(z|x', u) \int_X p(x'|x, u) b(x) dx \\ &\propto p(z|x', u) \int_X p(x'|x, u) b(x) dx \\ &= p(z|x', u) \sum_{x \in X} p(x'|x, u) b(x). \end{aligned}$$

While we cannot iterate over the belief points exhaustively to use PI or VI, we can try to find a why to succinctly represent the finite horizon value function and how this is changed with successive applications of T. Recall, the initial step of VI is to let $J_0(x) = 0$ for all $x \in X$. We may do the same here, let $J_0(b) = 0$ for all $b \in B$. This is equivalent to taking the zero vector $\mathbf{0}$ and writing $J_0(b) = \langle b, \mathbf{0} \rangle$. Then

$$\begin{aligned} J_1(b) &= \max_{u \in U} \left[\sum_{x \in X} g(x, u) b(x) + \gamma \sum_{z \in Z} p(z|b, u) J_0(b_u^z) \right] \\ &= \max_{u \in U} \left[\langle b, g(\cdot, u) \rangle \right] \\ &= \max_{\{\alpha_1^i\}_i} \langle b, \alpha_1^i \rangle \end{aligned}$$

where α_1^i form a set of vectors describing the value function after 1 iteration. We see that after one application of T, the single vector $\mathbf{0}$ describing J has grown to a set of vectors $|U|$ in length. Essentially, each of these vectors describes a hyperplane through the belief space, with the finite value function J_1 taking the maximum value over hyperplanes. It turns out that J_1 has the property of being a piecewise linear and convex function (PWLC). In fact, $J_1, J_2, J_3, \cdots$ all are PWLC functions [33]. PWLC functions can easily be expressed with a set of hyperplanes. In general, let $\{\alpha_n^i\}$ represent the set of vectors that describe J_n such that

$$J_n(b) = \max_{\{\alpha_n^i\}_i} \langle b, \alpha_n^i \rangle.$$

Denote that total number of vectors in the set $\{\alpha_n^i\}$ as $|J_n|$. Then we may write $J_{n+1}(b)$ as

$$\begin{aligned}
J_{n+1}(b) &= \max_{u \in U} \left[\langle b, g(\cdot, u) \rangle + \gamma \sum_{z \in Z} p(z|b,u) J_n(b_u^z) \right] \\
&= \max_{u \in U} \left[\langle b, g(\cdot, u) \rangle + \gamma \sum_{z \in Z} p(z|b,u) \max_{\{\alpha_n^i\}_i} \langle b_u^z, \alpha_n^i \rangle \right] \\
&= \max_{u \in U} \left[\langle b, g(\cdot, u) \rangle + \gamma \sum_{z \in Z} \max_{\{\alpha_n^i\}_i} \sum_{x' \in X} p(z|x', u) \sum_{x \in X} p(x'|x, u) b(x) \alpha_n^i(x') \right] \\
&= \max_{u \in U} \left[\langle b, g(\cdot, u) \rangle + \gamma \sum_{z \in Z} \max_{\{\alpha_n^i\}_i} \langle b, g_{u,z}^i \rangle \right]
\end{aligned}$$

where $g_{u,z}^i(x) = \sum_{x' \in X} p(z|x', u) p(s'|s, u) \alpha_n^i(x')$. Finally, if we define the vector

$$g_u^b = g(\cdot, u) + \gamma \sum_{z \in Z} arg \max_{g_{u,z}^i} \langle b, g_{u,z}^i \rangle,$$

then $J_{n+1}(b)$ takes the simple form of

$$J_{n+1}(b) = \max_{u \in U} \left[\langle b, g_u^b \rangle \right]$$

We see that given a representation of J_n with a set of hyperplanes and for a given belief point b, the next iteration, J_{n+1}, at point b can also be represented with a set of new hyperplanes. If B was a discrete set we could iterate over all $b \in B$ and find separate sets of hyperplanes, but this is unfortunately not the case. To alleviate this issue several region based algorithms have been developed. In particular, the *Witness* algorithm and *Incremental Pruning* are among the most popular [33, 34]. Both algorithms are similar in nature and decompose the optimal value function.

First, let Q_n^u be the value of taking action u, then acting optimally thereafter. Then

$$Q_n^u(b) = \sum_{x \in X} g(x,u)b(x) + \gamma \sum_{z \in Z} p(z|b,u) J_{n-1}(b_u^z),$$

and the optimal value function after n iterations can be written

$$J_n(b) = \max_{u \in U} Q_n^u(b).$$

We know that J_n is PWLC, but it is also true that each individual Q_n^u is PWLC and can be represented by a set of vectors [34]. Since this is the case, then J_n is just the maximum value over the union of set $\bigcup_{u \in U} Q_n^u$. Cassandra in [35] further decomposed J_n

$$\begin{aligned}
Q_n^{u,z}(b) &= \frac{\hat{g}(b,u)}{|Z|} + \beta p(z|b,u) J_{n-1}(b_u^z) \\
Q_n^u(b) &= \sum_{z \in Z} Q_n^{u,z}(b) \\
J_n(b) &= \max_{u \in U} Q_n^u(b).
\end{aligned}$$

Each of these functions are PWLC [34], and we let $J_n, Q_n^u, Q_n^{u,z}$ represent the set of vectors representing their corresponding functions over the belief space. The value of $J_n(b)$ is a combination of vectors from Q_n^u and $Q_n^{u,z}$. Finding the exact values of these vectors can be straightforward, but doing so for each b is infeasible. In order to calculate $J_n(b)$ for all $b \in B$, *Incremental Pruning* considers all possible combinations of vectors. This allows us to explicitly represent J_n, but at a large computational cost of calculating expensive cross-sum over the vectors. Let the cross-sum of two vector sets X and Y be defined as

$$X \bigotimes Y = \{x + y | x \in X, y \in Y\}$$

with $|X \bigotimes Y| = |X||Y|$.

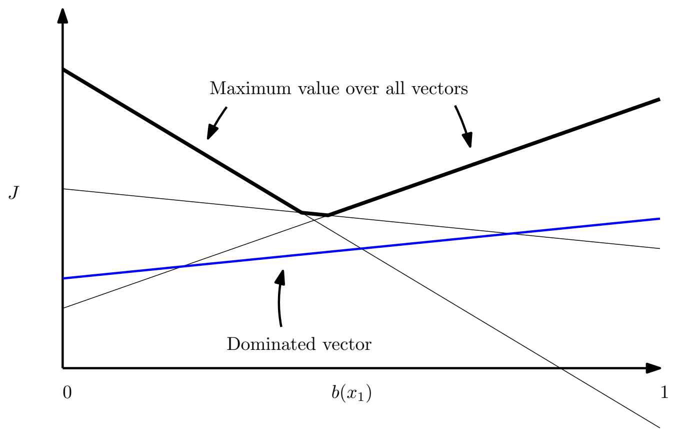


Figure 3.5: The value function expressed as a set of vectors. The bold black line indicates the maximum over the set of vectors. The blue line indicates a vector that is dominated by other vectors and less than the bold line for all belief points.

Once all possible vectors are calculated to represent J_n, many (if not most) may be removed from the set. Remember, $J_n(b)$ is the maximum value over a set of scalar products. Thus, if it is true for some $j \in J_n$ that $\langle b \cdot j \rangle < \langle b \cdot j' \rangle$ for all $j' \neq j \in J_n$, then j can be removed from the set without effecting the value function. We call finding and removing these vectors pruning. Assuming J_{n-1} has been calculated with a minimal set of vectors, then J_n may be found with a few straightforward steps and is described in the following pseudocode.

Algorithm 5 A simple implementation of incremental pruning. The algorithm represents the value function initially with a single **0** vector. It then sequentially updates J_n until the change in the value function is small. Dominated vectors are pruned each stage to minimize unnecessary computation.

```
$J \leftarrow \{\mathbf{0}\}$
**converged** $\leftarrow$ **false**
**repeat**
  Calculate $Q^{u,z}$
  Calculate $Q^u(b) = Prune(\oplus_{z \in Z} Z_n^{u,z})$
  Calculate $J' = Prune(\cup_{u \in U} Q_n^u)$
  **if** $|J'(b) - J(b)| < \delta \ \forall \ b \in B$ **then**
    **converged** $\leftarrow$ **true**
  **else**
    $J \leftarrow J'$
  **end if**
**until converged**
```

The details on pruning have been omitted, but essentially require solving a collection of linear programming problems. This step can be time consuming, but is necessary to avoid a rapid growth of vectors. For a more complete treatment of solving POMDP's with discrete states refer to [33, 34, 35].

	$\lvert J_n \rvert$ without pruning	$\lvert J_n \rvert$ with pruning
$n = 1$	3	3
$n = 2$	27	5
$n = 3$	2187	9
$n = 4$	$*$	10
$n = 5$	$*$	13

Table 3.4: A comparison of the size of J_n with and without pruning of dominated vectors on the Tiger problem. The $*$ indicates a size too large to calculate in a reasonable amount of time. We see that with pruning the number of vectors per iteration does not grow too fast.

3.2.3 Moving Towards Continuous State Spaces

We saw in the previous section how vectors of length $|X|$ can represent belief states for discrete state POMDPs, each element of the vector representing the probability that the agent is in a particular state. The set of all beliefs then forms a $|X|$ dimensional simplex and is thus infinite and continuous in nature. The value function of the simplex was exactly calculated with a set of piecewise linear and convex functions. The *Witness* algorithm in particular iteratively finds sets of linear functions to approximate the value function with increasing accuracy [33]. A major setback to this algorithm, and exact algorithms in general, is the growing sizes of the set of linear functions and the linear programming required to prune dominated functions from this set [35]. Another class of algorithms exist that select a discrete subset of points from the belief simplex, only updating these points during value iteration [36, 37, 38]. One in particular, *Perseus*, has been well studied over the last few years with promising results [36].

The key point in converting a POMDP to an equivalent belief state MDP is by analyzing the set of all probability distributions over states. For the discrete state space this set is continuous, with every possible distribution represented with a finite length vector (of dimension $|X|$). For continuous state spaces, though, this transition is not so gentle and can be illustrated with a simple example. Suppose the state space is an a point in the line segment $[0, 1]$. The belief states are then all probability distributions with support $[0, 1]$. This includes any truncated Gaussian, for any choice of μ and σ, any truncated exponential, and any probability mass function represented by a discrete number of points. An infinite-dimensional vector would be required to specify **every** distribution making the point based approaches using the belief simplex impossible. The next chapter will describe current methods to overcome these new challenges for continuous state spaces.

Chapter 4

Solving Continuous State POMDPs

Discrete state POMDPs can be converted into belief state MDPs where the new set of states are points on a belief simplex of dimension equal to the number of states. Exact algorithms using incremental pruning and the Witness algorithm [33, 35] were discussed as methods to solve in this new space. Point based methods such as Perseus [36] randomly sampled the belief simplex and only updated the value function at these points resulting in alternative algorithms with large state spaces. Models with a continuous state space have a corresponding belief space over the set of all possible probability distributions over that space requiring infinite dimensional vectors to exhaustively specify every possible distribution. An immediate extension of the Witness and Perseus algorithm (and similar variants) is not readily available to be applied to this new class of problems. There has, however, been significant progress to solving continuous state POMDPs.

Thrun in [39] pioneered some work on the continuous state space by representing beliefs as a weighted particle set over the admissible set of states.

He then used Monte Carlo techniques to update the beliefs after actions and observations, and also used a k-nearest neighbor scheme to interpolate the value function at belief points not specifically in his set. He used the Kullback Leibler divergence measure (effectively the non symmetric distance between two distributions) as the distance between two belief points. Agents training using this method were able to solve simple robot navigation problems at the cost of long training times. This is likely due to the size of the belief vector being equal to the number of particles representing the beliefs. Increasing the number of particles will increase accuracy, but at the cost of state space increase as well.

Porta and Spann in [40] provided strong theoretical contributions to solving POMDPs with continuous state spaces (and discrete action and observations). The value function over the belief space was shown to also be piecewise linear and convex represented with a set of α functions (rather than vectors). In addition, the n step value function was proven to have a discrete representation, although exponential with respect to number of observations. By assuming each belief can be represented by a discrete mixture of Gaussians they were able to apply there equations to solve a single corridor robot navigation problem using a modified Perseus algorithm adapted for the continuous state space. Their algorithm was successful in training the agent, but has a computational bottleneck in exponential growth in number of α functions during value iteration.

The work in [40] was further extended in [41] to handle continuous action spaces. Essentially, actions were sampled during the value iteration process. A wide variety of sampling methods are available for selecting actions, often sampling randomly as an exploration stage, and then sampling actions nearby known best actions seen thus far. Their algorithm still falls into the point based class, but the points are specific Gaussian distributions, rather than sampled states from the model. Their algorithm was compared to the traditional Perseus algorithm. The results between the discrete Perseus and continuous (Gaussian based) Perseus were comparable in terms of accumulated reward and time to train the agent, but did have a reduction in memory requirements (only 9 Gaussians versus 200 sampled states).

Brooks also investigated using POMDP models with continuous state spaces in [1, 8, 42]. His approach, called Parametric POMDP (PPOMDP), restricts beliefs to being in the set of multivariate Gaussians. The algorithm uses Monte Carlo methods and particle filtering to estimate the transition probabilities between a discrete set of Gaussian distributions representing likely belief states the agent my encounter naturally. The transition probabilities coupled with an expected reward over belief states allows the problem to be solved with traditional MDP value iteration rather than modeling the value function as a set of vectors (or functions). The agent then uses the calculated value function along with nearest neighbor interpolation to select good actions from any Gaussian belief state. He incorporated a forward search over the action space to improve the agent's performance. This algorithm is also capable of handling a continuous observation space, but uses a discretized action space as an approximation to model's with continuous actions. The algorithm was tested on simulated models as well as a real life trial showing strong results. The strength in this method is in the dimension reduction. Belief states for two dimensional Gaussians (representing the (x, y) location of the agent) require only a vector of length 4, a strong contrast to [39] where the vector length grows with the number of particles used.

Zhou extended the work of Brooks in [43, 44], producing the first theoretical error bounds for the difference between the true value function and the one calculated using the parametric belief distribution and estimated transition probabilities. She also generalized the set of beliefs available for the agent from multivariate Gaussians to the entire exponential family. Like Brooks, she represents beliefs with particle sets and projects them onto a distribution from the exponential family for dimension reduction. She also includes rigorous proofs of convergence as the number of particles tends to infinity.

The parametric POMDPs developed by Zhou and Brooks is the advocated method and the one explored in this thesis for modeling and solving POMDPs. The number of knobs and dials available to the user of this algorithm to twist and turn are vast. In particular, given a model, the user must decide on the belief distribution, choice of particle filter, number of particles used, belief set selection, discretization of action space, and action selection during online control. The selection of these options can have a dramatic effect on the reward obtain by the agent, memory demands, and length of time to compute the value function and select actions. The correct choices will be dependent on the selected model (robot navigation, inventory stock control) and the computational power of the user.

The rest of this chapter will be dedicated to making significant improvements to the state of PPOMDP. First we will detail the current standards of PPOMDP. We will then show that the current state of the PPOMDP algorithm wastefully spends computation on sampling more posterior beliefs than is necessary for training an intelligent agent. We will show this numerically on a variety of examples, and also show how extra samples can be effectively generated by using a k-nearest neighbor interpolation scheme. A novel forward search approach will be contrasted against Brooks recommended approach. We will show how our approach can search far deeper than Brooks allowing for intelligent agents to successfully accumulate rewards on sparse belief sets.

We will then create a novel threefold adaptive sampling scheme during agent training. We will show how this approach trains intelligent agents in a reasonable training time when compared to less sophisticated adaptive samplers. The number of samples drawn during the course of training an agent by our sampler will further verify our claim about the wasteful computational approach to PPOMDP initially.

4.1 Parametric POMDP

The introduction to this chapter described the inherent complications when generalizing the discrete state space to a continuous one, and also described the recent developments in alleviating these issues. In particular, we placed emphasis on the methodology developed by Zhou and Brooks in [1, 44]. The purpose of this section is to explain in detail how continuous state POMDPs can be converted and modified allowing traditional MDP solution methods to be employed. We will begin by defining the exponential family and showing how to project particle sets onto distributions from this family. We will then describe how to take a discrete subset of distributions and approximate transition probabilities given actions from an agent. A short introduction to nearest neighbor search will follow to illustrate how the transitions probabilities may be calculated efficiently. Finally, the basic PPOMDP algorithm will be presented.

4.1.1 Density Projection

To begin we take a closer look at the belief representation. Both Zhou and Brooks expressed beliefs states as a lower dimensional parametric distribution. The multivariate Gaussian distribution was employed by both, although Zhou produced results that can be applied to any distribution in the exponential family. Define Ω as the selected exponential family of densities chosen to represent the belief state of the agent. Let Θ be the set of valid parameters describing the densities in Ω. Then we can write

$$\Omega = \{f(\cdot, \theta) : \theta \in \Theta\}$$
$$f(x, \theta) \propto exp\left(\theta^T T(x) - A(\theta)\right)$$

where $x \in R^n$ corresponds to the state space defined by our POMDP model, $\theta \in R^m$ is a parameter vector , and $T(x) = [T_1(x) \;\; T_2(x) \cdots T_m(x)]$ is a vector of sufficient statistics [45], and $A(\theta) = log \int exp(\theta^T T(x))dx < \infty$. The multivariate Gaussian, exponential, gamma, chi-squared, beta, Bernoulli, and Poisson distribution (and many others) all belong to this family of densities.

Algorithms like Perseus select a discrete set of points over the continuous belief simplex. Similarly we will be selecting a discrete set of parameter vectors. Let $G = \{\theta_1, \theta_2, \cdots, \theta_{|G|}\}$ be the subset of allowable beliefs specified by parameter vectors, rather than explicit locations in the state space. From the perspective of the agent, these belief points are fully observable and define the state space for the MDP model we are converting to. We know that the MDP model has uncertainty in where an agent will transition given an action. Typically this is defined initially with a transition matrix. But, for our modified problem the transition probabilities will have to be estimated. Let $p(\cdot|\theta_i, u)$ represent the transition probabilities given an initial belief state and action. There are two main issues to approximating this distribution:

1. How do we incorporate the future measurement available to the agent in belief space rather than the explicit state space?
2. How can we be sure the belief distribution will remain in the exponential

family selected?

The first issue is circumvented by sampling possible states from the belief distribution θ_i. The initial POMDP model fully describes how the agent will transition between states given an action. Upon transitioning forward in time according to the state model, observations may be sampled as well according to the system dynamics. The new particle set along with the sampled observations represent a group of possible futures for the agent given and action. This set of future possibilities are not, however, directly comparable to the to our belief states which are represented parametrically rather than as a probability mass function over the state space (essential what a weighted particle set is).

The second issue involves using a technique known as density projection. Essentially density projection maps a weighted particle set back to a parameter in the exponential family of choice by either a maximum likelihood estimate or minimizing the Kullback-Liebler divergence [43, 46, 47]. The best fitting θ is unlikely to be in the belief set B. This is not a problem for the purposes of an agent maintaining a belief while executing actions and incorporating observations, however does pose an issue for calculating the transition probabilities. The forward beliefs generated will be "snapped" to nearby parameters in G. That is, an agent beginning in state θ_i and taking action u will transition to θ_j with positive probability if the forward beliefs are similar or nearby θ_j in parameter space after the particles are projected onto Ω. Details for explicitly calculating these transition matrices will follow in the next sections.

Kullback-Liebler divergence

Kullback-Liebler divergence was first published in 1951 and can succinctly be described as the non-symmetric distance or dissimilarity between two distributions [48]. We are interested in two distributions over the continuous state space: a weighted particle set, a probability density function $f(\cdot, \theta)$ in the family of exponential densities. Formally the Kullback-Liebler divergence is defined by

$$D_{KL}(b||f) = \int b(x) log \left(\frac{b(x)}{f(x)} \right) dx. \tag{4.1}$$

We may define a projection function $Proj_{\Omega} : B \to \Omega$

$$Proj_{\Omega}(b) = argmin_{f \in \Omega} D_{KL}(b||f). \tag{4.2}$$

Essentially, $Proj_{\Omega}(b)$ yields the parameter θ that best matches a particle set representing b. To solve for θ, first substitute $f(\cdot, \theta)$ for f in 4.1 yielding

$$\begin{aligned} D_{KL}(b||f(\cdot, \theta)) &= \int b(x) log \left(\frac{b(x)}{f(x, \theta)} \right) dx \\ &= \int b(x) log\left(b(x)\right) dx - \int b(x) log\left(f(x, \theta)\right) dx. \end{aligned} \tag{4.3}$$

The first term is constant as a function of θ, then the optimal theta can be found by maximizing

$$Proj_{\Omega}(b) = argmax_{\theta \in \Theta} \int b(x) log\left(f(x, \theta)\right) dx. \tag{4.4}$$

Zhou showed in [43] that the maximization is obtained when

$$E_b[T_j(X)] = E_{\theta}[T_j(X)], \text{for } i = 1, \cdots, m. \tag{4.5}$$

With this we have a principled way to convert weighted particle sets into a parameterized belief distribution in the exponential family. What remains to be discussed are the four main algorithms themselves used to convert the continuous POMDP into the MDP counterpart. Particularly we need algorithms that:

1. Update a parameterized belief θ given an action u and future observation z

2. Forecast future distributions given a belief θ and action u. Note the uncertainty in the future belief is from two sources: the noisy system dynamic, and the unknown future observation.

3. Calculate transitions between beliefs in the belief set G given a set of actions U.

4. Calculate a reward function for taking actions from beliefs in G.

PPOMDP Algorithm

In the previous chapter we discussed how an information state or belief state of an agent operating in a POMDP environment can be modeled as a fully observable state from an equivalent MDP model. The types of beliefs have been restricted to those from the parameterized exponential family of distributions. A key component to completing this transition is developing a method to calculate the transition function $f_I(I_k, u_k, z_k)$, where the information state I_k can really be replaced with the parameter θ_k. We simplify some of the notation in this chapter with the following table.

x_k	$\rightarrow$	x
x_{k+1}	$\rightarrow$	x^+
I_k	$\rightarrow$	θ
I_{k+1}	$\rightarrow$	θ^+
u_k	$\rightarrow$	u
z_{k+1}	$\rightarrow$	z^+

Table 4.1: This table contains a list of notation simplifications in notation for brevity purposes.

The following approach is based on particle filtering techniques and assuming that the exponential belief distribution chosen may easily be sampled. Essentially, particles are propagated forward in time according to an action. Observations are then sampled and compared to the given observation $z+$ to form a weighted particle set.

Algorithm 6 An implementation of the transition equation $f_I(\theta, u, z^+)$. The integer value N indicates the number of particles sampled.

```
Input: $\theta, u, z^+$
Output: $\theta^+$
for $i = 1 : N$ do
    Sample $x_i$ from $p(x_i|\theta)$
    Sample $x_i^+$ from $p(x_i^+|x_i, u)$
    Sample $z_i^+$ from $p(z_i^+|x_i^+)$
    Calculate $w_i = p(z^+|z_i^+)$
end for
Normalize $\hat{w}_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$
$\theta^+ = Proj_\Theta\left(\{x_i^+\}_{i=1}^N, \{\hat{w}_i\}_{i=1}^N\right)$
```

Algorithm 6 allows an agent to keep track of it's belief as it takes actions and makes observation. In addition, the agent must be able to reason about future beliefs given an action but devoid of the future observation. Brooks initially proposed a simple implementation of generating the posteriors in [1].

Algorithm 7 is similar to algorithm 6, but is missing the input of a known future observation. The algorithm compensates for the lack of observation by sampling one from each of the N particles generated. At this point algorithm 6 is essentially called with a fixed particle set for each sampled observation. One issue with generating a distribution of posteriors this way is that the complexity scales quadratically with respect to the number of particles generated. That is, if N particles are used then N posterior distributions are calculated. It will be shown in the following sections that some if this computation burden

Algorithm 7 A simple implementation of $GeneratePosteriors(\theta, u)$. Let N be the number of particles sampled and posteriors to be returned. This algorithm works by generating and propagating N particles forward in time according to the system dynamics. Each particle also samples a possible future observation. The particle set is then re-weighted N times assuming the correct observation belongs to one coming from an individual particle. This generates N differently weighted particle sets, but with the exact same particles. Each set is then projected back onto Θ

```
Input: $\theta, u$
Output: $\{\theta_1^+, \theta_2^+, \cdots, \theta_N^+\}$
Let $W$ be a matrix of size $N \times N$ holding the particle weights.
for $i = 1 : N$ do
    Sample $x_i$ from $p(x_i|\theta)$
    Sample $x_i^+$ from $p(x_i^+|x_i, u)$
    Sample $z_i^+$ from $p(z_i^+|x_i^+)$
end for
for $i = 1 : N$ do
    for $j = 1 : N$ do
        Calculate $W_{ij} = p(z_i^+|x_j^+)$
    end for
end for
Normalize the rows of $W$
for $i = 1 : N$ do
    $\theta_i^+ = Proj_\Theta\left(\{x_j^+\}_{j=1}^N, \{W_{ij}\}_{j=1}^N\right)$
end for
```

may be alleviated by limiting the number of posteriors to a value less than N.

One of the essential portions of any MDP model is the transition probability function $p(\cdot|x,u)$ and is typically known in advance. In our POMDP world we need a means of approximating $p(\cdot|\theta,u)$, the probability of transition to another belief in G given $\theta \in G$. A natural calculate this is to simulate the distribution over posteriors for each $\theta \in G$ and $u \in U$ combination. The posteriors that are close to beliefs already in G will weight those beliefs with positive probability. This is accomplished by a *nearest k-neighbor search* over the set G.

Algorithm 8 A simple implementation of $TransitionProbablities()$. This functions returns two jagged multidimensional arrays, $Tvalues$ and $Tindices$. $Tindices$ points to where in G particular θ and u combinations have positive transition probability. $Tvalues$ holds the transition probabilities. The jagged matrices allow the total memory requirements to be far than the order of $|G|^2|U|$ by not storing the transitions with 0 probability.

```
Input: $G = \{\theta_1, \theta_2, \cdots, \theta_{|G|}\}$
Output: An approximation to $p(\cdot|\theta_i, u)$ for $\theta_i \in G$ and $u \in U$ stored in the variables $Tindices$ and $Tvalues$.
Initialize $counter$ as an array of length $|G|$.
for $i = 1 : |G|$ do
  for $j = 1 : |U|$ do
    $\Delta = GeneratePosteriors(\theta_i, u_j)$
    Reset $counter$
    for $k = 1 : |\Delta|$ do
      Let $index$ be the index of the closest member in $G$ to $\Delta_k$
      $counter[index] \leftarrow counter[index] + 1$
    end for
    $Tindices[i][j] \leftarrow$ non-zero indices of counter
    $Tvalues[i][j] \leftarrow \frac{counter}{|\Delta|}$
  end for
end for
```

The overall runtime of algorithm 8 is a function of several sources. We write the following four functions that model these sources:

1. $C_1(x, x^+, z^+)$, the total cost of generating a particle x from initial belief θ, propagating it forward according to the system dynamics, and sampling the observation z^+.

2. $C_2(x^+, z^+)$, the cost of evaluating the particle weight $p(z^+|x^+)$.

3. $C_3(N)$, the cost of using a particle set of size N to project onto the parameterize family Θ.

4. $C_4(|G|)$, the cost of finding the nearest-neighbor in a set of size $|G|$.

Algorithm 8 has two outer loops enumerating over all possible combinations from the sets G and U. The inner loop is a function of the four above costs and the number of particles used in the filtering. Putting this altogether the complexity of run time can be written as

$$O\left(|G||U|\left(NC_1(x, x^+, z^+) + N^2C_2(x^+, z^+) + NC_3(N) + NC_4(|G|)\right)\right) \quad (4.6)$$

The cost of C_1 and C_2 are both model dependent, C_3 is a function of the chosen belief distribution, and C_4 is a function of the nearest-neighbor algorithm selected. Assuming C_1 and C_2 are close to constant, the projection function is simple (linear with the number of samples), and a proper data structure for similarity search is used for the nearest-neighbor search, then the dominating complexity is from the weighting of the particle sets. Equation 4.6 can then be approximately written as

$$O\left(|G||U|N^2\right).$$

The final necessary algorithm is an approximation to the reward received for taking actions from belief states. Assuming states can easily be sampled

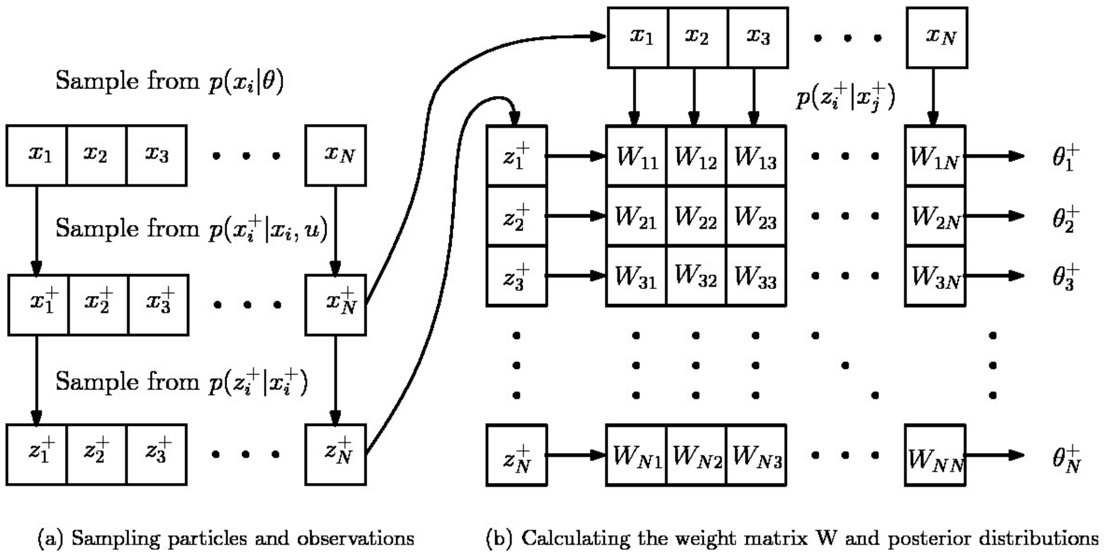

Sample from $p(x_i|\theta)$
x_1 x_2 x_3 x_N
Sample from $p(x_i^+|x_i, u)$
x_1^+ x_2^+ x_3^+ x_N^+
Sample from $p(z_i^+|x_i^+)$
z_1^+ z_2^+ z_3^+ z_N^+
(a) Sampling particles and observations
$p(z_i^+|x_j^+)$
W_{11} W_{12} W_{13} W_{1N} θ_1^+
W_{21} W_{22} W_{23} W_{2N} θ_2^+
W_{31} W_{32} W_{33} W_{3N} θ_3^+
W_{N1} W_{N2} W_{N3} W_{NN} θ_N^+
(b) Calculating the weight matrix W and posterior distributions

Figure 4.1: A depiction of algorithm 8. (a) illustrates the sampling of particles from the exponential distribution θ, the forward propagation according to the model dynamics, and the sampling of observations from particle states. (b) Shows how the weighting matrix W is calculated from the combination of particles and observations.

from the belief distribution, samples are drawn and the reward from these states executing the chosen action are averaged.

These four algorithms combined serve as the foundation for the Parametric POMDP algorithm. The next section briefly introduces four models to test our improvements as well as two competitor algorithms to establish a baseline performances. We then demonstrate our improvements.

4.2 Test Models and Competitor Algorithms

In this section we define our four continuous state POMDP models we will test our algorithms on. We also briefly discuss two competitor algorithms for creating policies for continuous state POMDPs.

Algorithm 9 A simple algorithm to calculate the reward from belief states.

```
Input: Belief state $\theta$ and action $u$, the number of samples to be drawn $N$
Output: $\hat{g}(\theta, u)$
$reward \leftarrow 0$
for $i = 1 : N$ do
    Draw $x$ from $p(\cdot|\theta)$
    $reward \leftarrow reward + g(x, u)$
end for
$reward \leftarrow \frac{reward}{N}$
$\hat{g}(\theta, u) \leftarrow reward$
```

4.2.1 Robot Navigation

Autonomous robot navigation has been a topic studied in several different flavors over the past decades. In particular, any robot navigation system is generally interested in one of the following:

- Localization: The act of sensing the environment to determine the robots position on given map. Position is typically described using a the tuple (x, y, θ), that is the (x, y) location on the map and the robot's pose, heading, or direction θ.
- Planning: Using knowledge of the environment to optimize plan actions that maximize the chance of the robot successfully reaching a goal state.

During the first attempts at equipping an agent with tools to autonomously navigate, it was assumed that the agent could easily maintain its correct position through simple a basic understanding of the robot's motion [49]. This technique, known as *dead reckoning*, was doomed to fail as small errors over time accumulated resulting in a loss of orientation [49]. This lead researchers to a Bayesian approach as a model for robot navigation. Simply put, the Bayesian approach allowed the agent to maintain a belief distribution over a discrete set of locations, and also allowed it to update the belief state in a mathematically principled manner given actions and sensory information [50]. Many problems assume that the model of the environment is known ahead of time, but work in [51] developed a Simultaneous Localization and Mapping algorithm the tracks the agents locations as well as discovers the obstacles making up the map. This thesis will, however, be assuming that the map is known *a priori* to the agent.

With the advances in localization, the agent's ability to keep a good approximation of its position, researchers were then allowed to pursue robot planning algorithms. While a variety of robot tasks can be imagined, a typical goal is for an agent to reach a desired location. An intelligently acting agent must balance between reaching the goal as quickly as possible, but also while avoiding hazards (obstacle collision). To naturally train an agent with this behavior, we use cast the robot navigation problem into a POMDP model:

- X: The set of allowable positions for the agent (the set of all legal (x, y, θ) tuples).
- U: The set of actions available to the robot: turn-left, turn-right, accelerate, etc...
- Z: The set of observations: Typically modeled as laser range sensing or odometry measurements.
- $g(x, u)$: The reward may be set as positive for actions from the goal state, and negative for other actions and collisions.

We saw in the former sections how POMDPs can be cast as an equivalent belief state MDP for both the discrete and continuous state space. The Gaussian and exponential belief distributions found in [1, 44] are fairly recent innovations in the POMDP field for tackling continuous state space problems. We argue that this approach is the most natural given the continuous nature of (x, y, θ) positions. During the infancy of this research, however, the state space was modelled as discretely by sections of portions of the world or creating a topological map [49, 50, 51, 52, 52, 53]. To begin, we specify a simple robot navigation problem where the state space is simply the robot's (x, y) position. We then increase the complexity of our robot navigation model by adding more components to the state space, that is a pose direction θ_p and also increasing the size of the world twofold.

4.2.2 Simple Robot Navigation

For our initial robot navigation model we assume a POMDP environment similar to [1]. The state space consists of all (x, y) pairs of points in a $20m \times 10m$ enclosed grid and not intersecting an obstacle placed on the map. For this model, the agent does not assume a pose direction and θ is omitted from the state space. The actions from the space U come in the form of (d, θ) pairs, that is the agent may select a direction and a distance to move. We discretize this set uniformly over 16 directions (θ divided uniformly in $[-\pi, \pi]$), and allow the choice of two distances $\{1, 2\}$. Additionally, a 33^{rd} action $(.1, 0)$ is added to allow the agent to make small movements. Observations are taken in the form of noisy range measurements in 4 equally spaced directions (North, West, South, East). The range detector only makes an observation if an obstacle or the map is within a range of 2 meters. Additionally, a collision detector indicates that the agent has collided with an obstacle or the border of the map. The goal of the agent is to navigate to an area $1m \times 1m$. The parameters of this model are summarized:

- X = the set of all legal (x, y) positions

- $p(x^+|x,u)$ is a two dimensional Gaussian with mean translated from the initial position x in the direction and distance indicated by u. The standard deviation is set to $.2d$ where d is the distance traveled. The covariance is set to 0.

- $z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 & z_d \end{bmatrix}^T$, where z_i is the range measurement in the i^{th} direction. z_d is a boolean variable indicating a collision.

- $p(z_i|x)$ is normally distributed with mean equal to the true range measurement with standard deviation $\sqrt{.5}$.

- $g(x,u) = \begin{cases} -.1 \text{ if the agent is not in the goal state} \\ \quad 10 \text{ if the agent is in the goal state} \end{cases}$

The belief state will be represented by a two dimensional Gaussian where the mean parameters correspond to the (x, y) locations and the variance represent uncertainty in either direction. Zero covariance is assumed.

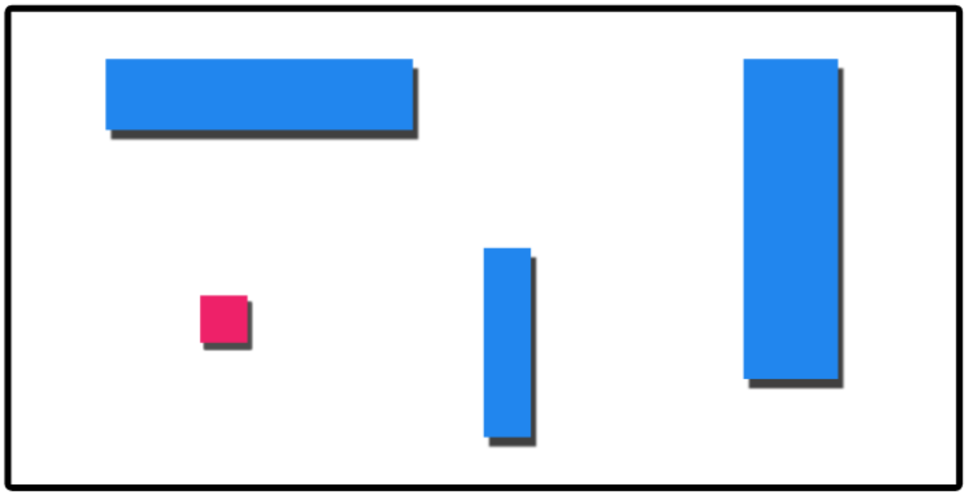


Figure 4.2: This figure illustrates a simple robot navigation model. The model is bounded by a 20×10 box. The red box indicates the goal position and the blue boxes indicate obstacles.

4.2.3 Robot Navigation with Pose

We are increasing the complexity of our robot navigation model by adding an additional state variable, the pose or heading of the agent indicated with θ_p. Similar models with a pose variable have been considered in [54] and [55]. In [54], a POMDP formulation is proposed and applied to an agent operating in an office setting. A graph model with transition probabilities is inferred from the map's corridors, entrances, and rooms. In addition to a range sensing model, they consider *landmark-based* navigation model. Essentially, a landmark is a unique feature of a map detectable by an agent aiding in localization. In [**?**], the state space is discretized and modeled graphically as well with a hierarchical tree of POMDP models.

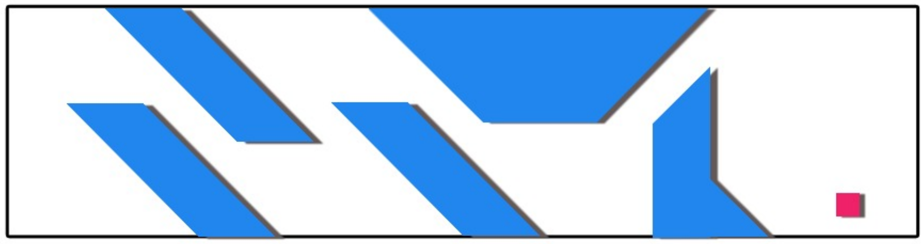


Figure 4.3: A figure illustrating a larger and more complex world for the agent to navigate. The blue areas indicate obstacles while the red region indicates the goal region.

Our pose navigation model is similar as the aforementioned model,but with a few exceptions adding complexity. The range sensor remains the same, except that the direction of the 4 sensors is dependent on the pose of the agent, rather than strictly than North, West, South, and East. Secondly, we incorporate obstacles that are not perfectly aligned with the x and y axis. This adds a slight extra difficulty taking range measurements and makes the independent Gaussian belief state too limited. The action space has been modified to a (d, θ_m) pair, where d indicates the distance to be traveled, and

θ_m indicates a rotation in pose before moving. Lastly, the map that the agent will test on has been expanded to a dimension of $40m \times 10m$, a two fold increase in area. The specifics of this model are as follows:

- X = the set of all legal (x, y, θ_p) positions. An illegal position is when the (x, y) coordinates are located inside an obstacle or outside of the map. We let θ_p take values in $[-\pi, \pi]$

- U = a set of 7 (d, θ_m) pairs. Six of the actions take the form $(d, -\pi/4)$,$(d, 0)$, and $(d, \pi/4)$ with d taking the value of .1 or 1. Lastly, the agent is allowed to take a small step directly backwards with the action $(-.5, 0)$.

- $p(x^+|x, u)$ is a two dimensional Gaussian with mean translated from the initial position x in the direction and distance indicated by u. The agent rotates according to θ_p first and then moves the set distance d. The standard deviation is set to $.2d$ where d is the distance traveled. The covariance is set to 0.

- $z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 & z_{\theta_p} & z_d \end{bmatrix}^T$, where z_1, z_2, z_3 and z_4 are range measurements in the $\theta_p, \theta_p + \pi, \theta_p + 2\pi$ and $\theta_p + 3\pi$ directions. z_{θ_p} is a noisy measurement of the true heading. z_d is a boolean variable indicating a collision.

- $p(z_i|x)$ is normally distributed with mean equal to the true range measurement with standard deviation $\sqrt{.5}$.

- $g(x, u) = \begin{cases} -.10 \text{ if the agent is not in the goal state} \\ 20 \text{ if the agent is in the goal state} \end{cases}$

The agent is allowed 200 actions (an increase over the 100 previously permitted) to reach a small goal region of size $1m \times 1m$. The total accumulated reward falls in the region $[-20, 20]$, although an agent acting perfectly will have a reward of about 15 due to the size of the map and the minimum number of actions needed to traverse it.

The beliefs available to the agent will be constrained to a 2 dimensional Gaussian over the (x, y) coordinates and 1 dimensional Gaussian over the pose θ_p. We assume that x and θ_p are independent as well as y and θ_p. A belief state can be summarized by the 7 length vector $\theta = [x, y, \sigma_x, \sigma_y, \sigma_{xy}, \theta_p, \sigma_p]$. If we modestly discretized the belief set choosing 10 different values for each parameter, this would result in a belief set size of 10^7. Clearly a belief set with ten million states would put the training time outside of acceptable bounds and a mesh of points will be necessary.

4.2.4 Car-on-a-Hill

Consider the simple task of driving a car up the hill of a fairly steep mountain, so steep that it is impossible to climb the mountain without some initial velocity. As the driver you have control over the accelerator thus defining the set of actions at your disposal. The only way to climb the mountain is to first reverse the vehicle to a location where you can accelerate to enough speed to climb to the top of the mountain. More details can be found in [56]. This is a simple but popular control problem modified to suit our POMDP purpose. The details are as follows:

- $X =$ the set of all (p, v) pairs where p is the vehicle's position and v is the vehicle's speed.
- $U =$ a discrete set of accelerations in the set $[-4, 4]$. For our purpose we discretize this set to a length of 5.
- Let the mountain's height be described (and depicted in 4.4) the function

$h(p)$ such that

$$h(p) = \begin{cases} p(1+p) \\ \frac{p}{\sqrt{1+5p^2}} \end{cases}$$

- $p(x^+|x,u)$ is a two dimensional Gaussian with mean translated from the initial position p and speed v according to well known physics describing motion.

- $z = \begin{bmatrix} z_p & z_v \end{bmatrix}^T$, where z_p and z_v are noisy observations of the vehicles position and speed.

- $p(z_p|x)$ and $p(z_v|x)$ are normally distributed with means equal to the true position and velocity with standard deviations σ_p and σ_v.

- The agent receives rewards for remaining in small area near the top of the mountain with constrained speed. All actions taken under different circumstances have a negative reward. The specifics are:

$$g(x,u) = \begin{cases} 0 & \text{if } p < -1 \text{ or } p > 1.5 \text{ or } |v| \geq 3 \\ 1 & \text{if } 1 < p < 1.5 \text{ and } |v| < 3 \end{cases}$$

We allow the agent to take a total of 100 actions. Beliefs are defined as the two dimensional gaussian $\theta = [p\ v\ \sigma_p\ \sigma_v]$. The value function for this model is illustrated in figure 4.4 on a uniform grid of beliefs with fixed standard deviations. Also a heat map of the optimal actions (acceleration) is depicted. We see that there is an area in the state space where being close to the top of the mountain with low speed has a low value function. The reasoning is that the agent must spend valuable time moving backwards first to gain the necessary velocity to climb the hill. The belief state for the Car-on-a-Hill will be represented with a two dimensional Gaussian with no zero covariance.

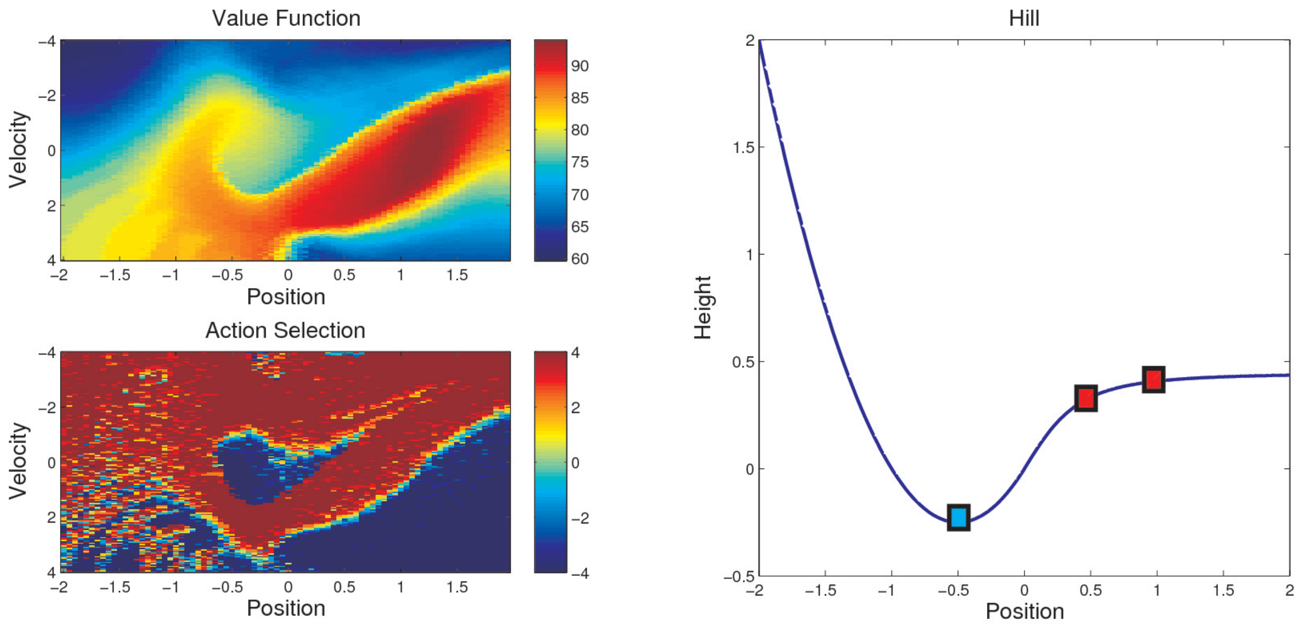


Figure 4.4: The top left figure indicates the value function for beliefs corresponding to position and velocity pairs. The standard deviations are of the belief parameters are ignored. The bottom image indicates the optimal action from beliefs. The color bar indicates the selected acceleration. Finally, the right plot illustrates the hill the car is balancing on. They cyan colored box indicates the vehicles initial position, while the red colored boxes indicate the bounds for the desired position.

4.2.5 Inventory Stock model

Consider the problem of managing the the inventory levels of an item in a store [57]. The manager needs to ensure that there are enough items to satisfy customer demand, but also is constrained by the amount of items stored due to holding costs and space considerations. The inventory level is measured as discrete times but it uncertain due to a variety of practical issues (misplacement, spoilage) [58].

We model this problem as a POMDP as seen in [58]. Let the total number of items stored by denoted with x and a holding cost of h incurred for storing each of the x items. At discrete times the stock my be replenished with exactly Q items or none at all. The demand d for an item is modeled as an exponential random variable with the parameter λ_d. A cost of s is also incurred if the demand exceeds the stock. The specifics of the model are as follows:

- X = the set of allowable item amounts, $[0, \infty]$.
- $U = \{0, 1\}$. The agent elects to restock with exactly Q or none of the item.
- $x^+ = max(x + uQ - d, 0)$
- The observation z is a gaussian distribution normally distributed on the true item stock level with standard deviation σ_x.
- The agent incurs a cost for holding each item and potentially not having enough supply to satisfy the demand:

$$g(x, u) = h\ max(x + uQ - d, 0) + s\ max(d - x - uQ, 0)$$

Model	$\lvert U\rvert$	$\lvert\theta\rvert$	Number of observations
Inventory	2	2	1
Navigate	33	4	5
Pose Navigate	7	7	6
Car-on-a-Hill	5	5	2

Table 4.2: A table listing the number of actions, length of belief state, and length of the observation vector for each model reviewed in this thesis.

The belief state is simply represented as a 1-dimensional Gaussian. We summarize all four models with the following table:

4.2.6 Competitor Algorithms

As a baseline we must compare our enhanced PPOMDP algorithm to the standard PPOMDP as well as other non PPOMDP substitutes. For the first batch of results we will look strictly at the performances of a discrete MDP method and Perseus algorithm. In both cases we will pay these algorithms every convenience in terms of memory and computation.

The discrete MDP solution [59] works by dividing the state space into a discrete number of cells. If we let s denote a cell, then we can approximate the cell transitions $p(s^+|s,u)$, the observation probabilities $p(z^+|s^+)$, and the reward $g(s,u)$ through sampling. Calculating $p(s^+|s,u)$ is accomplished by sampling from the cell s and using the system dynamics and action u to see what future cell the particle is moved to. Then $p(s^+|s,u)$ is equal to the number of particles landing in s^+ divided by the number of trials. For our calculating each cell s was sampled 50 times, and each particle sampled was propagated forward 5 different times yielding a total of 250 samples per transition calculating. It is unclear how to calculate $p(z^+|s^+)$ for the continuous case, therefore the range detection only yielded a boolean indication: either there was an obstacle within 2 meters or not. $p(z^+|s^+)$ was then approximated with a sampling scheme similar to $p(s^+|s,u)$.

Once the appropriate probabilities are calculated the POMDP becomes an MDP model and the corresponding value function may be calculated via VI. During the VI calculation the best action from each cell can be stored and used as a reference during plan execution. During the execution phase the agent keeps a probability distribution over the set of cells, updating the distribution according the the calculation of $p(s^+|s,u)$ and $p(z^+|s^+)$. The most likely state (cell with highest probability) indicates which action the agent will choose. The number of discrete cells for each model was set as 5000, a value far larger the number of beliefs for our PPOMDP (with the exception of Pose Navigate).

The Perseus solution is similar to the discrete MDP solution. First discrete number of points from the state space are selected and the appropriate transition probabilities between states are calculated via sampling. We then allow the agent to explore the this new state space holding a discrete probability distribution over the points. For training, a set of 2000 points are selected and 10,000 beliefs are collected to train the agent. The value function was then approximated according to the Perseus algorithm described in [36]. During plan execution, the best action was found by selecting the one that maximized the value function given by the hyperplane set output from the algorithm.

4.3 Generalizing the Transition Approximation

This section is dedicated to altering how the belief state transitions are approximated. We will show how a substantial amount of computation time can be saved and artificial sample can be generated.

4.3.1 Particles Sets and Posterior Beliefs

If we refer to algorithm 7 and the corresponding depiction in 4.1 we see that a matrix of size $N \times N$ of weights is created to calculate future belief states. The i^{th} column indicates the i^{th} particle sampled from a belief state θ representing a point in the model's state space. The j^{th} row is a set of weights assuming the agent truly received a measurement from the j^{th} particle. Thus, each row correspond to the same particle set of size N, but with a different set of weights as different observations are assumed. It is convenient to generate N weighted particle sets since the N observations are available. This assumes the the number of samples needed to represent θ as a particle set sets number of samples we must draw from $\theta^{+}|\theta, u$. In practice, the variety of beliefs after observing may be small. By taking belief samples than less N we can save considerable computation time during training From this point on we will discard the notation of N to describe both sample sizes in favor of:

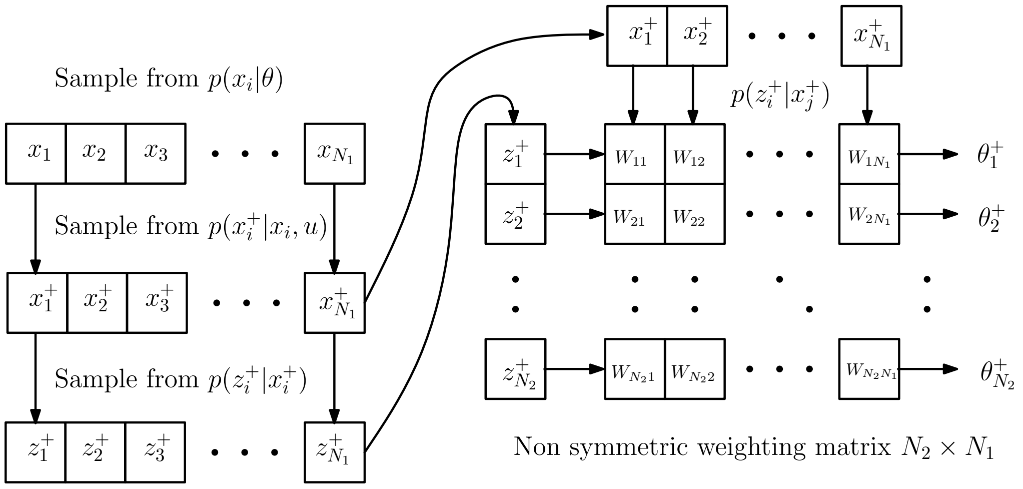
1. N_1: The number of particles sampled from a given belief state θ.
2. N_2: The number of weighted particle sets and belief states generated acting as samples from $\theta^{+}|\theta, u$.

We also add the constraint that $N_2 < N_1$ for efficient calculations. Note, we still use the full N_1 samples when projecting the weighted sets back to belief space, but only project N_2 times in total. The pseudocode and depiction of these changes can be found in algorithm 10 and figure 4.5

Computationally how is this going to effect our calculations during train?. Imagine a belief state that requires 1000 particles drawn for an accurate sample ($N_1 = 1000$). According to algorithm 7, we need to populate a matrix of size 1000×1000. Suppose, though, that the weighted particle sets are similar after sampling random observations. Then the number of future beliefs needed for

Algorithm 10 A slight modification to 10. Instead of a square matrix of weights of size $N \times N$, a non square matrix W is created of size $N_2 \times N_1$ where N_1 are the number of particles sampled from the given belief state and N_2 are the number of projected beliefs returned, assuming $N_2 < N_1$

```
Input: $\theta, u, N_1, N_2$
Output: $\{\theta_1^+, \theta_2^+, \cdots, \theta_{N_2}^+\}$
Let $W$ be a matrix of size $N_2 \times N_1$ holding the particle weights.
for $i = 1 : N_1$ do
    Sample $x_i$ from $p(x_i|\theta)$
    Sample $x_i^+$ from $p(x_i^+|x_i, u)$
    Sample $z_i^+$ from $p(z_i^+|x_i^+)$
end for
for $i = 1 : N_2$ do
    for $j = 1 : N_1$ do
        Calculate $W_{ij} = p(z_j^+|x_i^+)$
    end for
end for
Normalize the rows of $W$
for $i = 1 : N_2$ do
    $\theta_i^+ = Proj_\Theta\left(\{x_j^+\}_{j=1}^{N_1}, \{W_{ij}\}_{j=1}^{N_1}\right)$
end for
```



Sample from $p(x_i|\theta)$
x_1 x_2 x_3 x_{N_1}
Sample from $p(x_i^+|x_i, u)$
x_1^+ x_2^+ x_3^+ $x_{N_1}^+$
Sample from $p(z_i^+|x_i^+)$
z_1^+ z_2^+ z_3^+ $z_{N_1}^+$
x_1^+ x_2^+ $x_{N_1}^+$
$p(z_i^+|x_j^+)$
z_1^+ z_2^+ $z_{N_2}^+$
W_{11} W_{12} W_{1N_1}
W_{21} W_{22} W_{2N_1}
W_{N_21} W_{N_22} $W_{N_2N_1}$
θ_1^+ θ_2^+ $\theta_{N_2}^+$
Non symmetric weighting matrix $N_2 \times N_1$

Figure 4.5: A figure illustrating the calculation of future beliefs using a non-symmetric weighting matrix.

an accurate sample could be much smaller than 1000. Assume we have on good authority that this value is in the neighborhood of 100 ($N_2 = 100$). Then the weighting matrix need only be of size 100×1000, an entire 10 times less in terms of memory and computation. Additionally, only 100 projections of weighted particle sets must be computed rather than 1000. The complexity of training the agent changes from equation 4.6 to:

$$O\left(|G||U|\left(N_1C_1(x, x^+, z^+) + N_1N_2C_2(x^+, z^+) + N_2C_3(N_1) + N_2C_4(|G|)\right)\right) \tag{4.7}$$

Clearly we see that settings of $N_2 < N_1$ will be strictly faster than $N_1 = N_2$, but the question is how will this effect the agent's performance. To test our changes we fix $N_1 = 100$ while letting $N_2 \in \{5, 10, 20, 40, 60, 80, 100\}$. The settings $N_1 = N_2 = 100$ will allow us to compare to the standard PPOMDP procedure. We choose dense belief sets for all four models. Specifically, $|G|$ was set as 1000, 500, 5000, and 2000 for the Inventory, Navigate, Pose Navigate, and Car-on-a-Hill models respectively. Additionally, we compare our results to those accumulated by an agent that was trained using Perseus and a discrete MDP. Both Perseus and discrete MDP were allowed far greater training time and finer discretization than our PPOMDP parameters to establish a proper baseline to compare against as was described in the previous section.

Table 4.6 contains several interesting results. First we note that PPOMDP outperforms both Perseus and discrete MDP for each of the four models. We also notice that PPOMDP is unable to capture rewards near the theoretical optimal performance, something we we attempt to obtain in future sections. The most important result, however, it that values of $N_2 < N_1$ obtain comparable results to the standard PPOMDP setting of $N_1 = N_2$. Particularly, for the Car-on-a-Hill and Navigate models N_2 may be decreased to as little

N_2	Inventory	Navigate	Pose Navigate	Car-on-a-Hill
5	-39	8.5	-8.9	68
10	-22	8.9	-4.1	71
20	-21	9.2	-3.3	73
40	-22	9.2	3.3	72
60	-21	9.2	11.8	72
80	-21	9.3	11.8	73
100	-21	9.2	11.8	73
Perseus	-35	6.1	-20	65
Discrete MDP	-49	6.8	10.8	60
Max Reward	-13	9.3	14	84

Table 4.3: A table indicating the total rewards collected on average for all four models as a function of N_2. The bottom three rows indicate performance from the Perseus and Discrete MDP agent as well as an upper bound for rewards collected. N_1 is set constant at 100.

N_2	Inventory	Navigate	Pose Navigate	Car-on-a-Hill
5	.66s	9.6s	37s	2.8s
10	.80s	11.6s	42s	3.8s
20	1.05s	14.7s	53s	5.7s
40	1.53s	21.1s	71s	9.0s
60	1.91s	27.0s	90s	11.6s
80	2.27s	30.9s	101s	13.7s
100	2.61s	33.8s	108ss	15.6s

Table 4.4: A table indicating the total training time (seconds) for all four models as a function of N_2. N_1 is set constant at 100.

as 10 retain good performance. Then Pose Navigate model, however, clearly suffers from a good sample over the posterior beliefs, although there is still comparable performance when $N_2 = 60$. For smaller values of N_2 we see the performance predictably diminish. We refer to Table 4.7 to see the reduction in training times.

From Table 4.7 we see that a large portion of the training time must be spent on the projection of the particles sets onto belief space. The speed difference between $N_2 = 5$ versus $N_2 = 100$ is nearly 5 fold for all four models. We

Model	$N_2 < N_1$	$N_1 = N_2$	Train Time Reduction
Inventory	.80s	2.61s	31%
Navigate	14.7s	33.8s	43%
Pose Navigate	90s	108s	83%
Car-on-a-Hill	5.7s	15.6s	36%

Table 4.5: A table indicating the total training time (seconds) for all four models for settings of $N_1 = N_2$ and the smallest value of N_2 that receives comparable performance. The percentage of time is shown in the final column.

summarize the speed improvements in the following table where we compare the times when $N_2 < N_1$ with times when $N_1 = N_2$ and both have similar performance:

We see for the Navigate and Car-on-a-Hill models a dramatic reduction in training time when compared to the settings $N_1 = N_2$. For the Pose Navigate, there is a slight reduction in training time, but for smaller samples of the posterior belief the agent reaches the goal state less and less. The question is, can we effectively generate extra samples of the posterior belief with projecting more particles sets onto our parameterized distribution?

4.3.2 Generating Extra Posterior Belief Samples

If we look back to algorithm 8 we see that limiting N_2 is effecting the calculation of the transition probabilities between states in G. Each weighted particle set (N_2 per action and belief pair) is projected into the belief space and then "snapped" to the nearest belief in G. These "snapped" beliefs are then assigned nonzero transition probability. Zhou advocated "snapping" to only the single nearest neighbor in G, while Brooks uses an interpolation scheme that uses the nearest 10 neighbors. We argue that by assigning nonzero transition probability to more than just 1 neighbor for each of our samples $\theta_1^+, \cdots, \theta_{N_2}^+$, we effectively generate additional posterior belief samples. The only extra computational cost, while fixing N_2, is the extra burden performing a k-nearest neighbor search rather than the single nearest neighbor search. Algorithm 11 illustrates the changes to the transition approximation.

Algorithm 11 A generalization of $TransitionProbablities()$. As input this algorithm accepts the value N_3 which indicates the number of nearest neighbors to consider when approximating transitions between beliefs.

```
Input: $G = \{\theta_1, \theta_2, \cdots, \theta_{|G|}\}$, $N_3$
Output: An approximation to $p(\cdot|\theta_i, u)$ for $\theta_i \in G$ and $u \in U$ stored in the
variables $Tindices$ and $Tvalues$.
Initialize $counter$ as an array of length $|G|$.
for $i = 1 : |G|$ do
  for $j = 1 : |U|$ do
    $\Delta = GeneratePosteriors(\theta_i, u_j)$
    Reset $counter$
    for $k = 1 : |\Delta|$ do
      Let $index_m$ be the index of the $m^{th}$ closest member in $G$ to $\Delta_k$ for
      $m = 1 : N_3$
      $counter[index_m] \leftarrow counter[index_m] + 1$ for $m = 1 : N_3$
    end for
    $Tindices[i][j] \leftarrow$ non-zero indices of counter
    $Tvalues[i][j] \leftarrow \frac{counter}{|\Delta| N_3}$
  end for
end for
```

While Brooks used settings similar to our setting of $N_3 = 10$ (in his work the nearest neighbors were weighted according to distance from the sampled posterior), and Zhou used a setting of $N_3 = 1$, nobody has shown how settings $N_3 > 1$ can be interpreted as artificially inflating N_2. We are particularly interested in the performance of Pose Navigate as this is the only model that severely suffered as a consequence of small N_2. To test this generalization, we reran the previous experiment while varying the $N_3 \in 1, 2, 3, 4, 5$. We present the rewards and times for training while fixing $N_1 = 100$ and $N_2 = 10$ and varying N_3.

From Table 4.6 we can make several observations. We see that for the

N_3	Inventory	Navigate	Pose Navigate	Car-on-a-Hill
1	-22	9.2	-4.1	72
2	-22	8.9	-0.6	72
3	-21	8.7	11.8	73
4	-22	8.5	11.8	73
5	-22	8.1	11.8	71
Standard PPOMDP	-21	9.2	11.8	73
Perseus	-35	6.1	-20	65
Discrete MDP	-49	6.8	10.8	60
Max Reward	-13	9.3	14	84

Table 4.6: A table indicating the total rewards collected on average for all four models as a function of N_3. The bottom three rows indicate performance from the Perseus and Discrete MDP agent as well as an upper bound for rewards collected. N_1 is set constant at 100 while N_2 is set constant at the low value of 10.

Model	$N_2 = 10, N_3 = 3$	$N_2 = 100, N_3 = 1$	Train Time Reduction
Inventory	0.93s	2.61s	35%
Navigate	10.1s	33.8s	30%
Pose Navigate	43s	108s	40%
Car-on-a-Hill	3.6s	15.6s	23%

Table 4.7: A table indicating the total training time (seconds) for all four models for settings of $N_1 = N_2$ and the smallest value of N_2 that receives comparable performance. The percentage of time is shown in the final column.

Navigate model the total rewards collected on average slowly diminishes as N_3 is increased. But, the Pose Navigate model responds exactly as we had hoped. We see that for the setting of $N_2 = 10$ and $N_3 = 1$ the agent collects negative rewards, but for $N_3 >= 3$ the rewards collected become on par with the standard PPOMDP settings ($N_2 = 100$ and $N_3 = 1$). It appears, for models that require a large sample of posterior beliefs that using larger values of N_3 can effectively generate extra samples. This has to be done with care, though, as for the Navigate model suffers, while the other two models are indifferent. Finally, we reevaluate Table 4.7 with our updated results.

We have generalized the calculation of transition probabilities between belief states in G by introducing the parameters N_1, N_2, and N_3. We have shown that taking less samples over posterior beliefs than the number of particle samples representing $\theta \in G$ can result in comparable performance to the standard setting. We showed that values of $N_3 > 1$ can effectively generate posterior belief samples that boost the time savings for the Pose Navigate model. Our method trains agents from 2 to 5 times faster than the standard equivalent when we fix $N_1 = 100$. While these results are important, we do note that we failed to achieve performances that are comparable to theoretically optimal results. The next section reviews a forward search algorithm implemented by Brooks. We then implement our own forward search algorithm and demonstrate how it is capable of performing deeper searches than previously allowed while achieving higher performance.

4.4 Forward Searching without Future Observations

In the previous section we illustrated the average reward accumulated by an agent using a plan from a PPOMDP solution on a test bed of four models. A set of likely beliefs were selected and the transition probabilities between beliefs were approximated converting a continuous state POMDP into a discrete state MDP. While the MDP was solved during value iteration the best action from each state was easily stored at no extra computation cost and little additional memory. While acting the agent will never (excluding perhaps the initial state) hold a belief that exactly matches that from the belief set. The agent can quickly do a nearest neighbor search with the help of a KD-tree and find the closest belief state in the predefined set. Since the best action from this point was saved this serves as suitable action to select assuming a high density of beliefs. However, if the density of beliefs is sparse, there may arise situations where the snapping effect to the nearest belief results in very poor decision making as illustrated in Figure 4.6. But computationally, we prefer sparse belief sets in terms of ease of computation. Following and expanding upon the work of Brooks in [1], we develop forward planning techniques that allow the agent to search the belief space before snapping to the nearest belief in the set and show how this results in dramatically smaller belief sets and shorter training time needed for high performance on the previously mentioned robot navigation model.

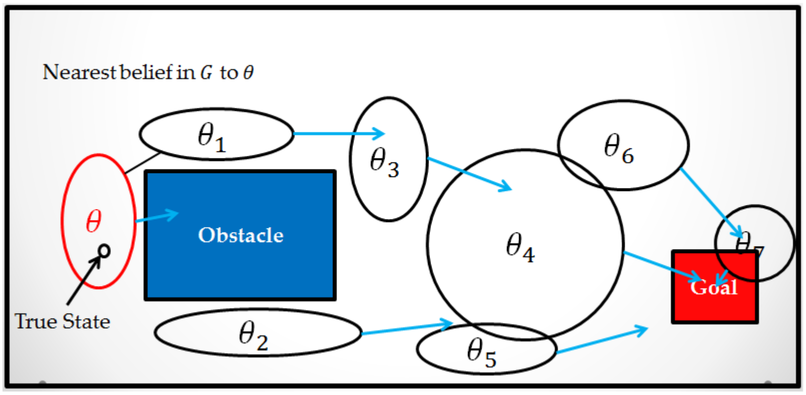


Figure 4.6: This figure depicts an agent that believes it is just right of an obstacle. The nearest reference belief, though, is clear of the obstacle and selects an action moving toward the right and towards the goal region. This dooms the agent to repeatedly collide with the obstacle even though the belief is closely aligned with its true position and illustrates the so called "snapping effect" when referencing the belief set for actions.

The following will describe forward planning algorithm applied to PPOMDP's presented in [1]. We will then show how it can be improved to drastically reduce computational cost. Two additional forward planning techniques will be introduced and their performance in terms of agent success and action selection time will be compared.

4.4.1 PPOMDP as a Game Tree

To prevent potentially poor decisions an agent needs to be equipped with an action selection strategy more informed than taking the action stored from the nearest belief point in the set. Consider the agent illustrated in 4.6. The agent here accurately has a belief that coincided with the actual unobserved true state. But the closest belief point used as a reference is slightly beyond an obstacle obstructing the agent. If the agent elects to choose the action stored in at that point it will inevitably run itself into the obstacle. This may repeat several times until the agent's belief shifts to one closer to a different belief point in the set that stored a better action suitable for the agent's position.

An intelligent agent should be able to predict its location in belief space after an action (or series of actions). Its goal is then to choose the action that results in a high reward and will leave the agent at a location where future rewards are also available. Calculating the reward from a belief state and action pair is straightforwardly approximated via algorithm 9. Deciding the worth of being in a belief state is also simple and similar to the original action selection. The agent simply finds the J value of the closest belief point calculated during value iteration. Brooks in [1] uses a weighted interpolation scheme over a set of belief points within range, while Zhou in [44] uses only the nearest neighbor. We also only use only the nearest neighbor saving some computation when comparing our results to their algorithm.

Typically, an agent benefits from expanding its forward search as far as possible. We see this particularly in the realm of game theory and in the analysis of chess artificial intelligence. In chess, two players alternate selecting actions. An individual player is trying to select an action that advances his board state closer to a game winning position, while preventing his opponent from doing the same. The farther a player can look ahead to consider the ramifications of an action, the more informed his decision will typically be (excluding a horizon effect) [60]. The term *ply* or *depth* refers to the number of actions considered sequentially before evaluating the board position. A board position is heuristically evaluated according to the number and position of chess pieces. One can exhaustively create a tree of all the possible board positions resulting from any combination of actions up until some predefined depth d. Unfortunately, this tree exponentially expands as a function of search depth. Once a tree is constructed a player may evaluate an action assuming the he will select the best action for himself and that his opponent will do the opposite. An algorithm of this type is called a *Min-Max* algorithm and can be used in any deterministic two player game [61]. This algorithm was significantly improved with the advent of the *Alpha-Beta* algorithm, mitigating the exponential explosion of board states allowing for deeper searches. Alpha-beta search operates by pruning out branches of the tree that are provably worse than an action already considered [62].

The agent in a PPOMDP setting has a similar decision to make as a player involved in a chess game, but without an adversary making decisions to maliciously thwart its goals. There is in some context, though, another player effecting our agent's progression. While an agent has complete control over its actions, the future true state and observed measurement are randomly decided. We can think of this effect as Nature effecting our position (or belief in our case), but in a random rather than adversarial manner. Additionally, since Nature's actions (or the agent's observations) are continuous, making a tree of future beliefs can only be accomplished by sampling a discrete number of observations. The leaves of our sampled tree hold potential future belief states after an action is selected and observation is collected. These leaves can be evaluated by the closest J value in the predetermined belief set. Unfortunately, since Nature is not knowingly competing against the agent, the Min-Max and Alpha-Beta algorithms are not at our disposal.

4.4.2 PPOMDP Game Tree with Observations

The game tree in the PPOMDP setting consists of two types of nodes each representing a point in belief space: *action* and *observation* nodes. There are $|U|$ edges leaving an action nodes, one for each of the available actions, and all edges transition to an observation node. An observation node represents the agent's belief immediately after it acts but before it incorporates an observation into its belief. The branching factor from an observation node should be $|Z|$ assuming a discrete set of measurements, but is a bit more complicated assuming an infinite set as in the robot navigation model. Instead of enumerating all possibilities at these nodes, future observations may be sampled limiting the branching factor. Recall, that we generalized how posterior beliefs were sampled. We use N_1 to represent the number of samples drawn from θ and N_2 to represent the number of future beliefs sampled $\theta^+|\theta, u$. Thus, in our implementation of a forward search the branching from an action node is N_2. The total number of leaves in this game tree considering a forward search of depth d results in a need to evaluate $(|U|N_2)^d$ belief states. The robot navigation model was defined as having 33 action. If we set N_2 at a very low value of 10, then a single depth search requires 330 evaluations. Searching one additional ply requires $(330)^2$! Both Chess and Go have (on average) a far smaller branching factor of 35 and 200 respectively [63], as well as being able to take advantage of the Alpha-Beta pruning. Thus, the exhaustive tree search for the PPOMDP controller will be limited to only 1 or 2 ply in the current state. This motivates us to find a method to increase our search depth. The main problem with this forward search is the double branching factor. Even if we elect to choose a very small value of N_2 (the number of future states we branch to from an action), this value will still have an exponential effect on our computation as a function of depth.

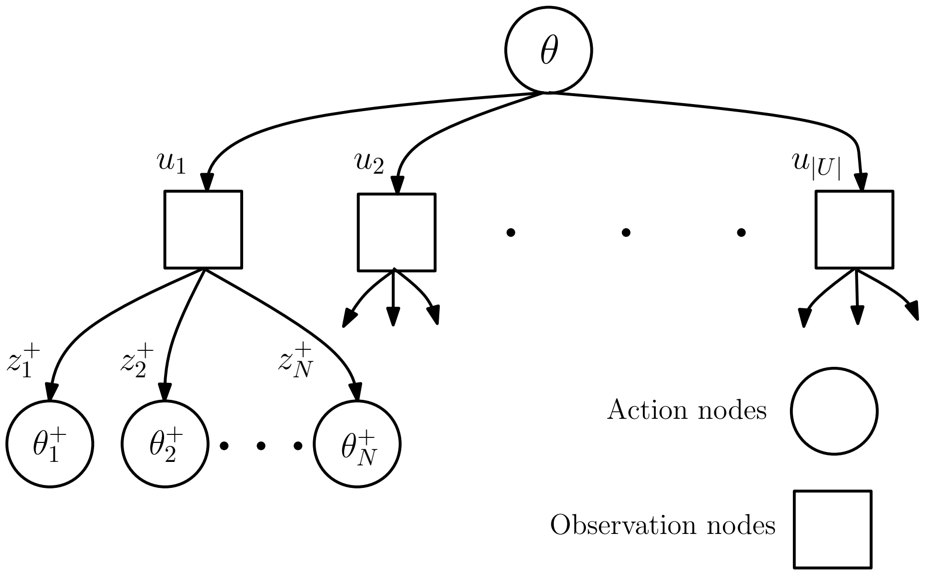


Figure 4.7: A figure adapated from [1] illustrating the the game tree resulting from an initial belief θ. First the agent branches out considering all actions, and then branches again according to possible future observations.

Koenig, in [64, 65], works on a robot navigation task similar to the Pose Navigate Model presented. The focus of his work is interleaving planning and plan execution for real time action selection on non-deterministic domains. The randomness in his navigation model is represented by the fictitious opponent *Nature*. The model, as with ours, has *Nature* acting randomly, but the agent in Koenig's work assumes that *Nature* is both intelligent and adversarial. That is, *Nature* will actively attempt to deny the agent access to a goal state. Applying this to our work is simple. Instead of taking the average over observations when evaluating the game tree we take the minimum. If we were to anthropomorphize the agent searching this way, we can consider it acting

Algorithm 12 A recursive forward search algorithm **EvaluateParticles** returning the value of being in an initial state and storing the predicted optimal sequence of actions. The subfunction **UpdateParticles** applies the selected action to each of the particles, **MakeMeasurements** stochastically samples a measurement from each particle, **CalculateStateActionReward** returns the weighted average reward of taking an action from each particle, **UpdateWeights** reweights the particle set given the assumed true measurement z_j^+.

```
Input: An initial particle set $\{x_1, x_2, \cdots, x_{N_1}\}$ with weights
$\{w_1, w_2, \cdots, w_{N_1}\}$, search depth $d$, and the total reward path $pathReward$.

Output: An approximation of the total expect future rewards and best next
action.
$actionCost \leftarrow 0$
$MaxValue \leftarrow -\infty$
Let $X$ denote $\{x_1, x_2, \cdots, x_{N_1}\}$
Let $W$ denote $\{w_1, w_2, \cdots, w_{N_1}\}$
for $i = 1$ : Number of actions do
  $action \leftarrow i^{th}$ action
  $\{x_1^+, x_2^+, \cdots, x_{N_1}^+\} \leftarrow$ UpdateParticles$(X, action)$
  Let $X^+$ denote $\{x_1^+, x_2^+, \cdots, x_{N_1}^+\}$
  $\{z_1^+, z_2^+, \cdots, z_{N_1}^+\} \leftarrow$ MakeMeasurements$(X^+)$
  Let $Z^+$ denote $\{z_1^+, z_2^+, \cdots, z_{N_1}^+\}$
  $meanValue \leftarrow 0$
  $actionReward \leftarrow$ CalculateStateActionReward$(X, W, action)$
  for $j = 1 : N_2$ do
    $\{w_1^+, \cdots, w_{N_1}^+\} \leftarrow$ UpdateWeights$(X^+, W, Z^+, z_j^+)$
    Let $W^+$ denote $\{w_1^+, \cdots, w_{N_1}^+\}$
    $R \leftarrow actionReward + pathReward$
    $meanValue \leftarrow meanValue +$ EvaluateParticles$(X^+, W^+, d-1, R)$
  end for
  $meanValue \leftarrow meanValue/N_2$
  if $meanValue > MaxValue$ then
    Store $action$ as best action for depth $d$
    $maxValue = meanValue$
  end if
end for
return $maxValue$
```

extremely risk adverse avoiding scenarios with possible low rewards even if the outcome is unlikely. If we take the minimum over observations rather than the average, then this would allows us to use results from *Min-Max* and *Alpha-Beta* literature. Recall, *Alpha-Beta* prunes out sections of the search tree that lead to provably worse scenarios than we know we can maneuver to. But, the double branch factor for each action still remains.

4.4.3 PPOMDP Game Tree without Observations

How can we reduce the number of evaluations for forward searching in the PPOMDP setting? What we need to realize is that the observations available to the agent are a convenience. Recall, the agent makes inferences about its location in state space based on its knowledge of the state transition function and the noisy measurement dynamics. If observations are completely neglected the agent can still make principled estimates of its location. We propose to modify the forward search by electing not to sample future observations. Essentially the agent simulates blind actions through state space. We expect the belief states generated from very deep searches to be very uninformed, that is the beliefs will have variance parameters. But, the search tree expansion only branches on the number of actions $|U|$. A search without considering observations to a depth of d has $|U|^d$ leaves opposed to $(|U|N_2)^d$ leaves if we evaluated via Brooks' forward search. Clearly our method is much more efficient computationally, but the leaves are our tree are more unrealistic beliefs. But, will the deeper depths allowed by our search allow for better performance.

To test the difference in performance between our proposed forward search and Brooks', we first evaluate the length of time it takes to select an action as a function of search depth and model. We set $N_1 = N_2 = 100$ when searching via Brooks' method. That is, we remove the sampling generalization we made

Inventory, $\lvert U \rvert = 2$		
Search Depth	No Observations	With Observations
1	.001s	.002s
2	.001s	.384s
3	.001s	NA
4	.001s	NA
5	.001s	NA

Table 4.8: A table of the times to select an action as a function of search depth for the Inventory model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

Navigate, $\lvert U \rvert = 33$		
Search Depth	No Observations	With Observations
1	.003s	.065s
2	.054s	204s
3	2.91s	NA

Table 4.9: A table of the times to select an action as a function of search depth for the Navigate model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

about state particles and posterior beliefs. For our search, we set $N_1 = 100$, N_2 is irrelevant as we are not sampling future observations.

From the previous tables, we see that forward searching via Brooks' algorithm is reasonably limited to a search depth of 1. A search depth any deeper results in action selection times greater than .1 seconds, which we set as a reasonable cut off on the time allotted for an autonomous agent to select an action. Note, for the Navigate model it takes a stunning 204 seconds to select a **single** action with a search of depth 2 when considering the impact of observations. On the other hand, for our search algorithm we are able to obtain reasonable searches of depths 5, 2, 3, and 5 for the Inventory, Navigate, Pose Navigate, and Car-on-a-Hill models respectively.

We test the performances as a function of realistic search depth for both

Pose Navigate, $\lvert U \rvert = 7$		
Search Depth	No Observations	With Observations
1	.003s	.022s
2	.008s	15.72s
3	.054s	NA
4	.391s	NA
5	2.87s	NA

Table 4.10: A table of the times to select an action as a function of search depth for the Pose Navigate model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

Car-on-a-Hill, $\lvert U \rvert = 5$		
Search Depth	No Observations	With Observations
1	.001s	.009s
2	.001s	4.51s
3	.005s	NA
4	.019s	NA
5	.084s	NA

Table 4.11: A table of the times to select an action as a function of search depth for the Car-on-a-Hill model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

Inventory, $\lvert U \rvert = 2$				
	$\lvert G \rvert = 20$		$\lvert G \rvert = 1000$	
Search Depth	No Obs.	W/Obs.	No Obs.	W/Obs.
0	-26	-26	-21	-21
1	-13.7	-14.2	-13.6	-13.9
2	-13.6	NA	-13.6	NA
3	-13.6	NA	-13.6	NA
Max. Reward	-13	-13	-13	-13

Table 4.12: A table of the average rewards collected as a function of search depth and search type for the Inventory model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

Navigate, $\lvert U \rvert = 33$				
	$\lvert G \rvert = 200$		$\lvert G \rvert = 500$	
Search Depth	No Obs.	W/Obs.	No Obs.	W/Obs.
0	7.4	7.4	9.2	9.2
1	8.0	8.2	9.2	9.2
2	9.2	NA	9.2	NA
Max. Reward	9.3	9.3	9.3	9.3

Table 4.13: A table of the average rewards collected as a function of search depth and search type for the Navigate model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

forward searches. In addition, we also test on sparse and dense belief sets. We are interested to see if our deep searches can compensate for sparse belief sets if they will outperform Brook's forward search. The results are summarized in the following tables.

We see that for the Inventory model, either forward search accumulates rewards very near the optimal values. Also the agent is indifferent to very sparse belief sets ($|G| = 20$), which is likely due to the low dimension state space (only 1). For the Navigate model we see that the agent accumulates very high rewards without a forwards search with a dense belief set. When this set

Pose Navigate, $\lvert U \rvert = 7$				
	$\lvert G \rvert = 2000$		$\lvert G \rvert = 5000$	
Search Depth	No Obs.	W/Obs.	No Obs.	W/Obs.
0	10.4	10.4	11.8	11.8
1	10.3	11.6	12.0	12.5
2	11.9	NA	12.5	NA
3	12.3	NA	12.7	NA
Max. Reward	14	14	14	14

Table 4.14: A table of the average rewards collected as a function of search depth and search type for the Pose Navigate model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

Car-on-a-Hill, $\lvert U \rvert = 5$				
	$\lvert G \rvert = 250$		$\lvert G \rvert = 2000$	
Search Depth	No Obs.	W/Obs.	No Obs.	W/Obs.
0	16	16	72	72
1	10.2	9.2	72.2	80.3
2	13.9	NA	79.8	NA
3	45.7	NA	81.7	NA
4	57.9	NA	82.1	NA
5	73.1	NA	83.1	NA
Max. Reward	84	84	84	84

Table 4.15: A table of the average rewards collected as a function of search depth and search type for the Inventory model. The sampling parameters are set as $N_1 = N_2 = 100$. NA corresponds to settings where the time to select an action becomes prohibitive.

is reduced to size 200 the zero depth is only able to accumulate rewards on average equal to 7.4. With our forward search of depth 2 we can bring this back to a value of 9.2. Due to the limited nature of searching with observations the competitor algorithm only obtains rewards equal to 8.2. We see similar results with Pose Navigate. Our search algorithm searches deeper and obtains rewards on a sparse belief set close to that of the dense belief set and out performs Brooks' algorithm. The most promising results are from the Car-on-a-Hill model. Our search to a depth of 5 on a sparse belief set accumulates rewards on average equal to 73.1. The competitor forward search may only search to depth 1 and achieve an average reward of 9.2. With our search we almost reach the peak rewards on an extremely sparse belief set.

4.5 Adaptive Sampling

In the previous section we saw that there are a variety of ways that the accuracy of the PPOMDP algorithm can be tuned that have a great effect on the training time, action selection time, and performance. In summary, during training the user must choose:

1. N_1, the number of particles sampled from a belief state.

2. N_2, the number of belief states to project from weighted particle sets.

3. N_3, the number of nearest neighbors the sampled belief states "snap" to when approximating transition probabilities.

Additionally, during plan execution the user may use different parameters (excluding N_3) to adjust search depth and accuracy. A wise choice (balancing performance and computational resources) of parameters will certainly change according to the agent's computational power and model. As well, in a fixed model there may exist places in the belief state space where more or less samples are necessary to satisfy a minimum degree of accuracy. This motivates us to incorporate an adaptive sampling scheme to ensure errors are minimized throughout training, plan execution, and across models. First we will review adaptive sampling techniques in the particle filter literature and then apply them to the PPOMDP algorithm. Then we will show how these techniques work across models and simplify the user's input while maintaining efficient training time, execution time, and performance.

4.5.1 Adaptive Sampling

Particle filters have been modified and tuned to solve problems in a diverse set of problems ranging from speech recognition to robot navigation [66]. This class of algorithms is attractive due to their ability to represent arbitrary probability density functions [16]. Throughout the years a number of improvements have been made to deal with specific deficiencies in the particle filter. In the review [13], Doucet explains a *Resample-Move* strategy that "herds" particles towards high likelihood locations in state space with Markov Chain Monte Carlo move steps. The point is to ensure the quality of the sample set. The Ensemble Particle Filter, seen in [67] and derived from the Ensemble Kalman filter in [68], has been shown to perform well on highly nonlinear model that exhibit multimodal behavior. Particle filters and genetic algorithms were merged in [69] as a resampling scheme to combat the well known degeneracy problem. The purpose of many of these improvements is to create a diverse set of particles that maintain an accurate representation of the underlying density function for a **fixed** number of particles.

The run time of any particle filter will be subject to the number of particles used. A tradeoff exists between using less particles for computational efficiency at the cost of possibly poor accuracy, and thus motivates an automated system for determining particle set size adaptively. During the course of a particle filter algorithm the complexity of the estimated probability distribution will vary over time [70]. For simple distributions we desire less particles, while for difficult and complex we require more. Consider the two areas of sampling in the PPOMDP algorithm:

1. Particles (or possibly true states of the agent) are sampled from belief states to estimate future distributions. For broad and uncertain belief states, we expect a large number of samples. For certain and low variance belief states we require less samples for an accurate particle representation.

2. Future belief states after acting are calculated (in the PPOMDP algorithm) by assigning different weights to the same particle set. The weights are a function of a predicted future observation. If a measurement conveys little information, the sets of weights will be similar requiring less projected beliefs to represent future possibilities.

Specifically, we need an algorithm that adjusts the values of N_1 and N_2 during training and plan execution, and also across models.

There have been attempts through the years to adjust the particles in a mathematically justified manner. In particular there is a likelihood-based adaptation method [71, 72] which adjusts the number of particles based on there non-normalized likelihoods. Samples are drawn until the sum of the likelihoods exceeds a user defined threshold. While this was shown to improve the quality of particle filters [71, 72], Fox explains in [73] how this approach does have deficiencies and samples less particles than is appropriate. He then proposes and derives an adaptive sampling scheme based on Kullback-Leibler distance (KL-distance) and will be our method of choice and explains in what follows.

4.5.2 Kullback-Liebler Distance Sampling

Kullback-Liebler Distance-sampling (KLD-sampling) in [70, 73] is motivated by the following quote from Fox

"At each iteration of the particle filter, determine the number of samples such that, with probability $1-\delta$, the error between the true posterior and the sample-based approximation is less than ϵ."

Recall the KL-distance between two probability distributions p and q

$$K(p,q) = \sum_x p(x) log\left(\frac{p(x)}{q(x)}\right).$$

The KL-distance establishes a measure (although not a metric, since it is not symmetric) that can be interpreted as a distance between probability distributions. $K(p,q)$ is zero if p and q are identical, and grows the larger the disparity between the two. Let $\underline{X} = \{X_1, \cdots, X_k\}$ be a sample of n draws from k bins. Assume $\underline{X}$ follows a multinomial distribution, that is $\underline{X} \sim Multinomial(n, \underline{p})$ where $\underline{p} = \{p_1, \cdots, p_k\}$ denotes the probabilities of drawing from each of the k bins. The maximum likelihood estimate (MLE) of $\underline{p}$ is given by

$$\underline{\hat{p}} = n^{-1}\underline{X} = \{n^{-1}X_1, \cdots, n^{-1}X_k\}$$

The likelihood ratio statistic can be written in terms of the KL-distance as

$$log\left(\lambda_n\right) = n\sum_{i=1}^{k} \hat{p}_j log\left(\frac{\hat{p}_j}{p_j}\right) = nK(\underline{\hat{p}}, \underline{p}).$$

The likelihood ratio is known to converge [74] to the chi-square distribution with $k-1$ degrees of freedom, that is

$$2log\left(\lambda_n\right) \rightarrow_d \chi^2_{k-1} \text{ as } n \rightarrow \infty$$

Fox shows in [73] that is n is selected to satisfy

$$n = \frac{1}{2\epsilon}\chi^2_{k-1,1-\delta}$$

then it can be guaranteed that the probability $P_{\underline{p}}(K(\hat{\underline{p}}, \underline{p}) = 1-\delta$. In practice, $\chi^2_{k-1,1-\delta}$ is approximated using

$$n = \frac{k-1}{2\epsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right)^3 \tag{4.8}$$

where $z_{1-\delta}$ follows the quantile of the standard normal distribution.

This gives us clear guidance for sampling from a multinomial distribution, but the distributions encountered so far have all ben continuous. This problem is alleviated by creating a uniform grid of bins over the support or using a multidimensional tree structure [75] (such as a KD-tree). When using a KD-tree, a point is “binned” at the nearest neighbor in the tree. A second problem is that the number of bins with non-zero probability are unknown during an iteration of the filter. That is, we don't know what value k should take. This is overcome by keeping track of the number of bins that the particles fall into, and updating the value of k as particles are sampled. This adaptively increases the number of particles to be sampled. Eventually, as the bins fill, the estimate of k is unchanged and the samples reach the value of n specified by 4.8. Algorithm 13 is presented and adapted from [70].

This gives us clear guidance for sampling from a multinomial distribution,

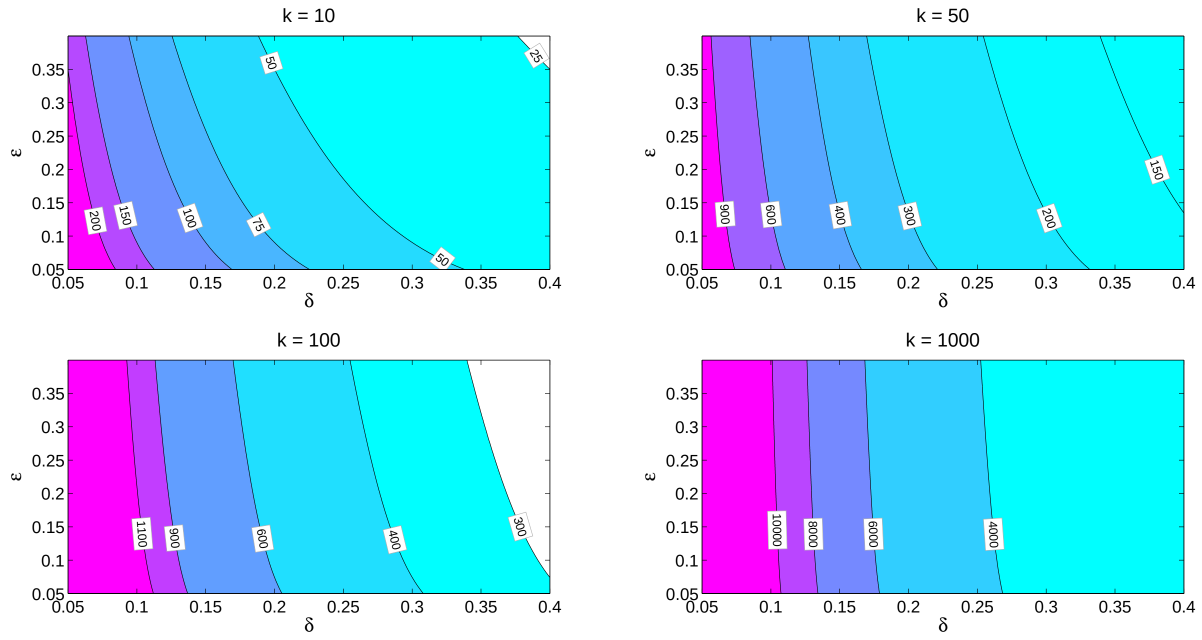


Figure 4.8: A figure showing four contour plots of the number of samples needed to satisfy the KLD-sampling criterion for values of ϵ and δ. Each figure corresponds to a different value of k, the number of bins with positive support.

Algorithm 13 The basic KLD sampling algorithm. This psuedocode ignores some of the details in particle sampling and weighting in exchange for readability. Also, the variable n_{min} is defined to ensure that at least n_{min} samples are drawn. Let n_χ be the minimum number of samples necessary to satisfy 4.8.

```
Let $k = n_\chi = n = 0$, $S =$
while $n < n_{min}$ and $n < n_\chi$ do
  Sample $x^{(n)} \sim X$
  $S \leftarrow S \cup x^{(n)}$
  if $x^{(n)}$ in an empty bin then
    $k \leftarrow k + 1$
    Mark the bin $x^{(n)}$ falls into as not empty
    $n_\chi \leftarrow \frac{k-1}{2\epsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right)^3$
  end if
  $n \leftarrow n + 1$
end while
return $S$
```

but the distributions encountered so far have all ben continuous. This problem is alleviated by creating a uniform grid of bins over the support or using a multidimensional tree structure [75] (such as a KD-tree). When using a KD-tree, a point is "binned" at the nearest neighbor in the tree. A second problem is that the number of bins with non-zero probability are unknown during an iteration of the filter. That is, we don't know what value k should take. This is overcome by keeping track of the number of bins that the particles fall into, and updating the value of k as particles are sampled. This adaptively increases the number of particles to be sampled. Eventually, as the bins fill, the estimate of k is unchanged and the samples reach the value of n specified by 4.8. Algorithm 13 is presented and adapted from [70].

There are two areas where KLD-sampling is applicable: sampling from the belief state and generating future belief states. From the previous section we set these values to N_1 and N_2, respectively. Ideally we can remove their specification and replace it with values more closely related with accuracy:

Model	Average N_1	Average N_2
Inventory	638.8	77.2
Navigate	96.8	6.3
Pose Navigate	218.3	9.8
Car-on-a-Hill	138.8	10.6

Table 4.16: A table of the average N_1 and N_2 selected for each of the four models

δ and ϵ. Algorithm 14 is an adapted two stage KLD-sampler specifically to accomplish this. Given an initial belief state, action, and N_3, 14 first generates a particle set along with sampled measurements. It then calculates weighted particles sets, projects them onto the belief space, generating n_3 future beliefs per weighted particle sets. Thus, each iteration (each weighted particle set) increases the number of samples in the second stage by n_3, and potentially increases k by n_3 as well. Before, we had the stipulation that $N_2 \leq N_1$, and similar reasoning applies here as well. Each particle yields one measurement and thus the number of future beliefs is limited by the number of particles sampled in the first stage of the algorithm. Additionally, this method requires two multidimensional trees for the binning process, one for the model's state space and another for the belief space.

We now have a mathematically principled way of sampling a particle representation of θ and distribution over posteriors $\theta^+|\theta, u$. Just by specifying the parameters ϵ and δ, algorithm 14 effectively adjust N_1 and N_2 during training and across models. In earlier sections we made the claim that in general $N_2 < N_1$ will provide adequate sampling for high performance. We demonstrated this numerically, but now we have the adaptive sampler to further verify this claim more systematically. We first compile frequency tables for the selected values of N_1 and N_2 across all four of our models during training. We also list the average values of N_1 and N_2.

Algorithm 14 The KLD algorithm adapted for the two stage sampling encountered in PPOMDPs. The first stage creates and particle representation of a given belief. The second stage creates and projects weighted particle sets until the KLD-sampling criteria is satisfied.

```
Let $k = n_\chi = n_a = 0$, $S_a = S_b = Z = \emptyset$
First stage: sampling particles from a given belief state
**while** $n_a < n_{min}$ and $n_a < n_\chi$ **do**
    Sample $x^{(n_a)} \sim X$
    Sample $x^{(n_a),+} \sim p(x^{(n_a),+}|x^{(n_a)}, u)$
    Sample $z^{(n_a)} \sim p(z^{(n_a)}|x^{(n_a),+})$
    $S_a \leftarrow S_a \cup x^{(n_a),+}$
    $Z \leftarrow Z \cup z^{(n_a)}$
    **if** $x^{(n_a)}$ in an empty bin **then**
        $k \leftarrow k + 1$
        Mark the bin $x^{(n_a)}$ falls into as **not empty**
        Update $n_\chi$
    **end if**
    $n_a \leftarrow n_a + 1$
**end while**
Let $k = n_\chi = n_b = counter = 0$
Second stage: sampling future beliefs
**while** $n_b < n_{min}$ and $n_b < n_\chi$ **do**
    **if** $counter == n_a$ **then**
        Halt the while-loop
    **end if**
    Let $newBelief$ be the projection of the particle set $S_a$ using measurement $z^{(counter)}$
    Find the $n_3$ nearest neighbors of $newBelief$ in $G$
    Update $k$ for all unbinned neighbors
    Update $n_\chi$
    $n_b \leftarrow n_b + n_3$
    $counter \leftarrow counter + 1$
**end while**
```

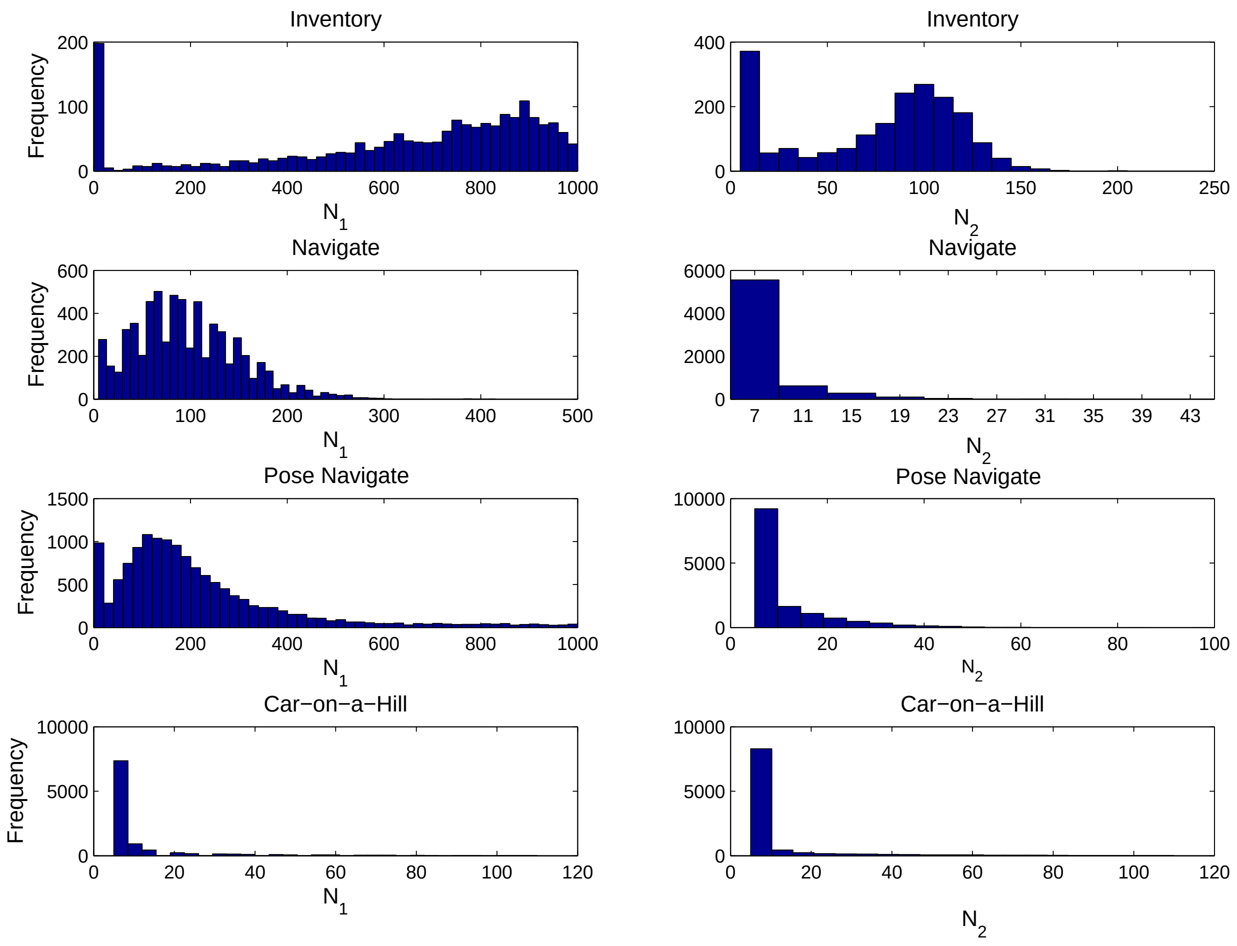


Figure 4.9: A figure showing the distributions of N_1 and N_2 selected by our adaptive sample while training an agent on all four models. Each row corresponds to a different model. The right column contains the frequencies for N_1, while the left row contains the frequencies for N_2.

From Figure 4.9 we see that our adaptive sampler is adjusting the values of N_1 and N_2 during the course of training. We see that the distribution of sampling parameters is also changing across the models as we had hoped. Table 4.5.2 verifies our claims about $N_2 < N_1$. We see for all four models that the adaptive sampler choose and order of magnitude less samples of the posterior belief distribution than the particles representing θ.

We need to verify that the adaptive sampler is capable of training an intelligent agent in a reasonable amount of time. We have no standards to measure against as we have the only PPOMDP adaptive sampler for training. We will, however, break our sampler into three versions of varying complexity. Recall that when considering a weighted particle set, the effective sample size can be measured as:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} \left(w^i\right)^2}. \tag{4.9}$$

We can save some computation time by using particles sets smaller than N_1 when we project them onto the belief space.

We list three versions of our adaptive sampler:

- Method #1: Adaptively sample N_1 and N_2. Limit the number of particles used in the projection of the particle set back onto belief space. Use the stopping convention of $N_{eff} > \frac{N_1}{2}$.

- Method #2: Adaptively sample N_1 and N_2. Use the full particle sets to project back to belief space.

- Method #3: Adaptively sample only N_1. Let the number of posterior beliefs be equal to N_1.

Method #1 is a small computation improvement over Method #2. We consider Method #3 to be a comparison to Brooks and Zhou if they were to implement

Model	$\lvert G\rvert$	Depth	M. #1	M. #2	M. #3	Fixed
Inventory	20	2	-13.4	-13.5	-13.4	-13.6
Navigate	200	2	9.1	9.1	9.2	9.2
Pose Navigate	2000	3	12.2	12.3	12.2	12.3
Car-on-a-Hill	250	5	73.0	73.5	72.3	73.1

Table 4.17: A table of average rewards collected for three adaptive samplers and one fixed. The fixed sampler used values of $N_1 = N_2 = 100$. The adaptive sampler used values of $\epsilon = \delta = .1$.

Model	$\lvert G\rvert$	Depth	M. #1	M. #2	M. #3	Fixed	M. #3 / M. #1
Inventory	20	2	.03s	.06s	.48s	.04s	16.0X
Navigate	200	2	4.7s	6.0s	19.4s	10.3s	4.1X
Pose Navigate	2000	3	50.6s	65.8s	208s	45.2s	4.1X
Car-on-a-Hill	250	5	.61s	1.19s	6.95s	1.62s	11.4X

Table 4.18: A table of average rewards collected for three adaptive samplers and one fixed. The fixed sampler used values of $N_1 = N_2 = 100$. The adaptive sampler used values of $\epsilon = \delta = .1$.

an adaptive sampler as both used the settings $N_1 = N_2$. The search times and performances of our algorithms are listed in the tables below.

From Table 4.5.2, we see no significant reduction in average rewards collected for our adaptive sampler. All three, and in particular Method #1, perform at the highest standards set thus far. We are of course interested in how the training times of all three methods compares. We of course do not want a sampler that "over-samples". Table 4.5.2 shows illustrates the times to train. In particular, we look at the difference between Method #1 and Method #3. We see that by allowing $N_2 < N_1$ that our adaptive sampler trains agents 4 to 16 times faster than the more simplistic sampler. Also, we note that our sampler trains a times less than the fixed settings $N_1 = N_2 = 100$, with the exception of the Pose Navigate model.

Chapter 5

Conclusions

In this thesis we proposed and developed several novel improvements to the Parametric POMDP algorithm for continuous state spaces and demonstrated their effectiveness on a variety practical POMDP models. Our improvements allow for a significant reduction in the computation aspects of training an agent and plan execution.

In the first part of our contributions we decoupled two areas of sampling used to approximate transitions probabilities between belief states from a discrete set of beliefs. Specifically, we allowed the number of particles representing a belief state to be of a different value of than the number of sampled posterior beliefs. We demonstrated that by allowing these values to be different we could save a considerable amount time training autonomous agents on a variety of models. Our methods were shown to be 2 to 5 times faster than the current standards when matching similar parameter settings for the sampling during belief state transitions. We also showed that we could effectively generate extra posterior belief samples by doing a k-nearest neighbor search when approximating the transition probabilities. Specifically, we demonstrated this on our adaptation of robot navigation with a pose state. We showed that an agent was able to collect high rewards a high number of samples were drawn. As we decreased the posterior sample size the rewards predictably declined as well. When only 10 samples were drawn the agent typically catastrophically failed to reach the goal region. As we introduced multiple neighbors during the training we saw training improve to the standards set by a full 100 samples.

Secondly, we implemented a new forward search algorithm. Our search algorithm branched only on the number of actions available to the agent rather than branching on number of actions and number of observations sampled. Our tree required the evaluation of a $|U|^d$ leaves where other similar approaches were required to evaluate a tree that branched twice. In the context of our parameters, the tree would have $(|U|N_2)^d$ leaves to evaluate. To reduce the branching factor so substantially, we ignored the impact of observations on future belief states altogether. We showed how this resulted in much more reasonable search times (where we define "reasonable" as being a search taking less than 1 seconds) for depths from 2 to 5 depending on the parameters of the model. We showed how the only other standard method may only search up to a depth of 1 before the action selection time becomes prohibitive. We demonstrated that the deeper searches of our method outperformed the other method on sparse and dense belief sets. We showed that rewards without a forward search were a lessened with the sparseness of belief set density. However, when our search algorithm was able to create nearly identical rewards when sparse sets were used and compared to rewards on a denser belief set. The standard search with observations was unable to search deep enough to make a similar claim.

Thirdly, we developed a threefold adaptive sampler based on Kullback-Liebler distance and effective sample sizes for training the agent. With our sampler we only require the setting of two accuracy parameters, ϵ and δ.

With these two values the sampler dynamically tunes the number of particles representing beliefs, the number of posterior beliefs sampled, and the size of weighted particle sets used to project back onto belief space for every $\theta \in G$ and $u \in U$ pair. We demonstrated that our sampler can train the agent in reasonable training times when compared to a fixed sampler. That is, our sampler does not take a prohibitive number of samples driving the computation time past what we would expect for the complexity of the model. We showed that without allowing N_1 and N_2 take different values an adaptive sampler takes 4 to 16 times as long to train without any boost in performance when tested on our four models. We also verified our claim that $N_2 < N_1$ by taking the average values of N_1 and N_2 selected by the adaptive sampler.

Bibliography

[1] Parametric pomdps for planning in continuous state spaces. *Robotics and Autonomous Systems*, 54(11):887 – 897, 2006. ISSN 0921-8890.

[2] R.E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[3] M. L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[4] D. Bello and G. Riano. Linear programming solvers for markov decision processes. In *Systems and Information Engineering Design Symposium, 2006 IEEE*, pages 90 –95, april 2006. doi: 10.1109/SIEDS.2006.278719.

[5] Sean Meyn. The policy iteration algorithm for average reward markov decision processes with general state space. *IEEE Trans. Automat. Control*, 42, 1997.

[6] Yue Wuyi Hu, Qiying. *Markov Decision Processes with Their Applications Markov Decision Processes with Their Applications.* Springer, 2008.

[7] Adam Shwartz Eugene A. Feinberg. *Handbook of Markov Decision Processes.* Kluwer Academic Publishers, 2002.

[8] Alex M. Brooks. *Parametric POMDPs for Planningin Continuous State Spaces.* PhD thesis, University of Sydney, Sydney, Australia, 2007.

[9] Nicolas Meuleau, Kee eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving pomdps by searching the space of finite policies. In *IN PROCEEDINGS OF THE FIFTEENTH CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE*, pages 417–426. Morgan Kaufmann, 1999.

[10] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up, 1995.

[11] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Finding approximate pomdp solutions through belief compression. Technical report, 2003.

[12] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains, 1994.

[13] Arnaud Doucet and Adam M. Johansen. A tutorial on particle filtering and smoothing: fifteen years later, 2011.

[14] Ioannis Rekleitis. A particle filter tutorial for mobile robot localization. Technical report, Centre for Intelligent Machines, Montreal, Quebec, Canada, McGill University, 2004.

[15] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

[16] M. Sanjeev Arulampalam, Simon Maskell, and Neil Gordon. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2002.

[17] Arthur Krener. The convergence of the extended kalman filter. In Anders Rantzer and Christopher Byrnes, editors, *Directions in Mathematical Systems Theory and Optimization*, volume 286 of *Lecture Notes in Control*

and Information Sciences, pages 173–182. Springer Berlin / Heidelberg, 2003.

[18] Bruce McElhoe. An assessment of the navigation and course corrections for a manned flyby of mars or venus. 1966.

[19] Branko Ristic. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2004.

[20] Sebastian Thrun. Particle filters in robotics. In *in Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI*, 2002.

[21] Rudolph van der Merwe, Arnaud Doucet, Nando De Freitas, and Eric Wan. The unscented particle filter, 2000.

[22] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *STATISTICS AND COMPUTING*, 10(3):197–208, 2000.

[23] Umut Orguner and Fredrik Gustafsson. Risk sensitive particle filters for mitigating sample impoverishment. In *Statistical Signal Processing, 2007. SSP '07. IEEE/SP 14th Workshop on*, pages 259 –263, aug. 2007. doi: 10.1109/SSP.2007.4301259.

[24] Seongkeun Park. A new evolutionary particle filter for the prevention of sample impoverishment. *Trans. Evol. Comp*, 13(4):801–809, August 2009. ISSN 1089-778X.

[25] Manya Afonso. Particle filter and extended kalman filter for nonlinear estimation: A comparative study. *IEEE Transactions on Signal Processing*, page 10, 2008.

[26] Jun Liu. Sequential monte carlo methods for dynamical systems. *J. Amer. Statist. Assoc.*, 93:1032–1044, 1998.

[27] Michael L. Littman. On the complexity of solving markov decision problems. pages 394–402, 1995.

[28] Christopher J. C. H. Watkins and Peter Dayan. Q-learning, 1989.

[29] Eyal Even-dar and Yishay Mansour. Learning rates for q-learning. In *Journal of Machine Learning Research*, pages 1–25, 2001.

[30] Csaba Szepesvsari. Reinforcement learning algorithms for mdps. 2009.

[31] Peng Dai and Daniel S.Weld. Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, page 29, 2011.

[32] Peng Dai and Daniel S.Weld. Focused topological value iteration. *Journal of Artificial Intelligence Research*, page 8, 2011.

[33] Michael Littman and Michael L. Littman. The witness algorithm: Solving partially observable markov decision processes. Technical report, 1994.

[34] Zhengzhu Feng. Region-based dynamic programming for partially observable markov decision processes. 2005.

[35] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 54–61. Morgan Kaufmann Publishers, 1997.

[36] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24: 195–220, 2005.

[37] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13: 33–94, 2000.

[38] Nicholas Roy and Geoffrey Gordon. Exponential family pca for belief compression in pomdps. In *In NIPS*, pages 1043–1049. MIT Press, 2003.

[39] S. Thrun. Monte carlo POMDPs. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1064–1070. MIT Press, 2000.

[40] Josep M. Porta, Matthijs T. J. Spaan, and Nikos Vlassis. Robot planning in partially observable continuous domains. In *In Robotics: Science and Systems I*, pages 217–224, 2005.

[41] Josep M. Porta, Nikos Vlassis, Matthijs T. J. Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. *JOURNAL OF MACHINE LEARNING RESEARCH*, 7:2329–2367, 2006.

[42] Alex Brooks and Stefan Williams. A monte carlo update for parametric pomdps. In Makoto Kaneko and Yoshihiko Nakamura, editors, *Robotics Research*, Springer Tracts in Advanced Robotics. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-14742-5.

[43] E. Zhou, M.C. Fu, and S.I. Marcus. A density projection approach to dimension reduction for continuous-state pomdps. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 5576 –5581, dec. 2008. doi: 10.1109/CDC.2008.4738730.

[44] Enlu Zhou, M.C. Fu, and S.I. Marcus. Solving continuous-state pomdps via density projection. *Automatic Control, IEEE Transactions on*, 55(5): 1101 –1116, may 2010. ISSN 0018-9286. doi: 10.1109/TAC.2010.2042005.

[45] L Fahrmeir and G Tutz. *Multivariate Statistical Modelling Based on Generalized Linear Models*, volume 91. Springer, 2001. URL `http://www.jstor.org/stable/2291687?origin=crossref`.

[46] Damiano Brigo. *Filtering by Projection on the Manifold of Exponential Densities*. PhD thesis, Free University of Amsterdam, 2006.

[47] Babak Azimi-Sadjadi. *Approximate Nonlinear Filtering with Applications to Navigation*. PhD thesis, University of Maryland, Maryland, USA, 2001.

[48] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):pp. 79–86, 1951. ISSN 00034851. URL `http://www.jstor.org/stable/2236703`.

[49] Nicholas Roy and Sebastian Thrun. Coastal navigation with mobile robots. In *IN ADVANCES IN NEURAL PROCESSING SYSTEMS 12*, pages 1043–1049, 1999.

[50] A.R. Cassandra, L.P. Kaelbling, and J.A. Kurien. Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In *Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 2, pages 963 –972 vol.2, nov 1996. doi: 10.1109/IROS.1996.571080.

[51] Dorian Spero and Ray Jarvis. On localising an unknown mobile robot. In *In 2 nd Int. Conf. Autonomous Robots and Agents, Palmerston North, New Zealand*, pages 13–15, 2004.

[52] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.

[53] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 963–972, 1996.

[54] Sven Koenig and Reid G. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998.

[55] Amalia Foka. Real-time hierarchical pomdps for autonomous robot navigation. In *in: IJCAI Workshop Reasoning with Uncertainty in Robotics*, 2005.

[56] *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning).* The MIT Press, March 1998. ISBN 0262193981. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262193981`.

[57] P. Naor M. Resh. An inventory problem with discrete time review and replenishment by batches of fixed size. *Management Science*, 10(109-118), 1963.

[58] Enlu Zhou. *Particle Filtering For Stochastic Control and Global Optimization.* PhD thesis, University of Maryland, Maryland, USA, 2009.

[59] Illah R. Nourbakhsh, Rob Powers, and Stan Birchfield. Dervish - an

office-navigating robot. *AI Magazine*, 16(2):53–60, 1995. URL http://dblp.uni-trier.de/db/journals/aim/aim16.html#NourbakhshPB95.

[60] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010. URL http://www.cs.berkeley.edu/~russell/aima/index.html.

[61] Introduction to ai techniques game search, minimax, and alpha beta pruning, 2009.

[62] Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:1203–1212, 1989.

[63] Jay Burmeister and Janet Wiles. The challenge of go as a domain for ai research: A comparison between go and chess. In *In Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems. IEEE Western Australia Section*, pages 181–186, 1995.

[64] Sven Koenig. Minimax real-time heuristic search. *Artif. Intell.*, 129(1-2):165–197, June 2001. ISSN 0004-3702. doi: 10.1016/S0004-3702(01)00103-5. URL http://dx.doi.org/10.1016/S0004-3702(01)00103-5.

[65] Sven Koenig and Reid G. Simmons. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *In Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 144–153, 1998.

[66] Arnaud Doucet, editor. *Sequential Monte Carlo methods in practice*. 2001. URL http://www.worldcatlibraries.org/wcpa/top3mset/839aaf32b6957a10a19afeb4da09e526.html.

[67] X. Xiong and I. M. Navon. 2005: Ensemble particle filter with posterior gaussian resampling. tellus, submitted, 2005.

[68] Geir Evensen. The ensemble kalman filter: theoretical formulation and practical implementation, 2003.

[69] S. Park, J. Hwang, K. Rou, and E. Kim. A new particle filter inspired by biological evolution: Genetic filter, 2006.

[70] Dieter Fox. Kld-sampling: Adaptive particle filters and mobile robot localization. In *In Advances in Neural Information Processing Systems (NIPS*, 2001.

[71] Daphne Koller and Raya Fratkina. Using learning for approximation in stochastic processes. In *In Proceedings of the International Conference on Machine Learning (ICML*, pages 287–295, 1998.

[72] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *IN PROC. OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI*, pages 343–349, 1999.

[73] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22:2003, 2003.

[74] John A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, April 2001. ISBN 0534399428. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0534399428`.

[75] Yu Huang and Joan Llach. Variable number of informative particles for object tracking, 2005.