REDUCING THE COMPLEXITY OF REINFORCEMENT LEARNING IN POMDPS BY DECOMPOSITION INTO DECISION AND PERCEPTUAL PROCESSES

by

RASOOL FAKOOR

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

July 2012

To Azade for her limitless support and love.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Manfred Huber, for his continued guidance, inspiration, support, and boundless energy. I appreciate his vast knowledge and skill in many areas, and his assistance in writing thesis and two papers. In addition, his great efforts to explain things clearly and simply to me. I could not have imagined having a better advisor and mentor for my graduate study. It was a privilege to work with him. I would also like to thank Dr. Sajal K. Das for being my co-adviser, the assistance he provided at all levels of the research project, and his guidelines and help. Finally, I would like to thank Dr. Gergely V. Zaruba for taking time out from his busy schedule to serve as my committee and for his thoughtful comments and help.

Lastly, I wish to thank my wife, Azade, without her love and encouragement, I would not have finished this thesis.

July 12, 2012

ABSTRACT

REDUCING THE COMPLEXITY OF REINFORCEMENT LEARNING IN POMDPS BY DECOMPOSITION INTO DECISION AND PERCEPTUAL PROCESSES

Rasool Fakoor, M.S.

The University of Texas at Arlington, 2012

Supervising Professors: Manfred Huber, Sajal K. Das

Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) are very powerful and general frameworks to model decision and decision learning tasks in a wide range of problem domains. As a result, they are widely used in complex and real-world situations such as robot control tasks. However, the modeling power and generality of the frameworks comes at a cost in that the complexity of the underlying models and corresponding algorithms grows dramatically as the complexity of the task domain increases. To address this , this work presents an integrated and adaptive approach that attempts to reduce the complexity of the decision learning problem in partially Observable Markov Decision Processes by separating the overall model into decision and perceptual processes. The goal here is to focus the decision learning on the aspects of the space that are important for decision making while the observations and attributes that are important for estimating the state of the decision process are handled separately by the perceptual process. In

this way, the separation into different processes can significantly reduce the complexity of decision learning. In the proposed framework and algorithm, a Monte Carlo based sampling method is used for both the perceptual and decision processes in order to be able to deal efficiently with continuous domains. To illustrate the potential of the approach, we show analytically and experimentally how much the complexity of solving a POMDP can be reduced to increase the range of decision learning tasks that can be addressed.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION AND RELATED WORK

1.1 Introduction

In most autonomous systems, such as robotic navigation systems, poker, and planning under uncertainty that need decisions to be made, it is impossible for the robot or agent to have complete information about the surrounding environment [1]. This incompleteness of information is caused by many factors such as the noise in sensors, existence of unknown obstacles, and insufficient memory in the agent to memorize every possible combination of the problem. As a result, the underlying environment state can not be completely recognizable by the agent [2]. In addition, the state spaces of these problems are usually large and continuous which makes learning decision policies a challenging job.

For this class of problems in which system states are not fully observable by the agent, Partially Observable Markov Decision Processes (POMDPs) are frequently used to model and solve the decision problem. This, together with the need to deal with continuous state spaces often makes learning of the solution computationally intractable using standard solution approaches. One of the difficulties here is that since the agent can only partially observe the state of the system, it needs to keep track of the history of past actions and observations. In order to handle this history, the POMDP uses the probability distribution of the system states to represent the state of its knowledge. For most problems, this implies that POMDPs can easily become impractical to solve using traditional methods since, when the number of state space variables increases, the complexity of solving the POMDP increases exponentially. In

addition, keeping track of the history accurately and in an analytic fashion requires an unbounded amount of memory. More importantly, the representation of the policy and the value function also increase exponentially and thus learning of the policy becomes a real problem and quickly computationally intractable. Hence, even though the POMDP is a useful tool to model problems, it can not be easily solved.

To address this, a number of methods have been proposed to solve POMDPs in such complex applications either by using dimensionality reduction techniques or by using approximation methods. In this work, we propose a novel approach to address the complexity of solving POMDPs for autonomous systems which combines sampling and approximation techniques with a new concept of decomposition of the problem into separate processes. In particular, we propose a sampling based algorithm in which the state space and the overall process description of the problem is divided into a perceptual and a decision component. The underlying rationale is here that in real world systems perceptual processing generally serves the purpose of extracting relevant information from the environment that can be used by the decision process to determine optimal actions. As a consequence, the decision process can frequently be modeled on a lower-dimensional state representation if the appropriate features can be correctly extracted by a perceptual process.

The perpetual process is here responsible to derive information that is important for the decision process. As such, it serves mainly an information extraction and partial belief state tracking purpose. On the other side, the decision process is responsible to use the information from the perceptual process and its own states to model the actual decision process on a reduced state space and to learn the optimal policy. In this way, the complexity is decreased in our framework through a reduction in the number of state variables that are considered in the decision process to a close to minimal set while ensuring the correct tracking of the corresponding belief state.

When using sampling based methods, fewer state variables in the decision part, which is itself modeled as a POMDP here, means a need for fewer samples to represent the decision problem and, in turn, lower complexity in the representation and learning of the corresponding policy. In this way, the improvement happens by providing relevant and processed perceptual information to the decision process model. Moreover, the perceptual process alleviates needs for the decision process to compute the perceptual parts of the POMDP's belief state. The decision part is represented as a POMDP on a state representation that only includes the decision-relevant state attributes but which is informed by the state of the perceptual processes.

For the perceptual and decision process, we use a particle filter to track the belief states. The non-parametric character and simplicity of the particle filter are the main reasons that we used it here. Through the split of the state space into two components in our model, one part of the belief update is handled by the perceptual process and the other part by the decision process. The interaction of the two belief components needed for the decision process is achieved by re-introducing the updated belief of the perceptual process as part of the modified belief update in the decision process. The empirical and analytical analysis shows the efficiency and applicability of our method and offers some insight into its potential in scaling learning to larger tasks in real world domains.

The thesis is organized as follows: The rest of this chapter reviews the related work dealing with complexity of solving POMDPs. Chapter 2 introduces the background and the underlying notation and formalisms of POMDPs, belief tracking, Reinforcement Learning, and the particle filter. Chapter 3 describes our approach and presents the formalization for our method. Chapter 4 describes our simulation setup and shows the experimental results. Finally, conclusion and future work are presented in the Chapter 5.

1.2 Related Work

The problem of scaling learning techniques for POMDP decision making to larger, more real-world problems has received significant attention in recent years and led to a number of different approaches to address the problems of representation and tracking of the belief state, and the computation of optimal solution policies and/or value functions. Even though various methods such as [1], [3], [4], and [5] have been proposed to solve POMDPs, dealing with problems with a large number of states is still challenging and complex, and becomes quickly computationally intractable. Since there may be no exact solution possible to deal with POMDPS, most of the recent research tried to find approximate solutions for these problems.

Littman in [1] discussed methods to find near-optimal polices for some problem domains. Using Reinforcement Learning, their methods can be used to solve POMDPs in problems with up to 100 states. Thrun in [3] used Monte Carlo methods and Reinforcement Learning (RL) to solve POMDPs in continuous state and action spaces. While this addresses the issue of infinite state and action spaces, it does not directly address the underlying learning complexity. Because when the underlying state space dimensions increase, the number of samples required by the Monte Carlo POMDP method to represent the state space still grows rapidly.

In order to solve continuous POMDPs, some of the proposed algorithms convert the continuous state space into an approximate discrete state space [6]. However, this is not always applicable since discretization is not possible in all problems and, more importantly, reduces the correctness of the solution. In [7], Roy proposed a method to address the solution complexity by reducing the dimensionality of the belief space. In this method, they used Exponential Family of Principal Components Analysis to reduce the dimensionality of the problem based on the introduction of structured

belief. However, they also mentioned in their paper that there exist problems in which such a dimensionality reduction technique will not work.

In [8], Brooks et al. proposed a parametric POMDP. In this method, the dimensionality of the belief state is fixed and the belief state is represented as a single or a mixture of Gaussians. However, there are cases, such as robot navigation, in which applying this method is not feasible due to the limitations in their belief state representation. To obtain a more general representation of the belief state, Brooks and Williams in [5] proposed a method to handle the belief space using a Monte Carlo method. More specifically, they transferred the parameterized belief using Monte Carlo sampling. In [9], Zhou et al reduce the dimensionality of the belief space using density projection. In their method, they projected the high dimensional belief space onto a parameterized low dimensional space. However, all these belief state compression methods impose limitations on the belief state that can be represented and do not inherently consider whether or not state attributes are decision-relevant, thus potentially requiring a significant number of non-relevant parameters or variables to be considered in learning the policies. In [10], Brunskill et al proposed a point-based POMDP that uses hybrid dynamics to solve the continuous POMDP. However, they mentioned that this method is not applicable for problems in which modeling the dynamics of the system is expensive using a weighted sum of Gaussians.

In [11], Kurniawati et al. developed a new point-based POMDP algorithm. In their method, optimal reachability of belief spaces has been exploited to enhance the computational efficiency. In other words, the algorithm maintains a belief tree and only explores the subtrees that were previously visited by the optimal policy. However, efficient pruning can not easily be achieved and comes with the expense of memory and computation power [12]. Poupart et al. in [13] proposed a method to reduce the gap between upper and lower bound on the optimal value function.

In their method, prioritized breadth first search is used to find the reachable belief state and linear programming is used for upper bound interpolation. However, their current method can become intractable with factored POMDPs due to the fact that the number of variables and constraints in linear programming can be exponential.

Poupart and Boutilier in [14] mathematically derived a set of conditions by exploiting the structure of the POMDP in order to compress the belief state without effecting the quality of decision making. In addition, they proposed an optimization algorithm to find the linear lossy compression. However, compressing the belief state can lead to suboptimal policies due to the fact that some relevant information may be discarded during the compression. Finding informative belief points by using point-based value iteration in solving the POMDP is the main contribution of [4]. In their method, only a small set of belief points were selected and tracked. More specifically, they used a stochastic trajectory to find the set of belief points. In addition, their method only selects reachable belief points and belief points which rapidly enhance their error bound, not random belief points .

Gradient based policy search methods have been studied as well to deal with continuous state spaces. Ng in [15] proposed a method to reduce the complexity of policy search in a POMDP by transferring it to a POMDP problem in which all transactions are deterministic. Since this method is gradient based, the local optima can be problematic. In addition, policy search methods can be data inefficient. In particular, reusing the sample trajectories from old policies is sometimes problematic. In [16], Coquelin et al. used policy gradient methods that replace the POMDP problem with an optimization problem. In their method, they consider the group of policies which are based on the particle filter estimated belief. Moreover, they proposed a Finite Difference technique by which the inconsistency and variance explosion of naive Bayes has been solved. However, this method only addressed the class of policies which are

based on a belief state constructed from a particle filter. In addition. due to the use of the gradient based method, local optima can be problematic as well.

In contrast to all of these methods, the approach presented in this thesis reduces the state space of the decision POMDP by separating it into separate but coupled decision and perceptual processes. In other words, the proposed algorithm can reduce the complexity of solving POMDPs by separating them into decision and perceptual processes. This, in turn allows the presented method to scale more efficiently in real-world problems.

CHAPTER 2

TECHNICAL BACKGROUND

Machine Learning algorithms are usually classified in four classes:

- Supervised Learning
- Unsupervised Learning
- Semi-Supervised Learning
- Reinforcement Learning

Assume a robot with a sequence of inputs $x_1, x_2, ..x_t$ where x_t is the input at time t [17]. The input can here for example be an image, audio, or a set of previously collected data. The sequence of outputs $y_1, y_2, ..., y_t$ represents the robot's actions, and desired outputs are provided to the learning system in Supervised Learning algorithms as well. Each of these desired outputs defines what the correct answer would be for a given input at time t. The Supervised Learning algorithm learns a model that maps the input data to the desired output given labeled training data. The goal of the algorithm is to allow the learned model to be used to predict the correct output when a new set of data is provided. On the other hand, in Unsupervised Learning the machine only receives the inputs $x_1, x_2, ..x_t$ [17]. The role of the algorithm here is to detect patterns and structure in the data which is otherwise recognized as pure unstructured noise [17]. Dimensionality reduction and clustering are examples for Unsupervised Learning algorithms. However, it is possible to have input data in which only some points are provided with the corresponding output at the time t. In other words, some of the data are labeled but others are unlabeled. Semi-Supervised

learning algorithms are designed to deal with this class of problems. Recommender systems are often a classic example for Semi-Supervised Learning.

Finally, Reinforcement Learning (RL) is the class of learning methods in which the agent interacts with a dynamic environment through actions and requires only qualitative feedback [17] [18]. Each of the agent's actions effects the state of the environment which the agent observes. After executing an action, the agent will also receive some scalar value which is called reward. Using this feedback, the agent has to determine which action to take in order to maximize the rewards obtained through the interaction with the environment [18]. By maximizing rewards, the agent can determine the optimal policy. It is worth noting that in contrast to other classes of machine learning methods, like Supervised learning in which the agent has been provided with a training set, an RL agent should act in the environment in order to explore its actions' effects and maximize the reward. The remainder of this chapter focuses on Reinforcement Learning and basic formalisms for POMDPs, the particle filter, and the learning methods used in experiments.

2.1 Reinforcement Learning

In Reinforcement Learning, the agent is only provided with a reward function. The reward function tells the agent when it is acting well or when it is acting poorly [19]. The agent's job is to execute actions over time to maximize the rewards [19]. In fact, interaction between a learning agent and its environment in terms of states, actions, and rewards can define a framework to be used in Reinforcement Learning [18]. Figure 2.1 shows the Reinforcement Learning structure. In general, a Reinforcement Learning system consists of the following elements [18]:

- A **policy** π defines the learning agent's way of behaving at a given time.

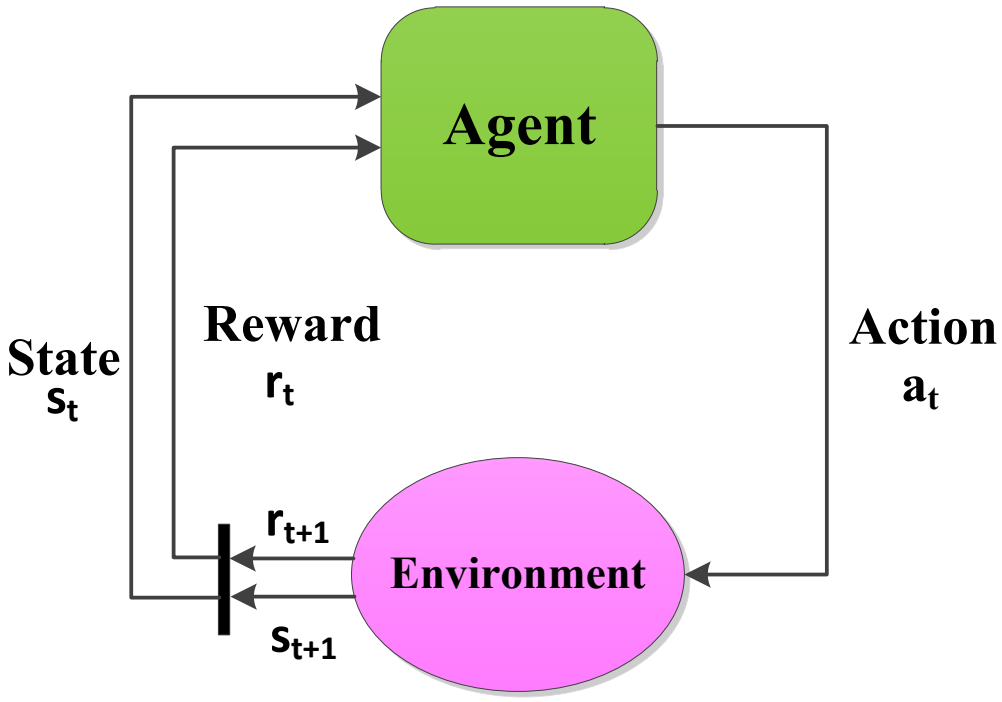

Figure 2.1. Reinforcement Learning Structure.

- A **reward function** $r_t(s)$ defines the relationship between the state of the system, s, and the resulting payoff.
- A **value function** V is a mapping from state to its utility, usually measured in terms of the expected total amount of discounted reward that an agent can collect.
- A Q **function** is like the V function except that it defines the best value for each action in each state.
- A **Model of the Environment** for capturing aspects of the behavior of the environment.

The most common underlying mathematical framework for the case when the state of environment is fully observable is the Markov Decision Process (MDP) [18] [20] [21]. Fully observable here means that the agent can get all required information from the environment to determine its state and make decisions. In other words, the agent assumes that the sensors can measure the complete state of the environment and are not noisy [20]. However, actions in Markov Decision Processes can still have stochastic effects. Consequently, a single sequence of actions is insufficient for planning [20]. In other words, every combination of states and actions should be considered in order to have enough information for planning. Hence, it is necessary to generate the actions for all possible states that the agent can be in [20]. Value iteration and Policy iteration are examples of Reinforcement Learning techniques in Markov Decision Processes.

In the real world, however, senors are usually noisy or the environment is only partially observable. As a result, the agent has insufficient information to make decisions based only on its current sensor readings and it needs methods to deal with the noisy projection of the states [20]. One mathematical framework to model this type of environment is the Partially Observable MDP (POMDP). Dealing with POMDPs is more complicated than with MDPs due to the fact that POMDPs need to represent the state in a way that can reflect the partial observability. In addition, POMDPs are a more general model than MDPs because they can model a larger class of problems. For this reason, in the next section , we briefly introduce the Markov Decision Processes and then, we present in more detail how POMDPs can be represented and some methods to solve them.

2.2 MDP

A MDP is defined by a tuple of four entities: $\langle S, A, R, T \rangle$. Here S is the finite set of world states denoted by s_t. A is the finite set of actions. T is the state transition

probability function, and R is the reward function. Finding the optimal policy is the main goal of MDP in reinforcement learning . Value iteration and policy iteration are example methods which can be used to find the optimal policy for given MDP [18].

2.3 POMDP

A POMDP is an extension of the MDP and is defined by a tuple of six entities: $\langle S, A, Z, R, T, O \rangle$. Here S is the set of world states denoted by s_t. Z is the set of observations. A is the set of actions, and T is the state transition probability function. Finally, O is the observation probability function, and R is the reward function. T, R, and O can be expressed as follows:

$$T(s, s', a) := P(s_t = s' | s_{t-1} = s, a_{t-1} = a)$$

$$O(s, z) := P(z_t = z | s_t = s)$$

$$R(s, a) = S \times A \longmapsto \mathbb{R}$$

Figure 2.2 shows the POMDP model. Since a POMDP has incomplete observations of the environment, it needs to find an optimal policy based on the complete sequence of observations and actions from the beginning or a sufficient statistic of observations and actions from the beginning [22]. This history is defined as follows:

$$h = \{z_t, a_t, z_{t-1}, a_{t-1}, ..., z_0, a_0\}$$

In a POMDP, the policy is the mapping from the histories to actions [3]:

$$\pi : h \longrightarrow a$$

Since the history can be infinite, it is common practice to use a sufficient statistic to represent the history [3] [22] and learn a mapping from that to actions instead [3].

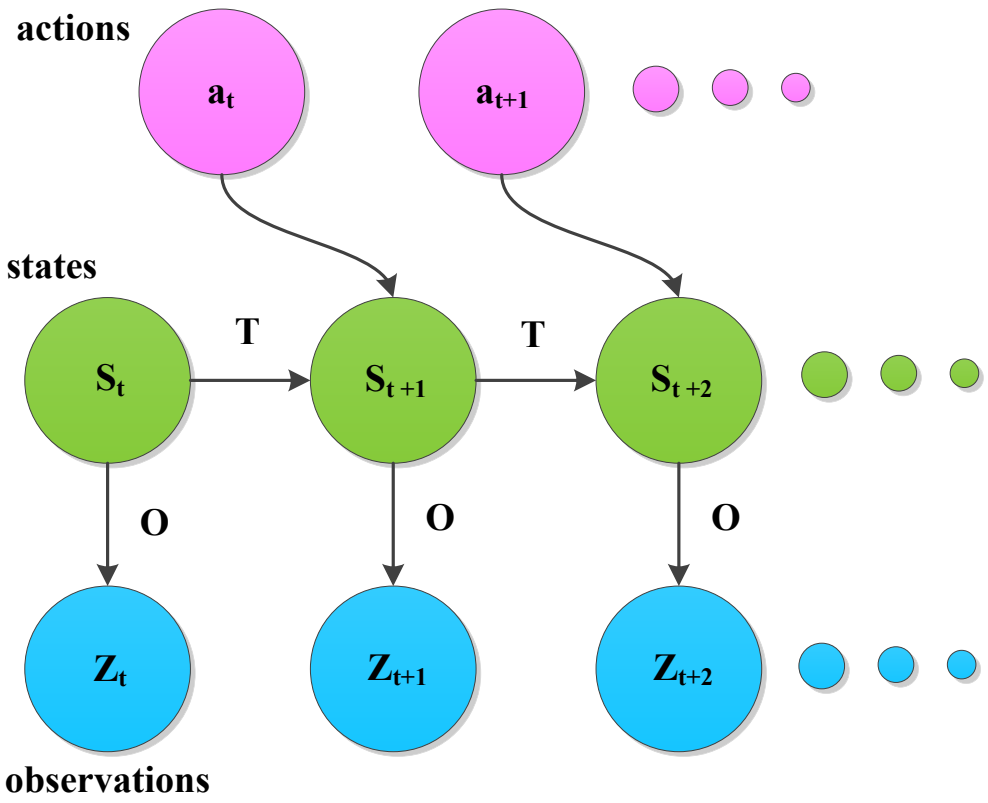

Figure 2.2. A POMDP Model.

The sufficient statistic in a POMDP is the belief state b. As a result, the policy is the mapping from the belief state to actions:

$$\pi : b \longrightarrow a$$

A belief state here represents a probability distribution over states, $bel(s)$, that reflects the complete state of knowledge about the system at time t, $bel(s) = p(s|z_t, a_t, ..., z_1, a_1, z_0, a_0)$. The belief can be calculated as follows:

$$bel(s') = \underbrace{\eta p(z|s')}_{\text{measurement update}} \underbrace{\int_s p(s'|s,a) bel(s) ds}_{\text{prediction}} \tag{2.1}$$

where s' and s represent the current and previous state, respectively, and η is a normalization factor. Tracking the belief state analytically in a continuous space is generally not tractable and has therefore to be either modeled using an approximation that imposes limitations on the distribution (e.g. exponential distributions or Gaussian) or that approximates it using sampling.

The goal in a POMDP is to learn a control policy, $\pi(b)$, that selects actions, a, in belief state b such as to maximize the expected discounted future reward:

$$V(b) = E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | b_t = b, a_t = a\right]$$

where E is the expectation and γ is discount factor. The goal is to find the optimal policy such that:

$$\pi^* = argmax_\pi V^\pi$$

In addition to the value function, the utility in a POMDP can be defined by the Q-value over the belief state:

$$Q^\pi(b,a) = E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | b_t = b, a_t = a\right]$$

where Q is the value function over belief state/action pairs. An optimal policy is here such that [18]:

$$Q^*(b,a) = argmax_\pi Q^\pi(b,a)$$

$$\pi^*(b) = argmax_a Q^*(b,a)$$

2.3.1 Methods to Solve POMDPs

Since the states are not observable in the POMDP [1] [20] [2], it is necessary to use the belief state to make decisions. In fact, to solve the POMDP, a value function [20] over belief states can be defined and used to derive an optimal policy:

$$V_t(b) = max_a \left[r(b,a) + \gamma \int V_{t-1}(b') p(b'|a,b) db\right] \tag{2.2}$$

The belief state is defined by a probability distribution over states [20] which is continuous. The problem with continuous belief is that its precise representation generally requires infinite memory and this makes the POMDP intractable [2]. Consequently, the value function in (2.2) and/or the policy that optimizes this value function can in general not be represented and computed analytically, but can only be approximated using iterative algorithms. A number of approximation methods have been proposed to deal with this problem such as [6], [7], and [23]. To deal with (2.2), we can classify the methods in two classes:

1. Finite State Space Methods
2. Continuous State Space Methods

For the first class, either the state space is inherently discrete (and finite) or the continuous state space is discretized. For the discretized class, method such as Replicated Q-Learning [1] and Linear Q-Learning [1] can be used. On the other hand, approximation methods can be used to deal with the second class such as Monte Carlo POMDP [3]. In the next two sections, Linear Q-Learning and Monte Carlo POMDP are described as they are used in this thesis.

It is worth noting that for finite horizon worlds with finite action and state spaces it is possible to represent the value functions as piecewise linear functions [1] [2]. However, piecewise linear functions can still be large and can not be easily solved.

2.3.1.1 Linear Q-Learning

One of the approximation methods to deal with POMDPs is Linear Q-Learning [1]. Linear Q-learning solves the decision POMDP by approximating the Q function over the belief space. In particular, Linear Q-learning tracks the belief over the finite

state space and uses a linear function approximator to estimate the Q-value of a belief state, b, using state/action values:

$$Q(b,a) = \sum_s bel(s)\ q(s,a)$$

Using this approximation, it learns the parameter values using a variation on the Q-learning rule:

$$\Delta q(s,a) = \alpha bel(s)(r + \gamma max_{a'} Q(b',a') - Q(b,a))$$

where α is the learning rate, γ is the discount factor,and a and a' are actions at time $t-1$ and time t, respectively. In addition, b and b' are the belief states at time $t-1$ and time t, respectively, and $b(s') = bel_t(s')$ is:

$$bel_t(s') = \frac{O(s',z_t)\sum_{s\in S} T(s',s,a)bel_{t-1}(s)}{\sum_{s'\in S} O(s',z_t)\sum_{s\in S} T(s',s,a)bel_{t-1}(s)} \tag{2.3}$$

Linear Q-learning generalizes the Q-learning rule to apply it to the vector-valued state distribution [1]. Moreover, it uses $q(s,a)$ as a single vector to approximate the Q function [1]. In other words, it calculates the Q for each action a as $Q(b,a) = q^T b = \sum_s b(s)\ q(s,a)$ [1]. Due to inaccuracies in the approximation, Linear Q-learning can sometimes result in poor policies [1]. However, for small POMDPs, this is usually not the case [1] and Linear Q-learning can be an efficient learning technique for such systems.

2.3.1.2 Monte Carlo POMDP

The MC-POMDP [3] is a Reinforcement Learning approach for POMDPs with continuous state spaces. It represents the belief state of the problem as a sample set. Similarly, it uses a sample set of belief states and the k-Nearest Neighbor function approximator to represent the value function. To integrate state tracking and

value function learning in this approach, the belief state is tracked by samples. When the belief state is updated, the algorithm first obtains its value-function value by calculating the nearest neighbors among the stored sample sets using KL-divergence as the distance function [3], and uses the corresponding stored parameter values to calculate the value of the current belief state by linearly averaging the neighbors' parameters. If there are sufficient neighbors for the current belief state, the value function update is performed by updating the stored function values. Otherwise the current state sample set is stored together with the estimate of the corresponding value. In order to approximate and track the belief states, the particle filter is used. The following pseudo-code shows the Monte Carlo POMDP algorithm in more detail [20]:

1: **repeat**
2: Sample $x \sim b(x)$
3: initialize X with M samples of b(x)
4: **repeat**
5: **for** all available actions u **do**
6: Q(u)=0
7: **repeat**
8: select random $x \in X$
9: $x' \sim p(x'|u,x)$, $z \sim p(z|x')$
10: $X' = Particle\_filter(X,u,z)$
11: $Q(u) = Q(u) + \frac{1}{n}(r(x,u) + \gamma V(X'))$
12: **until** N
13: **end for**
14: $V(X) = max_u Q(u)$

15: $u^* = argmax_u Q(u)$

16: $x' \sim p(x'|u, x)$

17: $z \sim p(z|x')$

18: $X' = Particle\_filter(X, u, z)$

19: $X = X'$

20: **until** episode over

21: **until** convergence

22: return V

2.4 Belief State Tracking

One of the important steps when applying POMDPs is belief state tracking. The most general algorithm for tracking belief state is given by the Bayes Filter algorithm [20]. Variations of Bayes Filter algorithms are classified into two categories:

- Parametric
- Non-Parametric

Gaussian filter or Kalman filter are examples for parametric filters. On the other hand, the particle filter is one of the most efficient and popular methods for non-parametric Bayes filtering.

In the following sections, first, the equations for belief tracking are introduced and then the particle filter is described in detail.

2.4.1 Bayes Filter for Belief State Tracking

The mathematical formulation of the Belief Update can be expressed as follows [3]:

$$bel(s') = p(s'|z_t, a_{t-1}..., z_1, z_0) \tag{2.4}$$

$$= \eta p(z_t|s', ..., z_0)p(s'|a_{t-1}, ..z_0) \tag{2.5}$$

$$= \eta p(z_t|s') \int_s p(s'|a_{t-1}, ..z_0, s)p(s|a_{t-1}, ..z_0, s)ds \tag{2.6}$$

$$bel(s') = \underbrace{\eta P(z|s')}_{\textbf{measurement update}} \underbrace{\int_s P(s'|s, a)bel(s)ds}_{\textbf{prediction}} \tag{2.7}$$

where s defines the state and η is the normalization factor. (2.5) is the result of applying Bayes rule to (2.4). In (2.6), to calculate $p(s'|a_{t-1}, ..z_0)$, the total probability rule is applied. It is necessary to say that since in a POMDP the observation is conditionally independent of past observation given the current state, $p(z_t|s', ..., z_0)$ can be rewritten as $p(z_t|s')$. In addition, $p(s'|a_{t-1}, ..z_0, s)$ is conditionally independent from observations given the current state and action. Hence, it can be written as $p(s'|s, a_{t-1})$.

2.4.2 Particle Filter

The particle filter is a non-parametric method that can be used to approximate the posterior probability or the belief state using Monte-Carlo sampling and a representation in the form of weighted particles, $\{(s^{(i)}, w^{(i)})\}$ [20]. The whole procedure of updating a particle filter consists of three steps:

- Prediction
- Update or Measurement update
- Resampling

The Prediction step uses the system model, T, to generate successor samples by randomly generating a successor state sample for each particle state according to the transition probability. The Measurement update uses the observation model, O, to assign weights to each particle according to the likelihood of the observation given the corresponding sample state. The resampling process is aimed at increasing particle

density in areas where particles have high probability, and lowering it in areas of the distribution with lower probability. To achieve this, resampling generates an equivalent particle set by sampling from the original set according to the probability distribution described by their weights. In this from, resampling can be used as one of the methods to solve the degeneracy problem [24] which is that after some iterations some particles will have very small weights. Since without resampling the variance of samples can only increase over time, the degeneracy is an unavoidable problem [24] which causes the particle filter module to waste a lot of effort on updating the almost zero weighted particles. To deal with the degeneracy, resampling is one of the methods which can be used.

Particles in this form can be consider as a tool to update and represent the belief state using samples.

2.4.2.1 Particle Filter Algorithm

In order to get more intuition about how (2.7) can be calculated in practice, the algorithm of the particle filter has been written according to [20] and [24]. This algorithm generates state samples, s_t^i, for time t based on the particles s_{t-1}^i. In this step , the state transition distribution is used to generate the next particle sets. After this step, an importance factor is calculated. The importance factor incorporates the measurement z_t into the distribution represented by the particle set. The last step of the particle filter is shown in line 5 through 8. This step is called the *resampling step*. Resampling transfers the particle set into another particle set of the same size [20]. By sampling them according to w^i, it reduces the number of particles with lower weights and increases the number of particle with higher ones without changing the represented belief state.

Algorithm 1 Particle Filter Algorithm

1: $\hat{S} = S = 0$

2: **for** $i = 1$ to N **do**

3: Draw $s_t^i \sim p(s_t|s_{t-1}^i)$

4: Assign the particle a weight $w_t^i = p(z_t|s_t^i)$

5: **for** $i = 1$ to N **do**

6: draw i with the probability $\alpha\ w^i$

7: add s_t^i to S_t

8: **end for**

9: Add the $\langle s_t^i, w_t^i \rangle$ to the sample set

10: **end for**

CHAPTER 3

PROPOSED FRAMEWORK

In this work, we propose a method to reduce the complexity of solving POMDPs by decomposing them into separate, coupled perceptual and decision processes which leads to a reduction of the state space size of the decision learning problem. In our method, we reduce the state space of the POMDP by handling some aspects of the state space outside of the decision POMDP. To achieve this, the whole problem state space is separated into separate state spaces for the decision and perceptual process. The perceptual process just serves to estimate aspects of the belief state while the decision process estimates the remainder and determines a policy. As a result, the decision process is modeled as a reduced state space POMDP. To allow the application of this method to continuous state spaces, the decision and the perceptual processes are here both handled by a sampling method within which this separation makes it possible to represent the POMPD with a smaller state space which leads to smaller sample sets for the decision POMDP and as a result reduced representational and decision learning complexity. The goal here is to focus decision learning on the aspects of the space that are important for decision making while the observations and attributes that are important for estimating the state of the decision process are handled separately by the perceptual process. In this way, the separation into different processes can significantly reduce the complexity of decision learning. In order to get a better intuition about how this framework can reduce the complexity of solving POMDP problems, we first derive the mathematical equation for the distributed coupled POMDP in a discrete space and after that we extend our ap-

proach to a continuous state space in which a sampling representation has been used to define our algorithm. In particular, in the continuous state space, Monte Carlo based sampling methods and corresponding sample set representations are used for both the perceptual and decision processes in order to be able to deal efficiently with continuous domains.

3.1 Mathematical Derivation of Equations in a Discrete State Space

In the proposed method, the belief update is divided into two separate parts: decision and perceptual processes belief update. The update in the decision process is handled in the form of a POMDP and the update in the perceptual process is handled by a sampling method such as a particle filter. By the separation, the POMDP only deals with the required information to find an optimal policy and the perceptual processes use the raw data from the sensor and send the processed state attribute to the POMDP. The separation promises the potential to lead to reduced complexity in the decision process by omitting non-necessary information. For example, shape and color information can be useful for the perceptual process but is often not essential for the POMDP decision making. In other words, our method tries to put all information inside the POMDP which is required to decide the action and all other information should be handled by the perceptual process. Due to this separation, the states of the perceptual process and the POMDP are different. Since the most expensive part of the overall computation is solving the POMDP, we want to have a separation in which the state of the POMDP decision component is as compact as possible. Unloading the POMDP states to the perceptual process, if done correctly, can reduce the overall computational complexity to a great extend. Moreover, handling the uncertainty separately in the perceptual process and decision process can further reduce the complexity. In particular, when the

uncertainty increases in either the perceptual process or the decision process, the other process does not need to handle that uncertainty as well. However, in the traditional model, when the uncertainty is increased either in the world and action model or in the sensors used for perception, this uncertainty has to be handled by the entire POMDP process, leading to a significant increase in overall complexity.

In our method, the state and observation of the whole system are divided into states and observations of the perceptual process and the POMDP. Z_f and S_f define perceptual process observations and states, respectively, and Z_d and S_d are the decision process observations and states:

$$Z = (Z_f, Z_d), S = (S_f, S_d)$$

Figure 3.1 shows the separation. In the figure, the decision process is built around the states in S_d and the perceptual processes are built for the states in S_f. Even though, we have just one POMDP in our method, we can have more than one perceptual processes. For example, when there is an agent tracking multiple objects in the environment, the tracking of each object can be handled independently by separate, loosely coupled perceptual processes to further reduce the complexity of the perceptual processes and thus of the overall system. The solid lines in Figure 3.1 show the direct relations and information flow between observations, perceptual process and decision process that are necessary in an ideal decomposition into perceptual and decision processes. On the other hand, the dash lines are used to show the additional information that has to be exchanged in situations where no optimal decomposition is - or can be - performed, requiring correction terms in the other process. For example, in some situations S_d can have an effect on the prediction step of the particle filter in the perceptual process. The formulation can explicitly reflect the relation among those terms. Hence, there can be correlations between states and observations

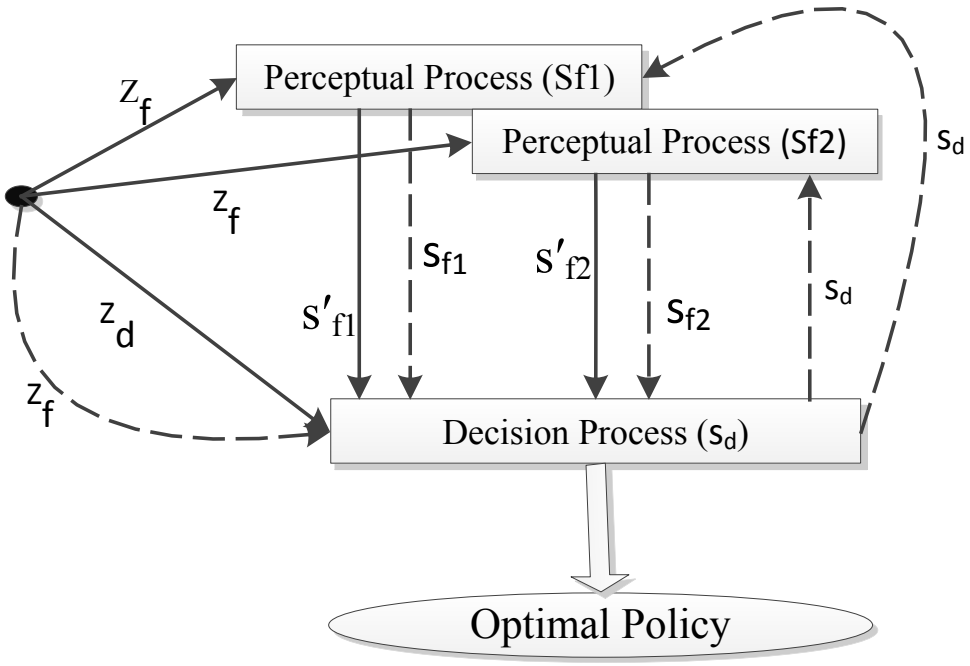

Figure 3.1. Dividing the Belief into Two Parts.

of the decision process and the perceptual process, but an ideal split would make S_d as compact as possible and achieve a situation where the dashed information flow can be avoided as much as possible. Such an ideal split is achievable under certain conditions which will be detailed in the formal derivation in the next section. In addition to providing a means to evaluate the quality of a split, these conditions also present a way to potentially automatically identify useful splits into perceptual and decision processes. In order to formulate our method and to illustrate its potential for complexity reduction as well as the conditions under which this decomposition is most effective and where correction terms are not required, we will in the next section derive the equations for the partially decoupled belief update. The formulation shows

that the belief update can be divided into the perceptual process and decision process belief update with additional correction terms that reflect the residual effects of remaining coupling between the processes. In the following section, we first describe the measurement update step and then, in the later section, the prediction step.

3.1.1 Update Formulation for Measurement Update

In this section, we define new equations for the perceptual process and the decision process measurement update. We solve $P(z|s')$ in a way that reflects the update of the perceptual and the decision processes separately:

$$\begin{aligned} P(z|s') &= P(z_d, z_f|s'_d, s'_f) \\ &= \underbrace{P(z_d|z_f, s'_d, s'_f)}_{*} \underbrace{P(z_f|s'_d, s'_f)}_{**} \end{aligned}$$

We can apply Bayes rule to term $*$:

$$P(z_d|z_f, s'_d, s'_f) = \frac{P(s'_f|z_d, z_f, s'_d)P(z_d|z_f, s'_d)}{P(s'_f|z_f, s'_d)} \tag{3.1}$$

Similarly, Bayes rule can be applied to term $**$ as well:

$$P(z_f|s'_d, s'_f) = \frac{P(s'_d|z_f, s'_f)P(z_f|s'_f)}{P(s'_d|s'_f)} \tag{3.2}$$

From (3.1) and (3.2), we have:

$$P(z_d, z_f|s'_d, s'_f) = P(z_f|s'_f, a)P(z_d|z_f, s'_d)C_o \tag{3.3}$$

$$C_o = \frac{P(s'_f|z_d, z_f, s'_d, a)}{P(s'_f|z_f, s'_d)} \frac{P(s'_d|z_f, s'_f)}{P(s'_d|s'_f)} \tag{3.4}$$

By using Bayes rule, the first part of the correction term C_o in (3.4) will be as follows:

$$\frac{P(s'_f|z_d, z_f, s'_d)}{P(s'_f|z_f, s'_d)} = \frac{P(z_d|s'_d, s'_f, z_f)P(s'_f|z_f, s'_d)}{P(s'_f|z_f, s'_d)P(z_d|z_f, s'_d)}$$

$$= \frac{P(z_d|s'_d, s'_f, z_f)}{P(z_d|z_f, s'_d)} \tag{3.5}$$

and the second part of (3.4) can be written by Bayes rule as:

$$\frac{P(s'_d|z_f, s'_f)}{P(s'_d|s'_f)} = \frac{P(z_f|s'_d, s'_f)P(s'_d|s'_f)}{P(s'_d|s'_f)P(z_f|s'_f)}$$

$$= \frac{P(z_f|s'_d, s'_f)}{P(z_f|s'_f)} \tag{3.6}$$

Finally, by putting (3.3), (3.5), and (3.6) together, the update equation for the measurement update is as follows:

$$P(z_d, z_f|s'_d, s'_f) = \underbrace{P(z_f|s'_f)}_{\text{I}} \underbrace{P(z_d|z_f, s'_d)}_{\text{II}}$$

$$\underbrace{\frac{P(z_f|s'_d, s'_f)}{P(z_f|s'_f)}}_{\text{(III)}} \underbrace{\frac{P(z_d|s'_d, s'_f, z_f)}{P(z_d|z_f, s'_d)}}_{\text{(IV)}} \tag{3.7}$$

(III) here represents the residual effect of the perceptual process on the decision process and (IV) is the residual effect of the decision process on the perceptual process. The residual effects define the part of the information which is not captured by the observation in each of the processes. For example, in the perceptual process, the residual effect addresses the information which is not captured by the perceptual process. It is worth noting that when the observations in the decision process or the perceptual process are complete, i.e. all information required for the respective state updates and observation predictions is captured, the terms (III) and (IV) will become one.

Thus, if the states of the perceptual process and of the decision process are complete in the sense that they encode all information necessary to predict the corresponding observation probabilities, the equation would reduce to:

$$P(z_d, z_f|s'_d, s'_f) = P(z_d|z_f, s'_d)P(z_f|s'_f)$$

This reflects the ideal case where measurement updates can be performed completely independently for the perceptual and decision processes, and represents part of a potential condition under which a separation into perceptual and decision process is optimal within the proposed framework.

3.1.2 Prediction Formulation for Belief Prediction

In this section, we show how the prediction step of the POMDP belief update, $\sum_{s\in S} P(s'|s,a)bel(s)$ can be expresses separately for the perceptual and the decision processes:

$$\sum_{s\in S} P(s'|s,a)bel(s) = \sum_{s\in S} P(s'_d, s'_f|s_d, s_f, a)bel(s_f, s_d)$$
$$= \sum_{s_d,s_f\in S} P(s'_d|s_d, s'_f, s_f, a)P(s'_f|s_d, s_f, a)bel(s_d, s_f)$$
$$= \sum_{s_d\in S_d} P(s'_d|s_d, s'_f, a) \sum_{s_f\in S_f} \left(\frac{P(s'_d|s_d, s_f, s'_f, a)}{P(s'_d|s_d, s'_f, a)} P(s'_f|s_f) \right.$$
$$\left. \frac{P(s'_f|s_d, s_f, a)}{P(s'_f|s_f, a)} bel(s_f, s_d) \right)$$

The belief, $bel(s_f, s_d)$, can be written as follows:

$$bel(s_d, s_f) = bel(s_d|s_f)bel(s_f) \tag{3.8}$$

Using Bayes rule this can be re-written as follows:

$$bel(s_d|s_f) = bel(s_d)\frac{P(s_d|s_f, z_0, .., z_{t-1})}{P(s_d|z_0, .., z_{t-1})} \tag{3.9}$$

Then, by using (3.8) and (3.9), we have:

$$\sum_{s_d\in S_d} P(s'_d|s_d, s'_f, a)bel(s_d) \sum_{s_f\in S_f} \left(\underbrace{\frac{P(s'_d|s_d, s_f, s'_f, a)}{P(s'_d|s_d, s'_f, a)}}_{(*)} \right.$$

$$P(s'_f|s_f)bel(s_f)\underbrace{\frac{P(s'_f|s_d,s_f,a)}{P(s'_f|s_f,a)}}_{(**)}\underbrace{\frac{P(s_d|s_f,z_0,..,z_{t-1})}{P(s_d|z_0,..,z_{t-1})}}_{(***)}\Bigg) \tag{3.10}$$

The term (**) in (3.10) represents the residual effect of s_d on s'_f that s_f does not explain. We would like this value to be one, implying that the perceptual process state contains all information that is required to update itself. Term (*) in (3.10) similarly represents the residual effect that the previous state of a particle has on the predicted state of the POMDP that is not provided by the output of the perceptual process. Finally, term (***) in (3.10) represents the correlation between the belief of the perceptual process and the decision process. In other words, how much of the belief in the decision process is already handled by the belief in the perceptual process. To put it more simply, if s_d and s_f are dependent, term (***) defines what part of the belief of s_d is integrated into the belief in s_f inside of perceptual process.

Finally, by putting (3.7) and (3.10) together, we have the whole belief update, $bel(s')$, separately expresses as the belief updates for the perceptual process and the decision process and an additional set of correction terms to compensate for imperfections in the separation into the different processes:

$$\sum_{s_d\in S_d}\underbrace{P(z_d|z_f,s'_d)P(s'_d|s_d,s'_f,a)bel(s_d)}_{\text{POMDP belief update}}\underbrace{\frac{P(z_d|s'_d,s'_f,z_f)}{P(z_d|z_f,s'_d)}}_{\text{correction term}} \tag{3.11}$$

$$\sum_{s_f\in S_f}\Bigg(\underbrace{P(z_f|s'_f)P(s'_f|s_f)bel(s_f)}_{\text{Perceptual process belief update}}\underbrace{\frac{P(s'_d|s_d,s_f,s'_f,a)}{P(s'_d|s_d,s'_f,a)}}_{\text{correction term}}$$

$$\underbrace{\frac{P(s_d|s_f,z_0,..,z_{t-1})}{P(s_d|z_0,..,z_{t-1})}\frac{P(z_f|s'_d,s'_f)}{P(z_f|s'_f)}\frac{P(s'_f|s_d,s_f,a)}{P(s'_f|s_f,a)}}_{\text{correction term}}\Bigg) \tag{3.12}$$

The terms in parts (3.11) and (3.12) of the complete belief update equation show that the separation leads to the complete belief update for the perceptual process and decision process with a number of correction terms which account for interactions and correlations between state variables and observations in the two types of processes. There will be a perfect split under certain assumptions where all correction terms become one. Through this, this formula provides insight into how to come up with the best separation into perceptual filter and POMDP, and how to assign state space components to the constituent processes. Besides, since each of the updates is separately done , it leads to each process handling relevant parts of the process rather than the whole of it. As a result, the uncertainty in one part has no direct effect on the other side.

In the next section, we describe the proposed sampling based algorithm to deal with (3.12).

3.2 Sampling Algorithm for Continuous State Spaces

Due to the continuity of the belief state, handling it in an efficient way can significantly help to solve the POMDP. Most previous work in this area, as mentioned previously, concentrated on how the belief state can be handled and the value function can be approximated. In our method we augment this with a new approach where part of belief state is handled by the perceptual process and the other part by the decision process. Through this, we handle the belief state in a way that leads to a smaller state space for the decision process. To achieve this, we divide the overall POMDP problem into a decision process and a perceptual process. The decision process can be modeled itself as a POMDP with a reduced state space which allows us to use the available frameworks for policy learning to solve the reduced decision problem. For the perceptual process, a particle filter is used to track the state statistics. It is worth

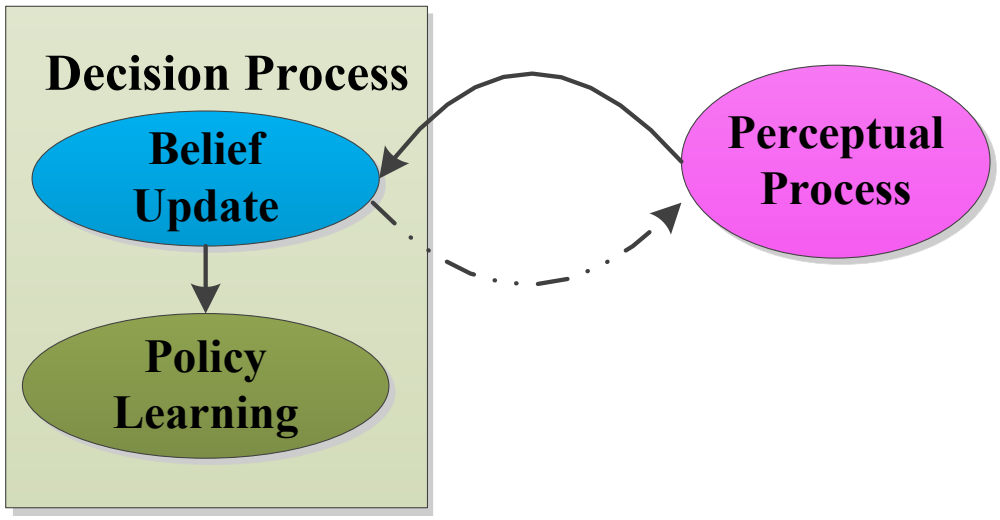

Figure 3.2. Overall System.

noting that the decision part uses the relevant parts of the belief state computed by the perceptual process in the same way as its local observations. How to determine the state spaces for the decision and the perceptual processes is generally difficult and beyond the scope of the thesis. However, through the mathematical and algorithmic analysis of our model, we provide some intuition how the separation should be done. In particular, the correction terms in the derivation provide some insight how state and observation variables should be separated to reduce POMDP complexity. Figure 3.2 shows the overall idea of our approach.

3.2.1 Separation of Belief Update in Continuous State Spaces

Since the belief update plays an important role in solving the POMDP, we derive here the equations to update the belief state in for the perceptual and decision

processes when applying our method in the case of a continuous state space. If S defines the state of the overall POMDP, the belief update equation in (2.7) can be rewritten for the combined state, s'_f, s'_d, of the two processes as $bel(s'_d, s'_f)$ and tracked as:

$$\int_{s_d} \int_{s_f} p(s'_d, s'_f | s_d, s_f, a) p(z_d, z_f | s'_d, s'_f) bel(s_d, s_f) ds_f ds_d$$

in which s_d and s_f define the states of the decision processes and the perceptual process, respectively. We assume here that the decision part is defined by the state/observation pair $\{s_d, z_d\}$ and the perceptual process by $\{s_f, z_f\}$. After applying Bayes rule and other probabilistic transformation rules as shown in the previous section for discrete state space, the following belief update formula for $bel(s'_d, s'_f)$ shows the separation into the two processes:

$$\int_{s_d} \Bigg(\underbrace{p(z_d | z_f, s'_d) p(s'_d | s_d, s'_f, a) bel(s_d)}_{\text{POMDP belief update}} \underbrace{\frac{p(z_d | s'_d, s'_f, z_f)}{p(z_d | z_f, s'_d)}}_{\text{correction term}}$$

$$* \int_{s_f} \Bigg(\underbrace{p(z_f | s'_f) p(s'_f | s_f) bel(s_f)}_{\text{Perceptual process belief update}} \underbrace{\frac{p(s'_d | s_d, s_f, s'_f, a)}{p(s'_d | s_d, s'_f, a)}}_{\text{correction term}}$$

$$\underbrace{\frac{p(s_d | s_f, z_0, .., z_{t-1})}{p(s_d | z_0, .., z_{t-1})} \frac{p(z_f | s'_d, s'_f)}{p(z_f | s'_f)} \frac{p(s'_f | s_d, s_f, a)}{p(s'_f | s_f, a)}}_{\text{correction term}} \Bigg) ds_f \Bigg) ds_d \tag{3.13}$$

(3.13) shows how the belief in the perceptual and decision process can be separately tracked. Meanwhile, it shows the correlation between s_d and s_f in terms of correction terms. Because, the decision process only needs to deal with decision-relevant state variables, it can lead to a smaller state space for the decision process which reduces the complexity of dealing with the POMDP. In addition, the decision process POMDP

only needs s_d to make decisions. As a result, it is necessary to calculate bel(s'_d) from bel(s'_d, s'_f). Hence, bel(s'_d) can be defined as follows:

$$bel(s'_d) = \int_{s'_f} bel(s'_d, s'_f) ds'_f \tag{3.14}$$

To solve (3.14), we propose a Monte Carlo sampling based algorithm to calculate bel(s'_d) by using (3.13) and (3.14). In the next section, we explore our algorithm in more detail.

3.2.2 Distributed Belief Tracking Algorithm

To explain the algorithm's operation, we classify terms in (3.13) into three classes:

- Sampling or Resampling terms that can be solved easily by pure sampling or resampling steps.
- Bias terms that bias the sampling distribution.
- Weight adjustments that adjust the weights of samples.

In our algorithm, we first deal with the perceptual process before updating the decision process. The reason for this is that the decision part uses feedback from the perceptual process to compute some of its terms (in particular, it uses the relevant parts of s'_f in the calculation of s'_d). Besides, this reflects the fact that the perceptual process mostly plays the role of a preprocessor for the decision part by handling and decoupling the state features that have no direct importance in the decision process but only serve to correctly track the belief of features that are relevant for decision making.

"Update Perceptual Process" in lines 2 through 3 of Algorithm 2 is responsible to calculate the bias term and track the belief of the perceptual process. Due to the simplicity of the particle filter and its ability to track multidimensional distributions

Algorithm 2 Calculate $bel(s'_d)$

1: Initialize sample sets of S_f and S_d

2: Update Perceptual Process

- Calculate the bias term $\frac{p(s'_f|s_d,s_f,a)}{p(s'_f|s_f,a)}$
- $\langle s'_f, w_f \rangle = Update\_Belief\_S_f(s_f, z_f)$

3: Correlate s_d with some sample from s_f.

4: Update Decision Process

- Calculate the bias term $\frac{p(s'_d|s_d,s_f,s'_f,a)}{p(s'_d|s_d,s'_f,a)}$
- $\langle s'_d, w_d \rangle = Update\_Belief\_S_d(s_d, z_d, s'_f, z_f)$
- Calculate the weight adjustment terms

5: $Feedback\_adjust(S'_d, S'_f)$

6: Down Sample to $(bel(s'_d))$

efficiently, it is used here to handle the belief tracking. It uses the observation and prediction model of the system to generate the particle sets for the given states. In line 3, the samples of s_d are correlated to the samples of s_f. If we assume that the size of the sample sets for s_d and s_f are m and n, respectively, the size of the sample set after correlation in the *worst case* will be $(n * m)$. "Update Decision Process" in line 4 is responsible to track the belief of the decision process. Rather than just using s_d, z_d to track the belief, $Update\_Belief\_S_d$ uses s'_f and z_f as well. Since $\frac{p(z_f|s'_d,s'_f,a)}{p(z_f|s'_f,a)}$ and $\frac{p(z_d|s'_d,s'_f,z_f,a)}{p(z_d|z_f,s'_d,a)}$ do not depend on the belief (i.e. they do not need samples to be computed as they come from the system model), their computation is delayed until all other terms' computation is completed. $Feedback\_adjust$ is responsible to calculate them. However, it is worth noting that we are interested in making these terms unnecessary, i.e. in making them one by selecting s_f and z_f accordingly. This

is achieved by making the observation model of the perceptual process maximally independent from the observation of the decision process.

In the most general case, lines 1 to 5 of the algorithm calculate bel(s'_d, s'_f). This is necessary in order to correctly capture the remaining interdependencies between the states in the perceptual and the decision processes. In order to correctly do this, the samples from s_d have to be correlated with samples from s_f. This step is here called "Up sampling". The reason for this step is to extract information from the belief of s_d which is already captured in $bel(s_f)$ and thus to avoid including it twice. Within the mathematical derivation, $\frac{p(s_d|s_f,z_0,..,z_{t-1})}{p(s_d|z_0,..,z_{t-1})}$ in (3.13) is responsible for this. On the other hand, to learn an optimal decision policy we are interested only in $bel(s'_d)$ which can be calculated using (3.14). To do this and to simultaneously reduce the size of the up-sampled state set, we down-sample it. For this, any resampling method can be used. The result of this step is a sample set for $bel(s'_d)$.

3.3 Policy Learning

The proposed framework can be used either in the discrete state space or continuous state space. In the discrete state space any policy learning method can be used with the proposed framework. However, in the discretized state space experiments performed here, we use the Linear Q-learning to learn a policy. For the continuous state space experiments, on the other hand, we use the value function learning method used in the Monte Carlo POMDP framework [3] to learn a policy. This allows us to represent the value function directly in terms of a sample set represented by belief state sample set/value parameter pairs, making this a convenient method to use with the proposed method. It is worth noting that the proposed framework is generally independent of the learning policy method. In other words, it can be used with any learning policy method which can deal with learning in a POMDP.

3.4 Practical Considerations and Analysis

Our proposed algorithm can reduce the complexity of the overall POMDP by reducing the number of state variables in the decision POMDP either in continuous or discrete state spaces. This, in turn, leads to a smaller sample set for the decision process POMDP in a continuous state space or fewer state variables for POMDPs in a discrete state space which reduces the complexity of solving the POMDP. In addition, because handling the sample set in the particle filter is just much cheaper than solving the POMDP, we prefer the perceptual process to handle the state space as much as possible. For this reason, during the separation, we want the perceptual process to avoid the feedback from the decision process. However, the states of the POMDP generally require some information or feedback from the perceptual process. If we can find a separation that can have completely isolated and independent observations for the perceptual process and decision part, some of the terms in (3.13) will automatically become ones, significantly simplifying the algorithm. In the next chapter, we compare our method with the conventional POMDP methods and show the efficiency of our algorithm.

CHAPTER 4

EXPERIMENTS AND RESULTS

In order to empirically demonstrate the applicability and effectiveness of our approach, we simulated our algorithm in two difference situations. In the first one, we used our method when the state space of the POMDP is discretized. In the second one, our method was used on a continuous state space. Since, the representations of the value functions are completely different in each of the state spaces, it is important to show how our algorithm behaves in each of these spaces. For the first case, we simulate the problem of tracking a bouncing ball using a robot with learning capability, and for the second experiment, a simplified version of the SpaceInvader game is used.

4.1 POMDP with Discretized State Space

To demonstrate the framework, we performed experiments with a simulated robot and a simulated bouncing ball in a grid world of size 9×8, where the actions of the robot, which move it between grid cells, are to be learned, and the ball moves in a continuous space, bouncing off the walls surrounding the grid world. The robot carries a camera which can measure the distance and angle to the bouncing ball but only with a certain level of sensor noise. In this simulation, the robot wants to track the bouncing ball by putting the camera and himself in the right position and direction to keep the ball in view. We used the particle filter to track the bouncing ball and Linear Q-learning as the method to solve the POMDP part. Figure 4.1 shows a snapshot of the simulation.

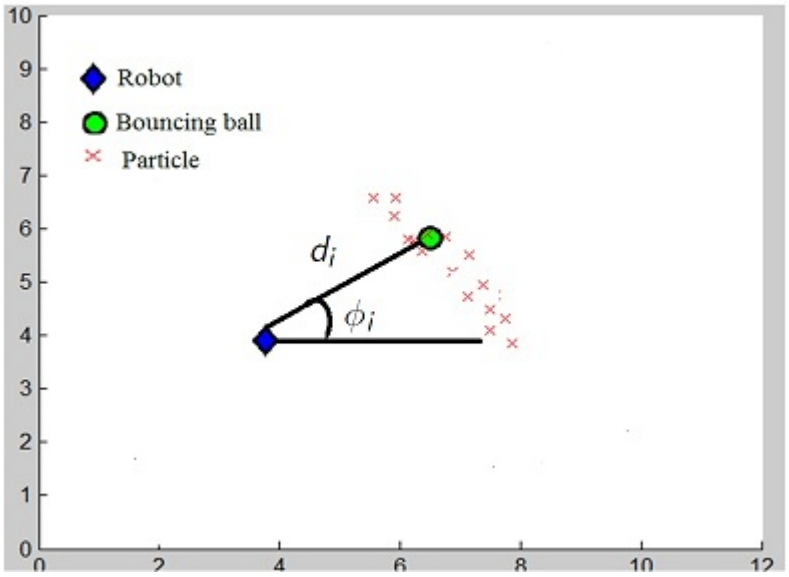

Figure 4.1. Bouncing Ball and Simulated Robot.

4.1.1 Simulation Model

Here, we describe how the object and robot are modeled in this environment.

4.1.1.1 System Model

In order to make the simulation more realistic, we used egocentric coordinates for the observations made by the robot camera. Here, the camera system provides coordinates relative to the robot rather than in terms of the global coordinate system. Using egocentric coordinate makes the perception more realistic and more noise resistant as it no longer relies on the robot location. In addition, egocentric coordinates can model the uncertainty in perception completely independent of the robot location. The outputs of the camera are distance and angle towards the bouncing ball. The samples, or particles, represent possible bouncing ball locations as a vector

of angle and distance. The equations for the angle and distance calculation in the simulation are as follows:

$$\phi_i = arctan(\frac{Y_i}{X_i}) \tag{4.1}$$

$$d_i = \sqrt{X_i^2 + Y_i^2} \tag{4.2}$$

where X_i and Y_i are the relative location of the object to the camera. These can be calculated using the following equations:

$$X_{i_t} = X_{o_t} - X_{r_t}$$

$$Y_{i_t} = Y_{o_t} - Y_{r_t}$$

Here, X_r and Y_r are the global location of the robot and X_o and Y_o are the global location of the bouncing ball. The model for the movement of the bouncing ball is:

$$X_{o_t} = X_{o_{t-1}} + \Delta t * V_{x_{t-1}} + \omega_{x_o}$$

$$Y_{o_t} = Y_{o_{t-1}} + \Delta t * V_{y_{t-1}} + \omega_{y_o}$$

In which V_x and V_y are the velocity in the x-direction and y-direction, respectively. ω_x and ω_y are Gaussian random variables with mean of zero and standard deviation of 0.23 and 0.22, respectively. In the simulation, when the ball hits the wall the velocity in the hitting direction will be reversed. The particle filter uses the following as its prediction model:

$$X_{i_t} = X_{i_{t-1}} + F_x(a) + \Delta t * (V_{x_{t-1}} + \zeta_x) + \omega_{x_i} \tag{4.3}$$

$$Y_{i_t} = Y_{i_{t-1}} + F_y(a) + \Delta t * (V_{y_{t-1}} + \zeta_y) + \omega_{y_i} \tag{4.4}$$

In (4.3) and (4.4), both ωs are Gaussian noise for the location with mean of zero and standard deviations of *0.1*. ζ is the Gaussian noise model for velocity with mean of zero and standard deviation of *0.25* in the x-direction and *0.21* in the y-direction. This

velocity noise, although not present in the simulated system, is necessary to allow the particle filter to estimate the system's velocity, and is partially compensated through the reduced position noise, ω. After calculating these values for each particle, we convert them to ϕ_i and d_i using (4.1) and (4.2). Index i indicates the particle numbers. Hence, the state of the particle filter can be represented as (ϕ_i,d_i,V_x,V_y). x_o,y_o can be converted to the ϕ_i and d_i, using equations (4.1) and (4.2). Since, we are using egocentric coordinates, the location of the robot is not important. However, the effect of the robot movement on the state of the particle filter should be considered. For this reason, we assume that, the camera is always facing in the right direction and the robot moves in one of four directions: (Up, Down, Right, Left). Given that, $F_x(a)$ and $F_y(a)$ define the effect of the instantaneous robot movement on the perceptual process. For example, when the robot moves left, all egocentric observations and states will move to the right. In other words, $F(a)$ defines how much the local features change through the robot movement. In our simulation, the actions move the robot by 1 square and thus the local features also move with a step size of one:

$$F_x(Right) = -1,\ F_x(Left) = 1,\ F_x(Up) = F_x(Down) = 0$$

$$F_y(Up) = -1,\ F_y(Down) = 1,\ F_y(Right) = F_y(Left) = 0$$

Finally, the observation model for the particle filter is:

$$d_{obs} = d + \tau$$

$$\phi_{obs} = \phi + \varrho$$

where τ and ϱ are zero mean Gaussian random variables with standard deviations of 0.18 and 0.09, respectively. The weights, $w^{(i)}$, of each particle are calculated as:

$$w^{(i)} = N_{(0,0.18)}(d^{(i)} - d_{obs}) * N_{(0,0.09)}(\phi^{(i)} - \phi_{obs})$$

where *N(x)* is the value of the normal distribution at point x.

4.1.1.2 Reward Function

The goal of this simulation is to learn a policy that allows the robot to put himself and camera in right position to track the bouncing ball without wasting valuable energy and considering its state of knowledge about its own location and the location of the bouncing ball. In order to do that, the reward function, r, is assigned based on the distance of the robot from the ball:

$$r = \begin{cases} -0.1 & \text{if } d = 0 \\ 5 & \text{if } d < 2 \\ 2 & \text{if } 2 \leq d < 4 \\ 0 & otherwise \end{cases}$$

4.1.2 Learning Method

We use Linear Q-Learning in these experiments to learn a policy for the POMDP decision process. Since there is no uncertainty in the robot motion and location, and because the state representation and observations chosen for the particle filter perfectly correlates the uncertainty in the POMDP with the one used for the particle filter, the belief update in the POMDP on which Linear Q-Learning is used, reduces to a direct transformation of the posterior probability or belief of the particle filter. This leads to the following value updates:

$$Q(b, a) = \sum w^{(i)} q(s^{(i)}, a)$$

$$\Delta q(s^{(i)}, a) = \alpha \; w^{(i)}(r + \gamma * max_{a'} Q(b', a') - Q(b, a))$$

where $w'^{(i)}$ and $w^{(i)}$ are weights of particles at time t and $t-1$, respectively, and i indicates the particle number. In addition, γ is the discount factor, which for these experiments is set to 0.7. α defines the learning rate which for these experiments

is 0.1. In order to explore the action space in each step of Linear Q-Learning, we use the $\varepsilon - greedy$ method with $\varepsilon = 0.1$. Finally, the states of the POMDP are $(x_r, y_r, x_o, y_o, V_x, V_y)$.

Since, in our simulation, the robot location and motion is certain, all correction terms in (3.7) for the POMDP and observation probability are one with the exception of the one for belief correlation between the particle filter and the POMDP (term (***) in (3.10)) which here cancels the belief term of the POMDP since all prior belief is already handled in the particle filter. Similarly, the correction terms in the particle filter are one due to the fact that there is no influence from s_d on s'_f and from s'_d on the z_f.

4.1.3 Traditional Method

In order to compare our method, we perform another simulation with the above configuration. In this case, instead of using a separate particle filter for updating the belief of a perceptual process separately from the belief of the POMDP, we use a monolithic POMDP with the egocentric camera observations and (2.3) to perform the belief update in order to solve the POMDP with the same Linear Q-Learning algorithm without any separation of perception and decision processes.

4.1.4 Experimental Results

In this experiment, the proposed method and the traditional, monolithic POMDP comparison method were executed for 3335 trials where each trial consists of 300 episodes, each consisting of 25 steps. Each of the methods was performed 10 times. To accommodate the used learning framework, we used the continuous belief state update for the particle filter and discrete states for the POMDP. Figure 4.2 shows the

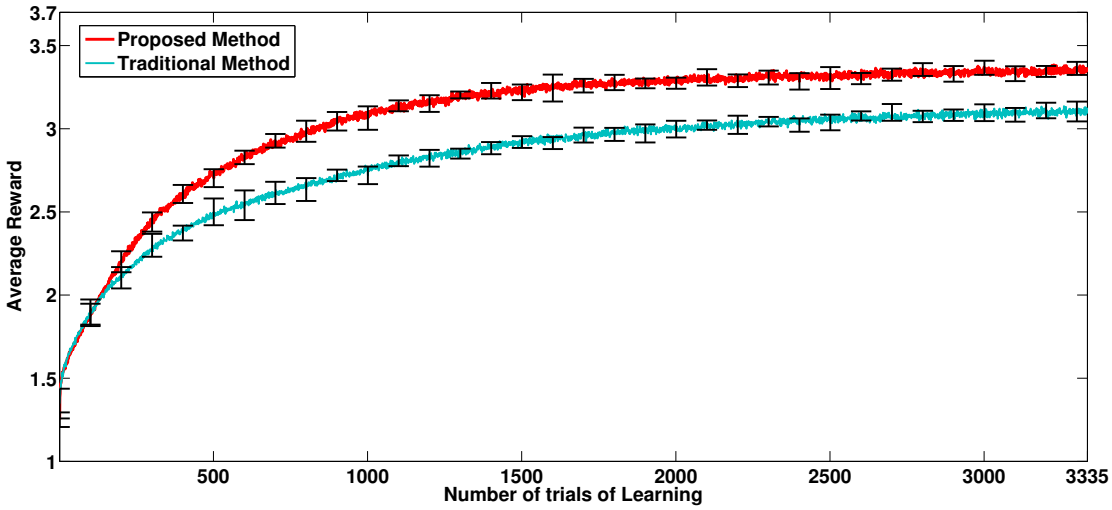

Figure 4.2. Learning Curve.

average learning curve for the 10 different executions. Each point in the curve shows the average reward for one trial and error bars represent one standard deviation.

These results show that the proposed method outperforms the traditional comparison method using the same POMDP learning algorithm both in terms of learning speed and final policy value. Part for the reason for this can be seen in the increased efficiency of the proposed separation into perceptual and decision processes which also allows the use of a discrete POMDP method while maintaining the belief state in the perceptual process in a continuous form (which is not possible in monolithic methods). It is worth noting that even though we used the sampling method, which is an approximate method, the result shows that at the end our method is achieving better final results. In addition, the results illustrate how the proposed method benefits from the potential complexity reduction discussed in the previous sections.

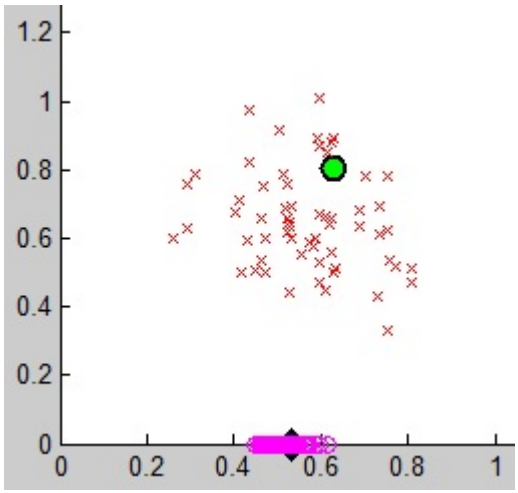

Figure 4.3. Space Invader.

4.2 POMDP in Continuous State Space

To demonstrate the efficiency and applicability of our algorithm in a continuous state space, we simulated a simplified version of the Space Invader game. In this game, a robot horizontally moves across the bottom of the screen and fires with a laser at a flying object. The overall goal is to learn the policy for the robot that leads to reward maximization. The reward is here related to the number of the times that the robot can shoot the object. Figure 4.3 shows a snapshot of the game.

The simulation has been performed with two different algorithms. For the first algorithm, we used the Monte Carlo POMDP [3] algorithm. In this algorithm, the entire problem has been modeled as the decision POMDP. For the other one, we used our proposed algorithm to track the belief states separately and to find the maximum policy in terms of $bel(s_d)$. For our algorithm, instead of modeling the complete problem as a monolithic POMDP, we divided it into a perceptual process responsible for correctly tracking the state of the object and a decision process. In order to solve the decision part in our algorithm, the same approach as in the Monte Carlo POMDP [3] has been used. For the perceptual process, we used a particle filter.

As described previously, our algorithm reduces the complexity of solving the overall PODMP by breaking it down into decision and perceptual parts. In the following section, we describe the simulation model.

4.2.1 State Space

Here we describe the state space for each algorithm.

4.2.1.1 Monte Carlo POMDP algorithm

For this algorithm, the state space of the problem is formed by the state variables $\langle X_o, Y_o, V_{x_o}, V_{y_o}, X_r \rangle$ representing the location and velocity of the object in the 2D space and the location of the robot in its 1D space.

4.2.1.2 Our Proposed Algorithm

For the proposed algorithm, we separate the perceptual and the decision process, resulting in two types of states:

- Decision Process POMDP : $\langle X_r, X_o \rangle$.
- Perceptual Process (Particle Filter): $\langle X_o, Y_o, V_{x_o}, V_{y_o} \rangle$.

In order to solve the decision process POMDP, we used the Monte Carlo POMDP algorithm augmented with the proposed approach to updating the corresponding belief state sample set, and for the perceptual process we used the particle filter to track its belief state. It is worth noting that the separation is done such that states in the POMDP and the particle filter components are sufficient to make decisions and track the belief state.

4.2.2 Simulation Model

Here, we describe how the object and robot are modeled in this environment.

4.2.2.1 Flying Object movement Model

The flying object moves in a world of size 2×2. When the object hits the wall, it bounce back towards the inside. In order to reflect the bounce, the velocity of the object is reversed in the bounce direction. Since the object area is a 2D space, X and Y represents the location of the object. Thus, the object moves according to the following equations:

$$X_{o_t} = X_{o_{t-1}} + \Delta t \times V_{x_{t-1}} + \omega_{x_o}$$

$$Y_{o_t} = Y_{o_t} + \Delta t \times V_{y_{t-1}} + \omega_{y_o}$$

where ω_x and ω_y represent a Gaussian noise with mean 0 and standard deviation of 0.01.

4.2.2.2 Robot movement Model

The robot only moves in a 1D space (i.e. X). When the robot hits the walls (right or left wall), it stays at the wall, i.e. in $X = 0$ for the left wall and $X = 2$ for the right wall. Thus, the robot moves according to following equation:

$$X_{t_r} = X_{t-1_r} + f(action) + \omega_{x_r}$$

where ω_{x_r} is a movement noise with standard deviation of 0.01. The available actions for the robot are left, right, and shoot. Since the actions are continuous, they effect the robot's position as follows:

$$f(right) = 0.2, f(left) = -0.2, f(shoot) = 0;$$

4.2.2.3 Observation Model for Robot

The only observation of the robot is a noisy, binary close range wall proximity sensor that indicates that the wall is close by. In order to calculate the observation

model for the robot, we calculate the probability of the sensor indicating a wall using a normal distribution with standard deviation 0.1 and mean 0 for the left and right wall:

- Left Wall:

$$W_{left} = \frac{N_{(0,0.1)}(x_r)}{N_{(0,0.1)}(0)}$$

- Right Wall:

$$W_{Right} = \frac{N_{(0,0.1)}(N_x - x_r)}{N_{(0,0.1)}(0)}$$

in which N_x is the width of the world and x_r is the location of the robot. The final values, W_{Right} and W_{Left} are the likelihood of he right and the left wall proximity sensor indicating a wall and are used to generate a corresponding wall proximity observation.

4.2.2.4 Observation Model for Flying Object

The observation of the object is obtained from a simplified "radar station" located in the lower left corner of the object area. It provides a noisy measure of the distance from position $(0, 0)$ and can be calculated by:

$$d_o = \sqrt{X_o^2 + Y_o^2} + \psi$$

in which ψ is Gaussian noise with a standard deviation of 0.02.

4.2.2.5 Reward Function

The goal of this simulation is to learn a policy that allows the robot to shoot the object as frequently as possible without wasting valuable energy and considering its state of knowledge about its own location and the location of the flying object. The reward function is modeled as follows:

$$r = \begin{cases} 100 & \text{if } |X_r - X_o| < 0.25 \wedge action = shoot \\ -0.1 & \text{if } action = right \\ -0.1 & \text{if } action = left \\ -80 & \text{if } action = shoot \wedge miss \end{cases}$$

4.2.3 Monte Carlo POMDP Implementation Detail

As we previously mentioned, the Monte Carlo POMDP [3] method is used as value function learning. The following pseudo-code shows the Monte Carlo POMDP algorithm in more detail [20]:

repeat
  Sample $x \sim b(x)$ and init X with M samples
  repeat
    for all available actions u **do**
      $Q(u) = 0$
      repeat
        select random $x \in X$
        $x' \sim p(x'|u, x)$, $z \sim p(z|x')$
        $X' = Particle\_filter(X, u, z)$
        calulate the distance of X' from table entries
        d_k=$KL\_distance(X', (V_0, ...V_n))$
        if there are K-nearset neighbor in V-table **then**
          $V(X')$=$\frac{\sum \frac{V_k}{d_k}}{\sum \frac{1}{d_k}}$
        end if
        $Q(u) = Q(u) + \frac{1}{n}(r(x, u) + \gamma V(X'))$
      until N

16: **end for**

17: $action = \epsilon\text{-}greedy(Q)$

18: d_k=$KL\_distance(X, (V_0, ...V_n))$

19: **if** there are K-nearset neighbor in table **then**

20: $V(X) = \frac{\sum \frac{V_k}{d_k}}{\sum \frac{1}{d_k}}$

21: Update the table

$V_k = V_k + \alpha\eta\frac{1}{d_k}(Q(action) - V_k)$

22: **else**

23: add $Q(action)$ to the table

24: **end if**

25: $x' \sim p(x'|action, x)$

26: $z \sim p(z|x')$

27: $X' = Particle\_filter(X, u, z)$

28: $X = X'$

29: **until** episode over

30: **until** convergence

The KL-divergence calculates the distance between two distributions. In our case, since we have samples sets, we need to use a Mixture of Gaussians in the $KL\_distance$ function. Calculating the Mixture of Gaussian is $O(kn^2)$ where n is the number of samples.

The $KL\_distance$ between two distributions can be calculated as follows:

$$KLD(P||Q) = \int_x P(x) \log(\frac{P(x)}{Q(x)})dx$$

In order to apply our framework to this algorithm, we made the following changes:

- In line **7**, two instances of particle filter functions are called. One with $\langle X_o, Y_o, V_{x_o}, V_{y_o} \rangle$ for the perceptual process and another one for the POMDP(decision process) with the $\langle X_r, X_o \rangle$ state variables .
- In line **8** and **15**, $KL\_distance$ uses just $\langle X_r, X_o \rangle$ as state variables.

In fact, in our method, the KL divergence method and V table both deal with a smaller number of state variables in comparison with the traditional Monte Carlo POMDP which in turns effects learning speed and complexity of the algorithm.

4.2.4 Simulation Results

To compare our algorithm with the Monte Carlo POMDP algorithm, we calculate two different quantities: i) learning curve in terms of the performance achieved as a function of the number of learning trials, each consisting of 25 steps, and ii) the time complexity in terms of the computation time required per step taken. The learning curve shows how fast the algorithm can learn the optimal policy. On the other hand, the time complexity measurement shows how long it takes for the algorithm to calculate each value function update. Since, our algorithm deals with a smaller set of state samples, it can convergence to an optimal policy much faster than the standard Monte Carlo POMDP approach. In addition, the time complexity of our algorithm is much lower due to the reduced amount of complexity in the computation of the decision part.

Figure 4.4 shows the time complexity of our method and traditional Monte Carlo POMDP. Since, our algorithm deals with a smaller state space in the decision process, the average running time of the algorithm is dramatically lower and also increases much more slowly as it requires significantly fewer value-function samples to be stored and compared. The main reason for the much lower time complexity is that our algorithm builds a significantly smaller table for the sample sets used to represent

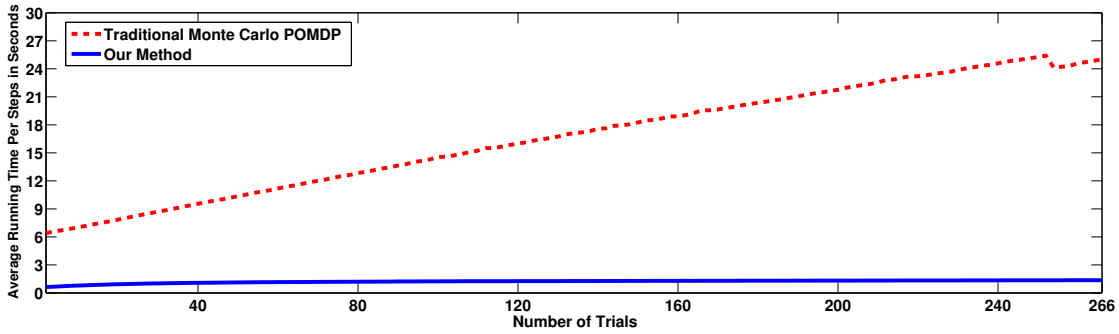

Figure 4.4. Time Complexity.

the value function which leads to a dramatically smaller number of computations of KL-divergence. Conversely, the Monte Carlo method has larger average running time because its sample table is much larger than the one for our method which leads to higher search time. Furthermore, the monolithic Monte Carlo POMDP deals with a larger state space which increases the complexity of calculating the value function and the sample set.

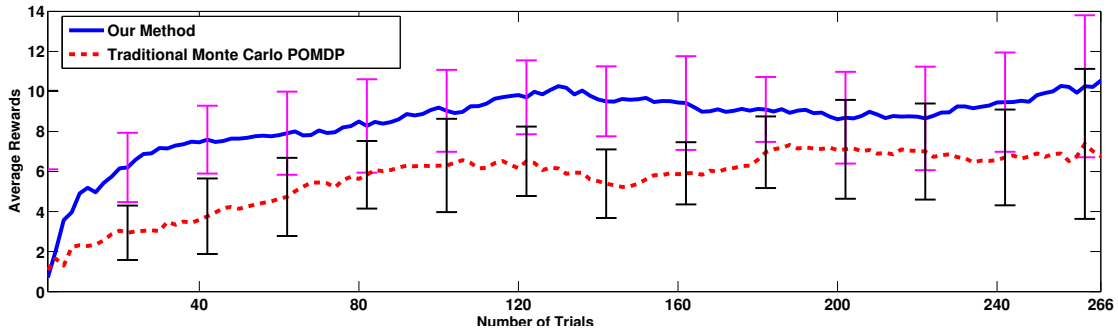

Figure 4.5. Learning Curve.

Figure 4.5 shows the learning curves, averaged over 18 runs for both algorithm in terms of the average reward per step over 22 trials. The error bars in the figure

represent one standard deviation. As the figure shows, the learning speed of our algorithm is much faster than for the monolithic Monte Carlo POMDP. Moreover, after 130 trials our algorithm is converged; however, the Monte Carlo POMDP still oscillates around a value significantly lower than the one achieved by our method. There are a couple of reasons for this behavior. First, the Monte Carlo POMDP has much larger value tables because it is dealing with 5 state variables as compared the two state variables in our method. In addition, since this method has to keep track of a larger belief state space, it takes more time to have a sufficient distribution to cover the space states. As a result, the Monte Carlo POMDP method requires more time to cover the value function space and more time to learn.

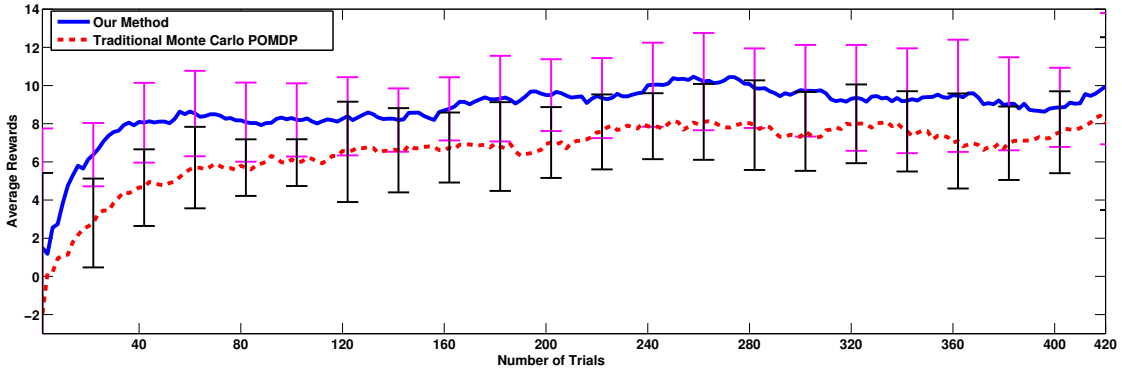

Figure 4.6. Learning Curve with tighter threshold.

In addition, we perform a different experiment where all parameters are the same as in the previous experiment except for the KL threshold. For this experiment, we choose the KL threshold tighter i.e. we require a close match for the K-nearest neighbor function approximator for the value function. In the previous experiment, the threshold was 0.5. However, this time, we make it tighter and assign 0.4. Fig-

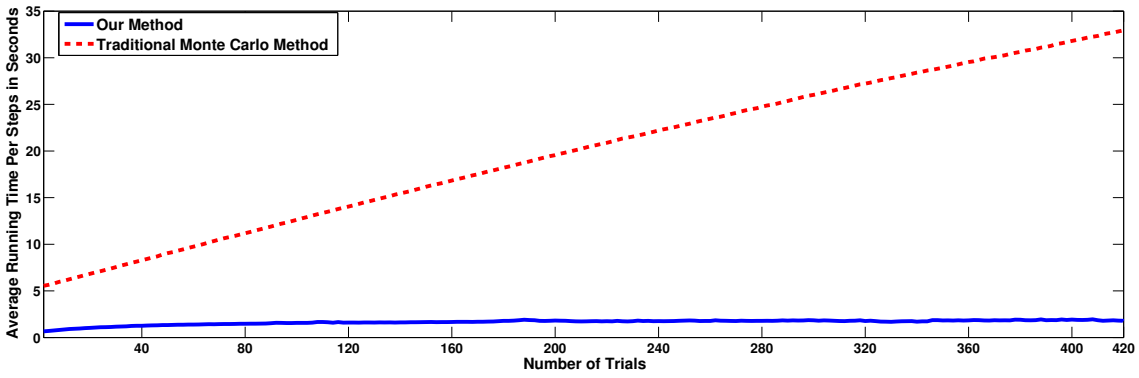

Figure 4.7. Time Complexity with tighter threshold.

ure 4.6 and 4.7 shows the learning curves and time complexity curves, averaged over 18 runs for both algorithm in terms of the average reward per step over 22 trials. It worth noting that, our algorithm after 180 trials is converged; however, the Monte Carlo POMDP again oscillates around a value significantly lower than the one achieved by our method. As the figures show, our algorithm outperforms the traditional Monte Carlo POMDP in terms of learning rate and learning time.

In conclusion, Figures 4.4, 4.5, 4.7, and 4.6 show that our method outperforms the Monte Carlo POMDP, indicating its potential to allow learning to scale to larger problems.

CHAPTER 5

CONCLUSION AND FUTURE WORK

POMDPs are a very powerful and general modeling and decision solution framework that can address a wide range of problem domains. However, it suffers from issues in terms of tractability as the complexity of POMDP solutions increases rapidly as the complexity of the state space increases. In this thesis, we proposed a framework to reduce the complexity of POMDPs by separating them into separate decision and perceptual processes. We showed formally that this separation is possible and derived the compensation terms that address interactions between the processes if no optimal process separation is achieved. In the proposed method, we used a particle filter as the perceptual process and a POMDP as the decision process. Using the proposed framework a significant reduction in the complexity of the POMDP solution can be achieved by performing a separation into perceptual and decision process in a way that removes many of the compensation terms. We mathematically and empirically show in a set of experiments that whenever a good separation is achieved, the complexity of solving the overall POMDP is reduced. In future work, we want determine analytically how much the complexity of the overall POMDP can be reduced and under what precise conditions. Furthermore, we will study the potential of using the derived formulation to automatically make decisions about the appropriate split between perceptual and decision processes.

REFERENCES

[1] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *International Conference on Machine Learning(ICML 95)*, 1995, pp. 362–370.

[2] D. Braziunas, "Pomdp solution methods," University of Toronto, Tech. Rep., 2003.

[3] S. Thrun, "Monte carlo POMDPs," in *Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 1064–1070.

[4] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: an anytime algorithm for pomdps," in *Proceedings of the 18th international joint conference on Artificial intelligence*, ser. IJCAI'03. Morgan Kaufmann, 2003, pp. 1025–1030.

[5] A. Brooks and S. B. Williams, "A monte carlo update for parametric pomdps." in *ISRR*, ser. Springer Tracts in Advanced Robotics, M. Kaneko and Y. Nakamura, Eds., vol. 66. Springer, 2007, pp. 213–223.

[6] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.

[7] N. Roy, "Finding Approximate POMDP solutions Through Belief Compression," *Journal of Artificial Intelligence Research*, vol. 23, 2000.

[8] A. Brooks, A. Makarenko, S. Williams, and D. H. Whyte, "Parametric POMDPs for planning in continuous state spaces," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 887–897, 2006.

[9] E. Zhou, M. C. Fu, and S. I. Marcus, "Solving continuous-state pomdps via density projection," *IEEE Trans. Automat. Contr.*, vol. 55, no. 5, pp. 1101–1116, 2010.

[10] E. Brunskill, L. Kaelbling, T. Lozano-Pérez, and N. Roy, "Continuous-state POMDPs with hybrid dynamics," in *International Symposium on Artificial Intelligence and Mathematics*, 2008.

[11] W. S. L. Hanna Kurniawati, David Hsu, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.

[12] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based pomdp solvers," *Autonomous Agents and Multi-Agent Systems*, pp. 1–51, 2012.

[13] P. Poupart, K.-E. Kim, and D. Kim, "Closing the gap: Improved bounds on optimal pomdp solutions," 2011.

[14] P. Poupart and C. Boutilier, "Value-directed compression of pomdps," in *In Neural Information Processing Systems 15*. MIT Press, 2002, pp. 1547–1554.

[15] A. Y. Ng and M. Jordan, "Pegasus: a policy search method for large mdps and pomdps," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, ser. UAI'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 406–415.

[16] P. A. Coquelin, R. Deguest, and R. Munos, "Particle filter-based policy gradient in pomdps," in *Neural Information Processing Systems(NIPS)*, 2008, pp. 337–344.

[17] Z. Ghahramani, "Unsupervised learning," in *Advanced lectures on machine learning*, ser. Lecture Notes in Artificial Intelligence 3176, O. Bousquet, G. Raetsch, and U. von Luxburg, Eds. Berlin: Springer-Verlag, 2004.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA,, 1998.

[19] A. Ng. (2011) CS229 machine learning. [Online]. Available: http://cs229.stanford.edu/notes/cs229-notes12.pdf

[20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*, ser. Intelligent robotics and autonomous agents. The MIT Press, 2005.

[21] S. Singh. (2010) Reinforcement learning: Tutorial + rethinking state, action and reward. [Online]. Available: $http://videolectures.net/mlss2010\_singh\_rlt/$

[22] M. T. Izadi and D. Precup, "Using rewards for belief state updates in partially observable markov decision processes," in *Proceedings of the 16th European conference on Machine Learning*, ser. ECML'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 593–600.

[23] J. M. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart, "Point-based value iteration for continuous pomdps," *JOURNAL OF MACHINE LEARNING RESEARCH*, vol. 7, pp. 2329–2367, 2006.

[24] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

BIOGRAPHICAL STATEMENT

Rasool Fakoor received his bachelors in Computer Engineering in 2006 from Azad University of Tehran. He started his Masters in Computer Science in 2010 in University of Texas at Arlington. His research interests are machine learning, social networks, dealing with big data, and cloud computing. He is continuing with his PhD after graduation.