Decision-Making for Autonomous Systems in Partially Observable Environments

A Dissertation

Presented to

the Faculty of the Department of Industrial Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

IN INDUSTRIAL ENGINEERING

by

Henry I. Ibekwe

May 2013

Decision-Making for Autonomous Systems in Partially Observable Environments

Henry I. Ibekwe

Approved:

Chair of the Committee
Ali K. Kamrani, Ph.D., P.E., Associate Professor
Industrial Engineering

Committee Members:

Qianmei Feng, Ph.D., Associate Professor
Industrial Engineering

Erhun Kundakcioglu, Ph.D., Assistant Professor
Industrial Engineering

Jagannatha R. Rao, Ph.D., Associate Professor
Mechanical Engineering

Keh-Han Wang, Ph.D., Professor
Civil & Environmental Engineering

Suresh K. Khator, Associate Dean,
Cullen College of Engineering

Gino J. Lim, Associate Professor
Chair of Department of Industrial Engineering

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Ali Kamrani, for his patience, guidance and support throughout this process. Without his constant encouragement and insightful discussions this dissertation would not have been successful. I would also like to thank my committee members, Dr. Feng, Dr. Kundakcioglu, Dr. Rao and Dr. Wang for their critical review and assessment of my work. Your support is highly appreciated as this dissertation significantly improved with your feedback.

Secondly, I wish to also acknowledge my loving parents, Mr. Augustine A. Ibekwe and (late) Mrs. Rebecca C. Ibekwe, for instilling in me a strong work ethic and providing the necessary educational foundations to achieve my academic goals. Also to my siblings Eric Ibekwe, Stella Ibekwe and Jane Ibekwe, thanks for putting up with me during some of the stressful periods. Perhaps, I may have some time to now relax and spend some time with you guys. I would also like to thank my good friend, Antoinette Baptiste, for her early support in my graduate studies. Many thanks to my colleagues, Maryam Azimi and Hazem J. Smadi for their useful discussions and encouragement during some of the more challenging times. I also wish to thank all other friends and family members not mentioned for their support.

Lastly, this dissertation is dedicated to my late loving mom, Mrs. Rebecca C. Ibekwe (1954-2011), whom while alive would listen to some of the ambitious and eccentric ideas I had for my dissertation. Thanks mom! You'll truly be missed!

Decision-Making for Autonomous Systems in Partially Observable Environments

An Abstract

of a

Dissertation

Presented to

the Faculty of the Department of Industrial Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

IN INDUSTRIAL ENGINEERING

by

Henry I. Ibekwe

May 2013

Abstract

Decision-making for autonomous systems acting in real world domains are complex and difficult to formalize. For instance, consider the task of autonomously navigating a mobile robot in an automated manufacturing facility. Its task is to transport hazardous material from a collection site to a disposal site. This is a navigation problem where the robot has to consider numerous variables such as collision avoidance, recognition of goal locations, accurate selection of the desired material, and knowledge of its location within the facility. The difficulty is often to reliably model the uncertainties and dynamics of the robot-environment interaction when the robot can only partially observe the states of the environment.

Therefore a principal problem in designing mobile robots that can efficiently navigate indoor domains to achieve a desired task autonomously is to construct robust models for efficient planning and motion control in stochastic domains. This is still a difficult and open problem despite significant advances. The robot must generate efficient *policies* to reliably accomplish its tasks while accounting for uncertainties in both its action and perception. In this dissertation we model the uncertainties in action selection and perception using a sequential decision-making model. The mathematical formalism adopted is the *Partially Observable Markov Decision Process* (POMDP), a generalization of the well-known *Markov Decision Process* (MDP). Though POMDPs represent a robust formalism for the modeling of agent-based decision making, it is still very difficult and often intractable to compute optimal solutions for problems with large state space due to the high

dimensionality of the underlying belief space. We propose a technique called *Goal-Specific Representation* (GSR) that exploits domain structure and generates policies over a subset of the state space given a map of the domain, a starting location and a goal location. We solve the resulting POMDP model using a *Point-Based Value Iteration* (PBVI) solver and apply the generated policies for navigation on an autonomous robot. We anticipate that the results from this work can be applied in manufacturing facilities to enhance automation and healthcare domains for assisted care.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of Agent-Based Decision Making

Autonomous agents typically interact in non-trivial domains that are dynamic and relatively unpredictable. In the case of biological agents, they actively gather information about their environment to generate perceptual models for use in achieving high-level goals or simply as a primitive low-level mechanism without any directed use of the internal models. Mechanical agents, such as robots can be designed to perform similar functions where the designer has a set of basic requirements for the robot to achieve. The robot could be designed to autonomously transport objects from an initial location to a predefined destination or could be designed to actively gather information and explore unknown terrain.

We may also consider decision-making from a more abstract perspective where the agent is an abstraction that can be clearly distinguished from an environment and has the ability to modify the *state* of the environment to meet a desired goal. The decision-maker or agent in this case need not be physical. For instance, it could be software designed to control and adapt to complex operations in a manufacturing facility (Bhatnagar et al., 1999). It can also be applied in healthcare for medical diagnosis or medical decision making (Alagoz et al., 2010). For this reason it is important to establish the context in which we wish to model the agent-environment interaction as the notion of an agent is commonly confused. We freely

use the term agent or system to denote the decision-maker and the environment or domain to denote the structure in which the agent is embedded and acts upon.

Agent-based decision making models can be modeled with a minimal set of constructs namely: (1) the agent or decision maker, (2) the environment, and (3) goals/rewards. We discuss the details of these constructs in subsequent subsections. For now their intuitive notion will suffice. An agent acting in an environment generally does so to meet certain goals. Interactions of practical interest are typical non-deterministic in that the system dynamics behave stochastically. By *stochastic* we mean that the effect of actions on the state of the environment is uncertain and sensory models of the environment are noisy and incomplete at best, resulting in uncertain state representation of the environment. This is an important characteristic as it serves as the basis of formulating a theoretical framework that models agent-based decision-making. In this work we are interested in investigating and formulating efficient policies for decision-making when the environment is partially observable and the action effects are stochastic. A *policy* is a description of how the agent should behave when it maintains a *belief* of the environment at a decision point.

Sequential decision-making problems can be modeled as an agent-environment interaction where the goal of the agent is to optimize some reward function over time. There are numerous approaches to modeling this problem as presented in Howard (1960), Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998), Hunt (2005), Barto, Bradtke & Singh (1995) and Puterman (1994). The minimal set of constructs required to model the sequential decision-making problem are the agent(s),

environment and rewards. In the following subsection we discuss these components in more detail and explore how various authors have modeled the problem. It is important to emphasize that we are primarily interested in finding optimal or efficient policies that guide the agent's behavior. This dissertation is broken down as follows: chapter 1 introduces a general overview of sequential decision-making for autonomous systems while chapter 2 discusses Markov Decision Processes (MDPs). We proceed to discuss Partially Observable Markov Decision Processes (POMDPs) in chapter 3. In chapter 4 we outline applications of POMDPs for decision-making in autonomous robots while in chapter 5 we present the proposed methodology. We conclude in chapter 6 to discuss lessons learned and the future work we wish to investigate.

1.1.1 Agents

When one thinks of an agent, we my may tend to imagine physical entities such as humans, animals or robots. However, this may not be the case as agents can be highly abstract concepts which may take the form of immaterial constructs such as software programs. Simply put, an agent is a decision-maker that receives sensory information, dynamically updates its beliefs and acts based on its beliefs (Gordon**,** 1999). Its beliefs may be able to perfectly represent the states of the environment in ideal cases. In many decision-making applications the context in which the agent is formalized will indicate how the agent is modeled. This can be quite challenging but taking a synthetic view can provide a more intuitive conception of an agent as discussed in Pfiefer & Scheier (1999) and Pfiefer & Bongard (2007).

At the agent-environment interface, the agent receives perceptual signals *z*. Along with a selected action *a,* it updates its belief *b* which it then uses to decide which action to take in order to optimize some reward function *R* over a definite or indefinite time period. This time period are commonly called *finite* or *infinite* planning horizon which is discussed later. The agent makes observations (or measurements) from a possible set *Z*. If this set of observations *Z* maps identically to a set of states *S*, then the described environment is fully observable. That is, if an observation probability function *O* accurately maps what is observed to the true state, then the state of the environment is known with certainty.

Russell & Norvig (2003) classify agents into five categories namely: (1) simple reflex agents; (2) model-based reflex agents; (3) goal-based agents; (4) utility-based agents; and (5) learning agents. Of these five we are most interested is goal-based agents. In goal-based agents there is a desired state which the agent wishes to achieve. This state generates the maximum reward for the agent. Therefore the agents actively chooses actions that will likely lead to observing the state. The reader may refer to the literature for details on the other types of agents.

1.1.2 Environments

The environment is part of the system that does not include the agent or decision maker. In some cases the agent itself may be considered as part of the environment—such as when multiple agent interactions are being considered. In autonomous robots for instance, the environment could be that domain where the robot has to navigate as well as the physical part of the robot itself. It may seem

counter-intuitive but recall that the agent need not be the physical. The boundary between agent and environment is more of an abstraction as discussed in the previous section. In the robot example the agent could be the computer controller that receives information from the environment including sensors attached to the robot itself (wheel encoders, joint sensors etc).

Sutton & Barto (1998) describes the environment as the *thing* the agent interacts with. It is instructive to restate that the agent-environment model is an abstraction that can be applied to diverse range of problem domains. The environment can be modeled as continuous spaces such as three-dimensional physical spaces or discrete spaces that can be *n*-dimensional such as the states of a manufacturing machine. This description can either take the form of high-level feature or its complementary low-level properties. For instance in robotics vision a high-level description of the state of the environment could be objects located in a room—such as a desk, computer, chair, telephone, lab coat etc. Conversely a low-level description of the state could be the pixilated image data expected from the sensory reading with no direct reference the object observed.

We assume in this work that the environment is *indifferent*. That is, the environment responds passively to the actions of the agent without any intent to subvert the actions effects of the agent. If this were *not* the case then the environment can be modeled as another agent with its own set of rules for decision-making which can possibly be competitive—or worse—combative to the primary agent. Multi-agent systems are common in *Game Theory* and *Markov Games.* If we accept that the environment is indifferent to the actions of the agent, this does not

necessarily imply that the environment does not change. In fact, the environment can be considered a *static* or *dynamic* system. In the static case, the states of the environment don't change and are intrinsically time-invariant systems whereas in the dynamic case the states of the environment may change at slow or rapid rate. If this change is slow enough then we can approximate the environment as a static system. As one might deduce, it is conceptually more challenging to formalize environments that are dynamic.

In machine learning, dynamic environments are often termed *non-stationary* environments whereas static environments are referred to as *stationary* environments. Sugiyama & Kawanabe (2012) provide some the latest theoretical approaches to handling non-stationary domains. However, we restrict this work to approximate stationary environments with stable dynamics or that which can be modeled as such.

1.1.3 Rewards

For any decision-making process to be meaningful it requires motivation or goals. Without motivation, the agent's behavior cannot be effectively evaluated. A way of generating goal-based behavior for an agent is by assigning rewards for the actions that lead to desirable states of the environment. Conversely, an agent may induce negative rewards or costs if it takes actions that do not lead its goals. An example of negative rewards could be the depletion of battery life of an autonomous robot roaming an environment without achieving its goal.

A rewards function is a part of the decision-making process that is distinct from the agent-environment interaction. It provides an objective means to evaluate the agent's performance (Littman, 1996). It is essentially the designer's task to model the reward function for which on occasion is an art rather than a science. From a theoretical perspective it is always desired to maximize the reward function while minimizing agent resources. As such, most agent-based decision-making problems are modeled algorithmically where the designed algorithm maximizes a reward function with minimal computational and execution time.

1.1.4 Policies

A *policy* is a full description of how the agent should behave in every state of the environment. Policies are different from *plans* in that plans specify how to behave in only a subset of states in state space. We discuss policies in the context of *Markov Decision Processes* in chapter 2. For now we should note that the purpose of a policy is for goal directed behavior of the agent. Our objective is to find an optimal policy for the agent.

Policy can be *stationary* or *non-stationary*. Stationary policies have a fixed state-action sequence for every decision or time step whereas in non-stationary policies the decision made in a time step may taken from a set of different policies. Consider this illustrative example: An environment has 3 states (s_1, s_2, s_3) and the agent has 2 actions ($a_{1,} a_2$) to choose from. If there are three decision (time) steps in which the agent has to make its decision, then a sample stationary policy for each of the decision step t_1, t_2, t_3 is:

$$\{[(s_1,a_2),(s_2,a_1),(s_3,a_1)]_{t=1},[(s_1,a_2),(s_2,a_1),(s_3,a_1)]_{t=2},[(s_1,a_2),(s_2,a_1),(s_3,a_1)]_{t=3}\}.$$

This policy states that for every decision step if the agent is in state s_1 it chooses action a_2; state s_2 the choice is action a_1; and in state s_3 the selected action is a_1. In a non-stationary policy such as:

$$\{[(s_1,a_2),(s_2,a_1),(s_3,a_1)]_{t=1},[(s_1,a_1),(s_2,a_2),(s_3,a_1)]_{t=2},[(s_1,a_2),(s_2,a_2),(s_3,a_2)]_{t=3}\},$$

if the agent is in state s_1 at decision step $t=2$ rather than use the same policy and choose action a_2, it chooses an action based on a different policy and selects action a_1. The policy differences are highlighted in red.

1.2 Decision-Making for Autonomous Robots

We have briefly discussed the essential constituents of agent-based decision making. Our goal in this work is to apply it by formulating an approach to facilitate solving open problems in autonomous robot navigation in partially observable, uncertain and potentially dynamic environments. This is a challenging problem that has received considerable research attention. The approach is to model the decision making problem as a *Partially Observable Markov Decision Processes (POMDPs)* and to compute practical efficient policies to control the robot motion during a given task.

1.2.1 Robot-Environment Model

The robot-environment dynamics can be modeled as a coupled dynamical system as shown in *figure 1.1*. The robot observes the state of the environment as

well as its internal state. It then maintains a belief according to some rule and acts on the belief to achieve a goal. The environment and robot are always in a certain *state*. A *complete* description of state variables is a complex and impracticable task as physical environments are continuous both spatially and temporally. The continuous nature of the real-world ensure that it is impossible to have perfect models of the environment as there are infinite set of possibilities. Most practical approaches simply use a suitable approximation. The environment is also highly contextual as the descriptions in say, a factory environment will significantly differ from descriptions in an outdoor environment. However, for practical purposes we consider the state to be the set of all structures of the robot-environment system that can be modeled in the appropriate operating context. In this case continuous states are discretized in addition to other discrete state variables such as the binary operation of a bumper sensor.

Continuous states variables may include the location of the robot with respect to some global reference frame, it's heading or direction, both rigid and non-rigid physical structures in the environment, the robot's present joint angle readings etc. As stated the possibilities are infinite and its representation is largely an art which depends on the designer's ability to abstract the features of the environment.

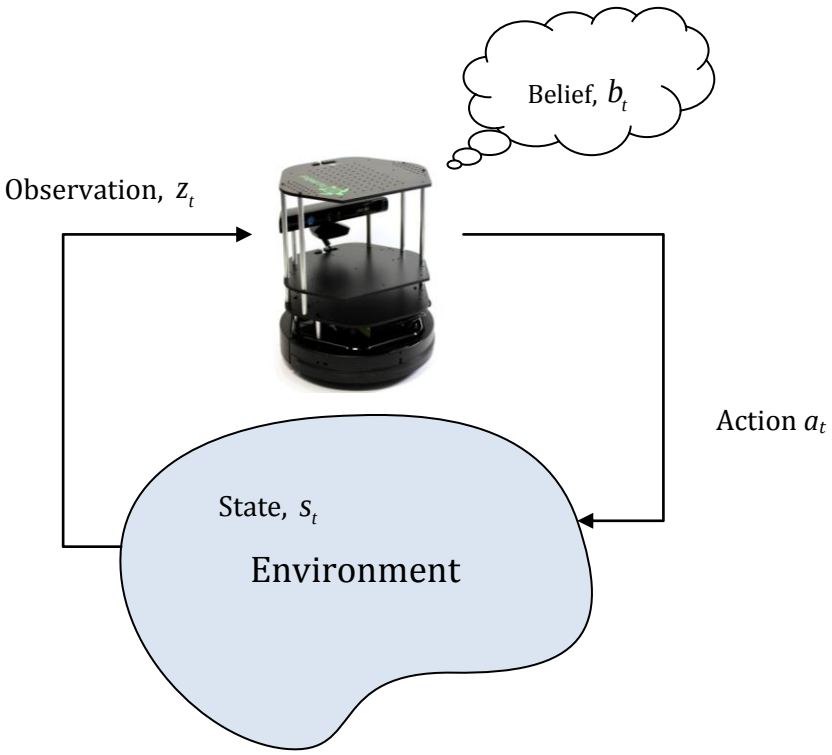


Figure 1.1 – Robot-Environment Model

The state variables of the system can be considered static or dynamic (Thrun, Burgard & Fox, 2006). In the environment, static state variables are the locations of rigid fixed objects and relevant features which may also be modeled as landmarks for navigation purposes. The location and orientation of the robot are considered static state variables. There are three variables for the location which represent the (X, Y, Z) 3-dimensional Cartesian space and three variables for the orientation namely the (*pitch, roll, yaw*). Conversely, objects in motion in the environment are considered dynamic state variables. Also the robot velocities as well as the joint and actuator velocities are considered to be the robot's dynamic state variables.

1.2.2 Sensory Observations

One of the primary ways the robot interacts with the environment is by making observations of the current state of the environment. This is achieved by using sensors such as laser range finder, cameras, ultrasonic sensor etc. to take measurements. The sensory data is used by the robot to maintain state estimates which are then used by the controller to choose appropriate actions. Typically, numerous observations can be made simultaneously and processed in parallel but this approach has its own challenges and drawbacks. In this work we consider only sequential processes where the observation data is acquired in discrete time steps.

1.2.3 Acting in the Environment

The objective of designing any mobile robot controller is for it to choose appropriate actions based on sensory information the robot receives from its environment. Classical approach to planning and navigation were idealized and assumed that the robot had perfect knowledge of the state of the environment. Its actions were fully deterministic (Nilsson, 1973; Shue and Xue, 1993). This approach is only sufficient for a very small subset of interesting real world problems where uncertainty can be ignored. However since uncertainty cannot be ignored in most real world problems and the robot-environment interaction is a coupled dynamical system we must account for the stochastic nature of this interaction. We are interested in applying decision-making models for autonomous robot navigation that models the uncertainty in action selection and perception. Therefore unmodeled stochastic interactions will result in poor performance of the controller.

For this reason the action effects are modeled using *Markovian* decision-making processes discussed in more detail in chapter 2 and its generalization discussed in chapter 3. At every time step the robot takes an action based on its *belief* resulting in a probabilistic transition to a new state and thus a new belief after observation are taken. Details of this process will be further elucidated in subsequent chapters. Nevertheless, a belief can be understood as a model of the environment with respect to the *true* state of the environment maintained by the robot.

1.2.4 Planning in Partially Observable Environments

We briefly mentioned that it was impractical to describe and list the complete state variables of the environment. Due to this only a contextual subset of relevant state variables are used to model the robot environment-interaction. The next issue is for the robot to the decide how to act in order to achieve its designed goals. Since we are interested in the navigation problem, the goal for the robot is thus to select appropriate control actions to effectively move from an initial location to a target location autonomously while avoiding obstacle. This is done by generating *control policies* or *plans* that describe the actions the robot should take at every state of the environment. A policy in this case is a complete specification of the actions the robot should take at every belief state in contrast to plans that only consider a subset of belief states from belief space.

Policies are implemented over a time horizon. Though time is naturally a continuous variable, for practical purposes we model it as a discrete variable of equally spaced intervals. Actions of the robot, which can also be referred as controls

or decisions, are taken sequentially. The robot starts off in some state s_0 at time t_0 but cannot directly observe this state. Thus it has to generate an initial belief b_0 of the starting state. A belief b can be modeled as a probability distribution over all the states. The robot then takes action a_1 and probabilistically transitions to a subsequent state s' at time t_1. It generates a posterior belief b' of the subsequent state by probabilistically observing z from a set of possible observations which are noisy projections of the true state of the system. Hence the updated belief is a posterior probability conditioned on the subsequent state, observation and action. It is sometimes referred to as the *state estimate*.

Thus the problem of planning in partially observable environments is to find optimal (or near-optimal) decision policies that the robot can execute during its navigation task. The planning algorithms discussed in this work produce action policies that are computed offline. Online algorithms such those considered in Ross et al. (2008) are not within the scope of this research. They however provide very interesting methods for solving the hard problems of planning in partially observable domains. Also extensive literature on other planning algorithms and techniques can be found in LaValle (2006).

1.2.5 The Autonomous Robot Navigation Problem

Simply stated, the autonomous navigation problem is as follows: Given a map of an environment find a collision free path that a mobile robot can efficiently navigate given an initial location and goal location with minimal external intervention while avoiding obstacles. This navigation can be applied in 2-dimensional environments

where the variables of interest are (x, y, θ). Here (x, y) is the planner location of the robot depicted as a point in 2D Euclidean space and θ the heading. Navigation in 3-dimensional environments are naturally more challenging to model and have received more research attention (Hornung et al., 2012).

To solve the navigation problem, path planning techniques such as those discussed in LaValle (2006) can be employed. Path planning involves finding a path or trajectory in the *physical* or *configuration space* that the robot should traverse to reach its destination location efficiently. Configuration space is the space of all possible representations of the robot's configuration. There are numerous path planning algorithms that have been proposed to solve the navigation problem. These plans generally fall into two categories: 1) Graph Search and 2) Potential Field Planning (Siegwart, Nourbakhsh & Scaramuzza, 2011). In graph search, the environment is represented as free space and occupied space then subsequently decomposed in some type of graph where various algorithms can be applied to calculating a collision free path. The most common types of graph search techniques are the Voronoi Diagram (Aurenhammer, 1991), Visibility Graphs, Exact and Approximate Cell Decomposition. Also A*, D* algorithms and Rapidly Exploring Random Trees (RRTs) are also popular methods used for path planning. Potential field planning in contrast plans over a gradient field map of the environment. The goal location is modeled as an attractive force while the obstacles are modeled as repulsive forces. The objective is for the attractive forces to "pull" the robot towards the goal location while the obstacle "pushes" the robot away from its location.

As will be extensively discussed Markovian decision processes and its partially observable variant have been applied in early literature to tackle the robot navigation problem due to the need to better handle uncertainty (Cassandra, 1996).

1.3 Related Work

Sequential decision-making in stochastic partially observable environments has received significant research attention both theoretically and practically. We are focusing our attention in applications for autonomous robotics—specifically in the area of motion control and planning. The primary issues in these areas are to formulate optimal plans or policies for efficient execution by the robot with minimal external intervention. The robot has to select appropriate actions that result in robust goal-directed behavior.

We tackle this problem by focusing on domain-specific model parameters of the POMDP formalism and implement it as a controller of the robot's actions. A closely related work is that of Kaplow, Atrash & Pineau (2010). One significant challenge and disadvantage of implementing POMDPs is that they are computationally intractable for problems with large state space so it suffices that most methods focus on its approximation. Also planning horizons and problem structure affects computability. Recently diverse algorithms using point-based methods have been proposed to solve POMDPs with large state space such as in Smith & Simmons (2004); Spaan & Vlassis (2005); Kurniawati, Hsu & Lee (2008); Poupart, Kim & Kim (2011). We use a state-of-the-art point-based solver to solve our problem.

1.3.1 POMDP in Robotics

Most applications of POMDPs in robotics are in navigation for mobile robots. However, some application in manipulation tasks does exist. Research for POMDPs in robotics originates from difficulties with hierarchical planners and behavior-based models in handling uncertainty. Probabilistic Robotics (Thrun, Burgard & Fox 2006)—a relatively new robotics paradigm—focuses on using probability theory to model the robot's behavior in the real world.

Early work by Cassandra (1996) use heuristics to solve the POMDP model for robot navigation while Koenig & Simmons (1998) propose a robot navigation architecture called *Xavier*. The robot in the latter performs delivery tasks in office environments and demonstrated reliability in uncertain sensory data and action effects. Pineau & Thrun (2002) introduced the application of a POMDP controller with uncertain sensory information for high-level control of the robot's behavior. Their model was applied in an assisted care facility. Also, Lopez et al. (2007) apply POMDP in assistant robots for robust navigation in a hospital environment by representing it as topological structures. They show their model to be robust to dynamic objects such as people in motion giving more credence to its potential application.

Other research efforts focus more on theoretical formulations that approximate solutions of the POMDP model. Roy, Gordon and Thrun (2005) apply *Exponential Principal Component Analysis* (EPCA) to reduce the belief space in order improve speed of computation. Pineau & Gordon (2005) also present a POMDP algorithm called PEMA and apply it in a nursebot. Their algorithm selects salient states of the

environment to improve its scalability. Ong et al. (2009) present a method using mixed observability to solve POMDPs. They suggest that parts of the environment may be fully observable thus reducing the dimensionality of the belief space. They also demonstrate that their algorithm significantly improved in performance over Pineau, Gordon & Thrun (2003).

Recent research in POMDPs for robot decision-making and navigation has witnessed positive results. That notwithstanding there is still significant progress to be made. Candido & Hutchinson (2011) propose a method to the solve navigation problem using Continuous POMDPs to find policies for minimizing collisions and successfully reaching its desired location in minimal time. Kaplow, Atrash & Pineau (2010) present a framework for decomposing the environment into non-uniform grids or variable resolution to exploit the facts that nearby states without obstacles can be merged in larger states thus reducing the computational demands of solving the POMDP. Ong, Png, Hsu and Lee (2010) discuss their approach of robot motion planning and navigation in uncertain environments by proposing a factored representation to model the fact that certain aspects of the environment may be fully observable as well as partially observable. They derive a reduced representation of the belief space of the POMDP.

These approaches and numerous others provided valuable contributions to decision-making and planning for mobile robotic systems. We complement these approaches by contributing a practical method to reducing the planning state space for the robot in order to achieve the desired tasks.

1.4 Contributions

Our main contribution in this dissertation is to present a methodology called *Goal-Specific Representation* (GSR) that exploits the intrinsic contextual properties of a task environment. Given a map of the domain decomposed into uniform cells, an initial location and a goal location, the proposed method selects only a subset of states relevant for task completion based on the domains structural properties and ignores other states. We show that there is not a significant difference in task completion between a complete state representation and a GSR state representation. The result is that the methodology can be applied to very complex domains were planning is only conducted over a precise region of the task domain.

We apply this to the robot navigation problem then proceed to formulate action policies using a POMDP model parameterized by a GSR algorithm. We solve the POMDP model using a point-based POMDP solver and perform empirical evaluation accordingly. The decision policies are tested on a real robot in an indoor environment that simulated an automated manufacturing facility layout. Our primary goal of this work is to advance research in decision-making for autonomous systems and formulate methods for autonomous robot navigation that can be utilize in applications such as automated material handling in a manufacturing domain or assisted care in healthcare domains.

Chapter 2

Markov Decision Processes

2.1 Overview

A Markov Decision Process (MDP) is a framework for decision-making under uncertainty that follows the *Markov property*. By Markov property, we mean a *memoryless stochastic process* where future states of the system depend only on the current state and not the history of previous states of the system. A stochastic process is a process whose time-based evolution has probabilistic elements, thus the sequence of states of the stochastic process is non-deterministic where the state transition is guided by a probability distribution over all states.

In MDPs the outcome of an action is partly under the control of the decision maker and partly random. The objective is to find the best policy to guide the decision-maker's actions. They are solved using *dynamic programming* or *linear programming*. Dynamic programming algorithms developed by Bellman (1957a, 1957b) and its variants are the preferred in literature. A detailed treatment of Markov Decision Processes can be found in Bellman (1957b) and Howard (1960), Puterman (1994). Bertsekas and Shreve (1996) also provide a mathematically rigorous analysis of solving discrete-time stochastic processes using dynamic programming.

2.2 The Markov Process

Let ***S*** be a sequence of random state variables $S_1, S_2, ..., S_t$ and let there be only one action available at each state. If $p(s_{t+1}|s_1, s_2, ..., s_t) = p(s_{t+1}|s_t)$ then the sequence is said to have the Markov property where the next future state variable of the sequence, s_{t+1}, only depends on the current state variable s_t and not the previous state variables $s_1, s_2, ..., s_{t-1}$. Any stochastic process that has the Markov property is called a *Markov process*. This process is also described as memoryless. In agent-based decision-making however, there are usually multiple actions available for the agent at any given state. Thus the Markov property can be written as:

$$p\{s_{t+1} = s' | s_t = s, a_t = a, s_{t-1}, a_{t-1}, ..., s_0, a_0\} = p\{s_{t+1} = s' | s_t = s, a_t = a\}. \qquad (2.1)$$

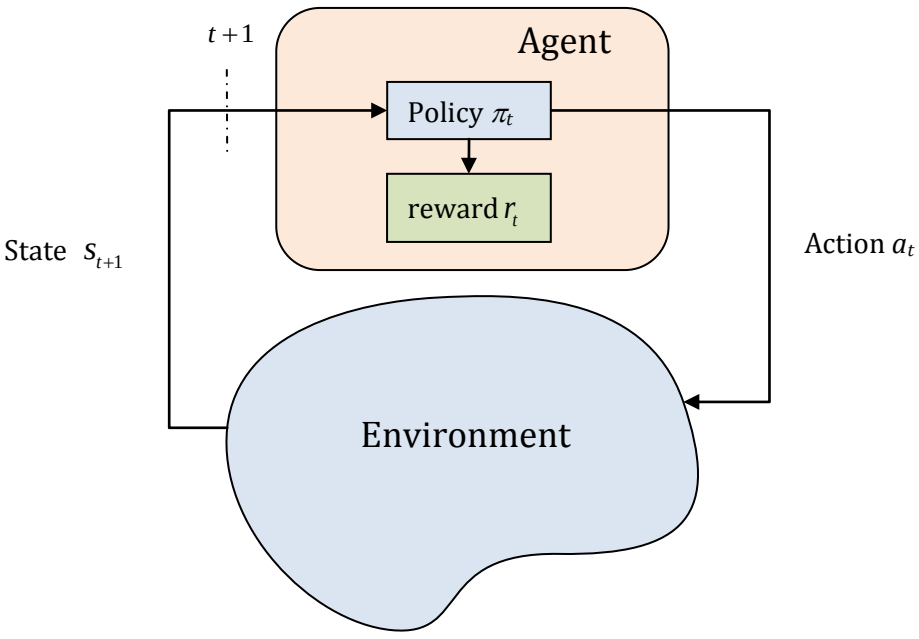


Figure 2.1 – An MDP Model of the Agent-Environment Interaction

This property is of significance because the behavior of the system modeled as an MDP may deviate from expectation if all the previous states and actions have to be accounted for in making our current decision. Also conditioning our present decision on all past decision will quickly result in computational intractability as the number of states and actions sequences increases exponentially resulting in the *curse of history*.

2.3 The MDP Model

When an agent or mobile robot operating in an environment has the ability to directly observe the state of the environment and perform actions to change the state of the environment, this behavior can be modeled as an MDP if the state space and action space are well-defined. An abstract model is shown in *figure 2.1* and *2.2*. In practice however, agents cannot directly observe the state of the environment and can only infer it from sensory information. This is called *partial observability* and we discuss this generalization in chapter 3. In this section we assume that the state of the environment is directly observable in order to develop an appropriate MDP model.

An MDP can be modeled as a set of states, a set of actions, a state transition probability function and a state transition reward function. It is the tuple $\langle S, A, T, R \rangle$ where:

- **States***:* $s \in S$ is a discrete and finite set of states of the environment, $S = \{s_1, s_2, ..., s_n\}$. The environment is assumed to always be in some state s at time t and initialized at state s_0.

- **Actions:** $a \in A$ is a discrete and finite set of actions: $A = \{a_1, a_2, ..., a_m\}$. This describes the actions the agent takes that can potentially alter the state of the environment.

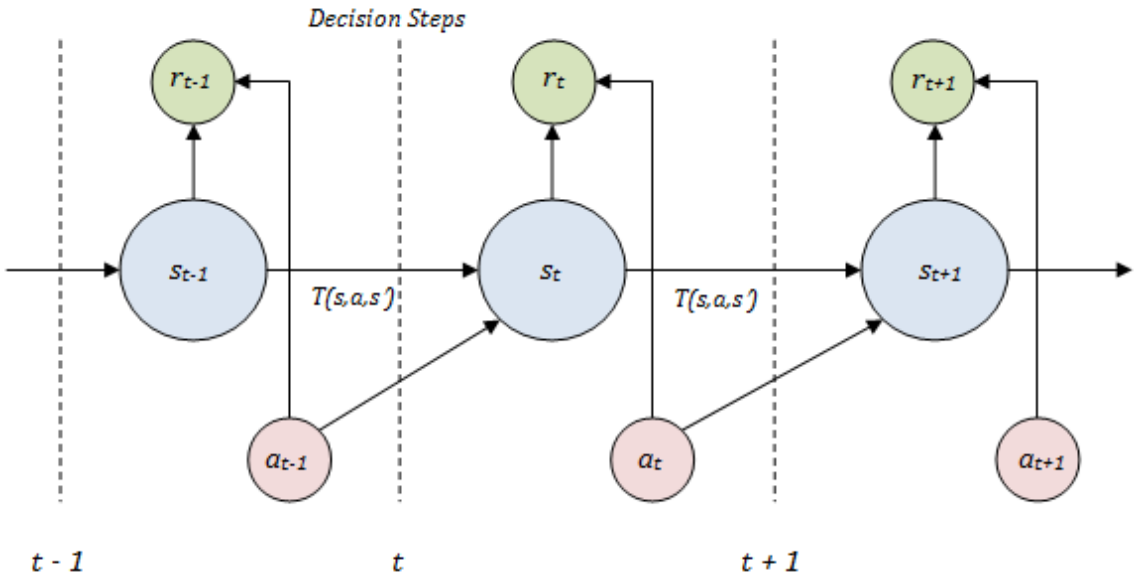


Figure 2.2 – An Abstract Representation of an MDP model

- **State transition probability function:** $T : S \times A \times S \to \mathbb{R}$ is the function that assigns a value in the interval [0,1] to the triplet (s, a, s'). It can also be described as $T(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$ which is the set of transition probabilities from state s to s' over time t to $t+1$ under action a. Furthermore the sum $\sum_{s' \in S} T(s, a, s') = 1, \forall (s, a).$

- **Reward Function:** $R : S \times A \to \mathbb{R}$ is the function that assigns a real number for every state-action pair. We define $R(s, a)$ as the set of *expected* immediate rewards the agent receives from selecting action a in state s. The reward function can also be defined as $r : S \times A \times S \to \mathbb{R}$ which is the state-action-state

triple that assigns an immediate reward for every subsequent state in state space given the current state and action. However it is usually convenient to represent the reward function in terms of the expected immediate reward $R(s,a)$. We express the relationship between the expected immediate and the immediate reward as:

$$R(s,a) = \sum_{s' \in S} T(s,a,s') r(s,a,s'). \tag{2.2}$$

In summary, an MDP model is define by the tuple $\langle S, A, T, R \rangle$ where S is the set of states, A is the set of actions, T is the transition probability function and R the reward function. The process starts at time t = 0 in state $s_0 \in S$ and takes action $a \in A$ specified by some arbitrary policy or decision rule. It transitions to the next state according to $T(s,a,s')$ and receives an expected reward $R(s,a)$. We discuss the objective of the model and how it is solved in the next sections.

2.3.1 Objective of an MDP

The objective of an MDP is to find a *policy*—a complete mapping of state to action—that maximizes the cumulative sum of the expected *discounted* reward R over a finite or infinite time horizon. A policy can be *stationary* or *non-stationary*. A stationary policy $\pi : S \to A$ is a fixed policy independent of the decision step. The action taken under the policy π is given by $a_s = \pi(s)$. This means that the policy is dependent only on the state of the system. In contrast, if a policy specifies an action to take at a certain state that is dependent on the decision or time step of selection

then the policy is non-stationary as the action selected at time *t* is taken from the policy set $\pi = \{\pi_1, \pi_2, ..., \pi_T\}$ where π_i is the policy selected at time i for $i = 1, 2, ..., T$.

For *finite-horizon* problems the reward function for a given policy π is denoted as:

$$R_\pi = E\left[\sum_{t=1}^{T} \gamma^t r_t\right], \tag{2.3}$$

where *T* is the horizon length, *t* is the time step and $0 \le \gamma \le 1$ is the *discount factor* which allows us to model the fact that future rewards are potentially less beneficial than present rewards and are thus geometrically discounted over the time horizon. If $\gamma = 0$ then the rewards are not considered whereas if the γ = 1 then we assign equal value of future rewards to the present reward. For finite horizon models we can sometimes assign $\gamma = 1$ since the problem is solvable within finite time.

There are challenges in solving finite-horizon problems optimally since it is typically not known when the decisionprocess will terminate. To overcome this difficulty *infinite-horizon* models with a discount factor are preferred. The cumulative expected reward for a given policy π is given by:

$$R_\pi = E\left[\sum_{t=1}^{\infty} \gamma^t r_t\right], \tag{2.4}$$

where $0 \le \gamma < 1$ is the discount factor. Note that for infinite horizon problems $\gamma \ne 1$ or else R_π will have infinite value. The policy that maximizes the expected cumulative discounted reward *R* is denoted as π^* and is given by:

$$R_{\pi^*} = \max_\pi E\left[\sum_{t=1}^{\infty} \gamma^t r_t \mid \pi\right]. \tag{2.5}$$

MDPs can be modeled as discrete-time or continuous-time processes. Discrete-time MDPs are used when actions are made in discrete time intervals whereas in continuous-time MDPs, actions are executed at any point in time. Puterman (1994) and Guo & Hernández-Lerma (2009) extensively discuss continuous-time MDPs. Continuous-time models pose its unique challenges in finding optimal solutions and are not within the scope of this work. We are primarily interested in discrete-time over discounted finite or infinite horizons. There are other interesting MDP formulations such as Decentralized MDPs formulated by Allen, Petrik & Zilberstein (2008).

2.3.2 MDP Solution Approach

A naïve approach to finding a solution for an MDP is to select the action that results in largest expected immediate reward for a given state. This might seem rather intuitive but since the decision process has probabilistic transitions and a planning horizon, choosing an action with a lower immediate reward may result in larger long term cumulative reward in the future. This adds to the complexity of computing optimal policies for the MDP. Thus, as equations (2.3) and (2.4) suggests it is preferable to compute the cumulative reward over the desired time horizon rather than simply considering the expected immediate reward. The notion of value or utility of a state that measures the expected cumulative reward from any decision step to the terminal step from that state is a consistent method to compute the optimal policy.

Howard (1960), Puterman (1994) and Bertsekas (2007) provide extensive theoretical analysis of methods and algorithms for solving MDPs. We discuss established algorithms for solving MDPs such as *value iteration* and *policy iteration*. A variant of the value iteration algorithm will be used to solve the generalized case of MDPs where the states are partially observable. We shall apply solutions to the robot motion planning and navigation.

2.3.3 Policies

We briefly discussed policies in section 2.3.1. As stated a policy, $\pi : S \to A$ is a mapping of state to action for every state in the state space. Consider a decision process with five states $\{s_1, ..., s_5\}$, three actions $\{a_1, a_2, a_3\}$ and finite time horizon of $T = 10$ time steps. For this case we have a total of $|A|^{|S|} = 3^5 = 243$ possible policies. This is the number of possible action combinations for every state. We may also be interested in the histories of the process. A *history* is the sequence of state-action pairs selected during the process. In this example there are $\left(|S| \cdot |A|\right)^{T-1} = (5 \cdot 3)^9 \approx 3.8443 \times 10^{10}$ possible state-action sequences for the 10 time steps. Note that the last decision step is the terminal step and is not included in the possible histories of the process.

Policies can be *deterministic* or *probabilistic*. In a deterministic policy the action selected at each state is fully determined whereas in probabilistic policies, the action selected for each state of the process come from a probability distribution over the all actions. In this work we only consider deterministic policies. That is, policies

that the agent must chose to during its decision-making process. The decision process can also be considered as a control problem. The difference being with the time scales under consideration. Policies could also be stationary or non-stationary. We also mentioned that a stationary policy is that which applies the same policy at every decision step while non-stationary policies select from a set of policies at various decision steps. The type of policy executed is important for selecting optimal decisions because when the time horizon is finite it may not be satisfactory to select from the same policy at the last decision point since this may not result in optimal agent behavior.

2.3.4 Decision Steps

An agent acting in an environment must make decisions at some point in time. Even the act of doing nothing may be considered a decision or action. Decisions are made in time steps which are sometimes called decision steps, decision periods or action steps. They are modeled as a discrete or continuous sets K of positive real numbers. K may either be finite or infinite. For the discrete finite horizon case K = {1, 2, 3, … , T} where $T < \infty$. That is, T is a set of finite positive integers whereas for the discrete infinite horizon K = {1, 2, 3,…}, the set of all positive integers. If the decision steps were continuous and the time horizon is finite then $K = [0, T]$ is the interval of positive real numbers where decisions can be made at any point in time.

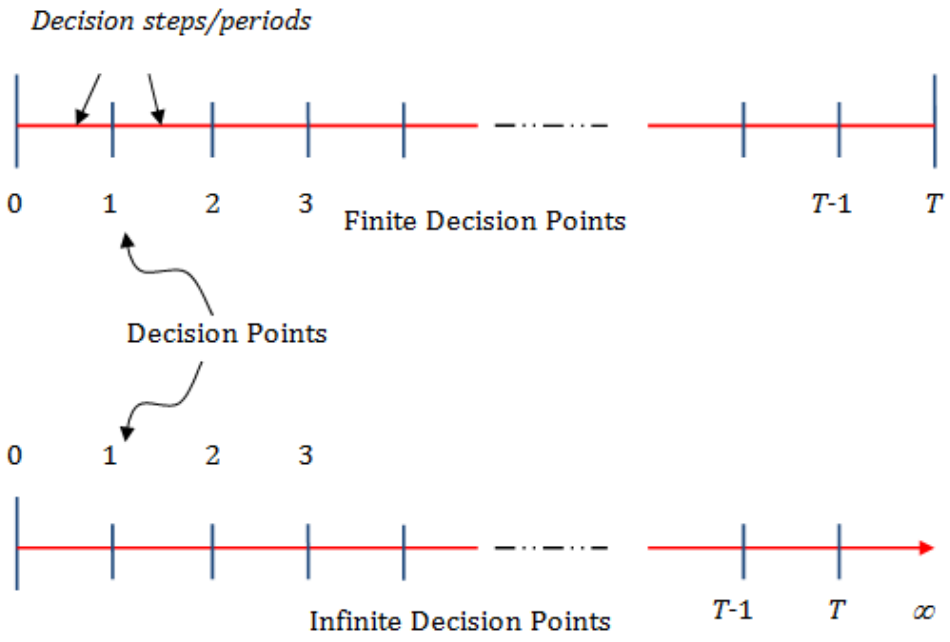


Figure 2.3 – Timeline for Discrete Decision Points

Note that there are an infinite number of possible decision points in a finite interval. For continuous and infinite time horizons the decisions are made in the interval $K = [0, \infty)$. *Figure* 2.3 illustrates the division of the time line into discrete decision steps or periods and decision points. As noted, decisions are made at each decision point except the last point *T*.

The mathematical theory and practical applications of continuous time problems still pose significant challenges. We restrict our focus to discrete time horizons where decisions are made sequentially at equally spaced intervals. We also assume the last action is taken in time step $T-1$. By convention we adopt *T* to denote the length of the time horizon and *t,* the t^{th} decision step of the process. We replace the horizon length and decision step with *N* and *n*, respectively when discussing the problem in terms of the backup update using a dynamic programming algorithm.

2.4 Algorithms for Solving MDPs

In this section we discuss the *Value Iteration* and *Policy Iteration* algorithms for solving MDPs. Other methods such as linear programming exist for computing optimal policies. However we focus more on the value iteration algorithm as this is the most widely used in research and practice. Some other recent interesting algorithms and techniques for solving variants of the MDP model are discussed in Guo & Zhu (2002) and Melo & Veloso (2011). Before we present the value iteration algorithm, we define the value function and how it can be used to compute optimal policies.

2.4.1 Value Functions

When an agent acts in an environment its motivation is modeled as a reward function. The reward function assigns a real value to the actions the agent performs for a given state. Since policies are a complete description of the agent's behavior for a given time or decision horizon we need a way to evaluate how desirable a policy would be if implemented. This is done by calculating the value as a function of state for a given policy. The value function at a time step t is the cumulative expected reward of starting at state s and executing the policy π for the remaining $T-t$ decision steps of the time horizon. If the time horizon were finite then the value function is computable. However, if the time horizon were infinite then the value function will have infinite value and also poses theoretical issues when attempting to compute it using dynamic programming methods. In order to mitigate this we add a discount factor $\gamma = [0,1)$ allowing for a unique solution. Also

for algorithmic reasons a stopping criteria $\|V_{t+1} - V_t\| < \delta$ is included where δ is some arbitrary small value.

We solve the MDP by computing an optimal value function. There can be multiple policies associated with the optimal value function. The agent simply selects an optimal policy arbitrarily. If the agent is a mobile robot and the policies are utilized for navigation then optimal policies associated with an optimal value function is selected based on the structure of the environments maps. This is discussed more extensively in subsequent chapters. For now, we present the value function for finite and infinite time horizons and discuss how to use them to solve MDPs using value iteration algorithm. White (1993) and Puterman (1994) discuss in greater mathematical detail these approaches along with proofs.

2.4.2 Value Function for Finite Horizon Models

A decision process has a finite horizon when there is a well-defined number of a decision steps that the process has to run. We consider these time steps to be discrete and the number of time steps the process runs is represented by *T*. As noted, optimal policies for finite horizon models are typically non-stationary since the policy selected at the early stages of the process may differ from the later stages in order to have optimal behavior. Thus for a given policy the long-term value the agent accumulates for a finite horizon is denoted by $V_{\pi,n}(s)$ which is the expected sum of rewards accumulated from starting in state *s* and implementing the non-stationary policy π for *n* decision steps. As mentioned, we avoid ambiguity between policy execution and the dynamic programming backup operation by using *n* to

represent the number of decision *step-to-go*. The notation *n* relates to *t* through the equation $n = T - t$. This value function is given by:

$$V_{\pi,n}(s) = R(s, a_n) + \gamma \sum_{s' \in S} T(s, a_n, s') V_{\pi,n-1}(s'). \tag{2.6}$$

This is the sum of the immediate expected reward for selecting action *a* with *n*-steps to go in state *s* and the expected discounted value for the $n-1$ remaining decision steps. Notice that the $n-1$ value function is multiplied by the transition function $T(s, a, s')$. The value function for the last step when $n = 1$ in the finite horizon problem is given by:

$$V_{\pi,1}(s) = R(s, a_1). \tag{2.7}$$

The value of the last step in the decision process is simply the expected reward of selecting action *a* at time step $n = 1$ in state *s* and we can safely assume that $V_{\pi,0}(s') = 0$. This implies that there is no decision made at the last decision point. Equation (2.5) can be solved via dynamic programming using the *principle of optimality* (Bellman, 1957b).

2.4.3 Value Function for Infinite Horizon Models

When the decision-making process does not have a time limit it is described using infinite-horizon models. At first glance the reader may observe that infinite time will result in infinite rewards the agent accumulates. However since the discount factor is $0 \leq \gamma < 1$, convergence and thus an optimal value function is guaranteed. The value function for infinite-horizon models is given by:

$$V_\pi(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s'). \tag{2.8}$$

Equation (2.7) is recursive and the decision step *n* is omitted since the horizon is infinite. Also notice that equation (2.7) results in a value function, V_π, whose values for each state can be computed from the set of |*S*| simultaneous equations with |*S*| unknowns variables where the unknowns variables are $V_\pi(s)$.

2.4.4 Optimal Value Functions and Policies

The purpose of solving an MDP is to find an optimal policy (a mapping of states to actions) that guides the agent's behavior. Infinite-horizon discounted models are more convenient to solve since the desired finite-horizon length is rarely known in practice. Details on deriving optimal finite-horizon value functions and policies are discussed in White (1993), Puterman (1994) and Kaelbling, Littman & Cassandra (1998). However we shall apply the finite-horizon equations for use in solving finite-horizon problems with partial observability.

Given a value function a *greedy policy* is given by:

$$\pi_V(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s') \right]. \tag{2.9}$$

A greedy policy is one that selects an action in every state that maximizes the sum of the immediate expected reward and the expected discounted value of the subsequent states. For finite-horizon models the optimal policy are potentially non-stationary, thus the optimal policy for the *n*-th decision step, π_n^*, is computed in

terms of the immediate expected reward and optimal value function of the $n-1$ step. The equation is given by:

$$\pi_n^*(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{n-1}^*(s') \right]. \tag{2.10}$$

The optimal *n*-th step value function is denoted by the following equation:

$$V_n^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{n-1}^*(s') \right]. \tag{2.11}$$

For the infinite-horizon model, the optimal value function can be found by executing a stationary optimal policy π^*. This optimal value function is given by the equation:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right]. \tag{2.12}$$

This is similar to the optimal value function for finite-horizons but the time steps have been omitted. Since it is not practical to have infinite time it is often convenient to compute the optimal value function using an approximate finite-horizon model whose time horizon approaches infinity. Thus we may define an arbitrarily small value as the stopping point between two successive backup computations of the set of equations of the value function *V*. The optimal policy for the infinite-horizon model is the equation:

$$\pi^*(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right]. \tag{2.13}$$

In the next section we discuss the most common method used for solving MDPs namely the value iteration algorithm. In the general case of Partially Observable MDPs we apply a variant of the value iteration algorithm.

2.4.5 Value Iteration Algorithm

The Value Iteration algorithm shown in *table 2.1* is the most commonly used algorithm to solve MDP. Its variant will be used to solve POMDPs and their approximations. Simply put, the value iteration algorithm is a recursive algorithm that calculates the value function for every action and finds the action that maximizes the value function. It terminates when the difference of two subsequent value functions is less than a small error value. This algorithm is derived from dynamic programming as described by Bellman (1957b).

The value iteration algorithm above terminates when $|V_n(s) - V_{n-1}(s)|$ is less than the error value ε known as the Bellman error magnitude. Puterman (1994) also shows that the difference between $V^*(s)$ and $V_n(s)$ of the optimal policy does not exceed $2\varepsilon\gamma/(1-\gamma)$ at any state of the process. Consequently we shall be more concerned with ε-optimal value functions such that $V^* : \max_s |V_n(s) - V^*(s)| \leq \varepsilon$. Most times the optimal policy π^* is found early in the iteration process. That is, $\pi = \pi^*$ before V_n approaches V^*.

2.4.6 Policy Iteration Algorithm

Another popular algorithm used in solving MDPs is the *Policy Iteration* algorithm. This algorithm is shown in *table 2.2* and finds the optimal policy by computing a sequence of policies that are monotonically improving in value. The sequence of policies thus converges to an optimal policy. If π is not optimal, then there's a policy π' with action *a* in state *s* such that $Q_\pi^a(s) < Q_{\pi'}^a(s)$.

Table 2.1 – Value Iteration Algorithm

Value Iteration Algorithm

Input $(S, A, R(s,a), T(s,a,s'))$

1. $n = 1$ (Initialize the decision step)
2. $V_1(s) \leftarrow 0, \forall s \in S$ (Initialize value for all states to zero)
3. **repeat loop**
4. $n \leftarrow n+1$
5. **for all** $s \in S$
6. **for all** $a \in A$
7. $Q_n^a(s) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{n-1}(s')$
8. **end for**
9. $V_n(s) \leftarrow \max_a Q_n^a(s)$
10. **end for**
11. **until** $\left| V_n(s) - V_{n-1}(s) \right| < \varepsilon \quad \forall s \in S$

Output: optimal policy, $\pi(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_n(s') \right]$

We first have to find the solution to the value function V_π for an arbitrary policy π by solving the $|S|$ simultaneous linear equations:

$$\begin{aligned} &c_{11}V_\pi(s_1) + c_{12}V_\pi(s_2) + \ldots + c_{1k}V_\pi(s_k) + b_1 = 0 \\ &c_{21}V_\pi(s_1) + c_{22}V_\pi(s_2) + \ldots + c_{2k}V_\pi(s_k) + b_2 = 0 \\ &\vdots \\ &c_{k1}V_\pi(s_1) + c_{k2}V_\pi(s_2) + \ldots + c_{kk}V_\pi(s_k) + b_k = 0, \end{aligned} \tag{2.14}$$

where $k = |S|$ is the total number of states and c_{ij} for $i, j = 1, 2, \ldots, k$ are the coefficients of the linear equations that simply represents the product of the immediate reward and the transition function parameters. After that, for every

state *s* and action *a* we compare $V_\pi(s)$ with $Q_\pi^a(s)$ and if there is any *a* such that $V_\pi(s) < Q_\pi^a(s)$ we set $\pi(s)$ which is the current action under to policy π to the new action *a*. This process is repeated until the policy cannot be improved upon again.

Table 2.2 – Policy Iteration Algorithm

Policy Iteration Algorithm

Input $(S, A, R(s,a), T(s,a,s'))$

1. $\pi' \leftarrow$ an arbitrary policy
2. **repeat loop**
3. $\pi \leftarrow \pi'$
4. $V_\pi \leftarrow$ solution to policy π using simultaneous linear equations
5. **for all** $s \in S$
6. **for all** $a \in A$
7. $Q^a(s) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')$
8. $Q(s) = \max_a Q^a(s)$
9. **if** $Q(s) > V_{\pi(s)}(s)$
10. **then** $\pi'(s) = \arg\max_a Q^a(s)$
11. **else** $\pi'(s) = \pi(s)$
12. **end for**
13. **end for**
14. **until** $\pi = \pi'$

Output: return π

It is important to note that the policy iteration does not guarantee a global optimal value function but guarantees an optimal policy in a finite number of

iterations since there are finite number of policies $|S|^{|A|}$ in the stationary case. Even though policy iteration is an interesting algorithm, we do not use it in the partially observable case for theoretical and computational reasons. The reader may consult White (1993), Puterman (1994), Sutton & Barto (1998) or Cassandra (1998) for more rigorous details of the algorithm. Pashenkova, Rish & Dechter (1996) also provided additional survey of the value iteration and policy iteration algorithms for MDPs.

2.5 Summary

In this chapter we presented and discussed the Markov Decision Process framework for sequential decision making when the actions effects of an agent operating in an environment is uncertain. We also outlined the formal MDP model with finite states and finite actions. The computation of the value function as they are applied to solving MDPs using the value iteration algorithm and policy iteration algorithm were also presented. The MDP model precedes our discussions of the generalized partially observable model. We shall use an extension of the value iteration algorithm to solve the partially observable case. We are interested in applying the model for decision-making and control in autonomous systems even though the framework can be applied to any decision-making process that can be modeled as an MDP.

Chapter 3

Partially Observable Markov Decision Processes

3.1 Overview

In chapter 2 we discussed the sequential decision making process where the agent has perfect knowledge of its environment but its action effects was uncertain. This model is suitable in practical applications where the uncertainty that results from sensing the state of an environment is minimal. In most interesting applications such as autonomous robot navigation or medical decision-making, the agent is operating in a potentially highly unpredictable and dynamic environment so it does not suffice to use sensory data solely as the true state of the environment. This results in a rapid increase in the cumulative error between the goal state and the agent's belief of the goal state. Consequently, the environment is only partially observable by the agent and the agent has both perceptual uncertainty and action uncertainty. The formal framework discussed in this chapter to account for perceptual uncertainty is *Partially Observable Markov Decision Processes (POMDPs)* a generalization of MDPs.

3.2 The POMDP Model

The POMDP model is the tuple $\langle S, A, T, R, Z, O, \gamma \rangle$. It is simply an MDP with the addition of a finite set of observations *Z* and an observation probability distribution function *O* as shown in *figure* 3.1. It is defined below as:

- **The MDP Model**: $\langle S, A, T, R \rangle$ described in section 2.3
- **Observations**: $z \in Z$ is the finite set of observations. The observations are a noisy projection of the true state of the environment. If the set of observations maps perfectly to the set of states, that is, some function $f : Z \to S$ for all $z \in Z$ and $s \in S$, then this function is a complete representation of the environment. It is said to be bijective thereby permitting the POMDP to be modeled as an MDP where each state simply maps to its corresponding measurement. However in practice this is rarely the case.
- **Observation Probability Function**: $O(z, a, s')$ is the function that describes the conditional probability of observing z given the action a and the subsequent state s'. Thus $O(z, a, s') = p(z \mid a, s')$.
- **Beliefs**: The agent must now maintain a belief of the state of the environment since it is unable to fully observe the state with certainty. All agents acting *rationally*[1] in physical environments must maintain a belief of its environment relative to idealized states of the environment. We define the belief *b* as a probability distribution over the set of states *S* and $b(s)$ to be the probability of observing a state *s*. The belief space is the space of all possible belief states which is a *simplex* of $|S| - 1$ dimensions. Also $0 \le b(s) \le 1$ and $\sum_{s \in S} b(s) = 1$. We show how to compute the belief for subsequent states in section 3.2.2.

[1] We use the term rational agents, as described in Russell and Norvig (2003), to be any autonomous system capable of exhibiting goal-based behavior.

- **Discount Factor**: The discount factor is the real value $\gamma \in [0,1)$. As with MDPs it indicates the desirability of decisions made in the future and ensures that the value iteration algorithm converges to a finite value.

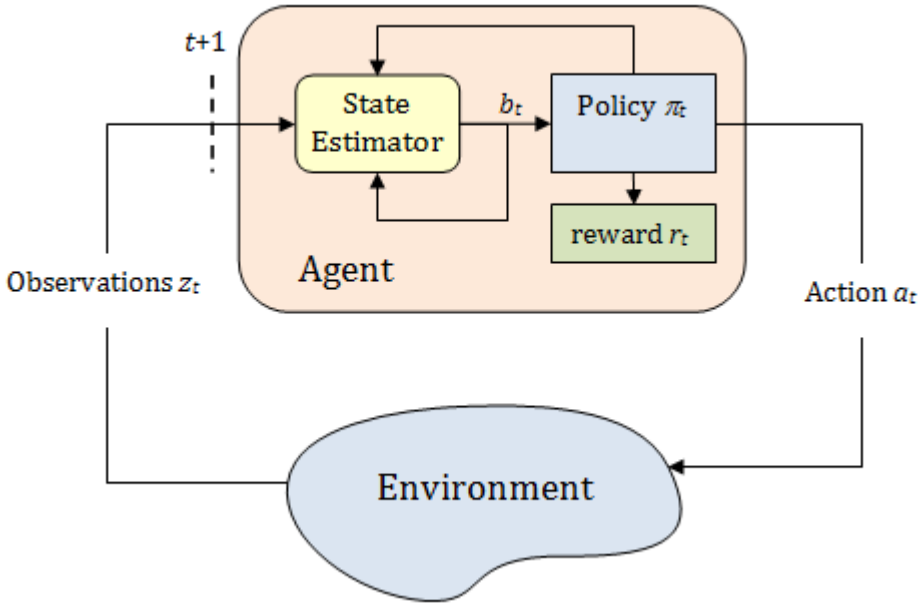


Figure 3.1 – An Abstract POMDP Model

Other interesting formulations exist. For instance Doshi-Velez (2009) model the problem as an infinite POMDP (iPOMDP) where the set of states are not explicitly represented but increases from an initialize size without bound as the agent acts in the environment.

3.2.1 Objective of the POMDP

As with MDPs the objective of solving POMDPs is to find the policy that maximizes the long term reward that the agent receives. This is equivalent to the rewards described in the MDP model. For the finite horizon model which we will be

more interested in, the maximum cumulative reward from the set of all possible policies $\pi \in \prod$ is given by:

$$R_{\pi}^{*} = \max_{\pi}\left(E\left[\sum_{t=1}^{T} \gamma^{t} r_{t} \right] \right). \tag{3.1}$$

The corresponding optimal policy that maximizes the expected future discounted reward is given by:

$$\pi^{*}(b_{t}) = \arg\max_{\pi} E\left[\sum_{t=1}^{T} \gamma^{t} r_{t} \middle| b_{t}, \pi \right]. \tag{3.2}$$

Since the agent cannot directly observe its domain. This optimal policy is a mapping from belief to action. That is, $\pi_{POMDP} : b \rightarrow a$ where $\pi(b)$ is the action selected for a given belief. Consequently, the reward function is a function of belief and not the states since the agent can only accumulate reward based on what it beliefs are. In the next section we discuss how the beliefs are constructed.

3.2.2 Belief Computation

The agent maintains a belief of the state of the environment and selects actions based on these beliefs. In autonomous robot navigation and path planning, the state of the environment are typically represented by the robot's pose (x, y, θ) and its respective velocity components $(\dot{x}, \dot{y}, \dot{\theta})$. State estimators such as Kalman filters are used to generate beliefs which can be used by a POMDP controller to execute appropriate actions.

Recall that the belief *b*, of an agent is the probability distribution over all states. In theory, the belief can be any formalism that can effectively capture the states of

the environment. During agent-environment interaction the agent has to update its beliefs at every decision step. It also has to maintain a history of previous observation and actions to deduce its current beliefs. However, Smallwood and Sondik (1973) show that the current belief state is a sufficient statistic given the past history of initial belief, observations and actions. This implies that belief computation is Markovian. That is, if the belief at time t is given by $b_t(s) = p(s_t = s \mid z_t, a_{t-1}, z_{t-1}, ..., a_1, b_0)$, it is equivalent to $b_t(s) = p(s_t = s \mid z_t, a_{t-1})$.

We compute the subsequent belief state $b'(s')$ of the agent using a variant of Bayes' Theorem and the Total Probability Theorem. The belief $b'(s') = p(s' \mid z, a, b)$ and can be derived as follows:

$$b'(s') = p(s' \mid z, a, b), \tag{3.3}$$

$$b'(s') = \frac{p(z \mid s', a, b)\, p(s' \mid a, b)\, p(a \mid b)}{p(z \mid a, b)\, p(a \mid b)}, \tag{3.4}$$

$$b'(s') = \frac{p(z \mid s', a, b) \sum_{s \in S} p(s' \mid s, a, b)\, p(s \mid a, b)}{p(z \mid a, b)}, \tag{3.5}$$

$$b'(s') = \frac{O(s', a, z) \sum_{s \in S} T(s, a, s') b(s)}{p(z \mid a, b)}. \tag{3.6}$$

The reciprocal of $p(z \mid a, b)$ can be considered a normalizing factor that allows $b'(s')$ to sum to 1.

Therefore $b'(s')$ can be given by:

$$b'(s') = \eta O(s', a, z) \sum_{s \in S} T(s, a, s') b(s), \tag{3.7}$$

where η is the normalizing factor $1/\ p(z \mid a, b)$.

3.2.3 POMDPs as Belief State MDPs

To compute an optimal policy for the POMDP we can use a variant of the value iteration algorithm. This algorithm is only computationally stable for finite horizon models. Recall that our policy now is the mapping $\pi_{POMDP} : b \to a$, where b is the belief (a probability distribution vector over states of the POMDP). Since the belief represents a sufficient statistic for the history of the agent's behavior we can convert the POMDP into a Belief-State MDP or simply Belief MDPs (BMDPs). It is very important to note that BMDPs has a continuous bounded state space but with infinite possible states.

For the clarity, if there are three states $S = \{s_1, s_2, s_3\}$, then the state space of the BMDP will be 2-dimensional simplex in a 3-dimensional space whose vertices will be at the points $(1,0,0),(0,1,0),(0,0,1)$ as shown in *figure* 3.2. The vertices are the belief points where the agent is certain to be in states s_1, s_2 and s_3 respectively. Notice that the belief point has a dimensionality of |S|=3. This is because it is the vector representing the distribution of over states. We can model the BMDP as the tuple $\langle B, A, \tau, r \rangle$. It is described as follows:

- **Belief States**: B is the set of belief states whose dimensionality is |S| and makes up the state space. It is represented as an $|S-1|$ hyperplane in |S|-dimensional space. The vertices of the hyperplane have a value of 1 which denotes the certainty of being in a specific state. Though the hyperplane is bounded, it is continuous—which implies it is uncountably finite. That is, there are an infinite number of belief points in the region. This poses

problems when solving the POMDP or BMDP optimally using the value iteration algorithm. We discuss how to overcome this issue in the next section

- **Actions**: A is the set of actions available at each decision point. For simplicity it is assumed that the same set of finite actions is available at every decision point
- **Transition Probability Function**: $p(b' \mid a, b)$ is the transitional probability function between beliefs. It can also be represented as $\tau(b, a, b')$. Since we are transitioning between belief states there are observation probabilities that should be accounted for. Thus, $p(b' \mid b, a) = \sum_{z \in Z} p(b' \mid b, a, z)\, p(z \mid b, a)$ where $p(b' \mid b, a, z) = 1$ if the $b' = b^{a,z}$ (i.e. the state estimate is b') $\therefore\ p(b' \mid b, a) = \sum_{z \in Z} p(z \mid b, a)$
- **Reward Function**: $r(b, a)$ is the reward received for executing action a in belief state b. This reward is related to $R(s, a)$ by the equation: $r(b, a) = \sum_{s \in S} b(s) R(s, a)$. This reward function considers the probability of observing the state s for every component of the belief point.

This model shows how we can convert a POMDP to a BMDP. The next step is to solve the BMDP using the value iteration algorithm discussed earlier. The primary difficulty with using the value iteration algorithm to solve the POMDP or BMDP is that we have to iterate recursively over every belief state. This is not possible since there are infinite numbers of belief states on the hyperplane representing the belief states. In the MDP case the number of states and actions were finite thus the MDP

was solvable in polynomial time (Puterman, 1994). Perhaps reducing the dimensionality of the belief space may improve the solvability of the problem. The problem with this approach is that the guarantee of optimality of the original problem no longer exists. Fordor (2002) provide a survey of some dimensionality techniques that may be of interest while Roy (2003) use EPCA to reduce the dimensionality of the POMDP model.

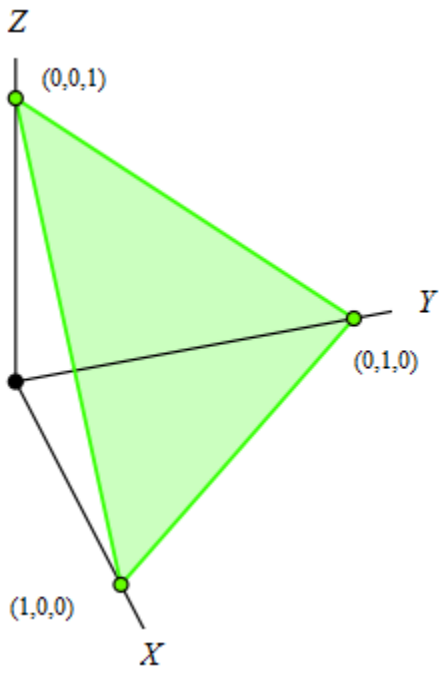


Figure 3.2 – A 2D Simplex in 3D Space Representing the Belief Space

3.2.4 POMDP Value Functions

To compute optimal policies for POMDPs we can compute optimal value functions for the POMDP and extract the optimal policy from the value function. However we focus on approximate value functions for practical reasons. Again, the value iteration algorithm is used to solve the POMDP as we did with the MDPs. In

MDPs the decision process for a finite horizon model has the possible sequence $(S_0 \times A_1 \times S_1 \times \ldots \times A_T \times S_T) = (S \times A)^T \times S$ where T is the time horizon length. For the infinite horizon model the sequence is $(S \times A)^\infty$. Whereas with a finite horizon POMDP the agent makes a sequence of actions and observations given by: $(A_1 \times Z_1 \times A_2 \times Z_2 \times \ldots \times Z_{T-1} \times A_T) = (A \times Z)^{T-1} \times A$. Since the observations and actions are a function of the belief state, they must be considered when computing the non-stationary policy for the POMDP. As we did with MDPs, when looking forward in time we use the notation t to denote a specific decision point and T the horizon length for finite horizon models.

Given any decision sequence $d = \{a_i^n \to z_k^{n-1} \to a_i^{n-1} \to \ldots \to a_i^2 \to z_k^1 \to a_i^1\}$, where n denotes the decision point with *n-steps to go*—(it should not be confused with the t^{th} decision point which is the time from the start of policy execution)—we can compute a value function for that sequence. In the sequence, a_i^n is the i^{th} action selected from the action set A with n-steps to go and z_k^{n-1} is the k^{th} observation made with $(n-1)$-steps to go. A complete decision sequence of actions conditioned on observations occurs when $T = n$. That is, when n is the horizon length. The decision sequence can also be modeled as a decision tree (Kaelbling, Littman and Cassandra, 1998).

- **Case 1**: $n = 1$

$$V_d(s) = R(s, a_d), \tag{3.8}$$

where a_d is the action selected for first time step executed in the decision sequence.

- **Case 2**: *n*-steps to go

$$V_d(s) = R(s, a_d) + \gamma \sum_{s' \in S} p(s' \mid s, a_d) \sum_{z_k \in Z} p(z_k \mid s', a_d) V_{z_k^d}(s'), \tag{3.9}$$

$$V_d(s) = R(s, a_d) + \gamma \sum_{s' \in S} T(s, a_d, s') \sum_{z_k \in Z} O(s', a_d, z_k) V_{z_k^d}(s'). \tag{3.10}$$

Recall that $T(s, a_d, s')$ is the transition probability function between states; $O(s', a_d, z_k)$ is the observation probability function and $V_{z_k^d}(s')$ is the value function for the $(n-1)$-step decision sequence after observation z_k is made.

The value function in equations (3.8)-(3.10) is defined over the state of the environment. However our agent can only make decisions based on its belief *b*. Thus the value function for a belief *b* given a decision sequence d is given by:

$$V_d(b) = \sum_{s \in S} b(s) V_d(s). \tag{3.11}$$

It can also be written in vector form where α_d is the value vector whose component is $V_d(s_i)$ for all $i = 1, 2, ..., |S|$. It is denoted as:

$$V_d(b) = b \cdot \alpha_d, \tag{3.12}$$

where $\alpha_d = \langle V_d(s_1), ..., V_d(s_n) \rangle$ is known as the α-vector. The value function for every belief vector is the expectation of the value function of states *S*.

3.3 Optimal POMDP Value Function

The optimal value function is found by selecting the value of the decision sequence d that maximizes the value function $V_d(b)$. That is, the optimal value function with *n*-steps to go is:

$$V_n^*(b) = \max_{d \in \Psi} \sum_{s \in S} b(s) V_d(s), \tag{3.13}$$

$$V_n^*(b) = \max_{d \in \Psi} V_d(b), \tag{3.14}$$

where Ψ is the set of possible decision sequences.

Computing the corresponding optimal policy is very difficult for large state spaces with long time horizons. This is because there can be numerous decision sequences to consider in order to find the optimal value function. Consider a decision sequence with $A = 4$ actions, $Z = 3$ observations and the planning horizon $T = 10$. Then possible decision sequences can be found by $(A \times Z)^{T-1} \times A$. Therefore there are $(12)^9 \times 4 = 20,639,121,408$ possible decision sequences. This is often referred to as the *curse of history*. Exact methods developed focused on exhaustive enumeration (Sondik, 1971). Other approximate solutions are generated by selecting from a set of belief points rather than the entire belief space and then pruning the value functions of undesirable decision sequences that do not contribute to the optimal value function. *Figure* 3.3 illustrates an arbitrary set of 4 value functions where the optimal value function is the upper surface of the collection.

Since we can represent the POMDP and a BMDP with a finite horizon, the value function backup equation is denoted as:

$$V_{n+1}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} p(b' \mid b, a, z)\, p(z \mid b, a) V_n(b') \right]. \tag{3.15}$$

When $b' = b^{a,z}$ then $p(b' \mid b, a, z) = 1$ since the transition between beliefs is perfectly modeled by the state estimator.

Thus we have:

$$V_{n+1}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} p(z \mid b,a) V_n(b') \right]. \tag{3.16}$$

The corresponding policy is:

$$\pi_{n+1}(b) = \arg\max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} p(z \mid b,a) V_n(b') \right]. \tag{3.17}$$

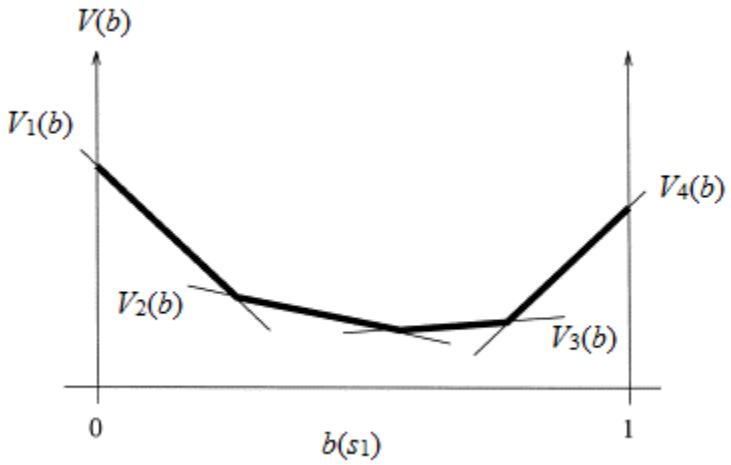


Figure 3.3 – A Sample Value Function for a 2 State POMDP

3.4 Algorithms for Solving POMDPs

In this section we briefly describe some early algorithms developed to solve the POMDP problem as well as algorithms presently implemented by current researchers. As mentioned, the main issue with solving POMDPs with value iteration is to iterate over belief space which is continuous and uncountably finite—i.e. there are infinite number of points in the bounded belief space hyperplane. However, Smallwood & Sondik (1973) showed that the value functions over belief

space are *Piecewise Linear Convex* (PWLC) which provides interesting properties that allow for the formulation of algorithms to solve the POMDP or BMDP.

Monahan (1982) discusses early algorithms used to solve the POMDPs such as the *One-Pass Algorithm* by Sondik (1971). Other early algorithms proposed are the *Witness Algorithm* by Littman (1994) and Kaelbling et al. (1998). The witness algorithms works by computing the best value function for a set of actions at a given time and takes their union to get the optimal value function at decision point *n*. Hansen (1997) focuses on using a policy iteration algorithm to solve the problem by representing the policy as a finite state controller. Cassandra, Littman and Zhang (1997) presented the incremental pruning algorithm which is an exact algorithm that prunes dominated vectors that do not add to the optimal value function.

More recent algorithms such as the *Point-Based Value Iteration (PBVI)* algorithm by Pineau, Gordon & Thrun (2003) have significantly improved the size of problems that can be solved. Their algorithm is an approximate method that chooses a small set of belief points and computes a single value function hyperplane and their resulting derivatives for those points. Hsu, Lee and Rong (2007) discuss why POMDPs are simpler to approximate in relation to computing exact solutions. He & Roy (2009) use a forward search technique that generates policies directly from the posterior belief distribution thereby avoiding the task of computing belief distributions by considering all the possible observations. The technique is used in continuous models. Kurniawati, Hsu & Lee (2008) propose an algorithm called *SARSOP* which is an acronym for Successive Approximations of the Reachable Space under Optimal Policies. It is a version of the Point-Based Value Iteration algorithm

that improves computational efficiency by focusing on reachable belief spaces rather than the entire belief space. They apply their algorithm to robot exploration tasks. We discuss the SARSOP algorithm further and employ a SARSOP based solver to find efficient policies for the proposed POMDP model.

Notwithstanding, other approaches for solving POMDP variants such as Decentralized POMDPs (DEC-POMDPs) using finite-state controllers have been proposed (Amato, Bonet & Zilberstein, 2010). Decentralized POMDPs are used when multiple agents are involved in the decision-making. Pajarinen and Peltonen (2011) also apply a finite-state controller to find a deterministic finite-horizon policy for the DEC-POMDP while Eker & Akin (2013) propose an algorithm that uses Genetic Algorithms to search the policy space then use a finite-state controller to represent the finite-horizon DEC-POMDP. In the case of infinite-horizon models which we do not consider, Li, Liao & Carin (2006) present an incremental least squares approach to solve the problem.

3.4.1 Point-Based Value Iteration Algorithms

Since solving partially observable problems is PSPACE-hard (Papadimitriou & Tsitsiklis, 1987) even for cases where the observation is deterministic it is often practical to search for near optimal or approximate solutions rather than exact optimal solutions except for the smallest problems. In fact it is often not desirable in practice to solve the POMDP optimally as the computational costs typically exceed its potential practical utility. The value iteration algorithm is generally adopted as a solution method for POMDPs. As mentioned exact value iteration is undesirable

since they iterate over the entire belief space which is enumerable infinite—that is, the belief space is continuous. To alleviate this difficulty, Point-Based value iteration techniques approximate the solution for exact value iteration by identifying a small set of belief points then computing its value and its derivative for only those selected points (Lovejoy, 1991 and Pineau, Gordon & Thrun, 2003).

The formulations by Lovejoy (1991) centered on selecting an arbitrary set of belief points then pruning any of the α-vectors from the value function that were not optimal from the set of belief subsets. A problem with this approach is that from the belief points selected there was a likelihood that those belief points will not be reached. Pineau, Gordon & Thrun (2003) propose selecting a set of *reachable* belief points by selecting a sequence of actions and observations. Thus the Bellman backup equation can be modified to include only the selected actions and observations sequence.

Given a POMDP model $\langle S, A, T, R, Z, O, \gamma \rangle$ as described in section 3.2 and an α-vector, which is a set value functions for each state *s*, the value function at each selected belief from the decision sequence is computed by the modified Bellman backup equation as follows:

$$V'(b) = \max_{a \in A} R(b,a) + \gamma \sum_{z \in Z} p(z \mid b,a) V(b^{a,z}), \tag{3.18}$$

$$V'(b) = \max_{a \in A} R(b,a) + \gamma \sum_{z \in Z} p(z \mid b,a) \max_{\alpha \in V} \sum_{s' \in S} \alpha(s') b^{a,z}(s'), \tag{3.19}$$

$$V'(b) = \max_{a \in A} R(b,a) + \gamma \sum_{z \in Z} p(z \mid b,a) \max_{\alpha \in V} \sum_{s' \in S} \alpha(s') \frac{O(s',a,z) \sum_{s \in S} b(s) T(s,a,s')}{p(z \mid b,a)}, \tag{3.20}$$

$$V'(b) = \max_{a \in A} R(b,a) + \gamma \sum_{z \in Z} \max_{\alpha \in V} \left[\sum_{s \in S} b(s) \sum_{s' \in S} \alpha(s') O(s', a, z) T(s, a, s') \right], \quad (3.21)$$

$$V'(b) = \max_{a \in A} R(b,a) + \gamma \sum_{z \in Z} \left[\max_{\alpha \in V} \sum_{s \in S} b(s) \alpha^{a,z}(s) \right], \quad (3.22)$$

where $p(z \mid b, a) > 0$ and $\alpha(s) = \sum_{s' \in S} \alpha(s') O(s', a, z) T(s, a, s')$. $R(b,a)$ can be written in vector form as $R(b,a) = r_a \cdot b$. Equation (3.22) in vector form we have:

$$V'(b) = \max_{a \in A} r_a \cdot b + \gamma \sum_{z \in Z} \max_{\alpha \in V} b \cdot \alpha^{a,z} . \quad (3.23)$$

The backup operator that generates a new α-vector for a specific belief *b* is given by:

$$f(V, b) = \arg\max_{\alpha \in V, a \in A} b \cdot \alpha_a^b , \quad (3.24)$$

where $\alpha_a^b = r_a + \gamma \sum_{z \in Z} \arg\max_{\alpha \in V} (b \cdot \alpha_a^b)$. The function $f(V,b)$ prunes the vectors that are dominated twice thereby reducing the computational costs of the procedure. The complexity of the point-based backup procedure for a set of belief point $|B|$ is $O(|A| \times |Z| \times |V| \times |S|^2 + |A| \times |S| \times |Z|)$ in comparison with exact backup procedure which has $O(|A| \times |Z| \times |V| \times |S|^2 + |A| \times |S| \times |V|^{|Z|})$. Further details on the complexity analysis and comparison can be found in Shani, Pineau and Kaplow (2012). A point-based value iteration algorithm is shown in *table 3.1* and the corresponding optimal policy for a point-based value function is:

$$\pi_V(b) = \arg\max_a R(b,a) + \gamma \sum_{z \in Z} p(z \mid b, a) V(b^{a,z}) . \quad (3.25)$$

The application of point-based value iteration techniques for solving problems with large state spaces have proven successful. As mentioned, the initial framework was present by Pineau, Gordon & Thrun (2003). Since then numerous variant

algorithms such as HSVI by Smith & Simmons (2004, 2005); FSVI by Shani, Brafman & Shimony (2007); Perseus by Spaan & Vlassis (2004, 2005) and SARSOP by Kurniawati, Hsu & Lee (2008) have been formulated. Other algorithms such as GapMin by Poupart, Kim & Kim, (2011) have been formulated to improve the bounds of the value function generated by the point-based algorithms. It is based on initial work by Poupart (2005). Porta, Spaan & Vlassis (2005) also postulates an extension of the discrete state model to continuous state spaces. Naturally modeling continuous state spaces is nontrivial as the expected value functions over states are defined by integrals that typically cannot be computed in closed form. However, they show that the optimal value function over an infinite dimensional belief space is also piece-wise linear convex. These findings permit the use of point-based value iteration algorithms to solve the problem.

These advances improve the problem space that can be tackled lending credence to their potentially diverse application in sequential decision-making problems and in our case the autonomous navigation problem. In this section we have succinctly discussed the point-based VI algorithms and how it significantly improves results in solving POMDP problems with large state space applicable in complex domains. Kaplow (2010), Shani, Pineau & Kaplow (2012) present the most up to date and rigorous survey of the current state-of-the-art point-based POMDP solvers. They focus on comparing the various algorithms by performing empirical analysis on several well-known benchmark problems such as tag and coastal navigation. Since certain solvers will be well-suited for certain domains it is important that their advantages and disadvantages are also highlighted. The survey analysis does this.

So how exactly does the point-based technique perform? To answer this we must understand the methods by which belief points are selected from the belief space.

Table 3.1 – A Point-Based Value Iteration Algorithm

Point-Based Value Iteration Algorithm

Input (*POMDP Model,* B_0)

1. $B \leftarrow B_0$ (initialize the set of belief points)
2. **repeat loop**
3. **repeat loop**
4. **for all** $b \in B$
5. $\alpha \leftarrow f(b,V)$ (randomly execute a backup operation for all belief points)
6. $V' \leftarrow V \bigcup \{\alpha\}$
7. $V \leftarrow V'$
8. **end for**
9. **until** $\left|V' - V\right| < \delta$ (stop when V and V' has converged for all belief points)
10. $B' \leftarrow B$
11. **for all** $b \in B$
12. $next(b) \leftarrow \{b^{a,z} \mid p(z \mid b,a) > 0\}$
13. $B' \leftarrow B' \bigcup \underset{b' \in next(b)}{\arg\max} \left\| B, b' \right\|_L$ (add the next belief *b* furthest from all points in *B*)
14. $B \leftarrow B'$
15. **end for**
16. **until** $\left|V^* - V\right| < \varepsilon$ (stop when V and V^* has converged for all belief points)

Output: B'

Most early approach used ad hoc methods and heuristics that provided poor results of the true optimal solution of the POMDP. As discussed more recent methods use a more structured approach in belief point selection. Rather than

develop algorithms that iterate over the entire belief space, which is enumerable infinite, the point-based technique approximates the solution to an exact value iteration by identifying a small set of belief points then computing its value and its derivative for only those selected points.

3.5 Summary

This chapter briefly presents the POMDP framework and algorithms that have been formulated to solve them and their approximations. It is not meant to be an exhaustive treatment of POMDPs. More rigorous treatment in literature can be found in Sondik (1973, 1978), Monahan (1982), Kaelbling (1998), Cassandra (1998), Pineau, Gordon & Thrun (2003) and Shani, Pineau & Kaplow (2012). Most authors that conduct research on POMDPs focus on computing efficient solutions using point-based techniques to avoid planning over the entire belief space which is known to be computationally intractable. We are interested in POMDPs because it has shown promising results in solving complex decision-making in autonomous systems where the agent-environment interactions are not fully predictable. In regard to this research we wish to further investigate its potential for generating efficient policies for autonomous navigation in mobile robots with possible application in industry.

Chapter 4

Decision-Making for Autonomous Robots using POMDPs

4.1 Overview

In the previous chapter we discussed the basic theory of POMDPs. We also outlined the difficulties of finding exact solutions of the POMDP model and discussed some approaches that have been implemented to overcome these difficulties but at the cost of optimality. However, in this work we are interested in practical implementation of POMDPs in autonomous robots. This implies that we are willing to accept the cost of guaranteed optimality of the POMDP solution in lieu of efficient performance of the autonomous robot. As mentioned, robots operating in real-world environments encounter uncertainty from both the environment dynamics and its internal sensory-motor dynamics. We model these uncertainties using a probabilistic framework and apply POMDPs primarily for action selection given the perceptual data. The POMDP model is essentially a high-level model. We are not concerned with how motor signals are executed electronically or how sensory signals are processed as this is robot specific. Thus in this chapter we shall discuss POMDPs as it relates to the generation of goal-based behavior in mobile robots. We focus more on navigation in partially structured environments as this is still an open problem in robotics.

It is instructive to note that numerous other planning methods exist for robot navigation. They all basically attempt to formulate an algorithm that can efficiently

navigate a robot from an initial location to a goal destination by planning a trajectory path. Accurate localization and mapping is a necessary precondition for navigation of a mobile robot. We do not consider details localization and mapping techniques in this work as this a separate research area in its own right. However we adopt Bayesian state estimation techniques such as Kalman filters for localization and belief estimates.

4.2 The Autonomous Robot Navigation Problem

When we mention decision-making in relation to robotics we are refer to navigation and task completion in an operating domain. The robot navigation problem is a well studied problem. Numerous methods and models have been proposed. We define the problem as follows:

> *Given a map of a known environment such as an office or factory floor with static and dynamic obstacles, an autonomous robot is required to navigate from an initial location to a goal location. We wish to formulate an effective decision-making policy that results in the robot arriving at its goal location using minimal resources. The robot receives perceptual information from its sensors and translates this information to motor commands based on the decision policy.*

Recall that a robot is simply a computer controlled mechanical device. For the robot to behave autonomously, it must reliably take perceptual information, process this information, and select appropriate motor actions that result in the robot reaching its goal state. Thus from the problem statement above our goal in this work is to find an efficient decision policy—that is a description of how to robot should select

its motor command at every state of the environment to guide the robot's behavior. It is also noteworthy to specify that the mechanical composition of the robot significantly affects the performance of the controller and its overall behavior. In this section we briefly describe components of the robot used for sensory observation and motor control then describe how it will be used in the POMDP model. We also discuss basic mobile robot kinematics.

4.2.1 Mobile Robot Kinematics

Mobile robot kinematics deals with the analysis of motion without regard to the forces that create the motion. It is necessary to understand mobile robot kinematics for motion planning and navigation. A mobile robot can be modeled as a rigid body on a 2-dimensional plane. Its position is defined with respect to a global reference frame. This (x_g, y_g) position denoted by *P* along with its heading or orientation θ is known as the *pose* of the mobile robot. Most times the pose is considered the *state* in regard to the planning models. It is represented as the vector $s_G = [x_g, y_g, \theta]^T$. The robot can also have its own reference frame called the local reference frame and the axis orthogonal to the plane which describes the heading. In general, the kinematic model can be described with six variables. The first three represents its 3-dimensional spatial coordinates while the last three describe its three Euler angles—namely the pitch (θ), roll (ϕ) and yaw (ψ) (note that the heading θ should not be confused with the pitch θ). *Figure 4.1* illustrates the robot pose in a global coordinate system.

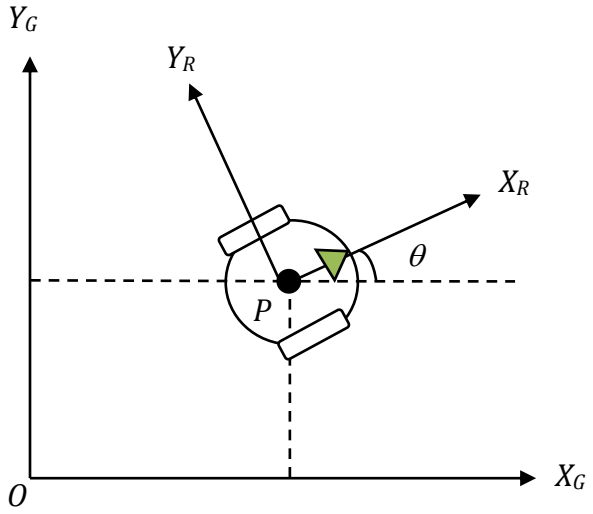


Figure 4.1 – Global and Local Coordinate Systems of the Robot Pose in 2D

The axes (X_G, Y_G) is the global reference coordinate system with origin *O* while the body attached axes is (X_R, Y_R) with center *P*. The center of the body attached axes *P*, has the location (x_g, y_g) with respect to the global reference frame. The heading θ is the angular difference between the global reference and the robot local reference frame.

In order to describe the motion of the robot with respect to the global reference frame we simply use an *orthogonal rotation matrix* that maps the motion along the global reference frame axes to that of the robot's local reference frame. The rotation matrix is given by:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \qquad (4.1)$$

Thus the robot's pose with respect to the robot's local reference frame can be computed by:

$$s_R = R(\theta)s_G, \tag{4.2}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_R = R(\theta)\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_G. \tag{4.3}$$

Correspondingly, the velocity component of the pose in the global reference is denoted by the vector $\dot{s}_G = [\dot{x}_g, \dot{y}_g, \dot{\theta}]^T$. The velocity with respect to the robot's reference frame can computed in a similar manner to equation (4.2),

$$\dot{s}_R = R(\theta)\dot{s}_G. \tag{4.4}$$

Further discussion on mobile robot kinematics can be found in the Seigwart, Nourbakhsh & Scaramuzza (2011).

4.2.2 Probabilistic Kinematic Model

Since we are interested in mobile robots that operate in probabilistic domains we must consider the error difference between desired actions and actual motion actions. This is important in the robot's decision-making because it directly reflects the state transition model. Given that the robot's state is its pose $s_t = [x, y, \theta]^T$ at time *t*, the state transition model is given by $p(s_{t+1} | a_t, s_t)$. It can also be denoted as $p(s_t | a_{t-1}, s_{t-1})$, which is simply a matter of convention. The transition model is the posterior probability distribution over the subsequent states given current states s_t and action a_t. As an example the robot's odometer may provide the execution of the

action control a_t. The action control a_t indicates the motion commands that are sent to the robot motor for execution.

It is also possible to provide action controls using a velocity model by considering the *rotational* and *translational velocities* as discussed in Thrun, Burgard & Fox (2006). The translational velocities at any point in time can be denoted by υ_t while for rotational velocities it is denoted as ω_t. In this formulation the action control is $a_t = [\upsilon_t, \omega_t]^T$ where the translational velocities produce forward linear motion and the rotational velocities produce counterclockwise rotation of the robot wheels. Thus to compute the subsequent state at $t+1$ a desired algorithm would receive as input the pose $s_t = [x, y, \theta]^T$ and action $a_t = [\upsilon_t, \omega_t]^T$ while outputting the subsequent state $s_{t+1} = [x', y', \theta']^T$. The POMDP we formulate does not consider rotational and translational velocities.

4.2.3 Sensory Model

An important part of the POMDP model for decision-making is the *observation model*. There are varieties of sensors used in mobile robots. Some are laser range finder, ultrasonic sensors, stereovision cameras, wheel encoders, contact and proximity sensors, temperature sensors etc. They are typically classified in two functional categories namely *proprioceptive* and *exteroceptive* sensors. Proprioceptive sensors measure the robot's internal state while exteroceptive sensors measure the environment's state. Since we are interested in observations in partially observable environments, the sensory model must handle sensory noise

and measurement errors. This is due to the intrinsic uncertainties in the robot's perceptual system. The observation model is given by $p(z_t|s_t, a_{t-1})$, where z_t defines the robot's observation at time t, s_t defines that state of the system which is the robot's pose at time t and a_{t-1} the previous action controls at time $t-1$. We can use both the action model and sensory to update the state of the system as information is acquired during operation. Also the type of sensory data collected affects how it is modeled and the amount of computing time required to process the data. For instance 3D data requires considerably more computing power than 2D data.

4.2.4 Recursive State Estimation

A key issue in robotics and decision-making in general is to estimate the state of the system after an action and observation has been made. Without doing so it will not be possible to determine the necessary controls for the robot to achieve its task. This is equivalent to the belief update we discussed in 3.2.2. It is especially important since it allows us to continuously localize the robot in its environment. To compute the state estimates we typically use a variant of the *Bayes Filter* algorithm which is a recursive algorithm. We present the algorithm but do not discuss its variant such as the *Gaussian Filters*, *Particle Filters*, *Kalman Filters* or *Extended Kalman Filters*. Further details can be found in Bar-Shalom & Li (1998), Gustafsson et al. (2002) and Thrun, Burgard & Fox (2006).

The Bayes' filter algorithm computes the belief $b(s_t)$ recursively from observation z_t, action a_t and previous belief at time $t-1$, $b(s_{t-1})$. To describe the algorithm better, consider that we have a set of data $d_t = \{a_0, z_1, ..., a_{t-1}, z_t\}$ at time t that consists of the history of observations and actions. The sensor or observation model is given by $p(z_t | s_t)$, the action model is given by $p(s_t | a_t, s_{t-1})$ and the prior probability is given by $p(s)$. The problem is to estimate the posterior $b(s_t) = p(s_t | z_t, a_{t-1}, ..., z_1, a_0)$. We assume the system is Markovian and neglect sensory noise. The Bayes filter can be computed as follows:

$$b(s_t) = p(s_t | z_t, a_{t-1}, ..., z_1, a_0), \tag{4.5}$$

$$p(s_t | z_t, a_{t-1}, ..., z_1, a_0) = \frac{p(z_t | s_t, a_{t-1}, ..., z_1, a_0) p(s_t | a_{t-1}, ..., z_1, a_0)}{p(z_t | a_{t-1}, ..., z_1, a_0)}, \tag{4.6}$$

$$b(s_t) = \eta p(z_t | s_t, a_{t-1}, ..., z_1, a_0) p(s_t | a_{t-1}, ..., z_1, a_0), \tag{4.7}$$

$$b(s_t) = \eta p(z_t | s_t) p(s_t | a_{t-1}, ..., z_1, a_0), \tag{4.8}$$

$$b(s_t) = \eta p(z_t | s_t) \int p(s_t | s_{t-1}, a_{t-1}, ..., z_1, a_0) p(s_{t-1} | a_{t-1}, ..., z_1, a_0) ds_{t-1}, \tag{4.9}$$

$$b(s_t) = \eta p(z_t | s_t) \int p(s_t | s_{t-1}, a_{t-1}) p(s_{t-1} | a_{t-1}, ..., z_1, a_0) ds_{t-1}, \tag{4.10}$$

$$b(s_t) = \eta p(z_t | s_t) \int p(s_t | s_{t-1}, a_{t-1}) b(s_{t-1}) ds_{t-1}. \tag{4.11}$$

Equation (4.6) is simply the Bayes' Theorem. In equation (4.7) η is known as the normalizing constant given by $\eta = 1 / p(z_t | a_{t-1}, ..., z_1, a_0)$. In equation (4.8) and (4.11) we apply the Markov assumption and we use the law of total probability in equation (4.9). The reader can notice the similarity between equation (4.11) and (3.7). Also in equations (4.9)-(4.11) integrals are used instead of summation due to the fact

robots operate in real worlds which are both spatially and temporally continuous. However in practice the state is usually discretized allowing for the use of summations. Therefore equation (4.11) can be written as:

$$b(s_t) = \eta\, p(z_t \mid s_t) \sum_{s_{t-1} \in S} p(s_t \mid s_{t-1}, a_{t-1}) b(s_{t-1}). \qquad (4.12)$$

The belief update component of the algorithm is shown in *table 4.1*. In order to compute the belief recursively it is initialized with $b(s_0)$. This initial belief may be a Gaussian distribution over the state space if the initial state is known with some degree of confidence, a uniform distribution if we are completely unaware of the initial state or a probability mass function over a specific state if we are certain of the initial state. In general the algorithm simply consists of a predictive step and an observation update step.

Table 4.1 – Bayes Filter Algorithm

Bayes Filter Algorithm

Input $(a_t, z_t, b(s_0))$

1. **for all** $s_t \in S$
2. $b_*(s_t) = \sum_{s_{t-1} \in S} p(s_t \mid s_{t-1}, a_{t-1}) b(s_{t-1})$
3. $b(s_t) = \eta\, p(z_t \mid s_t) b_*(s_t)$
4. **end for**

Output: return $b(s_t)$

State estimation is also very important in solving a fundamental problem in robotics called *Simultaneous Localization and Mapping* (SLAM). The problem arises when the robot has to navigate unknown environments without the use of a map. It

must then generate a map. But to generate a map it needs to know its pose within the environment and to know it pose it needs a map. Therefore it must simultaneously localize and map. We do not consider the SLAM problems in this work as we assume the map of the environment is given. However is still important to mention since the formulation of SLAM techniques and methods are highly researched. Thrun (2008) reviews the three important approaches to solving the SLAM planning which apply the *Extended Kalman Filters* (EKFs), *Sparse Graphs* and *Particle Filters*.

4.3 Autonomous Robot Navigation using POMDPs

In this section we discuss the literature on implementation of POMDPs for decision-making in autonomous robots. We especially focus on its application in navigation path planning and motion control. Ibekwe & Kamrani (2008) provided a general overview on robotics while Seigwart, Nourbakhsh & Scaramuzza (2011) discuss specifically on autonomous mobile robots. Currently active research is dominant in the *Artificial Intelligence* and *Computer Science* communities however POMDP research originated from the *Operations Research* field. Much of the current research in POMDPs has been focused on formulating algorithms that solves models with large state space over longer time horizons. Applying it to robot navigation is still an open problem. Essentially, the navigation problem reduces to a controls problem where we want to select control actions given the current knowledge and perceptual information. As is the main focus of this work, the models should handle the inherent uncertainty of the robot interactions. In the real-world on the other

hand, the use of POMDPs for localization are limited. Theocharous, Murphy and Kaelbling (2004) provide a method which uses a Dynamic Bayes Network (DBN) to represent a Hierarchical POMDP for the localization in multi-resolution spatial maps for indoor navigation. Even though we focus on navigation tasks, it is not the only application of POMDPs in robotics. Glashan et al. (2007) highlight the use of POMDPs for manipulation tasks while Zhang, Sridharan & Washington (2013) introduce a hierarchical decomposition of a POMDP that includes adaptive observation functions, automatic belief propagation and constrained convolutional policies that allow a team of robots to preserve task achieving behaviors using a visual sensing technique.

4.3.1 Robot Navigation with POMDPs in Literature

Most approaches for autonomous robot navigation focus on indoor environments. Littman, Cassandra & Kaelbling (1995), Cassandra, Kaelbling & Kurien (1996) and Koenig & Simmons (1998) provide some of the early noteworthy attempts for the application of POMDPs for robot navigation. They present a POMDP architecture that, at the time significantly outperformed a landmark-based approached. Pineau & Thrun (2002) describe the implantation of POMDPs for high-level control of robot behaviors. López et al. (2007) outline a global navigation architecture using a POMDP called SIRAPEM used for assisted care for the elderly and disabled. On the other hand some researchers (Likhachev & Stentz, 2006) have opted to take a different approach citing the hardness of the POMDP planning. Likhchev & Stentz (2006) use a probabilistic planner called PPCP in partially known

environments that was able to scale to environments with thousands of states while Marder-Eppstein et al. (2010) describe a method for indoor navigation of a *PR2* robot using voxel-based 3D mapping that models unknown spaces in indoor environments.

Ross, Chaid-Draa & Pineau (2008) present a Bayesian reinforcement learning method using Continuous POMDPs. The paper proposes the use of a particle filter algorithm to compute the posterior distribution of the model parameter. It then applies a planning algorithm that uses trajectory sampling to determine the actions. Their method selects actions optimally that tradeoff between state estimation, environment exploration and knowledge utilization. Some research involving multi-robot models using POMDP have been conducted. Chuang, Gerkey, Gordon & Ng (2005) put forth an open-loop planning method performance metrics in the benchmark problem *tag*. They argue that even though open-loop plans are less robust than full policies they still provide reliable performance that can be applied in certain scenarios. Roy, Gordon & Thrun (2005) present a method of compressing the belief space using an Exponential Family Principal Component Analysis (E-PCA). The result is that the problem space that can be solved is considerably increased. A recent algorithm called *Automated Model Approximation* (AMA) (Grady, Moll & Kavraki, 2013) focuses on constructing approximations of the state and action space that are domain-specific to compute efficient policies. This method and others described are approximate techniques that attempt to solve POMDPs which in practice has demonstrated success.

4.3.2 Sampling-Based Motion Planning for Mobile Robots

It is instructive to discuss other approaches to the motion planning problem for autonomous robots in order to have an intuition on how they contrast with the POMDP approach. Sampling-based motion planning uses a graph model to approximate the connectivity of the search space of the planning problem whereas in POMDPs we generates planning policies over the entire belief space in the exact case. Sample-based planning methods are computationally complex (Latombe, 1991) so approximations such Rapidly-expanding Random Trees (RRT) (LaValle, 1998 and LaValle & Kuffner, 2001) and Probabilistic Roadmap Method (PRM) (Kavraki et al., 1996) have been proposed to minimize the complexity.

Alterovitz, Siméon & Goldberg (2007) present a sampling-based motion planning framework called *Stochastic Motion Roadmap* (SMR) that constructs a roadmap by sampling collision-free states in configuration space and locally sample motions at each state to estimate state transition probabilities for every possible action. The roadmap is then used to formulate an MDP to generate optimal plans. Bhatia, Kavraki & Vardi (2010) propose a geometric approach where a high-level planner formulates high-level plans for the discrete abstraction of the system model while a low-level sampling-based planner uses both the physical model of the system along with the high-level plans to search the state-space for a feasible solution of the desired trajectories. They show that the geometry-based approach improves computational speed of benchmark sampling-based techniques. Bhatia, Maly Kavraki & Vardi (2011) outline an extension of the geometric approach called a multilayered synergistic planning framework which includes temporal goals over

a subset of the workspace. Recent advances have been made in providing simulation tools to explore sampling-based algorithms such as the OMPL (Şucan, Moll & Kavraki, 2012). An example of an implementation is shown in *figure 4.2*.

Figure 4.2 – The Open Motion Planning Library on ROS (Courtesy of ROS.org)

One of the more recent methods we describe is the use of *Task Motion Multigraphs* (TMMs) to solve the simultaneous task and motion planning (STAMP) problem (Şucan & Kavraki, 2012). A TMM is a directed acyclic multigraph $G_M = (V_M, E_M)$ such that $V_M = \{\upsilon \mid Q(\upsilon) \subset S\}$ is a finite set of vertices and every vertex υ is associated with a set of robot states $Q(\upsilon) \subseteq S$. S is the complete state space of the robot while $Q(\upsilon)$ is a sampled set of states. E_M is a finite multiset of edges that represents all the motion planning possibilities between every pair of nodes $(v_i, v_j) \in V_M \times V_M$. MDPs as described in chapter 2 along with TMMs are used to compute robust task plans.

The sampling-based approach are computationally less expensive than POMDPs hence its pervasive use in cutting edge motion-planning research. However POMDPs handle both the uncertainty from the motion and the noise from sensory observation to generate future plans in a more principled manner. The difficulty is that it must construct such plans in belief space. Nevertheless recent research show that empirically promising result can still be obtained as described in Porta, Spaan & Vlassis (2005), Candido & Hutchinson (2011) and Marthi (2012). When solutions to a motion planning problem are computed, they often have to be recomputed when the model condition changes. Also Kurniawati et al. (2012) outline a Guided Cluster Sampling motion planner that considers three sources of sensory uncertainty during active sensing. A sampling distribution based on the sensory data is used to partition a set of selected belief points into smaller subsets and an optimal policy is computed over this subset. Furthermore Kurniawati & Patrikalakis (2013) propose a *Point-Based Policy Transformation* (PBPT) algorithm that transforms the original solution to the POMDP by modifying the set of sampled belief points.

4.4 Summary

In this chapter we discussed the navigation and motion planning using POMDP controllers and sampling-based techniques. POMDPs provide a robust framework for modeling robot-environment interaction in uncertain and dynamic environments. Classical methods such as deliberative or behavior-based models either fail under inherent system uncertainty or are not scalable to larger more complex robot platform. As we have now seen, although POMDPs provided a

reliable framework there are still numerous limiting factors for widespread implementation in real-world domains such as office, home or factory environments. A diverse range of algorithms exists that have been applied in robot motion planning and navigation. In this work we do not focus on theoretical analysis of algorithms but rather propose a method for exploiting domain-specific features. We compute efficient policies using a state-of-the-art POMDP solver. We anticipate that by exploiting structured domain-specific features of the operating environment we should have greater success of implementing a POMDP-based robot controller in more complex situations. We discuss our proposed methodology in the next chapter and present experiments to demonstrate its validity.

Chapter 5

A Methodology for Goal-Specific Representation of POMDP Model Parameters

5.1 Overview

In preceding chapters we considered cases where the agent or decision-maker was perfectly aware of the world state after it selects an action though the effects of the selected actions were unpredictable. We mentioned that this is not a practical model of real-world agent-environment interactions as the agent or decision maker has to consider uncertainties in its perception as well. We then discussed using a POMDP model since they explicitly model perceptual uncertainty. Solving the POMDP model required generating policies—a description of how the agent should act based on its beliefs rather than the true state of the world. This led to computational issues as the agent's beliefs are potentially infinite. We then methodically described a variety of state-the-art methods and algorithms researchers have developed to overcome these difficulties in order to solve the POMDP model efficiently. Most techniques are not optimal but are within an upper and lower bound of optimality as heuristics are used to sample the belief space for computational tractability.

While the techniques described have advanced the state-of-the-art and solved relatively large sequential decision-making problems they are still ineffective in many complex real-world scenarios requiring lengthy time horizons to make

decisions. Minimal attention has been given to how the properties of the task environment affect the formulation of problem parameters. Recently Shani, Pineau & Kaplow (2012) have recognized this and allude to the importance of considering the properties of the task domain in selecting a solver rather than selecting the lastest and fastest solver. Researchers often assume that if we have fast solvers, then problems modeled by the POMDPs should be trivial without consideration of the task properties. This is a flawed assumption as complex real-world problems require prohibitively large number of parameters that must be enumerate to effectively solve them and these parameters must be able to capture the essential features of the problem.

As a result we proposed a methodology for modeling the task environment called *Goal-Specific Representation* (GSR) that we anticipate will improve the computational efficiency in solving complex real-world problems by exploiting structural properties of the task environment. Along with the task requirement we can significantly reduce the parameters required to model problem space. The idea behind the GSR approach is based on the fact that every agent operates within a context local only to that environment. A case in point is in the robot navigation problem where it is desired that the robot exhibit autonomous goal-based behaviors in its task environment. If we are given a known map, the starting location and the goal location within this map then we can ignore states that do not contribute to the robot navigating to its goal location. In a sense we are generating *sub-maps*. Also no two tasks are identical in the real-world even if the maps do not change. Obstacles can be added or removed between tasks or even while the robot is performing its

task. So it is also necessary to account for the dynamic nature of the environment. In this work we don't explicitly model the dynamics of the task domain. To the best of our knowledge this approach has not been explicitly formulated in literature and we anticipated that the methodology described can be adapted to complex domains. From our literature survey, the work most closely related to our proposed methodology is the variable grid decomposition method by Kaplow, Atrash & Pineau (2010).

5.2 The GSR Methodology

We apply the GSR methodology to the autonomous robot navigation problem. The goal is to generate policies that guide the robot's behavior. We model the decision-making process using a POMDP model and solve the POMDP with a state-or-the-art solver called APPL based on the SARSOP Algorithm by Kurniawati, Hsu and Lee (2008). We proceed to conduct empirical analysis on policies generated and test them on a real robot. We do not propose a new algorithm to solve the POMDPs but a methodology to reduce the parameter space required to model complex environments such as warehouses, offices or hospitals. Ultimately our goal is to deploy the proposed methods for industrial and service applications.

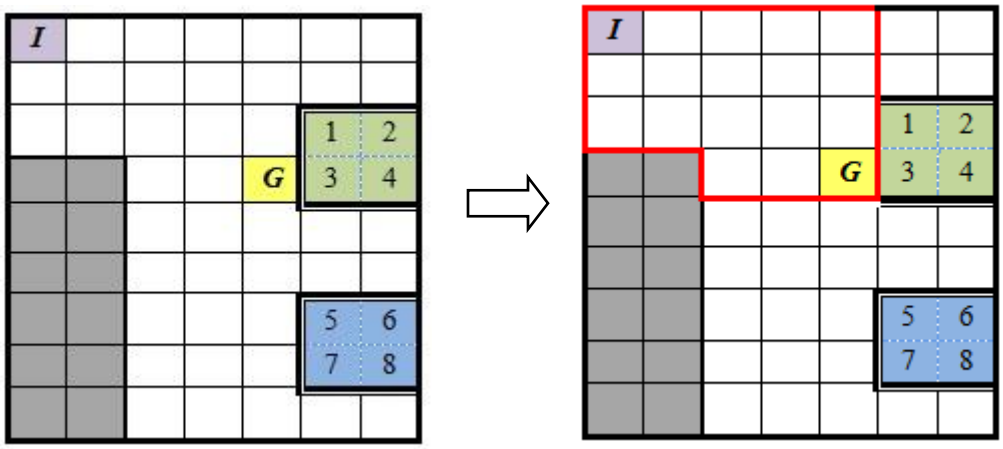
Consider a robot navigating an underwater domain. The model of this domain will differ from that of a robot operating on the planet Mars. The underwater domain will have to be modeled as a 3D voxel grid. Therefore it is imperative the state space is reduced. The GSR approach is robot invariant, meaning that it can be applied to any platform but obviously may perform differently between two varying

platforms. As highlighted in preceding chapters, various approaches exist to solve the robot navigation and motion planning problem. Adopting the POMDP formulation has met with varying degrees of success. If we have a map of the environment we only consider states closest to the initial state and goal state. The states of the environment are first decomposed to disregard the set of likely non-reachable states. Then planning is performed only on the reachable states in the sub-map. This indirectly reduces the dimensionality of the belief space the robot plans over. An optimal set of reachable belief states $\Re^*(b_0)$ are then extracted from the set of reachable beliefs $\Re(b_0)$ which in turn are sampled from the belief space B. This is presented by Kurniawati, Hsu & Lee (2008). The result is that we only consider states and beliefs that are relevant for reaching the goal state. By implication, the tradeoff is that the computed control policies are approximations as the entire state-space is not considered. However since we are more interested in reliable and efficient behavior of the robot rather than theoretically optimal solutions, these approximations will suffice.

5.2.1 Goal-Specific Representation

Operating environments for autonomous robot possess structure. In an office environment for instance desks, cabinets and other furniture are rarely moved. Thus they can be modeled as stationary objects. Typically the most dynamic aspects of the environment are people moving and other salient motions such as flying journal papers or changing the position of staplers that do not generally interfere with the task achievement of the robot. Another environment could be a warehouse

or an automated manufacturing facility where products and packages are moved to different positions repeatedly. Some equipment in a manufacturing facility are rarely moved and can also be considered stationary. We assume problem scenarios where a map of the environment is given. The GSR algorithm is describes in *table 5.1*. The algorithm simply prunes areas of the map that are not necessary for goal completion. It does this by selecting free cells in the rows of the grid that are both above and below the initial and goal states then prunes them from the map. It also does this for the columns of the grid by selecting columns before and after the initial and goal state then prunes them from the map. An illustration of this is shown in *figure 5.1*.



a b

Figure 5.1 – (a) A Map of a Task Environment Depicting the Initial and Goal State (b) A GSR-Map of the Task Environment in Red Outline

Table 5.1 – A Goal-Specific Representation (GSR) Algorithm

Goal-Specific Representation (GSR) Algorithm

input $\left(domain\_map(m \times n), (i, j)_{initial}, (i, j)_{goal}, (i, j)_{landmark}\right)$

1. $i \leftarrow [1:m]$
2. $j \leftarrow [1:n]$
3. **if** $(i, j)_{goal}$ is within $(i, j)_{landmark}$ *(checks if goal state is within landmark state)*
4. **then** $(i, j)_{goal} \leftarrow (i, j)_{landmark}$
5. **end if**
6. **for all** free cells in rows i *(scans all rows and delete free cells)*
7. **if** $i < i_{initial}$ and $i < i_{goal}$
8. **then** delete free cells in rows $1:i$
9. **end if**
10. **if** $i > i_{initial}$ and $i > i_{goal}$
11. **then** delete free cells in rows $i:m$
12. **end if**
13. **end for**
14. **for all** free cells in columns j *(scans all columns and deletes free cells)*
15. **if** $j < j_{initial}$ and $j < j_{goal}$
16. **then** delete free cells in columns $1:j$
17. **end if**
18. **if** $j > j_{initial}$ and $j > j_{goal}$
19. **then** delete free cells in columns $j:n$
20. **end if**
21. **end for**
22. **return** GSR_map

output (*GSR map*)

If a map is not given then the robot must actively explore the environment and generate a map. Popular techniques used to generate maps are *Simultaneous and Localization and Mapping* (SLAM) algorithms such as graphSLAM or EKF SLAM. We do not consider such cases in this work as we assume a map is given. Most applications of POMDP in robot motion planning and navigation discretize the task domain into uniform resolution grids and formulate policies over those states. Recall that physical environments are continuous both spatially and temporally and must be approximated using discrete models. As referred to, a related method to our approach is a variable resolution technique postulated by Kaplow, Atrash & Pineau (2010). The technique assigns aspects of the domain with larger grid sizes where there are open spaces and smaller grids sizes closer to walls. Also in Kurniawati et al. (2011), they use a method called *Milestone Guided Sampling* which plans over a compact set of states in state-space. As a consequence the sampled belief space is reduced. An example of a robot generated map where the proposed method can be applied is shown in *figure 5.2*. It is a map generated by a *Turtlebot* robot that explores an office domain.

Figure 5.2 – A Sample Map Generated by a Turtlebot (Courtesy of ROS.org)

We consider states of the environment that are represented as 2-dimensional discrete grids to approximate a continuous space 2D maps. *Figure 5.3* shows a sample of a robot generated maps with a grid overlay.

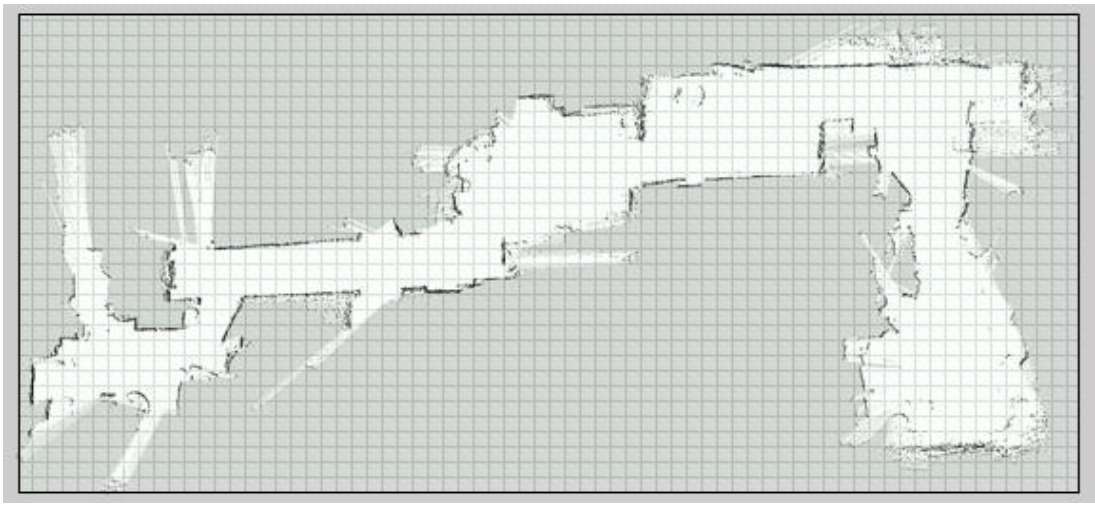


Figure 5.3 – A Sample Map Generated by a Turtlebot with a Grid Overlay

In practice as the scales of the task domain increase in size of the grid resolution decreases to maintain computational tractability since computing an efficient control policy tends to be infeasible for high resolution grids. A GSR-map can be generated for the *figure 5.3* if we know the initial location and the desired goal location. To find the initial location we may use state estimation technique such as Bayes filters or particle filters. An example of the corresponding GSR-map may look like *figure 5.4*. We use the locations on the GSR-map as well as the orientation as state inputs for the POMDP model along with a suitable transition function, observation function and reward function. We then solve the POMDP using a point-based solver and generate a control policy that guides the robot's actions. The

transition function, possible observations and observation functions are modeled based on the structure of the environment and the level of abstraction required.

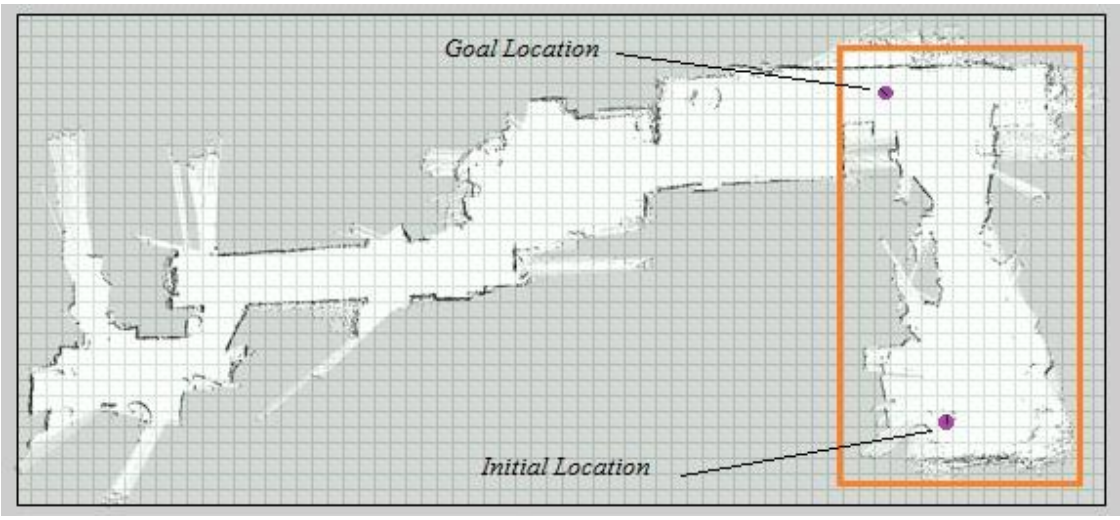


Figure 5.4 – A GSR-Map Generated by a Turtlebot with a Grid Overlay Depicting the Initial Location and Goal Location

The goal state is absorbing and receives the largest reward. Most POMDP models are fairly abstract in that they do not describe details of the control signals required to actuate the robot. It is intuitive to reason that low level controls depend on the type of robot and quality of its sensory hardware. Once we have a control policy we use it to guide the behavior on a real robot. The robot of choice is the Turtlebot 2® that runs on the Robot Operating System® (ROS). The experimental setup, analysis and discussion are described in the sections that follow.

5.3 A Sample Problem

In this section we describe a sample problem to understand the GSR methodology. The problem captures the essential features of a task environment.

The problem scenario can be applied to similar tasks in increasing order of complexity.

5.3.1 Problem Scenario

The problem scenario is as follows: *An autonomous robot is deployed in an automated manufacturing environment. Its goal is to efficiently transport products to one of eight specified collection locations in the facility as shown in figure 5.5. Only the initial location is specified since the robot has not been given a goal location.*

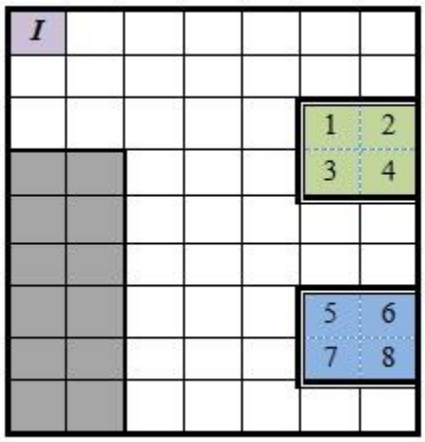


Figure 5.5 – A Grid Map of a Task Environment Depicting the Initial Location of the Robot

5.3.2 Problem Description

The robot starts at location s_{start} and arrives at a goal location s_{goal}. The map of the environment is decomposed into a finite set of uniform $m \times n$ grids where *m* is the number of rows of the grids while *n* is the number of columns. For instance location $s_{3,4}$ is the cell in the 3rd row and 4th column. If the task is to move the

product to location 3, then the goal location is either $s_{4,5}$ or $s_{5,6}$. Likewise if the task is to move the product to location 6 then the goal state is $s_{6,7}$.

The problem is as follows: Given a map of an environment decomposed into uniform grids, a starting state and a goal state, formulate the POMDP model such that only a subset of states required to reach the goal state is used for planning in the POMDP model. We call this a Goal-Specific Representation (GSR) of the POMDP model and the resulting state GSR-states. For instance suppose our goal state is location 3 with orientation 0^o then the map represented with GSR states are $\{s_{1,1}, s_{1,2}, ..., s_{5,6}\}$. The robot selects the goal state that is the most reachable state from its initial state. From the example it is $s_{4,5}$ from the set $\{s_{4,5}, s_{5,6}\}_{goal}$. In this specific problem we assume that the goal location in the front of collection site for 1, 3, 5 and 7. While the collection site 2, 4, 6 and 9 are located to side of the collection zone.

5.3.3 The GSR POMDP Model

The selection of the model parameter such as the transition function and observation function are done heuristically based on the capabilities of the robot. As alluded to these parameters are highly dependent on the type of robot adopted for the described task.

The GSR POMDP model can be described as $\langle S_{GSR}, A, T, R, Z, O, b_0 \rangle$

- **Set of States:** The set of states are the robot's configuration (x, y, θ) where $(x, y) \Rightarrow s_{i,j}$, $\forall i = 1, 2, ..., 9; j = 1, 2, ..., 7$. The robot has one of 8 possible orientation angles $\{0°, 45°, -45°, 90°, -90°, 135°, -135°, 180°\}$. With this configuration there

are a total of 504 possible states if obstacle cells are included and 344 possible states if obstacle cells are ignored. Once the GSR algorithm is applied the states are reduced to GSR states depending on the initial location and goal location. In the example described in 5.3.2 the GSR state are the grid cells $\{s_{ij} \mid \forall i = 1,...,4;\ j = 1,...,7\}$ where $s_{i,j} \neq s_{4,1}, s_{4,2}$ since they are not free cells along with all 8 orientations. This is illustrated in *figure 5.6b*. Thus the states are reduced to 144 possible states for the specific problem.

- **Set of Actions:** There are a total of 6 actions that the robot can perform. These actions are signals send to the motor to rotate at a certain angular velocity that corresponds to the desire length of travel. It can move forward with a heading of 0^o, 45^o, -45^o, left at 90^o and right at 90^o. The last is the *do_nothing* action.

  $\{a_1 = forward(0°), a_2 = forward(45°), a_3 = forward(-45°), a_4 = left(90°),$

  $a_5 = right(-90°),\ a_6 = do\_nothing\}$.

- **Transition Probability Function:** Since transitions in real domains are uncertain we model the transition function $T(s,a,s')$ with $p = 0.9$ success that the robot translates to the desired state when it performs action a and $p = 0.0125$ that it translates to any of the 8 possible adjacent states when it performs the same action a. There is a probability $p = 0$ for the all other states since it is safe to assume the robot cannot teleport.

- **Reward Function:** $R(s,a)$ = +10 for goal state and -0.05 for all others, bumping into wall is -1 and boundary states -2.

- **Set of Observations:** There are 3 possible observations the robot can make from it sensors. They are $\{z_1 = free, z_2 = obstacle, z_3 = goal\}$. We consider these observations to be independent of the robot orientation.
- **Observation Probability Function:** The robot observes the correct state probabilistically by $O(s', a, z)$. However we may model the observation function with just the subsequent state and ignore the actions that led to the subsequent state. That is, we have $O(z, a, s') = O(z, s'), \forall a \in A$. This further simplifies the model allowing for computational efficiency. If the subsequent state is free the robot observes $z_1 = free$ with $p = 0.9$ probability. The 2 other observations have the probability $p = 0.05$ of being observed. Due to sensory noise if the subsequent state has an obstacle such as a wall or machine, it is observed with probability $p = 0.85$ while the other 2 observations have a probability $p = 0.075$ respectively. Lastly the goal state is observed with $p = 0.95$ while the other 2 observations are observed with $p = 0.025$. This is shown in *table 5.2*.

Table 5.2 – Observation Probabilities

True State	Observation
Free	$z_1 = 0.90, z_2 = 0.05, z_3 = 0.05$
Obstacle (wall, machine etc.)	$z_1 = 0.075, z_2 = 0.85, z_3 = 0.075$
Goal	$z_1 = 0.025, z_2 = 0.025, z_3 = 0.95$

- **Initial Belief:** The initial belief is $b_0 = 1$ for the initial location. Since we assume the robot knows it initial location with certainty. We use the discount factor $\gamma = 0.95$ to model the desirability of future rewards.

Representing model parameters and data collection can be an extremely tedious task as the state space increases. Recall that all the data has to be enumerated since they are countably finite. For instance the transition probability function $T(s,a,s')$ can be generated as a lookup table with 1,524,096 entries. Luckily most of the entries are $p=0$ and can be ignored since only transitions to the 8 adjacent cells are considered. For a robot operating environment there is almost zero probability of being in a none-adjacent cell. This will only occur if there is a problem with the motor system and the robot fails to follow the motor commands. It may exhibits erratic runaway behavior. Also the observation probability function $O(z,a,s')$ can have as much as 9,072 entries. But since we have defined all observations as equally likely for every action the data is reduced. Likewise the reward function $R(s,a)$ will have 3024 entries if completely enumerated.

Some GSR maps are illustrated in *figure 5.6*. Notice that all the GSR maps outlined in red have at least 2 corner grid cells that are either the goal state or initial state. This tightly binds the number of states required for planning. Recall the original domain map has 504 states. The corresponding GSR state for the maps in *figures 5.6b*, *5.6d*, *5.6f* and *5.6h* is shown in *table 5.3*.

Table 5.3 – States of the Domain Map and GSR Map

	Domain Map	**GSR MAPs**
Map 1	$\|S\|=344$	$\|S\|=144$
Map 2	$\|S\|=344$	$\|S\|=216$
Map 3	$\|S\|=344$	$\|S\|=216$
Map 4	$\|S\|=344$	$\|S\|=248$

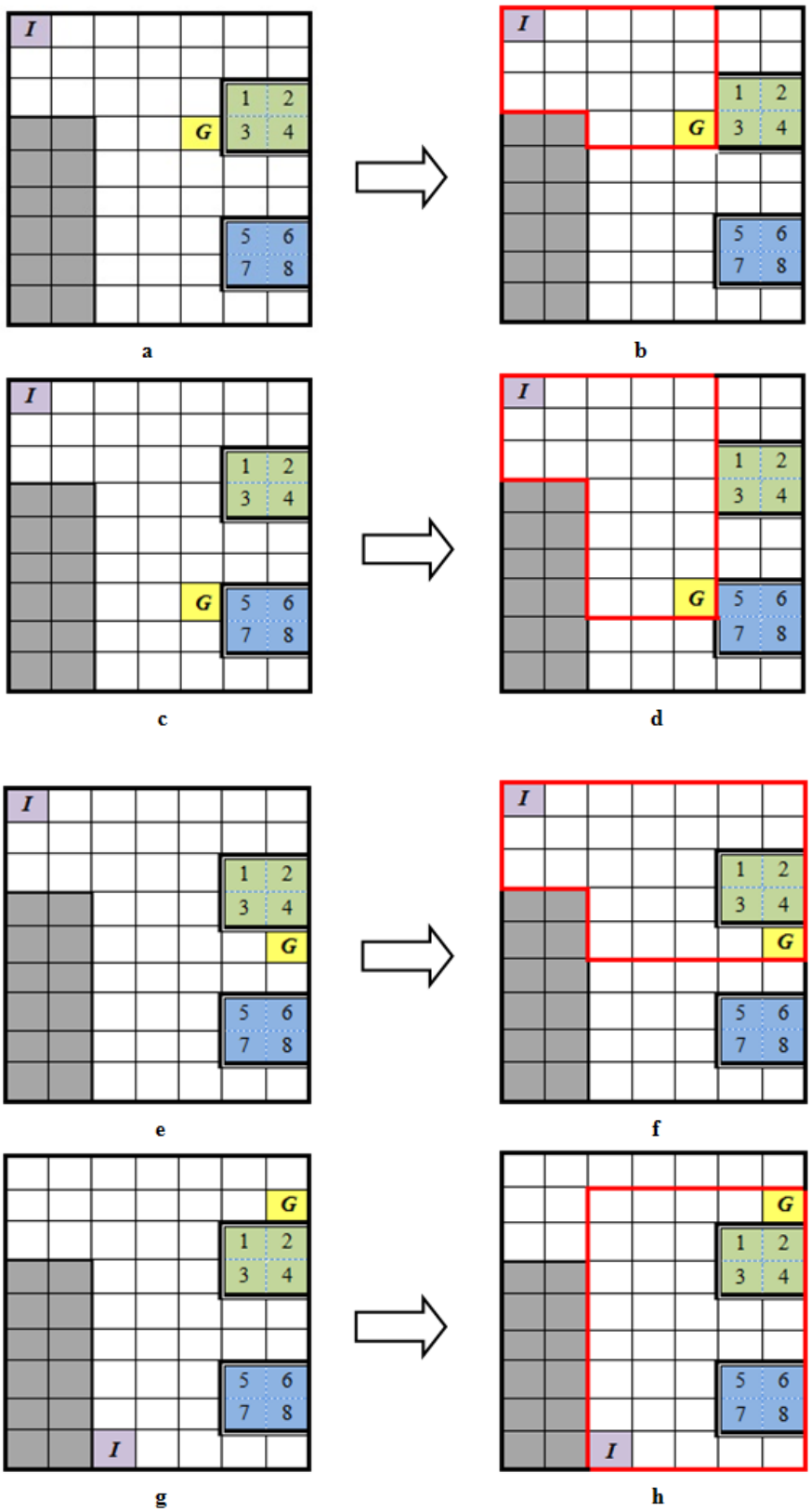


Figure 5.6 – Sample GSR Maps of the Domain Map

5.4 Empirical Analysis

In this section we describe the problem used to conduct experiment. The domain selected is adapted from the *Hallway2* layout present by Littman, Cassandra & Kaelbling (1995). The layout has served as a benchmark problem for POMDP research. We call the adapted model *automation* since we are interested in a manufacturing layout. We create corresponding GSR maps for the environment and plan over the GSR maps.

5.4.1 POMDP Model for the Automation Problem

The layout for the automation problem is shown in *figure 5.7*. In the automation problem, the robot is given some parts at an initial location in the domain map (grid cell with green robot).

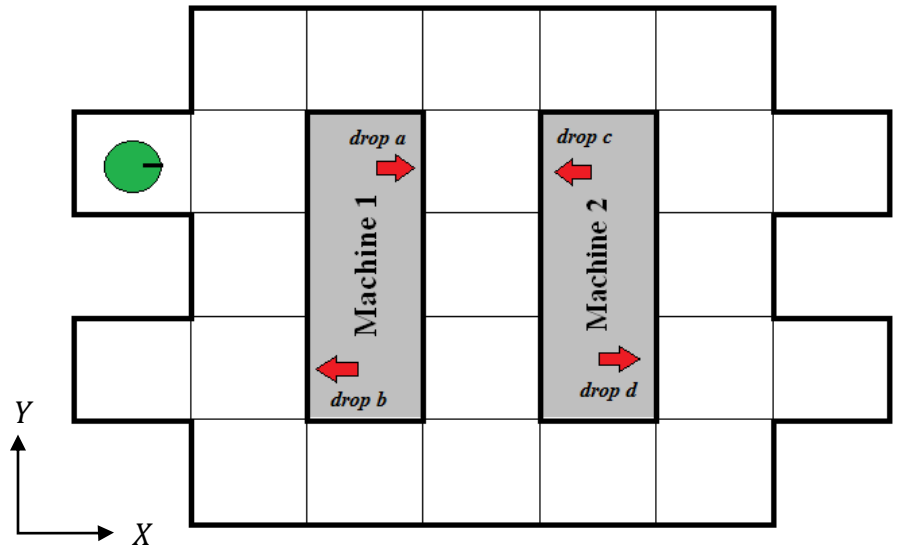


Figure 5.7 – Domain Map for the Automation Problem (Layout Adapted from Littman et al., 1995)

Its task is to efficiently move the part from this initial location to one of 4 drop off locations. The drop off location must be specified to the robot *a priori*. Based on the

drop off location the robot must autonomously plan a route to the desired location given the domain map and a good estimate of its initial location. Although it is possible to have numerous possible orientations for computational reasons we assume that the robot is in one of 4 possible orientations.

On the implemented solver they are $(0°, -90°, 90°, 180°)$ where the $0°$ orientation is parallel to the *Y-axis* and the states in each grid are indicated clockwise. The robot can perform one of 5 possible actions at a time in one of the 4 orientations $(do\_nothing, forward, right, left, rotate(180°))$. The robot all has 4 sensors in each of its quadrants which can detect the presence of an obstacle by registering an on-off switch. All 4 sensors operate at a given time so the robot can make a total 16 possible observations. Also the robot observes a signal when it has arrived at its goal location. It receives a reward only when it has arrived at the designated drop off location. Since the robot cannot know precisely where it is except for the initial location and goal location, it must maintain a belief of the current state throughout its navigation. The problem is to provide a policy to guide the robot's actions based on its beliefs. For the 4 drop off locations we created GSR maps that the robot uses to guide action selection. The model parameters are as follows:

POMDP model automation: $\langle S, A, T, R, Z, O, b_0 \rangle$:

- **Set of States:** $|S| = 92$ for 4 possible orientations in each grid cell.
- **Set of Actions:** $|A| = 5$ $(do\_nothing, forward, right, left, rotate(180°))$
- **Set of Observations:** $|Z| = 17$ one for each combination of the 4 quadrant sensors plus the observation of the goal state.

- **Transition Probability Function:** The transition $T(s,a,s')$ between states is approximately $p=0.7$ in the desired direction and $p=0.1$ in the 3 other primary adjacent directions. There is also a small probability of translating to a different orientation depending on the current orientation. See *appendix A* on where a complete specification of the transition probability can be found.
- **Observation Probability Function:** The robot observes the correct state given all the actions at approximately $p=0.73$. The distribution for all other states are according to how close the observation made is to the actually state. Again see *appendix A* where a complete specification can be found.
- **Reward Function:** The robot receives no reward until it has arrived it location. The reward $R(s,a)=1$ for the *correct* orientation in the goal state. This is important since the robot may be in the correct location but in a wrong orientation.
- **Initial Belief:** The initial belief is $b_0=1$ for the starting state.
- **Discount Factor:** The discount factor is $\gamma=0.95$

5.4.2 The GSR Maps of the Automation Problem

We now describe and illustrate the GSR maps for the automation problem. As discussed in *section 5.3,* the domain maps are generated as a grid of size (5×7). The cells $(1,1),(1,7),(3,1),(3,7),(5,1),(5,7)$ are automatically deleted from the domain map to resemble domain in *figure 5.7*. The resulting domain map is generated and the states for each map are labeled as shown in *figure 5.8*. Labels 1-4 represents the

location of the robot plus each of the 4 orientations. However when solving the POMDP the syntax require that we label from 0-91 (still 92 total states). Due to the configuration of the domain we place a landmark at a grid cell that generates a map that is navigable.

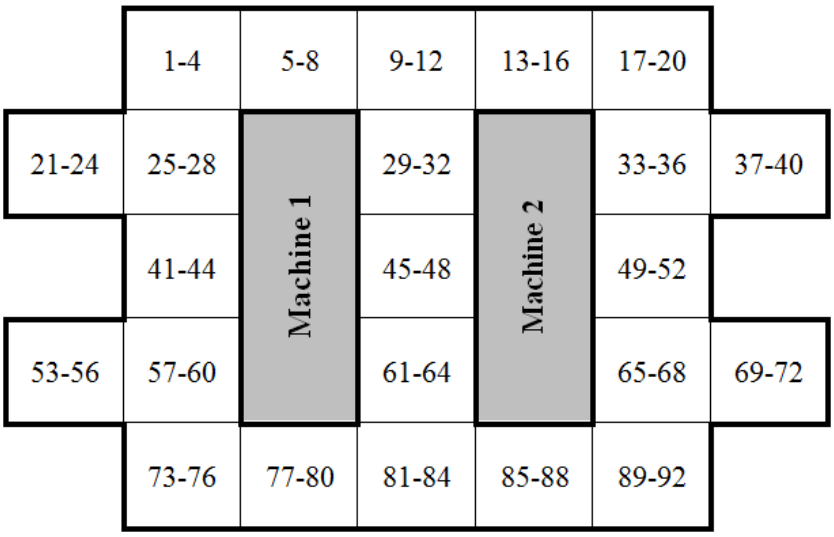
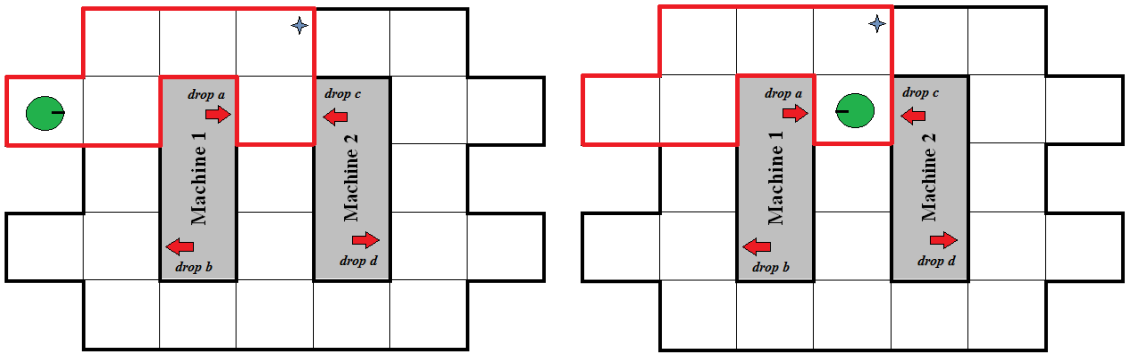


Figure 5.8 – State Labels for the Automation Domain Map

We now illustrate each of the GSR maps based on the goal location in *figures 5.9-5.12*.

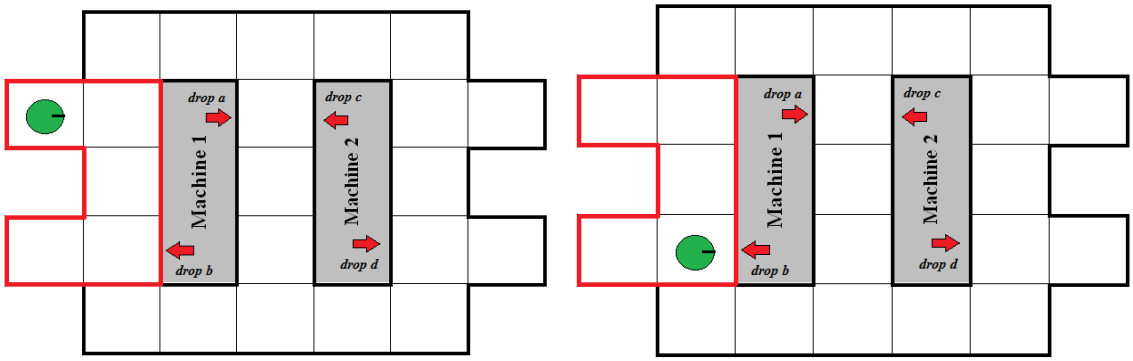
- **<u>Drop Location A:</u>**



a b

Figure 5.9 – GSR Map for Drop Location A

- **Drop Location B:**

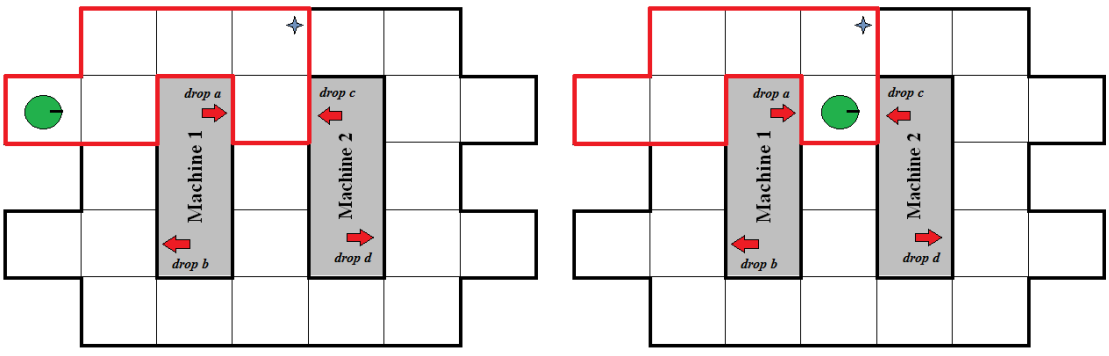


a b

Figure 5.10 – GSR Map for Drop Location B

Notice that in *figure 5.9* and *5.11* are similar with the exception of the orientation of the robot. In drop location A the robot must be oriented at 180° (w.r.t. the global reference frame) in grid $(2,4)$ while it has to be oriented at 0° in the same grid location for drop location C.

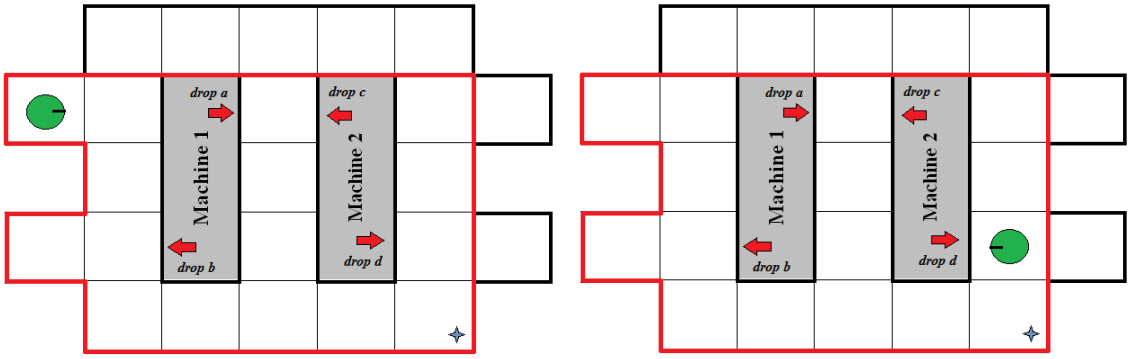
- **Drop Location C:**



a b

Figure 5.11 – GSR Map for Drop Location C

- **Drop Location D:**



a b

Figure 5.12 – GSR Map for Drop Location D

In *figure 5.9* and *5.12* we had to add a landmark as described in the GSR algorithm. If we did not do so the generated map will not be navigable as shown in *table 5.1*. The landmark is depicted with a blue star.

5.4.3 Solving the GSR POMDP

To solve the GSR POMDP model, we used a point-based POMDP solver called **APPL ver. 0.95**. It is based on the paper by Kurniawati, Hsu & Lee (2008) and Ong, Png, Hsu & Lee (2009). This solver was selected since policies are solved over the set of optimal reachable belief points $\Re^*(b_0)$ called the optimal reachable belief space rather than the set of reachable belief points $\Re(b_0)$. Both are subsets of the belief space *B*. The solver significantly improves the computational efficiency of solving POMDPs when compared to other existing solvers. APPL ver. 0.95 is based on **ZMDP** solver by Smith (2007). Both the APPL and ZMDP are based on the **POMDP-Solve ver. 5.3** software and use file format developed by Cassandra

(1998). Details on the discussed software can be found at the websites listed in *table 5.4*. The APPL solver was implemented on a PC that runs a 64-bit Linux-based OS Ubuntu 12.04 LTS (Precise) with a 2.4 GHz Intel® Duo Core processor and 4GB of RAM. The **GapMin** solver is one of the latest solvers that run on MATLAB. It is based on the paper written by Poupart, Kim & Kim (2011). The authors present an algorithm for improving the bounds on the upper and lower bound of the optimal value function of the POMDP. We do not apply this solver in this problem.

Table 5.4 – Websites for POMDP Solvers

APPL ver. 0.95	http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl
ZMDP ver. 1.1.7	http://www.cs.cmu.edu/~trey/zmdp
POMDP-Solve ver. 5.3	http://www.pomdp.org/pomdp/code
GapMin ver. 2011-06-13	https://cs.uwaterloo.ca/~ppoupart/software.html

5.4.4 Turtlebot 2 Robot

The *Turtlebot 2* is an innovative platform developed by Willow Garage Inc. and distributed by Clearpath Robotics Inc. It is shown in *figure 5.13*. We selected the turtlebot because it provides a state-of-the-art robotic research platform with advanced sensory capabilities at an affordable price. It is a highly suitable platform for autonomous navigation research. Also its performance is sufficiently adequate in undertaking a diverse range of other robotics related research. It is designed for an indoor environment and will be used to demonstrate proof of concept for our algorithm. The specifications are shown in *table 5.5*. It is important to note that the

GSR-POMDP model is solved offline and the generate policies are based on the defined model parameters. The solved policy is simply implemented on the robot.

Figure 5.13 – A Turtlebot 2 Robot

The turtlebot 2 runs on ROS which is an open-source meta-operating system developed on a Linux-based platform such as Ubuntu 12.04 LTS (Precise). It offers similar capabilities that traditional operating systems do, which include the direct control of low-level device, hardware abstraction, libraries, code building, multiple computer networking and a host of other functionalities. Details of the functionality of ROS for robotic devices can be found at www.ros.org.

An autonomous navigation task was performed on the turtlebot in a home environment. *Figure 5.14* illustrates a screenshot of a path travelled. The cluster of coordinate axes indicates the robot's position estimate, while the coordinate on the bottom toward the left indicates the global reference frame. The coordinate toward the bottom right indicates the robot's goal location. We selected 2 destination

points and it performed the navigation task with relative success. Curiously, it sometimes exhibited a rotating behavior when assigned to its goal location. We could not immediately ascertain the cause of this. Further inquiry will be conducted to understand what necessitated such behaviors. It may perhaps be a mechanical issue. Such behavior further highlights the need to model the uncertainties in the robot's action and perception. On a positive note, two goal locations that were selected were eventually successfully navigated by the robot. The robot achieved this task at a speed of roughly 20 cm/sec. Our preliminary results on the turtlebot are promising. It is important to note that there is a navigation package for the turtlebot that could support other motion planning methods.

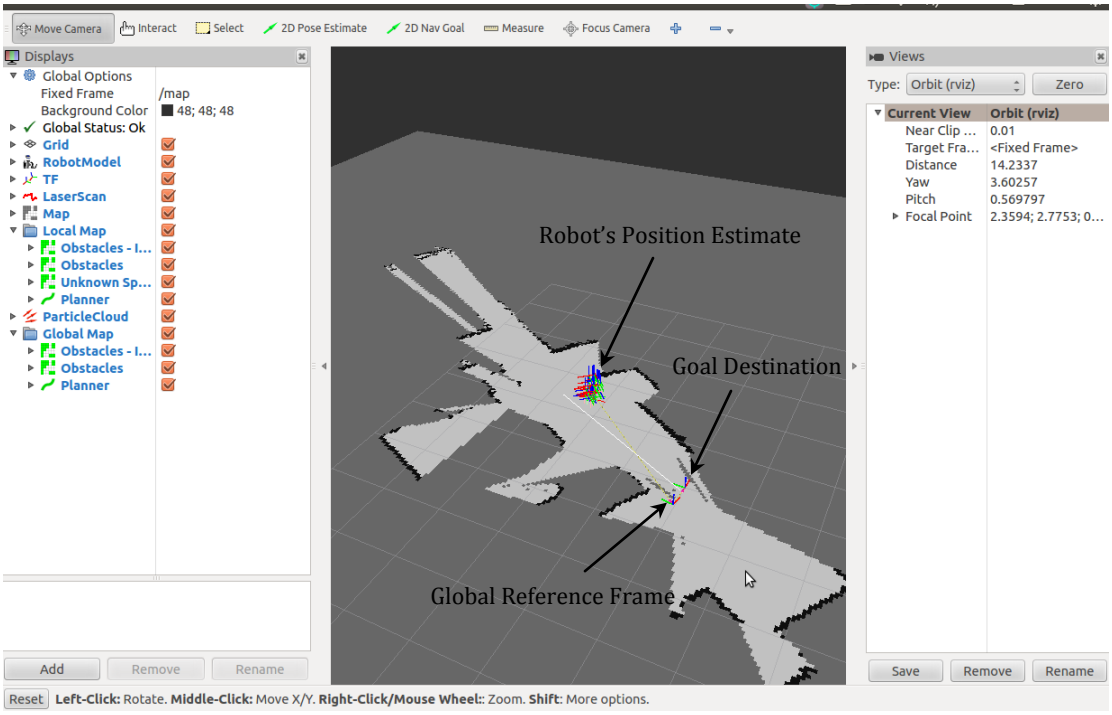


Figure 5.14 – An Autonomous Navigation Task on the Turtlebot

Table 5.5 – Turtlebot Specifications

Turtlebot Specifications	
Dimensions	35.4 cm x 35.4 cm x 42.0 cm
Wheels	7.6 cm diameter
Weight	6.3 kg
Ground Clearance	15 cm
Max Speed	65 cm/s
Max Rotational Speed	180 deg/s
Max Payload	5 kg
Battery	2200 mAh Li-Ion (with 4400 mAh extended)
User power	5V & 19V @ 1A, 12V @1.5A, 12V @5A
3D Vision Sensor	Microsoft Kinect
Camera Color	640 x 480 px, 30 fps
Depth Camera	640 x 480 px, 30 fps
Encoders	25700 cps; 11 ticks/mm
Gyroscope	100 deg/s
Bumper Sensor	3x forward
Other Sensor	2x cliff sensor and 2x wheel drop
Computer	ASUS with 1.6GHz Intel Atom N2600 processor and 1GB RAM
Operating System	Robot Operating System (ROS®) running on Ubuntu 12.04 LTS (Precise)

5.5 Results and Discussion

A snapshot of the results from the experiments is described in *table 5.6*. We follow up with a detailed discussion of the finding.

Table 5.6 – Results of GSR-Model Implementation

	GSR POMDP Model	Expected Reward	Execution Time (secs)
Task 1	States $\|S\|$: 92 GSR States $\|S\|_{GSR}$: 24 Initial location: $((2,1),0°)$ Goal location: $((2,4),180°)$ Actions $\|A\|$: 5 Observations $\|Z\|$: 17	(100 Simulations) Domains States: 0.662648 GSR States: 0.837799 (200 Simulations) Domains States: 0.691717 GSR States: 0.833081	Domain States: 300.79 GSR States: 300.71
Task 2	States $\|S\|$: 92 GSR States $\|S\|_{GSR}$: 20 Initial location: $((2,1),0°)$ Goal location: $((4,2),0°)$ Actions $\|A\|$: 5 Observations $\|Z\|$: 17	(100 Simulations) Domains States: 1.090130 GSR States: 2.732290 (200 Simulations) Domains States: 1.082360 GSR States: 2.748350	Domain States: 302.4 GSR States: 300.65
Task 3	States $\|S\|$: 92 GSR States $\|S\|_{GSR}$: 24 Initial location: $((2,1),0°)$ Goal location: $((2,4),0°)$ Actions $\|A\|$: 5 Observations $\|Z\|$: 17	(100 Simulations) Domains States: 0.653846 GSR States: 0.949199 (200 Simulations) Domains States: 0.685472 GSR States: 0.943234	Domain States: 301.95 GSR States: 301.57
Task 4	States $\|S\|$: 92 GSR States $\|S\|_{GSR}$: 64 Initial location: $((2,1),0°)$ Goal location: $((4,6),180°)$ Actions $\|A\|$: 5 Observations $\|Z\|$: 17	(100 Simulations) Domains States: 0.567472 GSR States: 0.681748 (200 Simulations) Domains States: 0.558172 GSR States: 0.684719	Domain States: 301.8 GSR States: 301.13

5.5.1 Data Analysis

In this section we analyze the result from the experiments. The data utilized had to be configured to the *filename.pomdp* format. *Appendix A* directs the reader to where this format can be located. *Figure 5.15* illustrate the state labels according to this format.

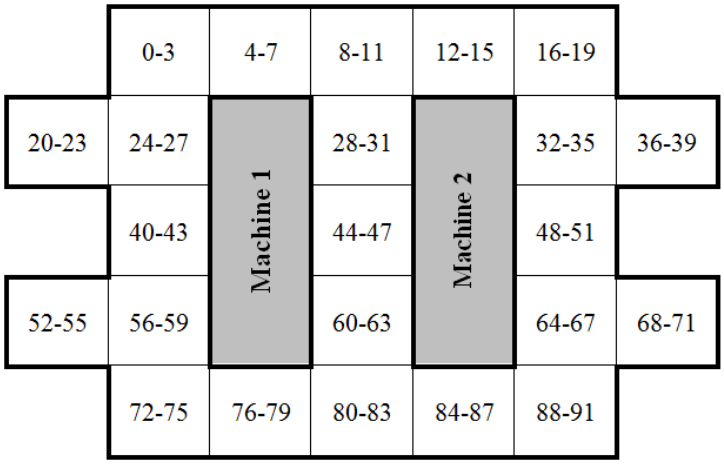


Figure 5.15 – Domain Map with Labels in the *filename.pomdp* syntax

Recall that the goal of the GSR approach is to reduce the size of the state space with the expectation that computational efficiency will be improved. To accurately compare the results of the policies generated from all 8 maps (4 for the domain map state and 4 for the GSR map), we ran the solver for approximately the same amount of time (300 seconds). It is possible that the solver run indefinitely if the level of precision between updates in reduced. The resulting α-vectors, number of beliefs sampled, number of backups and the upper and lower bounds on the optimal expected value (rewards) were recorded. Details of the results can be found in *Appendix B*. The α-vectors, beliefs, backups and expected rewards were compared for all 8 policies generated. The generated policies were evaluated with 100 and

200 simulations and the expected cumulative rewards (value) for both simulations were documented. *Table 5.7* outlines the results that were compared. We also averaged the expected rewards from both simulations. We found some interesting results from the comparisons as illustrate in *figures 5.16, 5.17, 5.18 and 5.19*. The first were from the α-vectors comparison in *figure 5.16*. At first glance one might interpret from the chart that an excessive number of α-vectors we generated for the GSR maps. This is simply due to the fact that a large number of value iteration backups were performed as illustrated in *figure 5.18*.

Table 5.7 – Results from POMDP Solver with Expected Cost

Maps	#Alphas	#Beliefs	#Backups	Exp Rewards (100 Sim)	Exp Rewards (200 Sim)	Exp Rewards (Avg.)
Domain 1	181	1671	14233	0.662648	0.691717	0.677183
GSR 1	484	1719	22138	0.837799	0.833081	0.835440
Domain 2	251	1603	14321	1.090130	1.082360	1.086245
GSR 2	491	1963	19720	2.732290	2.748350	2.740320
Domain 3	266	1651	13597	0.653846	0.685472	0.669659
GSR 3	475	1771	23317	0.949199	0.943234	0.946217
Domain 4	229	1432	16668	0.567472	0.558172	0.562822
GSR 4	441	1409	17933	0.681748	0.684719	0.683234

Also recall that the solver ran for virtually the same time as indicated in *table 5.6* so the large number of value iteration backups cannot be due to longer computation time for the GSR-based maps. This observation suggests that the GSR model performed as anticipated. Notice that quantity of beliefs generated were relatively the same even for the reduced GSR states. A more careful observation will only

imply a marginal increase. This is likely due to that fact that the APPL solver selects beliefs over an optimal reachable belief space and not the entire belief space.

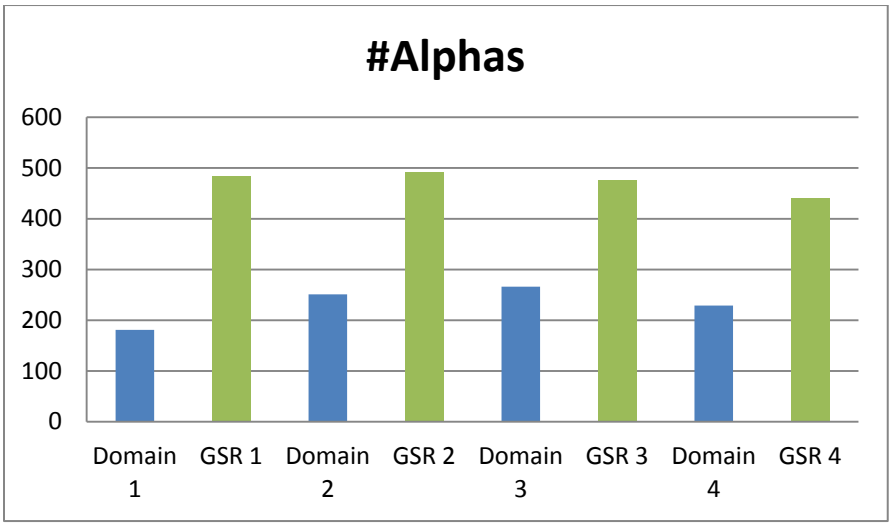


Figure 5.16 – Chart Comparing the Number of *α-vectors* Generated

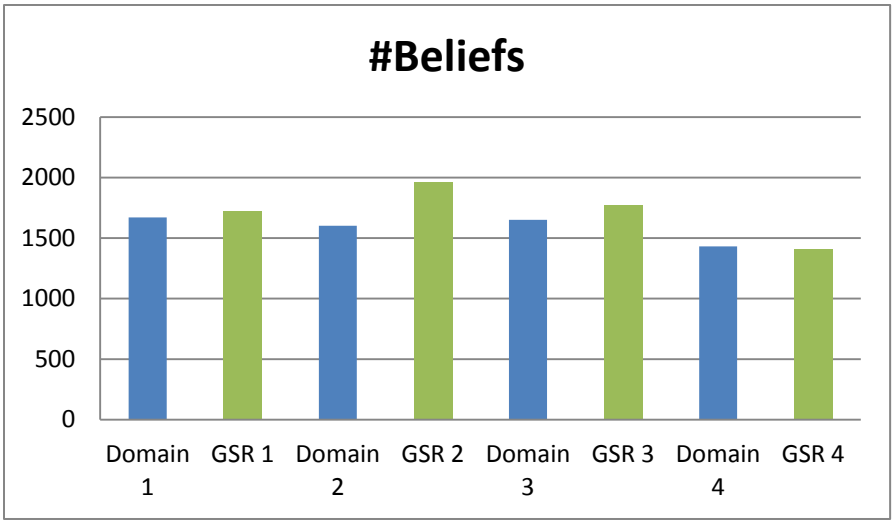


Figure 5.17 – Chart Comparing the Number of Belief Generated

The number of beliefs generated for the GSR map for task 2 were the largest due to the fact that the number of states in the GSR map is considerably smaller thus the corresponding belief space for generating a policy is significantly reduced. This is

exactly what we expected to observe and why the GSR methodology is proposed as an alternative to the complete state representation of model parameters.

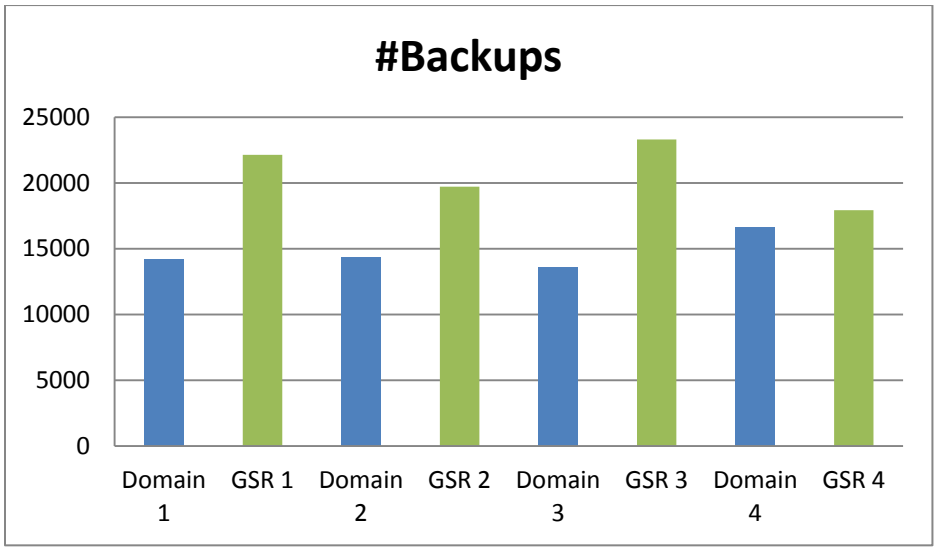


Figure 5.18 – Chart Comparing the Number of Backups Computed

Figure 5.19 – Chart Comparing the Expected Rewards from the Policy Evaluation

There was no significant difference in the expected rewards for the 100 simulations and 200 simulations run. The GSR map for task 2 has greater rewards because the number of initial states and the goal state are very close together as shown in *figure*

5.9. The upper and lower bounds on the expected cumulative rewards can be found in *Appendix B*. The number of value iteration backups for the GSR task 1 and GSR task 2 are very close. They are 22138 backups and 23317 backups respectively. This is because they have very similar configurations as shown in *figure 5.9* and *figure 5.11*. In fact the only difference is that their goal state has different orientations.

5.6 Summary

In summary, we presented an approach called Goal-Specific Representation (GSR) to reduce the state space required for planning with POMDPs. Results were encouraging and demonstrated that this methodology is viable for complex domains. A drawback is that the policies generated still plan over a relatively short horizon as the solver quickly runs out of memory. Solving long horizon navigation problems using POMDPs is still considerably difficult since the time complexity is exponential in the horizon length. We shall look into applying a non-uniform grid model to further reduce the model parameters and generate more efficient solutions. We also applied the policy computed offline to the Turtlebot 2 robot platform for autonomous navigation. The turtlebot successfully navigated an indoor environment. Future work is aimed at applying the model directly in complex 3-dimensional domains and also in real manufacturing facilities. Of course doing so will be more computationally tasking but exploiting the structure of the domain should significantly reduce the problem parameter as we have successfully demonstrated.

Chapter 6

Summary and Future Work

6.1 Overview

In this dissertation we thoroughly discussed sequential decision-making for autonomous systems in partially observable environments. We were particularly interested in applying the Partially Observable Markov Decision Process (POMDP) model in the robot motion planning and navigation problem. For decades there has been significant interest in designing robotic systems that can operate autonomously in uncertain and dynamic environments. A major hindrance is that modeling robot-environment interactions is still a difficult problem. If current research methods prove successful, it is anticipated that potential application domains include healthcare for assisted-care of the elderly, hazardous material management, search & rescue, autonomous material handling etc.

A POMDP model—a generalization of Markov Decision Processes—is a robust theoretical formalism that explicitly accounts for uncertainty of an agent's action and observation as it interacts within its task environment. This ensures its suitability in modeling real-world problems that are intrinsically stochastic by nature. A major drawback with adopting POMDPs for decision-making is that it is computationally intractable to solve problems with large state space. Researchers have addressed this problem by formulating approximate techniques resulting in

solution to problems with state space in orders of magnitude large than previously possible.

We formulated a novel approach to reduce the state space of the POMDP parameters by exploiting the structures of the problem which include the method by which the domain is modeled, the goal location within this domain and the type of action and observation the agent can perform. This was applied in robot navigation problem to generate control policies that guide the robot to autonomously completing a described task with a specific domain. We modeled the domain as an automated manufacturing facility. We called the proposed methodology Goal-Specific Representation (GSR) which reduces the state-space to only states reachable from the initial location to the goal location. This ensured that we computed policies over only GSR states thus reducing computational costs. We then solved the GSR model using a point-based POMDP solver called APPL v 0.95 and performed empirical analysis and evaluation on the solution it generated. We also applied to the policy solved offline to navigate a real robot.

6.2 Limitations

Though our approach to planning for autonomous systems where the operating domain is stochastic has been shown to be useful, there are still some disadvantages that have been observed. The first and most obvious is that we do not generate policies for the complete task environment which may be undesirable in certain cases. The second is that the generated policies are currently limited by the precision and efficiency of the latest state-of-the-art POMDP solvers. Developing

better solvers is still an active area of research. The third limitation in this work is that the model parameters described are relatively high-level in that the effects of the robot design specifications are abstracted out of the model. This may be important to consider in future work. Also one issue rarely discussed in literature which we encountered is the prohibitively large data set required as input parameters for the POMDP model even for smaller models. This may require further research to investigate more efficient means of representing the problem parameters.

6.3 Future Work

We have learned important lessons in this dissertation on how to efficiently model decision-making tasks for complex autonomous systems where the environment is partially observable. We were interested in the autonomous robot navigation problem even though the models can be extended to other domains such as medical decision-making, computer network management, metropolitan city evacuation, and computational biology just to name a few. In the navigation task described we only discussed the scenario where the robot is operating in a 2-dimensional environment. Further research will be conducted to explore 3-D dimensional environments. Clearly the state space of the problem will significantly increase in 3D so the GSR methodology will provide considerable utility in taking advantage of the domain structures. We deliberately did not present a real automated facility in order to focus on the decision-making model. Ultimately we anticipate that our contributions in this work can be applied in industrial and

manufacturing environments with teams of fully autonomous robots performing tasks with minimal human intervention.

References

1. **Alagoz, O., Hsu, H., Schaefer, A. J., Roberts, M. S**. (2010). *Markov Decision Processes: A Tool for Sequential Decision Making under Uncertainty*, Medical Decision Making, Vol. 30, No. 4, pp 474-483.

2. **Allen, M.**, **Petrik, M., Zilberstein, S.** (2008). *Interaction Structure and Dimensionality Reduction in Decentralized MDPs*, In Proceedings of the 23rd AAAI Conference on Artificial Intelligence.

3. **Alterovitz, R. Siméon, T., Goldberg, K.** (2007). *The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty*, In Proceeding of Robotics: Science and Systems, pp 246-253.

4. **Amato, C., Bonet, B., Zilberstein, S.** (2010). *Finite-State Controllers Based on Mealy Machines for Centralized and Decentralized POMDPs*, In Proceedings of the 24th Conference on Artificial Intelligence.

5. **Aurenhammer, F.** (1991). *Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure*, ACM Computing Surveys, Vol. 23, No. 3, pp 346 – 405

6. **Bar-Shalom, Y., Li, X-R.** (1998). *Estimation and Tracking: Principles, Techniques and Software*, YBS, Danvers, MA.

7. **Barto, A., Bradtke, S., Singh, S.** (1995). *Learning to Act using Real-Time Dynamic Programming*, Artificial Intelligence, Vol. 72, pp 81-138.

8. **Bellman, R.** (1957a). *A Markovian Decision Process*, Journal of Mathematics and Mechanics, Vol. 6, No. 5, pp 679- 684.

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

10. **Bertsekas, D. P.** (2007). *Dynamic Programming and Optimal Control Vols. 1 & 2*, Athena Scientific, Belmont, MA.

11. **Bertsekas, D. P., Shreve S. E.** (1996). *Stochastic Optimal Control: The Discrete-Time Case*, Athena Scientific, Belmont, MA.

12. **Bertsekas, D. P., Tsitsiklis, J.** (1996). *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA.

13. **Bhati, A., Kavraki, L. E., Vardi, M. Y.** (2010). *Sampling-Based Motion Planning with Temporal Goals*, In Proceedings of the International Conference on Robotics and Automation.

14. **Bhatia, A., Maly, M. R., Kavraki, L. E., Vardi, M. Y.** (2011). *A Multi-Layered Synergistic Approach to Motion Planning with Complex Goals*, IEEE Robotics & Automation Magazine, Vol. 18, No 3, pp 55-64.

15. **Bhatnagar, S., Fernández-Gaucherand, E., Fu, M. C., He, Y., Marcus, S. I.** (1999). *A Markov Decision Model for Capacity Expansion and Allocation*, In Proceedings of the 38th Conference on Decision and Control, Phoenix, AZ, pp 1380-1385.

16. **Candido, S. Hutchinson, S.** (2011). *Minimum Uncertainty Robot Navigation using Information-Guided POMDP Planning*, In Conference Proceedings of IEEE International Conference on Robotics and Automation, pp 6102-6108.

17. **Candido, S., Hutchinson, S.** (2011). *Minimum Uncertainty Robot Navigation using Information-Guided POMDP Planning*, In Proceedings of the International Conference on Robotics and Automation, pp 6102-6108.

18. **Cassandra, A. R.** (1994). *Optimal Policies for Partially Observable Markov Decision Processes*, MS Thesis, Computer Science Department, Brown University, Providence, RI.

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

20. **Cassandra, A. R., Kaelbling, L. P., Kurien, J. A.** (1996). *Acting Under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation*, In Proceedings of the International Conference on Intelligent Robots and Systems, Vol. 2, pp 963-972.

21. **Cassandra, A., Littman, M. L., Zhang, N. L.** (1997). *Incremental Pruning: A Simple Fast, Exact Method for Partially Observable Markov Decision Processes*, In Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence, pp 54-61.

22. **Doshi-Velez, F.** (2009). *The Infinite Partially Observable Markov Decision Process*, Advances in Neural Information Processing Systems (NIPS), Vol. 22, pp 477-485.

23. **Eker, B., Akin, H. L.** (2013). *Solving Decentralized POMDP Problems using Genetic Algorithms*, Autonomous Agents and Multi-Agent Systems, Vol. 27, No. 1, pp 161-196.

24. **Fodor, I. K.** (2002). *A Survey of Dimension Reduction Techniques*, Technical Report, Center of Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA.

25. **Glashan, R., Hsiao, K., Kaelbling, L. P., Lozano-Pérez, T.** (2007). *Grasping POMDPs: Theory and Experiments*, In Proceedings of Robotics: Science and Systems Workshop on Manipulation for Human Environments.

26. **Gordon, G. J.** (1999). *Approximate Solutions to Markov Decision Processes*, PhD Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA

27. **Grady, D., Moll, M., Kavraki, L. E.** (2013). *Automated Model Approximation for Robotic Navigation with POMDPs*, In Proceedings of the International Conference on Robotics and Automation.

28. **Guo, X., Hernández-Lerma, O.** (2009). *Continuous-Time Markov Decision Processes*, Springer-Verlag, Berlin.

29. **Guo, X., Zhu, W.** (2002). *Denumerable-State Continuous-Time Markov Decision Processes with Unbounded Transition and Reward Rates under the Discounted Criterion*, Journal of Applied Probability, Vol. 39, No. 2, pp 233-250.

30. **Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., Nordlund, P. J.** (2002). *Particle Filters for Positioning, Navigation,*

and Tracking, IEEE Transactions on Signal Processing, Vol. 50, No. 2, pp 425-437.

31. **Hansen, E. A.** (1997). *An Improved Policy Iteration Algorithm for Partially Observable MDPs*, In Proceedings of 10th Neural Processing Systems, Denver, CO.

32. **Hansen, E. A.** (1997). *An Improved Policy Iteration Algorithm for Partially Observable MDPs*, In Proceedings of the 10th Neural Information Processing Systems Conference, Denver, CO.

33. **He, R., Roy, N.** (2009). *Efficient POMDP Forward Search by Predicting the Posterior Belief Distribution*, Technical Report MIT-CSAIL-TR-2009-044, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

34. **Hornung, A., Phillips, M., Jones, E. G., Bennewitz, M., Likhachev, M., Chitta, S.** (2012*). Navigation in Three-Dimensional Cluttered Environments for Mobile Manipulation*, In Proceedings of the IEEE International Conference on Robotics and Automation, pp 423-429.

35. **Howard, R. A.** (1960). *Dynamic Programming and Markov Processes*. MIT Press, Boston, MA.

36. **Hsu, D., Lee, W. S., Rong, N.** (2007). *What makes some POMDP Problems easy to Approximate*, Advances in Neural Information Processing Systems (NIPS).

37. **Hunt, F. Y.** (2005). *Sample Path Optimality for a Markov Optimization Problem*, Stochastic Processes and their Applications, Vol. 115, pp 769-779.

38. **Ibekwe, H. I., Kamrani, A. K.** (2008). *Robotics and Autonomous Robots*, In book *Collaborative Engineering*, Eds. **Kamrani, A. K., Nasr, E. A.**, pp 173-206, Springer Science+Business Media, NY.

39. **Kaelbling, L. P., Littman M. L., Cassandra, A. R.** (1998). *Planning and Acting in Partially Observable Stochastic Domains*, Artificial Intelligence, Vol. 101, pp 99-134.

40. **Kaplow, R.** (2010). *Point-Based POMDP Solvers: Survey and Comparative Analysis*, MSc Thesis, Department of Computer Science, McGill University, Montreal, Quebec, Canada.

41. **Kaplow, R., Atrash, A., Pineau, J.** (2010). *Variable Resolution Decomposition For Robotic Navigation Under a POMDP Framework*. In Proceedings of the International Conference on Robotics and Automation (ICRA), Anchorage, AK.

42. **Kavraki, L. E., Švestka, P., Latombe, J.-C., Overmars, M. H.** (1996). *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*, IEEE Transactions on Robotics and Automation, Vol. 12, No. 4, pp 566-580.

43. **Koenig, S., Simmons, R. G.** (1998). *Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models*, In book *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, Eds. **Kortenkamp, D., Bonasso, R. P., Murphy, R. R.**, pp 91-122, MIT Press, Cambridge, MA.

44. **Kurniawati, H., Bandyopadhyay, T., Patrikalakis, N.** (2012). *Global Motion Planning under Uncertain Motion, Sensing, and Environment Map*, Autonomous Robots, Vol. 33, pp 255-272.

45. **Kurniawati, H., Du, Y., Hsu, D., Lee W. S.** (2011). *Motion Planning under Uncertainty for Robotic Tasks with Long Time Horizons*, International Journal of Robotics Research, Vol. 30, No. 3, pp 308-323.

46. **Kurniawati, H., Hsu, D., Lee W. S.** (2008). *SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces*, In Conference Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland.

47. **Kurniawati, H., Patrikalakis, N. M.** (2013). *Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models*, In Proceedings on the 10th Workshop on the Algorithmic Foundations of Robotics, pp 493-509.

48. **Latombe, J.-C.** (1991). *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA.

49. **LaValle, S. M.** (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*, Technical Report 98-11, Computer Science Dept., Iowa State University, IA.

50. **LaValle, S. M.** (2006). *Planning Algorithms*, Cambridge University Press, New York, NY.

51. **LaValle, S. M., Kuffner, J. J.** (2001). *Randomized Kinodynamic Planning*, International Journal of Robotics Research, Vol. 20, No. 5, pp 378-400.

52. **Li, H., Liao, X., Carin, L.** (2006). *Incremental Least Squares Policy Iteration for POMDPs*, In Proceedings of the National Conference on Artificial Intelligence, Vol. 21, No. 2, pp 1167.

53. **Likhachev, M., Stentz, A.** (2006). *PPCP: Efficient Planning with Clear Preferences in Partially-Known Environments*, In Proceedings of the National Conference on Artificial Intelligence (AAAI).

54. **Littman, M. L.** (1994). *The Witness Algorithm: Solving Partially Observable Markov Decision Processes*, Technical Report: CS-94-40, Computer Science Department, Brown University, Providence, RI.

55. **Littman, M. L.** (1996). *Algorithms for Sequential Decision Making*, PhD Dissertation, Department of Computer Science, Brown University, Providence, RI.

56. **Littman, M., Cassandra, A. R., Kaelbling L. P.** (1995). *Learning Policies for Partially Observable Environments: Scaling Up*, In Proceedings of the 12th International Conference on Machine Learning, pp 362-370.

57. **Lovejoy, W. S.** (1991). *Computationally Feasible Bounds for Partially Observed Markov Decision Processes*, Operations Research, Vol. 39, No. 1, pp 162-175.

58. **López, M. E., Barea, R., Bergasa, L. M., Ocaña, Escudero, M. S.** (2007). *Global Navigation of Assistant Robots using Partially Observable Markov Decision Processes*, In book *Perception and Navigation* book, Ed. **Kolski, S.**, pp 263-298, Pro Literatur, Verlag, Germany.

59. **Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., Konolige, K.** (2010). *The Office Marathon: Robust Navigation in an Indoor Office Environment*, In Proceedings of the International Conference on Robotics and Automation.

60. **Marthi, B.** (2012). *Robust Navigation Execution by Planning in Belief Space*, In Proceedings of Robotics: Science and Systems, pp 37.

61. **Melo, F. S., Veloso, M.** (2011). *Decentralized MDPs with Sparse Interactions*, Artificial Intelligence, Vol. 175, pp 1757-1789.

62. **Monahan, G. E.** (1982). *A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms*, Management Science, Vol. 28, No. 1, pp 1-16.

63. **Nilsson, N. J.** (1973). *A Hierarchical Robot Planning and Execution System*, Technical Report 76, SRI Report 1187, Artificial Intelligence Center, Menlo Park, CA.

64. **Ong, S. C. W., Png, S. W., Hsu, D., Lee, W. S.** (2009). *POMDPs for Robotics Tasks with Mixed Observability*, In the Conference Proceedings of Robotics: Science and Systems, Seattle, WA.

65. **Ong, S. C. W., Png, S. W., Hsu, D., Lee, W.S.** (2010). *Planning Under Uncertainty for Robotic Tasks with Mixed Observability*, International Journal of Robotics Research, Vol. 29, No. 8, pp 1053-1068.

66. **Pajarinen, J., Peltonen, J.** (2011). *Periodic Finite State Controller for Efficient POMDP and DEC-POMDP Planning*, In Proceedings of the 25th Annual Conference on Neural Information Processing Systems.

67. **Papadimitriou, C. H., Tsitsiklis, J. N.** (1987). *The Complexity of Markov Decision Processes*, Mathematics of Operations Research, Vol. 12, No. 3, pp 441-450.

68. **Pashenkova, E., Rish, I., Dechter, R.** (1996). *Value Iteration and Policy Iteration Algorithms for Markov Decision Problems*, In Proceedings of the National Conference on Artificial Intelligence (AAAI) Workshop on Structural Issues in Planning and Temporal Reasoning.

69. **Pfeifer, R., Bongard, J.** (2007). *How the Body Shapes the Way we Think: A New View on Intelligence*, MIT Press, Boston, MA.

70. **Pfeifer, R., Scheier, C.** (1999). *Understanding Intelligence*, MIT Press, Boston, MA.

71. **Pineau, J., Gordon G., Thrun, S.** (2003). *Point-Based Value Iteration: An Anytime Algorithm for POMDPs*, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico, pp 1025-1032.

72. **Pineau, J., Gordon, G.** (2005). *POMDP Planning for Robust Robot Control*, In Proceedings of the International Symposium on Robotics Research (ISRR), San Francisco, CA.

73. **Pineau, J., Thrun, S.** (2002). *High-Level Robot Behavior Control with POMDPs*, In the AAAI Workshop on Cognitive Robotics, Edmonton, Canada.

74. **Porta, J. M., Spaan, M. T. J., Vlassis, N.** (2005). *Robot Planning in Partially Observable Continuous Domains*, In Proceedings of Robotics: Science and Systems I, pp 29, Cambridge, MA.

75. **Poupart, P.** (2005). *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*, PhD Dissertation, Department of Computer Science, University of Toronto, Toronto, Canada

76. **Poupart, P., Kim, K-E., Kim, D.** (2011). *Closing the Gap: Improved Bounds on Optimal POMDP Solutions*, In Proceedings of the 21st International Conference on Automated Planning and Scheduling, Freiburg Germany.

77. **Puterman, M. L.** (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons Inc., NY.

78. **Ross, S. Pineau, J., Paquet, S., Chaib-Draa, B.** (2008). *Online Planning Algorithms for POMDPs*, Journal of Artificial Intelligence Research, Vol. 31, pp 663-704.

79. **Ross, S., Chaib-Draa, B., Pineau, J.** (2008). *Bayesian Reinforcement Learning in Continuous POMDPs with Application to Robot Navigation*, In Proceedings of the International Conference on Robotics and Automation, pp 2845-2851.

80. **Roy, N.** (2003). *Finding Approximate POMDP Solutions through Belief Compression*, PhD Dissertation, Carnegie Mellon University, Pittsburgh, PA

81. **Roy, N., Gordon, G., Thrun, S.** (2005). *Finding Approximate POMDP Solutions Through Belief Compression*, Journal of Artificial Intelligence Research, Vol. 23, pp 1-40.

82. **Russell, S. J., Norvig, P.** (2003). *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ.

83. **Siegwart, R., Nourbakhsh, I. R., Scaramuzza, D.** (2011). *Introduction to Autonomous Mobile Robots*, MIT Press, MA.

84. **Shani, G., Brafman, R., Shimony, S.** (2007). *Forward Search Value Iteration for POMDPs*, In Proceedings of the International Joint Conference on Artificial Intelligence.

85. **Shue, P. C. Y., Xue, S. Q.** (1993). *Intelligent Robotic Planning Systems*, World Scientific Publishing, River Edge, NJ.

86. **Siegwart, R., Nourbakhsh, I. R., Scaramuzza, D.** (2011). *Introduction to Autonomous Mobile Robots*, MIT Press, Cambridge, MA.

87. **Smallwood, R. D., Sondik, E. J.** (1973). *The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon*, Operation Research, Vol. 21, No. 5, pp. 1071-1088.

88. **Smith, T.** (2007). *Probabilistic Planning for Robotic Exploration*, PhD Dissertation, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

89. **Smith, T.**, **Simmons, R.** (2004). *Heuristic Search Value Iteration for POMDPs*, In Proceedings of the Conference on Uncertainty in Artificial Intelligence.

90. **Smith, T., Simmons, R.** (2005). *Point-Based POMDP Algorithms: Improved Analysis and Implementation*, In Proceedings of the Conference on Uncertainty in Artificial Intelligence.

91. **Sondik, E.** (1971). *The Optimal Control of Partially Observable Markov Decision Processes*, PhD Dissertation, Stanford University, Stanford, CA.

92. **Spaan, M., Vlassis, N.** (2004). *A Point-Based POMDP Algorithm for Robot Planning*, In Proceedings of the International Conference on Robotics and Automation, pp 2399-2404.

93. **Spaan, M.**, **Vlassis, N.** (2005). *Perseus: Randomized Point-Based Value Iteration for POMDPs*, Journal of Artificial Intelligence Research, Vol. 24, pp 195-220.

94. **Şucan, I. A., Kavraki, L. E.** (2012). *Accounting for Uncertainty in Simultaneous Task and Motion Planning using Task Motion Multigraphs*, In Proceedings of the International Conference on Robotics and Automation, pp 4822-4828.

95. **Şucan, I. A., Moll, M., Kavraki, L. E.** (2012). *The Open Motion Planning Library*, IEEE Robotics & Automation Magazine, Vol. 19, No. 4, pp 72-82.

96. **Sugiyama, M., Kawanabe, M.** (2012). *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation*, MIT Press, Boston, MA.

97. **Sutton, R. S., Barto, A. G.** (1998). *Reinforcement Learning: An Introduction*, MIT Press, Boston, MA.

98. **Theocharous, G., Murphy, K., Kaelbling, L. P.** (2004). *Representing Hierarchical POMDPs as DBN for Multi-Scale Robot Localization*. In Proceedings of the International Conference on Robotics and Automation, Vol. 1, pp 1045-1051.

99. **Thrun, S.** (2000). *A Programming Language Extension for Probabilistic Robot Programming*, In Workshop Notes of the IJCAI Workshop on Uncertainty in Robotics.

100. **Thrun, S.** (2008). *Simultaneous Localization and Mapping*, In Robotics and Cognitive Approaches to Spatial Mapping, Eds. **Jefferies, M. E., Yeap, W-K**., Vol. 38, Springer-Verlag Berlin Heidelberg.

101. **Thrun, S., Burgard, W., Fox, D.** (2006). *Probabilistic Robotics*, MIT Press, Boston, MA.

102. **White, D. J.** (1993). *Markov Decision Processes*, John Wiley and Sons, New York, NY.

103. **Yu, C-H., Chuang, J., Gerkey, B., Gordon, G., Ng, A.** (2005). *Open-Loop Plans in Multi-Robot POMDPs*, Technical Report, Computer Science Department, Stanford University, Stanford, CA.

104. **Zhang, S. Sridharan, M., Washington, C.** (2013). *Active Visual Planning for Mobile Robot Teams using Hierarchical POMDPs*, IEEE Transactions on Robotics, Vol. 29, No. 4.

Appendix

A. Data Syntax

Data acquisition on a real robot is extremely tedious. Even abstracts models like that presented in this work can be extremely cumbersome to document. Since we are interested in assessment of the proposed methodology we opted to use existing benchmark data found in POMDP literature. The data used in the work closely related to the *Hallway2* problem presented in Littman, Cassandra and Kaelbling (1995). The input parameters followed the format require for the **pomdp-solve ver. 5.3** solver. The file naming convention is: *filename.pomdp*. The **APPL ver. 0.95** requires a XML input format. It is capable of converting *filename.pomdp* to *filename.pomdpx* which is the corresponding XML format for the pomdp file. The APPL ver. 0.95 solver can generate policies, solve then evaluate them. The original data for the hallway2 problem can be found at the link below. The data used in this work has been considerable modified for the problem space. We do not show the complete data set since it is prohibitively large. However we only illustrate the input data for GSR Map 2 in section A.1.

http://www.pomdp.org/pomdp/examples/index.shtml

A.1 Input Data for GSR map Task 2

```
# A grid world layout applied to the GSR Algorithm
# Adapted from "Hallway2" benchmark POMDP problem
# Model for GSR-Map for task2

discount: 0.950000
values: reward
states: 20
actions: 5
observations: 17

start:
0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000

# Transition Probabilities

T: 0 : 0 : 0 1.000000
T: 1 : 0 : 5 0.050000
T: 1 : 0 : 0 0.950000
T: 2 : 0 : 0 0.100000
T: 2 : 0 : 1 0.700000
T: 2 : 0 : 2 0.100000
T: 2 : 0 : 3 0.100000
T: 3 : 0 : 0 0.100000
T: 3 : 0 : 1 0.150000
T: 3 : 0 : 2 0.600000
T: 3 : 0 : 3 0.150000
T: 4 : 0 : 0 0.100000
T: 4 : 0 : 1 0.100000
T: 4 : 0 : 2 0.100000
T: 4 : 0 : 3 0.700000
T: 0 : 1 : 1 1.000000
T: 1 : 1 : 5 0.800000
T: 1 : 1 : 1 0.200000
T: 2 : 1 : 0 0.100000
T: 2 : 1 : 1 0.100000
T: 2 : 1 : 2 0.700000
T: 2 : 1 : 3 0.100000
T: 3 : 1 : 0 0.150000
T: 3 : 1 : 1 0.100000
T: 3 : 1 : 2 0.150000
T: 3 : 1 : 3 0.600000
T: 4 : 1 : 0 0.700000
T: 4 : 1 : 1 0.100000
T: 4 : 1 : 2 0.100000
T: 4 : 1 : 3 0.100000
T: 0 : 2 : 2 1.000000
T: 1 : 2 : 5 0.050000
T: 1 : 2 : 2 0.950000
T: 2 : 2 : 0 0.100000
T: 2 : 2 : 1 0.100000
T: 2 : 2 : 2 0.100000
```

```
T: 2 : 2 : 3 0.700000
T: 3 : 2 : 0 0.600000
T: 3 : 2 : 1 0.150000
T: 3 : 2 : 2 0.100000
T: 3 : 2 : 3 0.150000
T: 4 : 2 : 0 0.100000
T: 4 : 2 : 1 0.700000
T: 4 : 2 : 2 0.100000
T: 4 : 2 : 3 0.100000
T: 0 : 3 : 3 1.000000
T: 1 : 3 : 5 0.025000
T: 1 : 3 : 7 0.025000
T: 1 : 3 : 3 0.950000
T: 2 : 3 : 0 0.700000
T: 2 : 3 : 1 0.100000
T: 2 : 3 : 2 0.100000
T: 2 : 3 : 3 0.100000
T: 3 : 3 : 0 0.150000
T: 3 : 3 : 1 0.600000
T: 3 : 3 : 2 0.150000
T: 3 : 3 : 3 0.100000
T: 4 : 3 : 0 0.100000
T: 4 : 3 : 1 0.100000
T: 4 : 3 : 2 0.700000
T: 4 : 3 : 3 0.100000
T: 0 : 4 : 4 1.000000
T: 1 : 4 : 6 0.800000
T: 1 : 4 : 8 0.050000
T: 1 : 4 : 3 0.050000
T: 1 : 4 : 4 0.100000
T: 2 : 4 : 4 0.100000
T: 2 : 4 : 5 0.700000
T: 2 : 4 : 6 0.100000
T: 2 : 4 : 7 0.100000
T: 3 : 4 : 4 0.100000
T: 3 : 4 : 5 0.150000
T: 3 : 4 : 6 0.600000
T: 3 : 4 : 7 0.150000
T: 4 : 4 : 4 0.100000
T: 4 : 4 : 5 0.100000
T: 4 : 4 : 6 0.100000
T: 4 : 4 : 7 0.700000
T: 0 : 5 : 5 1.000000
T: 1 : 5 : 0 0.050000
T: 1 : 5 : 10 0.050000
T: 1 : 5 : 1 0.025000
T: 1 : 5 : 3 0.025000
T: 1 : 5 : 5 0.850000
T: 2 : 5 : 4 0.100000
T: 2 : 5 : 5 0.100000
T: 2 : 5 : 6 0.700000
T: 2 : 5 : 7 0.100000
T: 3 : 5 : 4 0.150000
T: 3 : 5 : 5 0.100000
T: 3 : 5 : 6 0.150000
T: 3 : 5 : 7 0.600000
T: 4 : 5 : 4 0.700000
```

```
T: 4 : 5 : 5 0.100000
T: 4 : 5 : 6 0.100000
T: 4 : 5 : 7 0.100000
T: 0 : 6 : 6 1.000000
T: 1 : 6 : 0 0.025000
T: 1 : 6 : 2 0.025000
T: 1 : 6 : 10 0.800000
T: 1 : 6 : 3 0.050000
T: 1 : 6 : 6 0.100000
T: 2 : 6 : 4 0.100000
T: 2 : 6 : 5 0.100000
T: 2 : 6 : 6 0.100000
T: 2 : 6 : 7 0.700000
T: 3 : 6 : 4 0.600000
T: 3 : 6 : 5 0.150000
T: 3 : 6 : 6 0.100000
T: 3 : 6 : 7 0.150000
T: 4 : 6 : 4 0.100000
T: 4 : 6 : 5 0.700000
T: 4 : 6 : 6 0.100000
T: 4 : 6 : 7 0.100000
T: 0 : 7 : 7 1.000000
T: 1 : 7 : 0 0.050000
T: 1 : 7 : 10 0.050000
T: 1 : 7 : 3 0.800000
T: 1 : 7 : 7 0.100000
T: 2 : 7 : 4 0.700000
T: 2 : 7 : 5 0.100000
T: 2 : 7 : 6 0.100000
T: 2 : 7 : 7 0.100000
T: 3 : 7 : 4 0.150000
T: 3 : 7 : 5 0.600000
T: 3 : 7 : 6 0.150000
T: 3 : 7 : 7 0.100000
T: 4 : 7 : 4 0.100000
T: 4 : 7 : 5 0.100000
T: 4 : 7 : 6 0.700000
T: 4 : 7 : 7 0.100000
T: 0 : 8 : 8 1.000000
T: 1 : 8 : 4 0.800000
T: 1 : 8 : 16 0.025000
T: 1 : 8 : 18 0.025000
T: 1 : 8 : 8 0.150000
T: 2 : 8 : 8 0.100000
T: 2 : 8 : 9 0.700000
T: 2 : 8 : 10 0.100000
T: 2 : 8 : 11 0.100000
T: 3 : 8 : 8 0.100000
T: 3 : 8 : 9 0.150000
T: 3 : 8 : 10 0.600000
T: 3 : 8 : 11 0.150000
T: 4 : 8 : 8 0.100000
T: 4 : 8 : 9 0.100000
T: 4 : 8 : 10 0.100000
T: 4 : 8 : 11 0.700000
T: 0 : 9 : 9 1.000000
T: 1 : 9 : 4 0.050000
```

```
T: 1 : 9 : 18 0.050000
T: 1 : 9 : 9 0.900000
T: 2 : 9 : 8 0.100000
T: 2 : 9 : 9 0.100000
T: 2 : 9 : 10 0.700000
T: 2 : 9 : 11 0.100000
T: 3 : 9 : 8 0.150000
T: 3 : 9 : 9 0.100000
T: 3 : 9 : 10 0.150000
T: 3 : 9 : 11 0.600000
T: 4 : 9 : 8 0.700000
T: 4 : 9 : 9 0.100000
T: 4 : 9 : 10 0.100000
T: 4 : 9 : 11 0.100000
T: 0 : 10 : 10 1.000000
T: 1 : 10 : 4 0.025000
T: 1 : 10 : 6 0.025000
T: 1 : 10 : 18 0.800000
T: 1 : 10 : 10 0.150000
T: 2 : 10 : 8 0.100000
T: 2 : 10 : 9 0.100000
T: 2 : 10 : 10 0.100000
T: 2 : 10 : 11 0.700000
T: 3 : 10 : 8 0.600000
T: 3 : 10 : 9 0.150000
T: 3 : 10 : 10 0.100000
T: 3 : 10 : 11 0.150000
T: 4 : 10 : 8 0.100000
T: 4 : 10 : 9 0.700000
T: 4 : 10 : 10 0.100000
T: 4 : 10 : 11 0.100000
T: 0 : 11 : 11 1.000000
T: 1 : 11 : 4 0.050000
T: 1 : 11 : 18 0.050000
T: 1 : 11 : 11 0.900000
T: 2 : 11 : 8 0.700000
T: 2 : 11 : 9 0.100000
T: 2 : 11 : 10 0.100000
T: 2 : 11 : 11 0.100000
T: 3 : 11 : 8 0.150000
T: 3 : 11 : 9 0.600000
T: 3 : 11 : 10 0.150000
T: 3 : 11 : 11 0.100000
T: 4 : 11 : 8 0.100000
T: 4 : 11 : 9 0.100000
T: 4 : 11 : 10 0.700000
T: 4 : 11 : 11 0.100000
T: 0 : 12 : 12 1.000000
T: 1 : 12 : 17 0.050000
T: 1 : 12 : 12 0.950000
T: 2 : 12 : 12 0.100000
T: 2 : 12 : 13 0.700000
T: 2 : 12 : 14 0.100000
T: 2 : 12 : 15 0.100000
T: 3 : 12 : 12 0.100000
T: 3 : 12 : 13 0.150000
T: 3 : 12 : 14 0.600000
```

```
T: 3 : 12 : 15 0.150000
T: 4 : 12 : 12 0.100000
T: 4 : 12 : 13 0.100000
T: 4 : 12 : 14 0.100000
T: 4 : 12 : 15 0.700000
T: 0 : 13 : 13 1.000000
T: 1 : 13 : 17 0.800000
T: 1 : 13 : 13 0.200000
T: 2 : 13 : 12 0.100000
T: 2 : 13 : 13 0.100000
T: 2 : 13 : 14 0.700000
T: 2 : 13 : 15 0.100000
T: 3 : 13 : 12 0.150000
T: 3 : 13 : 13 0.100000
T: 3 : 13 : 14 0.150000
T: 3 : 13 : 15 0.600000
T: 4 : 13 : 12 0.700000
T: 4 : 13 : 13 0.100000
T: 4 : 13 : 14 0.100000
T: 4 : 13 : 15 0.100000
T: 0 : 14 : 14 1.000000
T: 1 : 14 : 17 0.050000
T: 1 : 14 : 14 0.950000
T: 2 : 14 : 12 0.100000
T: 2 : 14 : 13 0.100000
T: 2 : 14 : 14 0.100000
T: 2 : 14 : 15 0.700000
T: 3 : 14 : 12 0.600000
T: 3 : 14 : 13 0.150000
T: 3 : 14 : 14 0.100000
T: 3 : 14 : 15 0.150000
T: 4 : 14 : 12 0.100000
T: 4 : 14 : 13 0.700000
T: 4 : 14 : 14 0.100000
T: 4 : 14 : 15 0.100000
T: 0 : 15 : 15 1.000000
T: 1 : 15 : 17 0.025000
T: 1 : 15 : 19 0.025000
T: 1 : 15 : 15 0.950000
T: 2 : 15 : 12 0.700000
T: 2 : 15 : 13 0.100000
T: 2 : 15 : 14 0.100000
T: 2 : 15 : 15 0.100000
T: 3 : 15 : 12 0.150000
T: 3 : 15 : 13 0.600000
T: 3 : 15 : 14 0.150000
T: 3 : 15 : 15 0.100000
T: 4 : 15 : 12 0.100000
T: 4 : 15 : 13 0.100000
T: 4 : 15 : 14 0.700000
T: 4 : 15 : 15 0.100000
T: 0 : 16 : 16 1.000000
T: 1 : 16 : 8 0.800000
T: 1 : 16 : 19 0.025000
T: 1 : 16 : 17 0.025000
T: 1 : 16 : 15 0.050000
T: 1 : 16 : 16 0.100000
```

```
T: 2 : 16 : 16 0.100000
T: 2 : 16 : 17 0.700000
T: 2 : 16 : 18 0.100000
T: 2 : 16 : 19 0.100000
T: 3 : 16 : 16 0.100000
T: 3 : 16 : 17 0.150000
T: 3 : 16 : 18 0.600000
T: 3 : 16 : 19 0.150000
T: 4 : 16 : 16 0.100000
T: 4 : 16 : 17 0.100000
T: 4 : 16 : 18 0.100000
T: 4 : 16 : 19 0.700000
T: * : 17
0.052642 0.052631 0.052631 0.052631 0.052631 0.052631 0.052631 0.052631
0.052631 0.052631 0.052631 0.052631 0.052631 0.052631 0.052631 0.052631
0.0 0.052631 0.052631 0.052631
T: 0 : 18 : 18 1.000000
T: 1 : 18 : 8 0.025000
T: 1 : 18 : 10 0.025000
T: 1 : 18 : 17 0.800000
T: 1 : 18 : 15 0.050000
T: 1 : 18 : 18 0.100000
T: 2 : 18 : 16 0.100000
T: 2 : 18 : 17 0.100000
T: 2 : 18 : 18 0.100000
T: 2 : 18 : 19 0.700000
T: 3 : 18 : 16 0.600000
T: 3 : 18 : 17 0.150000
T: 3 : 18 : 18 0.100000
T: 3 : 18 : 19 0.150000
T: 4 : 18 : 16 0.100000
T: 4 : 18 : 17 0.700000
T: 4 : 18 : 18 0.100000
T: 4 : 18 : 19 0.100000
T: 0 : 19 : 19 1.000000
T: 1 : 19 : 8 0.050000
T: 1 : 19 : 17 0.050000
T: 1 : 19 : 15 0.800000
T: 1 : 19 : 19 0.100000
T: 2 : 19 : 16 0.700000
T: 2 : 19 : 17 0.100000
T: 2 : 19 : 18 0.100000
T: 2 : 19 : 19 0.100000
T: 3 : 19 : 16 0.150000
T: 3 : 19 : 17 0.600000
T: 3 : 19 : 18 0.150000
T: 3 : 19 : 19 0.100000
T: 4 : 19 : 16 0.100000
T: 4 : 19 : 17 0.100000
T: 4 : 19 : 18 0.700000
T: 4 : 19 : 19 0.100000

# Observation Probabilities
O: * : 0
0.000949 0.008549 0.008549 0.076949 0.000049 0.000449 0.000449 0.004049
0.008549 0.076949 0.076949 0.692550 0.000449 0.004049 0.004049 0.036464
0.0
```

O: * : 1
0.000949 0.008549 0.008549 0.076949 0.008549 0.076949 0.076949 0.692550
0.000049 0.000449 0.000449 0.004049 0.000449 0.004049 0.004049 0.036464
0.0
O: * : 2
0.000949 0.000049 0.008549 0.000449 0.008549 0.000449 0.076949 0.004049
0.008549 0.000449 0.076949 0.004049 0.076949 0.004049 0.692550 0.036464
0.0
O: * : 3
0.000949 0.008549 0.000049 0.000449 0.008549 0.076949 0.000449 0.004049
0.008549 0.076949 0.000449 0.004049 0.076949 0.692550 0.004049 0.036464
0.0
O: * : 4
0.085737 0.004512 0.004512 0.000237 0.771637 0.040612 0.040612 0.002137
0.004512 0.000237 0.000237 0.000012 0.040612 0.002137 0.002137 0.000120
0.0
O: * : 5
0.085737 0.004512 0.004512 0.000237 0.004512 0.000237 0.000237 0.000012
0.771637 0.040612 0.040612 0.002137 0.040612 0.002137 0.002137 0.000120
0.0
O: * : 6
0.085737 0.771637 0.004512 0.040612 0.004512 0.040612 0.000237 0.002137
0.004512 0.040612 0.000237 0.002137 0.000237 0.002137 0.000012 0.000120
0.0
O: * : 7
0.085737 0.004512 0.771637 0.040612 0.004512 0.000237 0.040612 0.002137
0.004512 0.000237 0.040612 0.002137 0.000237 0.000012 0.002137 0.000120
0.0
O: * : 8
0.009024 0.081225 0.000474 0.004275 0.081225 0.731024 0.004275 0.038475
0.000474 0.004275 0.000024 0.000225 0.004275 0.038475 0.000225 0.002030
0.0
O: * : 9
0.009024 0.000474 0.081225 0.004275 0.000474 0.000024 0.004275 0.000225
0.081225 0.004275 0.731024 0.038475 0.004275 0.000225 0.038475 0.002030
0.0
O: * : 10
0.009024 0.081225 0.000474 0.004275 0.081225 0.731024 0.004275 0.038475
0.000474 0.004275 0.000024 0.000225 0.004275 0.038475 0.000225 0.002030
0.0
O: * : 11
0.009024 0.000474 0.081225 0.004275 0.000474 0.000024 0.004275 0.000225
0.081225 0.004275 0.731024 0.038475 0.004275 0.000225 0.038475 0.002030
0.0
O: * : 12
0.000949 0.008549 0.008549 0.076949 0.000049 0.000449 0.000449 0.004049
0.008549 0.076949 0.076949 0.692550 0.000449 0.004049 0.004049 0.036464
0.0
O: * : 13
0.000949 0.008549 0.008549 0.076949 0.008549 0.076949 0.076949 0.692550
0.000049 0.000449 0.000449 0.004049 0.000449 0.004049 0.004049 0.036464
0.0
O: * : 14
0.000949 0.000049 0.008549 0.000449 0.008549 0.000449 0.076949 0.004049
0.008549 0.000449 0.076949 0.004049 0.076949 0.004049 0.692550 0.036464
0.0
O: * : 15

```
0.000949 0.008549 0.000049 0.000449 0.008549 0.076949 0.000449 0.004049
0.008549 0.076949 0.000449 0.004049 0.076949 0.692550 0.004049 0.036464
0.0
O: * : 16
0.085737 0.004512 0.004512 0.000237 0.771637 0.040612 0.040612 0.002137
0.004512 0.000237 0.000237 0.000012 0.040612 0.002137 0.002137 0.000120
0.0
O: * : 17
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
O: * : 18
0.085737 0.771637 0.004512 0.040612 0.004512 0.040612 0.000237 0.002137
0.004512 0.040612 0.000237 0.002137 0.000237 0.002137 0.000012 0.000120
0.0
O: * : 19
0.085737 0.004512 0.771637 0.040612 0.004512 0.000237 0.040612 0.002137
0.004512 0.000237 0.040612 0.002137 0.000237 0.000012 0.002137 0.000120
0.0


# Rewards
R: * : * : 17 : * 1.000000
```

B. Generated Policies for the Robot Tasks

The APPL solver was run for approximately 300 seconds. It reports the number of trials, the number of backups, the upper and lower bound on the optimal value function along the number of α-vectors. The number of belief points sampled were also reported. Approximately the last 60 seconds of computations are illustrated.

B.1 Output Data for Domain Map Task 1

```
#automat_map1 output

 239.17  286     12600    0.422828   0.963478   0.54065     148      1518
 240.61  288     12665    0.423164   0.963478   0.540314    148      1522
 242.41  289     12709    0.423226   0.963478   0.540252    151      1524
 243.41  290     12753    0.423259   0.963478   0.540219    150      1526
 244.34  291     12800    0.423276   0.963478   0.540202    148      1528
 246.36  292     12850    0.423285   0.963478   0.540193    149      1530
 247.39  293     12900    0.42329    0.963478   0.540188    144      1532
 249.24  295     12967    0.423297   0.963478   0.540181    147      1538
 251.18  296     13009    0.423301   0.963478   0.540177    148      1541
 252.31  297     13053    0.423304   0.963478   0.540174    149      1544
 252.97  298     13100    0.423306   0.963478   0.540172    153      1547
 254.98  299     13150    0.423308   0.963478   0.540171    152      1549
 255.97  300     13200    0.423308   0.963478   0.54017     153      1553
 257.25  302     13267    0.423309   0.963478   0.540169    150      1559
 259.17  303     13309    0.42331    0.963478   0.540168    147      1560
 260.32  304     13351    0.423392   0.963478   0.540086    147      1562
 261.57  305     13400    0.423479   0.963478   0.539999    144      1565
 264.89  306     13450    0.423617   0.963478   0.539861    145      1573
 268.7   308     13519    0.423941   0.963478   0.539537    153      1583
 269.82  309     13559    0.424046   0.963478   0.539432    158      1589
 272.59  310     13601    0.424207   0.963478   0.539271    164      1593
 274.38  311     13650    0.424359   0.963478   0.539119    165      1599
 276.18  312     13700    0.425407   0.963478   0.538071    160      1603
 279.25  313     13750    0.426664   0.963478   0.536814    162      1610
 281.57  314     13800    0.427599   0.963478   0.535879    163      1622
 284     316     13867    0.428654   0.963478   0.534824    170      1631
 286.48  317     13911    0.428935   0.963478   0.534543    172      1635
 288.23  318     13957    0.42912    0.963478   0.534358    180      1639
 289.08  319     14001    0.42924    0.963478   0.534238    166      1643
 291.02  320     14050    0.42943    0.963478   0.534048    166      1646
 294.03  321     14100    0.429702   0.963478   0.533776    165      1651
 296.2   323     14169    0.430336   0.963478   0.533142    161      1659
 298.9   324     14215    0.430638   0.963478   0.53284     165      1670

-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 300.79  324     14233    0.430638   0.963478   0.53284     181      1671
-------------------------------------------------------------------------------


-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             0.662648
 200             0.691717
```

B.2 Output Data for GSR Map Task 1

```
#automat_gsr1 output

 238.6   531     20000    0.81357    1.02818    0.214608    399      1538
 240.02  533     20059    0.813585   1.02818    0.214593    397      1545
 241.55  534     20100    0.81363    1.02818    0.214548    399      1549
 242.29  535     20150    0.813671   1.02818    0.214507    400      1551
 243.41  537     20215    0.814057   1.02818    0.214121    393      1556
 245.08  538     20255    0.814109   1.02818    0.214069    394      1561
 246.11  539     20300    0.814134   1.02818    0.214044    399      1566
 247.2   540     20350    0.814149   1.02818    0.214029    399      1571
 249.61  542     20411    0.81416    1.02818    0.214018    407      1576
 250.12  543     20450    0.814167   1.02818    0.214011    407      1577
 251.57  544     20500    0.814173   1.02818    0.214004    410      1582
 254.32  546     20567    0.814181   1.02818    0.213997    414      1590
 255.18  547     20609    0.814185   1.02818    0.213993    418      1593
 255.96  548     20650    0.814193   1.02818    0.213985    418      1596
 257.05  549     20700    0.814561   1.02818    0.213617    416      1600
 259.02  551     20767    0.814796   1.02818    0.213382    420      1603
 259.69  552     20805    0.814808   1.02818    0.21337     418      1605
 260.5   553     20850    0.814812   1.02818    0.213365    420      1608
 262.29  555     20900    0.814823   1.02818    0.213355    423      1610
 263.42  556     20959    0.814844   1.02818    0.213334    426      1614
 264.2   557     21000    0.814876   1.02818    0.213302    433      1617
 266.24  558     21050    0.814897   1.02818    0.21328     436      1620
 267.96  560     21113    0.814905   1.02818    0.213273    436      1626
 269.06  561     21153    0.814912   1.02818    0.213266    441      1629
 269.8   562     21200    0.814997   1.02818    0.213181    441      1631
 272.24  564     21269    0.815089   1.02818    0.213089    450      1638
 273.65  565     21311    0.815108   1.02818    0.21307     450      1642
 274.79  566     21350    0.815113   1.02818    0.213064    454      1645
 276.64  567     21400    0.815117   1.02818    0.213061    448      1648
 277.94  569     21465    0.815144   1.02818    0.213034    437      1652
 278.92  570     21503    0.815153   1.02818    0.213024    441      1655
 280.55  571     21550    0.815164   1.02818    0.213014    440      1657
 282.37  573     21619    0.815173   1.02818    0.213005    444      1665
 283.71  574     21659    0.815173   1.02818    0.213005    445      1668
 286.62  575     21701    0.815173   1.02818    0.213004    446      1674
 287.75  576     21750    0.815173   1.02818    0.213004    442      1676
 289.67  578     21815    0.815173   1.02818    0.213004    454      1684
 290.59  579     21851    0.815173   1.02818    0.213004    455      1686
 292.61  580     21900    0.815176   1.02818    0.213001    459      1691
 294.1   582     21959    0.815182   1.02818    0.212996    455      1696
 295.24  583     22000    0.815199   1.02818    0.212979    460      1701
 297.85  584     22050    0.815208   1.02818    0.212969    463      1708
 299.75  586     22100    0.815213   1.02818    0.212965    468      1714


-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 300.71  586     22138    0.815215   1.02818    0.212963    484      1719
-------------------------------------------------------------------------------

-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             0.837799
 200             0.833081
```

B.3 Output Data for Domain Map Task 2

```
#automat_map2 output

 238.52  384     12550    0.80649    1.41543    0.608943    219      1476
 240.06  386     12615    0.807188   1.41543    0.608245    218      1480
 241.73  387     12650    0.807235   1.41543    0.608198    219      1482
 243.32  389     12709    0.807274   1.41543    0.608159    218      1486
 244.74  390     12750    0.807281   1.41543    0.608153    219      1490
 247.92  392     12811    0.80733    1.41543    0.608103    223      1500
 248.9   393     12850    0.807392   1.41543    0.608042    223      1502
 250.6   395     12907    0.807462   1.41543    0.607971    225      1505
 252.41  396     12950    0.807469   1.41543    0.607965    226      1506
 254.29  398     13005    0.807478   1.41543    0.607955    221      1512
 255.32  399     13050    0.80748    1.41543    0.607954    224      1513
 257.79  401     13101    0.807484   1.41543    0.60795     217      1516
 259.04  403     13159    0.807485   1.41543    0.607948    219      1517
 260.12  404     13200    0.807485   1.41543    0.607948    224      1520
 262.7   406     13257    0.807485   1.41543    0.607948    223      1525
 264.11  407     13300    0.807487   1.41543    0.607946    222      1528
 265.74  409     13355    0.807488   1.41543    0.607945    223      1532
 267.87  410     13400    0.807489   1.41543    0.607945    223      1534
 269.34  412     13455    0.807599   1.41543    0.607835    223      1539
 270.89  413     13500    0.807653   1.41543    0.607781    210      1544
 274.05  415     13557    0.807681   1.41543    0.607752    210      1550
 275.57  416     13600    0.807815   1.41543    0.607619    211      1553
 277.17  418     13655    0.807916   1.41543    0.607517    212      1557
 279.61  419     13700    0.80793    1.41543    0.607504    213      1560
 281.29  421     13755    0.80794    1.41543    0.607493    219      1564
 282.35  422     13800    0.807942   1.41543    0.607491    217      1566
 285.51  424     13859    0.808073   1.41543    0.607361    217      1572
 286.61  425     13900    0.80812    1.41543    0.607313    217      1574
 288.23  427     13961    0.808152   1.41543    0.607281    216      1578
 290.32  428     14000    0.808157   1.41543    0.607277    216      1579
 291.85  430     14055    0.80816    1.41543    0.607273    219      1582
 292.98  431     14100    0.808161   1.41543    0.607272    219      1585
 296.1   433     14157    0.808163   1.41543    0.607271    223      1591
 297.11  434     14200    0.808163   1.41543    0.60727     225      1592
 299.3   436     14259    0.808203   1.41543    0.60723     233      1599
 300.33  437     14300    0.808227   1.41543    0.607206    234      1600


-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound     |UBound     |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 302.4   438     14321    0.808236   1.41543    0.607198    251      1603
-------------------------------------------------------------------------------

-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             1.09013
 200             1.08236
```

B.4 Output Data for GSR Map Task 2

```
#automat_gsr2 output

 239.68  536     17601    2.71021    2.81027    0.100059    445      1738
 240.73  537     17650    2.71021    2.81027    0.100058    448      1745
 242.22  539     17700    2.71022    2.81027    0.100054    441      1746
 243.27  541     17751    2.71022    2.81027    0.100048    444      1751
 244.42  542     17800    2.71023    2.81027    0.100045    445      1757
 246.17  544     17855    2.71023    2.81027    0.100043    449      1764
 247.03  545     17900    2.71023    2.81027    0.100037    445      1768
 248.31  547     17951    2.71026    2.81027    0.100015    451      1775
 250.27  549     18000    2.71026    2.81027    0.100014    455      1783
 251.1   550     18051    2.71027    2.81027    0.100006    461      1786
 252.42  552     18113    2.71027    2.81027    0.100001    465      1792
 253.94  553     18150    2.71027    2.81027    0.0999981   466      1796
 255.07  555     18213    2.71029    2.81027    0.099986    466      1801
 255.88  556     18250    2.71029    2.81027    0.0999818   469      1805
 258.17  558     18315    2.71029    2.81027    0.0999778   475      1813
 258.9   559     18350    2.71035    2.81027    0.0999249   475      1815
 260.23  561     18403    2.71037    2.81027    0.0999057   468      1821
 261.27  563     18461    2.71038    2.81027    0.0998951   466      1823
 262.99  564     18500    2.71038    2.81027    0.0998915   463      1827
 264.93  566     18559    2.71039    2.81027    0.0998853   468      1836
 265.95  567     18600    2.71039    2.81027    0.0998838   465      1840
 267.6   569     18650    2.71039    2.81027    0.0998773   470      1847
 270.31  571     18717    2.7104     2.81027    0.099872    477      1856
 271.25  572     18750    2.71042    2.81027    0.099853    481      1860
 272.54  574     18801    2.71044    2.81027    0.0998347   481      1866
 273.72  576     18853    2.71048    2.81027    0.0997961   481      1869
 276.07  578     18915    2.71049    2.81027    0.0997766   477      1875
 276.88  579     18950    2.71051    2.81027    0.0997591   471      1877
 278.51  581     19013    2.71052    2.81027    0.0997526   480      1885
 280.42  582     19050    2.71052    2.81027    0.0997513   476      1890
 281.33  584     19100    2.71053    2.81027    0.0997453   471      1891
 282.22  586     19150    2.71053    2.81027    0.0997415   475      1892
 284.19  588     19209    2.71053    2.81027    0.0997392   483      1901
 286.07  589     19250    2.71053    2.81027    0.0997388   481      1908
 287.82  591     19309    2.71054    2.81027    0.0997319   486      1915
 289.07  592     19350    2.71054    2.81027    0.0997305   485      1921
 290     594     19400    2.71055    2.81027    0.0997206   479      1922
 292.51  596     19455    2.71056    2.81027    0.0997141   479      1931
 293.9   597     19500    2.71056    2.81027    0.0997132   478      1938
 295.39  599     19551    2.71056    2.81027    0.099712    486      1945
 297.65  601     19613    2.71057    2.81027    0.0997027   491      1950
 298.8   602     19650    2.71058    2.81027    0.0996933   489      1955
 300.3   604     19707    2.71058    2.81027    0.0996916   487      1961


-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 300.65  604     19720    2.71058    2.81027    0.0996916   491      1963
-------------------------------------------------------------------------------

-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             2.73229
 200             2.74835
```

B.5 Output Data for Domain Map Task 3

```
#automat_map3 output

 239.12  271     12150    0.470563   0.963453   0.49289     240      1476
 241.23  273     12215    0.47077    0.963453   0.492684    242      1478
 242.42  274     12263    0.470849   0.963453   0.492604    240      1482
 244.96  275     12307    0.470876   0.963453   0.492577    243      1484
 246.41  276     12351    0.470941   0.963453   0.492512    243      1486
 248.26  277     12400    0.470995   0.963453   0.492459    242      1489
 250.31  278     12450    0.471025   0.963453   0.492428    241      1490
 251.92  279     12500    0.471043   0.963453   0.49241     248      1499
 253.04  280     12550    0.471052   0.963453   0.492401    235      1510
 255.83  282     12621    0.47121    0.963453   0.492243    232      1515
 257.13  283     12663    0.471281   0.963453   0.492172    235      1522
 259.81  284     12707    0.471948   0.963453   0.491505    232      1524
 260.97  285     12751    0.472292   0.963453   0.491162    228      1525
 262.68  286     12800    0.472427   0.963453   0.491026    233      1531
 265.22  287     12850    0.472598   0.963453   0.490856    235      1532
 266.32  288     12900    0.472677   0.963453   0.490776    236      1535
 268.46  290     12967    0.472719   0.963453   0.490734    228      1541
 270.47  291     13009    0.472725   0.963453   0.490728    232      1545
 271.97  292     13051    0.472735   0.963453   0.490718    230      1549
 273.02  293     13100    0.472743   0.963453   0.49071     234      1556
 275.86  294     13150    0.472804   0.963453   0.490649    230      1565
 278.27  295     13200    0.472848   0.963453   0.490605    237      1576
 281.38  296     13250    0.472867   0.963453   0.490586    236      1591
 285.92  298     13319    0.472914   0.963453   0.490539    226      1602
 288.67  299     13363    0.473316   0.963453   0.490137    228      1610
 290.09  300     13407    0.474211   0.963453   0.489242    231      1616
 294.42  301     13451    0.474805   0.963453   0.488648    239      1626
 296.96  302     13500    0.47501    0.963453   0.488443    240      1636
 298.62  303     13550    0.475078   0.963453   0.488375    243      1643


-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 301.95  304     13597    0.475102   0.963453   0.488352    266      1651
-------------------------------------------------------------------------------


-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             0.653846
 200             0.685472
```

B.6 Output Data for GSR Map Task 3

```
#automat_gsr3 output

 239.83  566     21167    0.901508   1.12468    0.223167    454      1619
 240.6   567     21205    0.901532   1.12468    0.223143    452      1622
 242.69  568     21250    0.901544   1.12468    0.223131    452      1625
 244.02  569     21300    0.901547   1.12468    0.223128    456      1632
 245.09  571     21355    0.901548   1.12468    0.223127    457      1634
 245.93  572     21400    0.901549   1.12468    0.223126    450      1636
 248.05  574     21461    0.901553   1.12468    0.223123    446      1640
 249.41  575     21503    0.901554   1.12468    0.223121    443      1645
 250.33  576     21550    0.901555   1.12468    0.22312     440      1647
 253.49  578     21617    0.901644   1.12468    0.223031    439      1657
 254.35  579     21651    0.901693   1.12468    0.222982    436      1660
 255.36  580     21700    0.902968   1.12468    0.221707    439      1663
 257.56  582     21763    0.903598   1.12468    0.221078    440      1667
 258.27  583     21803    0.903612   1.12468    0.221063    441      1669
 259.02  584     21850    0.903617   1.12468    0.221058    442      1671
 260.06  585     21900    0.903626   1.12468    0.221049    442      1675
 262.29  587     21959    0.903724   1.12468    0.220951    449      1679
 263.7   588     22003    0.903739   1.12468    0.220936    453      1683
 264.35  589     22050    0.903751   1.12468    0.220924    452      1684
 266.39  591     22117    0.903861   1.12468    0.220814    453      1688
 267.32  592     22159    0.903889   1.12468    0.220786    453      1692
 268.63  593     22200    0.903901   1.12468    0.220774    457      1695
 269.49  594     22250    0.903904   1.12468    0.220771    457      1697
 271.82  596     22317    0.903916   1.12468    0.220759    465      1703
 272.45  597     22355    0.90392    1.12468    0.220755    465      1704
 273.1   598     22400    0.903921   1.12468    0.220754    466      1705
 275.28  600     22467    0.903937   1.12468    0.220738    466      1707
 275.85  601     22503    0.903946   1.12468    0.220729    466      1708
 276.74  602     22550    0.904043   1.12468    0.220632    461      1710
 278.91  604     22613    0.90409    1.12468    0.220585    466      1714
 280.33  605     22650    0.904098   1.12468    0.220577    466      1718
 281.55  606     22700    0.904104   1.12468    0.220571    465      1722
 283.03  608     22757    0.904123   1.12468    0.220552    465      1728
 284.73  609     22800    0.904127   1.12468    0.220548    464      1729
 286.83  611     22867    0.904128   1.12468    0.220547    463      1735
 288.62  612     22907    0.904129   1.12468    0.220546    471      1741
 289.85  613     22950    0.904135   1.12468    0.22054     473      1746
 291.78  614     23000    0.904162   1.12468    0.220513    466      1749
 293.67  616     23057    0.904183   1.12468    0.220492    473      1755
 294.54  617     23100    0.904184   1.12468    0.220491    475      1757
 297.38  619     23165    0.904185   1.12468    0.22049     471      1762
 298.18  620     23203    0.905024   1.12468    0.219651    466      1764
 298.87  621     23250    0.905264   1.12468    0.219411    466      1765
 301.56  623     23317    0.905324   1.12468    0.219352    475      1771

-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 301.57  623     23317    0.905324   1.12468    0.219352    475      1771
-------------------------------------------------------------------------------

-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             0.949199
 200             0.943234
```

B.7 Output Data for Domain Map Task 4

```
#automat_map4 output

 239.02  331     15001    0.343245   0.769087   0.425842    210      1298
 241.39  332     15050    0.343252   0.769087   0.425835    206      1303
 242.89  333     15100    0.343258   0.769087   0.425829    205      1309
 244.46  334     15150    0.34331    0.769087   0.425778    212      1313
 246.6   335     15200    0.343375   0.769087   0.425712    210      1318
 248.4   336     15250    0.343434   0.769087   0.425654    206      1323
 251.69  338     15319    0.348479   0.769087   0.420608    189      1329
 252.71  339     15363    0.349714   0.769087   0.419373    184      1330
 253.91  340     15407    0.350384   0.769087   0.418703    185      1334
 255.82  341     15451    0.350762   0.769087   0.418325    185      1336
 256.85  342     15500    0.350989   0.769087   0.418098    190      1340
 257.74  343     15550    0.351122   0.769087   0.417966    187      1342
 260.1   344     15600    0.351204   0.769087   0.417883    183      1344
 261.7   345     15650    0.351258   0.769087   0.417829    192      1348
 263.2   347     15700    0.351367   0.769087   0.41772     189      1354
 265.36  348     15765    0.35151    0.769087   0.417577    189      1356
 266.35  349     15809    0.35154    0.769087   0.417547    185      1358
 267.37  350     15851    0.35156    0.769087   0.417527    185      1360
 269.41  351     15900    0.351593   0.769087   0.417494    180      1362
 270.51  352     15950    0.351649   0.769087   0.417438    180      1367
 273.44  353     16000    0.35176    0.769087   0.417328    186      1373
 275.57  354     16050    0.35186    0.769087   0.417227    195      1379
 277.41  356     16119    0.351938   0.769087   0.417149    194      1383
 280.05  357     16165    0.351951   0.769087   0.417136    193      1388
 280.98  358     16209    0.351959   0.769087   0.417128    193      1390
 282.54  359     16253    0.351965   0.769087   0.417122    196      1394
 285.25  360     16300    0.351971   0.769087   0.417116    197      1397
 286.94  361     16350    0.351976   0.769087   0.417112    204      1401
 289.82  362     16400    0.352009   0.769087   0.417078    209      1406
 292.07  363     16450    0.352085   0.769087   0.417002    209      1412
 293.62  365     16515    0.352199   0.769087   0.416888    210      1414
 295.96  366     16557    0.352221   0.769087   0.416866    213      1417
 297.06  367     16601    0.352234   0.769087   0.416854    215      1420
 298.63  368     16650    0.35224    0.769087   0.416847    215      1424


-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 301.8   368     16668    0.35224    0.769087   0.416847    229      1432
-------------------------------------------------------------------------------


-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             0.567472
 200             0.558172
```

B.8 Output Data for GSR Map Task 4

```
#automat_gsr4 output

 238.69  381     16250    0.580532   0.784755   0.204223    399      1285
 241.09  382     16300    0.580639   0.784755   0.204116    397      1289
 243.46  383     16350    0.580674   0.784755   0.204081    400      1297
 245.4   385     16419    0.580734   0.784755   0.204021    399      1301
 247.85  386     16465    0.580744   0.784755   0.204011    405      1305
 248.81  387     16509    0.580752   0.784755   0.204004    400      1307
 250.99  388     16553    0.580756   0.784755   0.203999    410      1310
 252.51  389     16600    0.580873   0.784755   0.203883    411      1314
 254.35  390     16650    0.580964   0.784755   0.203791    412      1317
 256.72  391     16700    0.580996   0.784755   0.203759    409      1321
 258.17  393     16765    0.581019   0.784755   0.203737    408      1325
 259.67  394     16811    0.581372   0.784755   0.203383    410      1329
 261.78  395     16855    0.581533   0.784755   0.203222    408      1333
 262.57  396     16900    0.581594   0.784755   0.203161    405      1335
 264.82  397     16950    0.58161    0.784755   0.203145    406      1337
 266.39  398     17000    0.581616   0.784755   0.203139    407      1340
 268.95  399     17050    0.581619   0.784755   0.203136    405      1347
 271.97  401     17117    0.581622   0.784755   0.203133    402      1351
 273.06  402     17159    0.581624   0.784755   0.203132    408      1354
 273.95  403     17203    0.581625   0.784755   0.20313     402      1356
 275.9   404     17250    0.581626   0.784755   0.20313     403      1358
 277.03  405     17300    0.581626   0.784755   0.203129    406      1361
 279.71  407     17367    0.582342   0.784755   0.202413    405      1367
 280.97  408     17409    0.58288    0.784755   0.201876    403      1371
 282.47  409     17455    0.583218   0.784755   0.201537    409      1375
 284.79  410     17500    0.583372   0.784755   0.201383    416      1379
 286.22  411     17550    0.58365    0.784755   0.201105    413      1382
 288.35  413     17621    0.58381    0.784755   0.200945    412      1386
 290.76  414     17667    0.583824   0.784755   0.200931    409      1390
 291.56  415     17709    0.584035   0.784755   0.20072     406      1392
 293     416     17755    0.584244   0.784755   0.200511    408      1395
 295.75  417     17800    0.584345   0.784755   0.20041     412      1399
 297.88  418     17850    0.584762   0.784755   0.199994    409      1405
 298.94  419     17900    0.584963   0.784755   0.199793    411      1406

-------------------------------------------------------------------------------
 Time    |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 301.13  420     17933    0.585013   0.784755   0.199742    441      1409
-------------------------------------------------------------------------------

-----------------------------------
 #Simulations  | Exp Total Reward
-----------------------------------
 100             0.681748
 200             0.684719
```