Graduate School
University of South Florida
Tampa, Florida

CERTIFICATE OF APPROVAL

Master's Thesis

This is to certify that the Master's Thesis of

NIKFAR A. KHALEELI

with a major in Computer Science has been approved by
the Examining Committee on April 4, 1997
as satisfactory for the thesis requirement
for the Master of Science in Computer Science degree

Examining Committee :

Major Professor: Sridhar Mahadevan, Ph.D.

Member: Lawrence O. Hall, Ph.D.

Member: Dmitry B. Goldgof, Ph.D.

A ROBUST ROBOT NAVIGATION ARCHITECTURE USING PARTIALLY OBSERVABLE SEMI-MARKOV DECISION PROCESSES

by

NIKFAR A. KHALEELI

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
University of South Florida

April 1997

Major Professor: Sridhar Mahadevan, Ph.D.

DEDICATION

This thesis is dedicated to my parents, Afsur and Farkhondeh, for teaching me to go wherever dreaming goes.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Mahadevan for fueling my interest in robotics, and for the ideas, suggestions, enthusiasm, and encouragement that he provided, when there seemed to be no light at the end of the tunnel.

Not only do I appreciate both Dr. Hall for getting me started off in the program, and Dr. Goldgof for teaching a really enjoyable Digital Image Processing course, but I would also like to thank them for being on my committee.

I would like to thank Billy Raulerson for coding the graphical interface.

Finally, I would like to thank my family and friends (you know who you are) for their love and support, without which this would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
SMDP	Semi-Markov Decision Process
POSMDP	Partially Observable Semi-Markov Decision Processes
ANN	Artificial Neural Network

A ROBUST ROBOT NAVIGATION ARCHITECTURE USING PARTIALLY OBSERVABLE SEMI-MARKOV DECISION PROCESSES

by

NIKFAR A. KHALEELI

An Abstract

Of a thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
University of South Florida

April 1997

Major Professor: Sridhar Mahadevan, Ph.D.

In order to perform delivery tasks in an office environment, an autonomous robot must be able to robustly navigate corridors for long periods of time, without getting lost. This thesis presents a robust and autonomous robot navigation system in an unstructured office environment based on partially observable semi-Markov decision processes (POSMDPs).

Two common traditional approaches to robot navigation are geometric-based and topological-based. Geometric-based navigation is able to take advantage of information about robot motion, but is vulnerable to map inaccuracies and dead-reckoning errors. By utilizing information about the sensed environment features, topological-based navigation is independent of geometric accuracy but suffers from problems of unreliable sensors occasionally not detecting landmarks and perceptual aliasing.

Using a partially observable Markov decision process model for navigation combines the advantages of the two approaches by modeling navigation uncertainty, such as sensor and odometric errors and approximate environment knowledge. However, this approach assumes that actions take uniform, discrete amounts of time. A more suitable and natural approach should incorporate time. One such approach is to use an event-based MDP model called a semi-Markov decision process model.

In an office environment robot sensors such as ultrasonic range sensors are prone to specular reflections. Occupancy grids use readings taken from several sensors over multiple points of views to create a spatial model of the environment, but are unable to compensate for these specularities. An artificial neural network is used to sense high level features, since it can be trained to robustly interpret occupancy data. Not only does this ensure better sensor reliability, it also allows the net to be retrained for different office environments.

We present a navigation architecture based on POSMDPs that provides a uniform framework with an established theoretical foundation for state estimation, path planning and robot control during navigation. Reliable sensor interpretation is

achieved through an artificial neural network. Implemented on PAVLOV, an actual indoor mobile robot, our experiments show that this approach leads to robust corridor navigation. PAVLOV was tested on the third floor of the Engineering building at the University of South Florida on runs totaling several kilometers over a period of many weeks.

Abstract Approved: _____

Major Professor: Sridhar Mahadevan, Ph.D.
Assistant Professor,
Department of Computer Science and Engineering

Date Approved: _____

CHAPTER 1

INTRODUCTION

To successfully perform routine delivery tasks in an office environment, it is important that autonomous robots navigate corridors robustly in order to reliably reach their goals. However, planning and following a path requires that the robot answer the question: "Where am I?" In doing so, the robot must be able to recover robust and useful spatial descriptions of its surroundings, using sensory information. This chapter briefly reviews a number of methods for mobile robot positioning. It then provides an introduction to the navigation system implemented on the robot PAVLOV.

1.1. Robot Navigation

In order to get anywhere or to plan a route to a goal location, a mobile robot must be able to estimate its position. Two common methods of position estimation are relative and absolute position measurements. Odometry and inertial navigation are techniques for relative position measurements. Methods for absolute position estimation are active beacons, artificial and natural landmark recognition and map-based positioning. Navigation systems are usually comprised of a combination of the two methods.

Odometry uses encoders to measure wheel rotation and steering orientation. It is self contained and is always capable of providing the robot with an estimation of

its position. The main disadvantage is that the position error grows without bound unless an independent reference is used to periodically reduce the error [1].

Inertial navigation uses gyroscopes and accelerometers to measure rate of rotation and acceleration. Measurements are integrated to yield position. Inertial navigation systems are also self-contained. Unfortunately, in addition to high equipment cost, inertial sensor data drifts with time because of the need to integrate rate data to yield position. Any small constant error increases with bound after integration. Therefore, inertial sensors are unsuitable for accurate positioning over an extended period of time [2].

Active beacons involve computation of the absolute position of the robot by measuring the direction of incidence of three or more actively transmitted beacons, located at known sites in the environment. The Navstar Global Positioning System (GPS) [3] uses a constellation of 24 satellites orbiting the earth every 12 hours at a height of about 10,900 nautical miles. The absolute three-dimensional location of any GPS receiver can be determined through trilateration techniques based on time of flight for uniquely coded spread-spectrum radio signals transmitted by the satellites. Introduced into the automotive racing world, Precision Technology' *Precision Location* [4] tracking and telemetry system utilizes a number of antennae situated at fixed locations around a racetrack. Comparison of the signals received by the various antennae to a common reference signal of identical frequency generated at the base station allows determination of relative changes in vehicle position with respect to each antennae.

A natural landmark positioning system generally has a sensor for detecting landmarks and contrasting them against their background, a method for matching observed features with a map of known landmarks, and a method of computing location and localization errors from matches [2]. Developed in Canada, ARK or the *Autonomous Robot for a Known Environment* [5] is a navigation system that uses

natural landmarks such as alphanumeric signs, semi-permanent structures, or doorways. The only criteria to be fulfilled is that landmarks be distinguishable from the background by color or contrast. The navigation module consists of a custom-made pan-and-tilt table, a CCD camera and an eye-safe IR spot laser rangefinder. Landmarks are learned by generating a three dimensional *grey-level* surface from an image obtained from the CCD camera. A range scan of the same field of view is performed by the laser rangefinder, providing depth information for each pixel in the grey-level surface. During operation, when the robot is approximately at the training position, the system searches for landmarks that are expected to be visible from the momentary position. The projected appearance of the landmark is computed and used to yield the robot's relative distance and heading with respect to it. Odometric positioning can be updated to within a few centimeters by finding a pair of natural landmarks of known position.

It is also possible to place distinctive artificial landmarks at known locations. These landmarks can be designed for optimal detectability, even under adverse environmental conditions. The advantage of this approach is that position errors are bounded, but detection of external landmarks and real-time position fixing may not always be possible. The *Mobile Detection Assessment and Response System* [6, 7] uses passive reflectors in conjunction with a pair of fixed-orientation polarized sensors on board the robot. Short vertical strips of retroreflective tape are placed on various immobile objects on either side of a virtual path segment. The exact x-y locations of the tape markers are encoded into the virtual path planner. Longitudinal position is updated to the known marker coordinates, while lateral position is inferred from sonar data. The polarized sensors respond only to the presence of a retroreflector, while ignoring specular surrounding surfaces. Caterpillar Industrial's *Self Guided Vehicle* [8, 9] for materials handling relies on a scanning laser triangulation scheme to provide positional updates to the vehicles on-board odometry system, through the

usage of bar-code targets at known locations. Line navigation is another type of landmark navigation used in industry. Pyroelectric sensors can be used to detect thermals paths created by heating the floor with a quartz halogen bulb [10, 11, 12], while a robot equipped with an odor sensing system is able to follow a previously placed odor trail [13, 14, 15].

Map-based positioning is a technique in which the robot uses its sensors to create a map of its local environment. This local map is then compared to a global map. If a match is found, then the robot can compute its actual position and orientation in the environment. The global map can be provided to the robot as *a priori* knowledge, or it can build its own environment map, through exploration [16, 17, 18]. In map-based positioning, there are two common approaches: *geometric* and *topological* maps [2].

Geometric maps represent objects according to their absolute geometric relationships. Geometric-map based navigation systems rely on metric maps of the environment, and are able to take advantage of information about the motion of the robot (*actions reports*), such as translation and rotation, derived from wheel encoder. One of the simplest ways to represent geometric map data is through certainty grids [19]. In this approach sensor readings are placed into the grid by using probability profiles that describe the certainty about the existence of objects at individual grid cells. This approach has a number of advantages. It is easy to build, represent and maintain. It allows for easy integration of data from different sensors [20]. Recognition of places, based on geometry, is non-ambiguous and view point independent. Unfortunately, disadvantages do exist. It is space consuming and requires accurate determination of the robot's *pose*[1]. Although odometry is very accurate over short distances, errors accumulate over time and decrease the dead-reckoning abilities of the robot.

[1] *Pose* refers to the position and orientation

Topological maps are based on the recording of the geometric relationships between the observed features, rather than their absolute position with respect to an arbitrary coordinate frame of reference. This representation takes the forms of graphs, where nodes represent recognizable locations (*landmarks*) and edges indicate connections between landmarks and how the robot should navigate between them. Information about the sensed features of the environment (*sensor reports*) enable the robot to identify its current location. Topological maps can be built and maintained without estimating the robot position. Such an approach allows integration of large maps, without suffering from accumulated odometric error, since all connections between nodes are relative, rather than absolute. Such maps permit efficient planning, since resolution depends on the complexity of the environment. Unfortunately, this approach is prone to problems of unreliable sensors occasionally not detecting landmarks, and *perceptual aliasing*, where the sensors are not able to distinguish between different landmarks with similar features.

1.2. The POSMDP Approach

In an office type environment, it would make sense to exploit the underlying structure. Therefore, a logical choice for an indoor navigation system would be to utilize a map-based approach. There are a number of advantages in using map-based navigation system. It utilizes the naturally occurring structure of typical indoor environments to derive position information without modifying the environment. The environment map can be updated and used in tasks such as global path planning or obstacle avoidance. This approach allows a robot to learn a new environment, and to improve positioning accuracy through exploration. However, both map-based approaches have disadvantages.

The robot must also be able to traverse corridors for long periods of time without getting lost. In any navigation task, there can be a number of sources for uncertainty, which can contribute to the robot getting lost. Motion commands are not always carried out perfectly due to wheel slippage and mechanical tolerances, resulting in *dead reckoning error*. Unreliable sensors can produce *false positives* (features that are not present) or *false negatives*. While interpreting sensor data, the robot might not be able to clearly distinguish between a wall and an opening. Lengths of corridors might not be known exactly. The environment can change as people and obstacles move in the environment. Both geometric and topological approaches represent only a single pose that is believed to be the current pose of the robot. If this proves to be incorrect, the robot is lost and has to relocalize itself.

A navigation system using a partially observable Markov decision process model [21, 22] is able to explicitly account for various forms of uncertainty. Partially observable Markov decision processes integrate topological and approximate metric information, utilizing both action and sensor reports in determining the robot pose. A partially observable Markov decision process represents all possible states of the robot. Instead of maintaining a single estimate of its current pose, the robot uses a partially observable Markov decision process to maintain a probability distribution over all possible states. The robot always has some belief about its true pose, and is never completely lost. Bayes rule is used to update the pose distribution after each action and sensor report.

In the Markov model, actions (decisions) occur in discrete time [23]. Navigation in a dynamic environment results in varying amounts of time to complete actions. For example, if the robot has to avoid a number of obstacles in traversing a distance of one meter, it will take a longer time than if it had a clear path. This thesis shows how a partially observable semi-Markov decision process (POSMDP) can provide a more natural model by extending the standard POMDP approach to account for

continuous time. Decisions are allowed only at discrete points in time. In between decisions the state can change, unlike in an MDP, where state changes are solely due to decisions.

1.3. Thesis Overview

Figure 1. The Mobile Robot PAVLOV.

The overall structure of the POSMDP-based autonomous layered navigation system, implemented on the robot PAVLOV (Figure 1), is displayed in Figure 2. The navigation system consists of three separate layers: an observational layer, a reactive behavior layer, and a planning layer. Ultrasonic sensors value are used to

construct local occupancy grids, which are used by an artificial neural network to produce observations. The reactive behavior based layer receives actions commands from the planning layer and translates these into motor commands. Responsibility for obstacle avoidance and alignment fall to this layer. The planning layer is responsible for generating a path and estimating the robot pose. It uses sensor and action reports to update the partially observable Markov decision process, which is then used in estimating the pose of the robot.

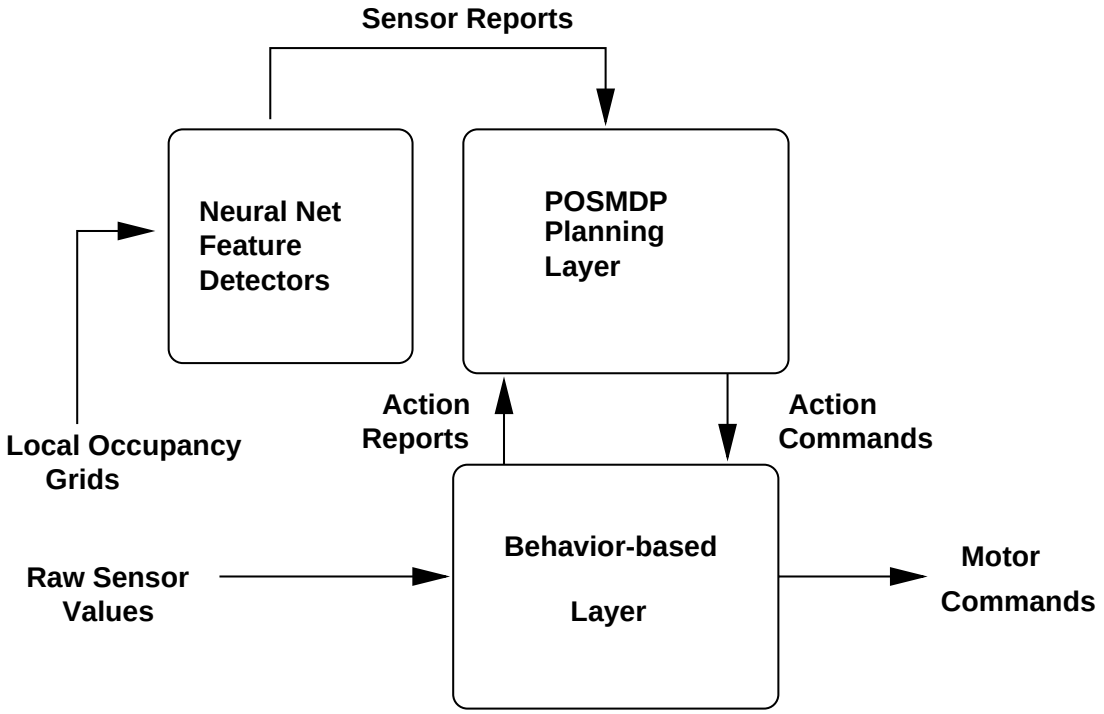


Figure 2. The Overall Architecture.

The planning layer is supplied with approximate metric information between nodes via a *topological* map, which is a graph of nodes and edges that contain information about the connectivity of the environment. Figure 3 represents the map made available to the planning layer. Given this map, one might question the necessity of

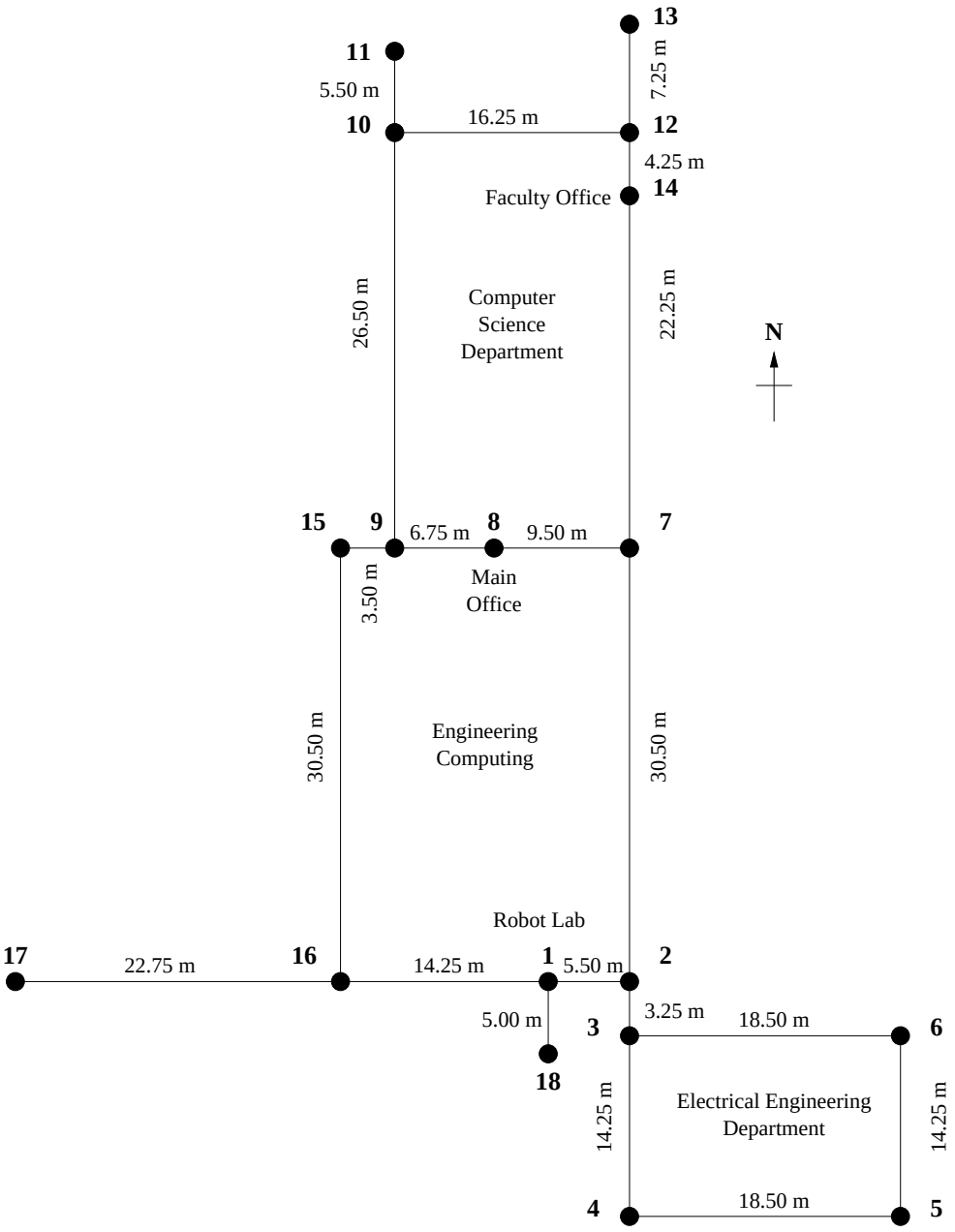


Figure 3. The Topological Map. A topological map of the third floor of the engineering building, showing landmarks. The landmarks (topological nodes) correspond to recognizable locations, such as corridor intersections and entry ways. Edges, containing approximate metric information, represent the distance between the landmarks. The topological map has 17 nodes and 19 edges

any complex navigation system. The robot could simply follow the metric information provided. When it believes itself to be close to a node, it could start anticipating the appropriate landmark (an intersection, a doorway, etc.).

Experiments with PAVLOV have shown that this is not feasible because of odometric errors, sensor unreliability and *perceptual aliasing*. Odometry is accurate for short distances, but the errors accumulate. Turns at intersections cause the internal angular representation to drift. As the robot avoids obstacles, more errors are introduced. These errors become very noticeable over large distance. Instructing the robot to travel 20 meters East, after traversing the Computer Science Department, could not only result in it attempting to travel into a wall but there is no guarantee that it will accurately traverse the specified distance. It is also not possible to completely rely on sensors. Not only are sensors and interpretation of sensor data prone to errors, but numerous places in the environment could look the same. This is known as *perceptual aliasing*. For example, since both node 7 and 12 are three way intersections, the robot has no way of differentiating between them if it were to completely rely on sensors. Using partially observable Markov decision processes in navigation unites both map-based approaches and explicitly accounts for the sources of uncertainty.

This thesis extends the partially observable Markov decision process approach to incorporate time. A modification of the value iteration algorithm [24] to account for varying action times results in better path planning. Experiments on PAVLOV show that it is possible to learn to avoid crowded corridors. Results from using a partially observable Markov decision process approach show that PAVLOV always has some belief about its true pose, and is able to relocalize itself, in the event that it gets lost. In addition, we demonstrate that an artificial neural network can be easily trained to sense high level features. Results from PAVLOV show that the artificial neural network functions robustly in highly specular environments.

1.4. Chapter Outline

Chapter 2 provides background information on Markov decision processes, dynamic programming, occupancy grids and artificial neural networks. This chapter also reviews different approaches in navigation systems that maximize the advantage of geometric-based and topological-based navigation systems.

Chapter 3 describes the implementation of the layered navigation system, comprised of the planning layer, the reactive behavioral layer and a learned observational model.

Chapter 4 presents the results of a number of experiments performed on PAVLOV to test the robustness of the navigation system and the results from the learned observational model.

Chapter 5 summarizes the results and contributions of this thesis and discusses areas of future research.

CHAPTER 2

THEORETICAL FOUNDATIONS

Decisions can have immediate and long-term consequences. Decisions made in isolation do not account for the relationship between the present and the future and can adversely affect overall performance. Markov decision processes are a model for sequential decision making under uncertainty which takes current and future decision outcomes into account. This chapter discusses Markov decision processes, and methods to extend this to continuous time and situations of incomplete knowledge. Occupancy grids, artificial neural networks and related navigation architectures are also reviewed.

2.1. Markov Decision Processes

A sequential decision problem is one where the agent perceives the environment to be in one of a set of states that it can inhabit, a set of actions it can take in any environment state, and a reward for being in any state. Given a state, the policy tells the agent what action to perform. Solving a Markov decision problem [24, 23] requires calculating an optimal policy in an observable, stochastic environment with a transition model that satisfies the Markovian property.

A dynamical system specified in terms of a transition model and a reward function is called a Markov decision process. A transition model is one which gives, for each state s and action a, the resulting distribution of states if a was executed in s. A Markov decision process is a mathematical model of a discrete-time sequential

decision problem [24]. Formally a Markov decision process is defined by the four tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$, where

- $\mathcal{S}$ is the finite set of environment states
- $\mathcal{A}$ is the set of actions
- P is the set of action dependent transition probabilities
- R is the reward function

P, the state transition model of the environment, is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$. $P(s'|s,a)$ represents the probability that the environment will make a transition from state s to state s' under action a.

$R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the expected reward for taking each action in each state. $R(s,a)$ represents the immediate reward for taking action a in state s.

In the evolution of a Markov decision process, for every sequence of observed states,x_0, x_1, ..., x_{t-1}, x, y, and the corresponding actions, a_0, a_1, ..., a_{t-1}, a, the probability of being in the current state y is given by

$$P(X_{t+1} = y | X_0 = x_0, a_0, X_1 = x_1, a_1, \ldots, X_t = x_t, a_t = a)$$
$$= P(X_{t+1} = y | X_t = x, a_t = a)$$

where $t = 0, 1, \ldots$, is the decision epoch.

This is known as the *Markovian Property* [24], and essentially says that the current state and action provide all of the information available for predicting the next state. Knowledge of the current state is all that is required in making a decision.

2.1.1. Policies and the Value Function

At discrete times in the process evolution, known as decision epochs, the state of the process is observed. An action is then taken by the controlling agent. When the choice of actions at each step is based solely on the current process state, the policy is *stationary*. The agent must have some function $\pi(s)$ that is used in choosing actions. The policy, π, defined as $\pi : \mathcal{S} \rightarrow \mathcal{A}$, is a complete mapping from states to actions. Given a state, it tells the agent what action to perform. A policy can be calculated from a transition model and a utility function.

In order to identify an optimal policy, we must be able to distinguish between different policies. The *value function* for a particular policy is defined as a mapping $V^{\pi} : \mathcal{S} \rightarrow \Re$ that determines the expected utility of each state s, if actions are chosen according to the policy π. The reward function assigns a numeric value to a state, based on the short term view, while the value function assigns a real-valued number to a state taking a longer term view. For example, an agent could take an action with a positive reward. It is not to the agents advantage if the action takes it into a state with a negative utility value. A better strategy would be to take an action that promises a larger long term reward. With this in mind, the *expected total discounted reward* of policy $\pi(s)$ is defined to be

$$V_{\gamma}^{\pi}(s) = E_s^{\pi} \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.1}$$

for $0 \leq \gamma < 1$. The discount factor γ measures the present value of one unit of reward received one epoch in the future. Discounting arises to account for the time value of rewards and forces the decision maker to value policies according to the expected

total reward. The aim is to maximize the value function over all states by identifying a policy π where $V_\gamma^\pi(s) \geq V_\gamma^{\pi_i}(s)$ for all s and for all π_i

$$V^*(s) = \max_\pi V_\gamma^\pi(s), \quad \forall s \in \mathcal{S} \tag{2.2}$$

V^* refers to the optimal value function and V_γ^π is the value function under policy π. Therefore,

$$V^*(s) = V_\gamma^{\pi^*}(s), \quad \forall s \in \mathcal{S} \tag{2.3}$$

where π^* is the discount optimal policy. More that one optimal policy might exist, but they all define the same value function.

2.1.2. Dynamic Programming

Generally the aim is to maximize the expected reward over a sequence of decisions. In the ideal case, the agent should take actions that maximize future rewards. In a Markov decision process, the expected future reward is dependent only on the current state and action. Therefore there must exist a stationary policy which will guarantee that the maximum expected rewards are received, if taken starting from the current state [25, 26]. The goal of any method to solve Markov decision processes is to identify the optimal policy. The optimal policy is one that will maximize the expected reward starting from any state.

Theoretically it might be possible to enumerate all of the possible policies for a state space, and then pick the one with the maximum expected value function. This method can be intractable, since the number of policies is exponential in the size of the state space. Methods of policy generation take advantage of the fact that the optimal policy will be locally optimal for each individual state.

The traditional approach to solving sequential decision problems is dynamic programming [23]. It is a model based approach assuming availability of complete information. Application of dynamic programming techniques requires precise knowledge of P, the transition probabilities and R, the reward function. Unfortunately, dynamic programming is computationally expensive in large state spaces.

The optimal *value* of a state is the expected infinite discounted sum of rewards the agent will receive if it starts in that state and executes the optimal policy. Using π as a complete decision policy, it is written

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \tag{2.4}$$

The optimal value function is unique and can be defined as the solution to the Bellman equations:

$$V^*(s) = \max_{a}\left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s)V^*(s')\right), \forall s \in \mathcal{S} \tag{2.5}$$

Given the optimal value function, the optimal policy can be specified as

$$\pi^*(s) = \operatorname*{argmax}_{a}\left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s)V^*(s')\right) \tag{2.6}$$

where $argmax_x(f(x))$ signifies the x that maximizes $f(x)$.

Two methods of calculating optimal policies in discounted Markov decision processes are value iteration and policy iteration. Both these methods attempt to modify the utilities of the neighboring states such that they will satisfy the Bellman equations. Repetition of this local modification process at each state in the state space for enough iterations will cause the utilities of the individual states to converge to their correct values.

1. Let k be the time step, initialized to zero
2. Initialize the value function, $V_k(s) = 0$, $\forall s \in \mathcal{S}$
3. Select $\epsilon > 0$
4. Repeat
   (a) Increment k
   (b) Compute $V_k(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s|a,s') V_{k-1}(s') \right)$, $\forall s \in \mathcal{S}$
5. Until $|V_k(s) - V_{k-1}(s)| \leq \epsilon \left(\frac{1-\gamma}{2\gamma} \right)$

Figure 4. The Value Iteration Algorithm.

A value iteration algorithm for a discrete time Markov decision process is shown in Figure 4. It will find a stationary policy that is ϵ-optimal within a finite number of iterations. The generated policy might be optimal, but the algorithm provides no means of determining this. Combination of this algorithm with methods for identifying suboptimal actions [24] can often ensure that the algorithm terminates with an optimal policy. In practice choosing ϵ small enough ensures that the algorithm stops with a policy that is very close to optimal.

2.2. Semi-Markov Decision Processes

Markov decision process models assume discrete time. Decisions occur at equally spaced points in time, after every state transition. For problems such as navigation, where the time to complete actions can vary, a more natural model would be one that incorporates time. Semi-Markov decision processes extend the usual discrete-time Markov decision process model by incorporating a continuous model of time. Actions are allowed only at discrete points in time. In between actions, the state can change. This is unlike a Markov decision process, where state changes are solely due to actions.

A semi-Markov decision process is a mathematical model of continuous time decision problems [24]. Formally a semi-Markov decision process is defined by the five tuple $\langle \mathcal{S}, \mathcal{A}, P, R, F \rangle$, where

- $\mathcal{S}$ is the finite set of environment states
- $\mathcal{A}$ is the set of actions
- P is the set of action dependent transition probabilities
- R is the reward function
- F is a function giving probability of transition times for each state action pair

As in the Markov decision process, P is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$. $P(s' \mid s, a)$ denotes the probability that the environment will make a transition from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ under action $a \in \mathcal{A}$. It is important to note that P describes the transitions at decision epochs only.

F is a function where $F(t|s,a)$ is the probability that the next decision epoch occurs within t time units, after the agent chooses action a in state s at a decision epoch. Let Q denote the joint probability that the system will be in state s' for the next decision epoch, at or before t time units after choosing action a in state s. The expected transition time between decision epochs can be calculated from Q, which can be computed from F and P by

$$Q(t, s'|s, a) = P(s'|s, a)F(t|s, a) \tag{2.7}$$

In general, the reward function for semi-Markov decision processes is more complex than in the Markov decision process model. In addition to the fixed reward

$k(s,a)$, accrued due to an action taken at a decision epoch, an additional reward may be accumulated at *rate* $c(s',s,a)$ for the time the natural process remains in state s' between the decision epochs. The natural process may change state several times between decision epochs. Therefore, the rate at which the rewards are accumulated between decision epochs may vary.

A discounted framework is assumed in this thesis. The goal is to compute an optimal stationary policy that maximizes the discounted sum of rewards. The continuous time discounting model is defined as follows: a reward of 1 at the current time will be worth $e^{-\beta t}$ at a time t units in the future. Given that the robot was in state s and chose action a, the expected reward between two decision epochs can be expressed as the sum of the lump-sum cost and the expected discounted rate cost over the transition time by

$$r(s,a) = k(s,a) + E_s^a \left\{ \int_0^\tau e^{-\beta t} c(W_t, s, a) dt \right\} \tag{2.8}$$

where τ is the transition time to the second decision epoch, and W_t denotes the state of the natural process. Here, E_s^a represents the expectation with respect to the transition time distribution $F(t|s,a)$.

Let $\pi : \mathcal{S} \rightarrow \mathcal{A}$ represent a stationary policy from states to actions. The value of a state s under a stationary policy π can be expressed as sum of the expected immediate reward until the next decision epoch and the expected discounted value of the resulting new state.

$$V_\beta^\pi(s) = r(s,a) + \sum_{s' \in S} \int_0^\infty e^{-\beta t} Q(dt, s'|s,a) V_\beta^\pi(s') \tag{2.9}$$

where $Q^{\pi}(dt, s'|s, a)$ is the joint probability distribution. By defining a discounted transition function

$$m(s'|s, a) = \int_0^{\infty} e^{-\beta t} Q(dt, s'|s, a)$$

the discounted value function can be expressed in a more conventional form by

$$V_{\beta}^{\pi}(s) = r(s, a) + \sum_{s' \in S} m(s'|s, a) V_{\beta}^{\pi}(s') \tag{2.10}$$

The aim is to compute an optimal discounted policy π^* that maximizes the value function $V^{\pi^*}(s)$ over all states. The value iteration algorithm can compute such a policy, given the underlying probabilistic transition model, and the cumulative distribution function.

2.3. Partially Observable Markov Decision Processes

In many real world situations it is not possible for the agent to have perfect and complete perception of the state of the environment. Partially observable Markov decision processes are a class of formal models suitable for controlled stochastic dynamic systems, such as robots and factories, that deal with the problem of calculating an optimal policy in an environment where complete state information is not available [27]. Because of the inability to determine the true state, partially observable Markov decision processes do not assume the Markovian property.

When the state is not completely observable, a model of observations must be added. This includes a finite set, $\mathcal{O}$, of possible observation and $O : S \times A \rightarrow \mathcal{O}$, a function mapping elements of $A \times S$ into discrete probability distributions over $\mathcal{O}$.

$O(o|s,a)$ represents the probability of making observation $o \in \mathcal{O}$ from state s after having taken action a.

A simplistic approach would be to take the set of observations as the set of states, and treat a partially observable Markov decision process as a Markov decision process. The problem is that the process would not necessarily be Markov since multiple states in the environment that are indistinguishable from immediate perceptual input could require different responses from the system. This can result in an optimal policy having arbitrarily poor performance.

2.3.1. State Estimation

A *belief state* is a discrete probability distribution over the set of environment states, $\mathcal{S}$, representing for each state the probability that the agent is currently in that state. Let $\mathcal{B}$ be the set of belief states. The probability assigned to state s when the agent's belief state is b is denoted by $b(s)$. The axioms of probability require that $0 \leq b(s) \leq 1$ for all $s \in \mathcal{S}$ and that $\sum_{s \in \mathcal{S}} b(s) = 1$.

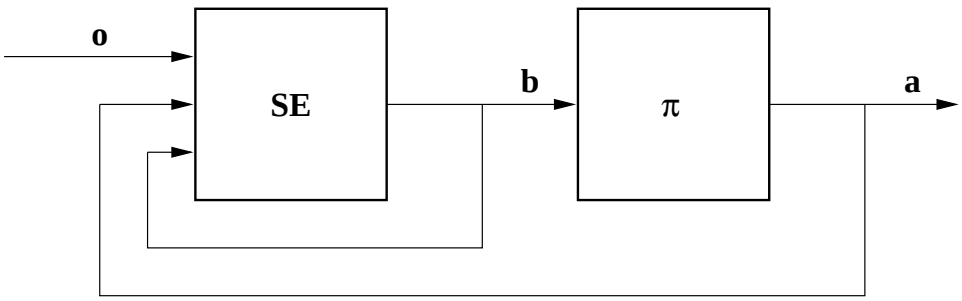


Figure 5. Controller for a POMDP.

The problem of acting in a partially observable environment can be decomposed as shown in Figure 5. The component labeled "SE" is the *state estimator*. It takes

as input the last belief state, the most recent action and the most recent observation, and returns an updated belief state. The second component, labeled π, is the *policy*. This updated belief state is used by the policy to select an action.

The state estimator can be constructed from P and O by straightforward application of Bayes' rule. The state estimator output is a belief state, represented as a vector of probabilities, one for each environment state, that sums to one. The component corresponding to state s', written $SE(s'|b,a,o)$, can be determined from the previous belief state b, the previous action a, and the current observation o by

$$\begin{aligned} SE(s'|b,a,o) &= Pr(s'|a,o,b) \\ &= \frac{Pr(o|s',a,b)Pr(s'|a,b)}{Pr(o|a,b)} \\ &= \frac{O(o|a,s')\sum_{s\in\mathcal{S}} P(s'|a,s)b(s)}{Pr(o|a,b)} \end{aligned} \tag{2.11}$$

where $Pr(o|a,b)$ is a normalizing factor defined as

$$Pr(o|a,b) = \sum_{s'\in\mathcal{S}} O(o|a,s') \sum_{s\in\mathcal{S}} P(s'|s,a)b(s) \tag{2.12}$$

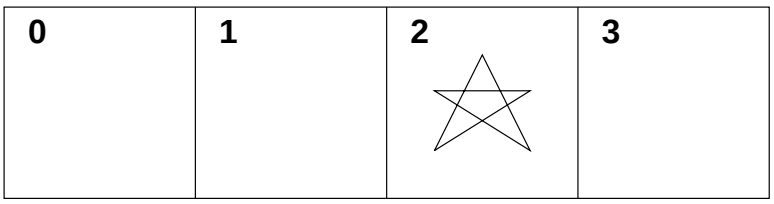


Figure 6. A Simple POMDP Environment.

A simple example of a partially observable Markov decision process is shown in Figure 6 [27]. Of the four states, state 2 is designated as the goal state. An agent

is in one state at all times. The two allowable actions, left and right, move it in one state in either direction. If it attempts to move into a wall, it stays in the same state. If the goal state is reached, the agent receives a reward of 1, and is then moved, with equal probability, into state 0, 1, or 3. If the agent can observe the what state it is in, the problem becomes trivial. It becomes more difficult when it can only observe whether or not it is currently in the goal state.

When the agent cannot observe its true state, it can represent its belief of where it is with a probability vector. After leaving the goal, the agent moves to one of the other states with equal probability. This is represented by a belief state of $\langle \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3} \rangle$. After taking a "right" action and not observing the goal, there are only two states from which an agent could have moved: 0 and 3. The agent's new belief vector is $\langle 0, \frac{1}{2}, 0, \frac{1}{2} \rangle$. If a "right" action is taken again and the goal is not encountered, the agent can be sure that it is now in state 3 with belief state $\langle 0, 0, 0, 1 \rangle$. This example has deterministic actions, resulting in the agent's uncertainty shrinking at each step. Generally, some actions in some situations will decrease the uncertainty while others will increase it.

2.3.2. Planning

A variety of algorithms have been developed for explicitly solving partially observable Markov decision processes [28]. However the problem is so computationally challenging [29] that most techniques are too inefficient to be used on all but the smallest problems. The Witness algorithm [30, 31] finds exact solutions to discounted finite-horizon partially observable Markov decision processes using value iteration. Although it has been used in problems with up to 16 states, it is not efficient enough to be used for larger problems. The hybrid approach of Littman *et. al.*[32] is capable of determining high quality policies for partially observable Markov decision

processes with nearly 100 states. However this is not a feasible alternative to handle the thousands of states needed to address realistic problems.

The key to finding optimal policies in the partially observable case is that the problem can be cast as a *completely observable* Markov decision process. This "belief MDP" can be defined as follows:

- $\mathcal{B}$ is the set of belief states, comprising the state space
- $\mathcal{A}$ is the set of actions
- τ is the state transition function
- ρ is the reward function on belief states

The belief state set is $\mathcal{B}$ and the action set is $\mathcal{A}$. For a current belief state b and action a, there are only $|\mathcal{O}|$ possible successor belief states b'. The new state transition function τ can be defined as

$$\tau(b'|a,b) = \sum_{\{o \in \mathcal{O} | SE(b'|b,a,o)\}} Pr(o|a,b) \tag{2.13}$$

The reward function on the belief states can be constructed from the original reward function on world states and is given by

$$\rho(b,a) = \sum_{s \in \mathcal{S}} b(s) R(s,a) \tag{2.14}$$

The reward function might seem strange, since the agent is rewarded for believing that it is in a good state. However, since the state estimator is constructed from a correct observation and transition model of the world, it is not possible for the agent to purposely delude itself into believing that it is in a good state when it is not.

The partially observable Markov decision process planning problem can be transformed in a planning problem for a completely observable Markov decision process [21, 22]. Since the decision maker can never be sure of the state of the partially observable Markov decision process, the set of executable actions $\mathcal{A}$ must be the same for every state s. This belief Markov decision process is such that an optimal policy for it will give rise to optimal behavior for the original partially observable Markov decision process [33, 34]. This means that there always exists a partially observable Markov decision process policy that maximizes the expected total reward. Since this policy can be pre-computed, the decision maker just calculates the current state distribution and then looks up which action to execute.

2.4. Occupancy Grids

Occupancy grids [19] are a spatial representation that describe space as a Cartesian grid where each cell has a certain probability of being occupied. Initially, each of these cell probabilities should be set to the estimated prior probability of cell occupancy [35]. For example, if one fifth of the space in a given area is occupied, the prior probability could be set to 0.20. However, in practice, occupancy grids tend to be insensitive to errors in the prior probability, and an estimate of 0.5 works well.

Sensor inputs are used in updating the occupancy grids by using a sensor model that describes the probability of cells being occupied given the reading received. In the case of time-of-flight sonars, a sonar transducer emits a short pulse of sound and then listens for the echo. The time reading between the sound pulse emission and echo reception is measured and referred to as a *range reading*. It corresponds to half the distance the sound pulse traverses in that time.

The sonar model is divided into two areas: a cone-like freespace hypothesis, where the posterior probability of occupancy will be lowered, and an arc-like surface

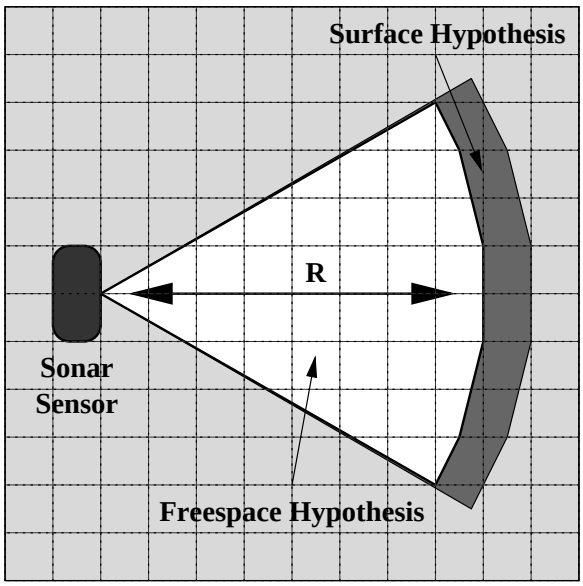


Figure 7. Sonar Sensor Interpretation. Cells in the freespace hypothesis have their occupancy probabilities lowered, while cells in the surface hypothesis have their occupancy probabilities raised. Occupancy probabilities of other cells remain unmodified.

hypothesis, where it will be raised (Figure 7). The sonar emits a pulse of sound in a cone that expands as it gets farther from the transducer. When the pulse hits an obstacle, it is reflected back to the sensor. A range reading of R indicates that an obstacle has been detected somewhere along the sonar arc at range R, so the occupancy probability of all cells along this arc should be increased. At the same time, this indicates that no obstacle was detected at a range closer than R, so all cells within the sonar cone at range less than R should have their occupancy probabilities reduced. Cells at ranges beyond R are not affected, since obstacles at range R prevented the sonar from obtaining any information about them.

Given X, representing information from a sensor reading, the probability that a cell is occupied is $p(o|X)$, and the probability that the cell is not occupied is $p(\neg o|X)$. From Bayes' theorem [35]:

$$\frac{p(o|X)}{p(\neg o|X)} = \frac{p(X|o)}{p(X|\neg o)} \times \frac{p(o)}{p(\neg o)} \tag{2.15}$$

where $p(X|o)$ is the probability of receiving information X given that this cell is occupied, $p(X|\neg o)$ is the probability of receiving information X given that this cell is not occupied, $p(o)$ is the prior probability that any given cell is occupied, and $p(\neg o)$ is the prior probability that any given cell is unoccupied, where $p(o) + p(\neg o) = 1$.

Let A represent the current grid state, and B represent the information from a new sensor reading. The cell occupancy probability can be combined [20] using

$$\frac{p(o|A \cap B)}{p(\neg o|A \cap B)} = \frac{p(o|A)}{p(\neg o|A)} \times \frac{p(o|B)}{p(\neg o|B)} \times \frac{p(\neg o)}{p(o)} \tag{2.16}$$

This assumes that A and B represent independent information. This is not necessarily true, since a particular point can be sensed more than once, either by the same or different sensors. For example, if sonar cones overlap for two sensor readings, cells in

the common region would have their probabilities updated twice. This means that although the numerical occupancy probabilities are not reliable. However, the overall occupancy results tend to be accurate.

When the odds ratio involves a probability and its complement, odds and the probabilities can be interconverted by the following relationship

$$Odds(A) = \frac{p(A)}{1 - p(A)} \Longleftrightarrow p(A) = \frac{Odds(A)}{1 + Odds(A)} \tag{2.17}$$

Using an odds formulation, B, the new sensor information, can be combined with the current grid state A by

$$Odds(o|A \cap B) = \frac{Odds(o|A) \times Odds(o|B)}{Odds(o)} \tag{2.18}$$

Utilizing a logarithmic formulation allows each cell update to be computed with a single addition.

2.5. Artificial Neural Networks

Artificial neural networks are computer models of the mental processes in the brain [36]. They consist of processing elements, or units, that attempt to model some properties of neurons. The units that form the input layer act as "sensors", receiving their inputs from outside the network. Units not in the input layer receive their inputs from other units. In some implementations, a signal is sent to the other units only if the total strength of all the inputs to a unit exceeds a certain threshold. Implementations not using thresholding compute an output that is the sum of the weighted inputs. The outer layer units produce the "activity", the external output of the system.

Various rules can be used to adjust the internal processing. The adjustment of the weights controls the learning of the artificial neural network. The power of this representation lies in the interconnections between units. No single unit provides a clue to the overall picture. The overall pattern of interaction among units determine the properties of the network.

Artificial neural networks can process information and carry out solutions almost simultaneously, because of their parallelism [36]. Knowledge is distributed throughout the network as a dynamic response to the inputs and the network architecture. They provide a general and practical method for learning functions from labeled examples and tend to be robust to error in the training data.

The network most widely used in addressing problems requiring recognition of complex patterns and performing nontrivial mapping functions is the backpropagation network [37, 38]. The backpropagation algorithm learns weights in multi-layer feed-forward networks, where interconnections are given and fixed. The network learns a set of input-output example pairs using a two-phase propagate-adapt cycle. Input patterns are propagated through the net to generate an output. The output pattern is compared to the desired output in computing the error signal for each output unit. The error signals are transmitted backward from the output layer to each node in the intermediate layer contributing directly to the output. Each unit receives only a portion of the total error, based roughly on the relative contribution it made to the original output. This process is repeated for all layers. Gradient descent is performed by simultaneously refining the weights of all network units by attempting to minimize the squared error between the network outputs and the target value for these outputs, provided by the training data. This causes the network to converge toward a state that allows all the training patterns to be encoded.

As the network trains, the units in the intermediate layers are organized such that different units learn to recognize different features of the total input space. After

training, when presented with an arbitrary input pattern that is noisy or incomplete, the units in the hidden layers of the network will respond with an active output if the new input contains a pattern that resembles the feature the individual units learned to recognize during training. If the inputs pattern does not contain the feature that the units were trained to recognize, their outputs will be inhibited. Backpropagation networks provide an effective means to examine data patterns that may be incomplete or noisy, and to recognize subtle patterns from the partial input.

NETtalk [39] is an example of a neural network that used the backpropagation algorithm to learn to read out loud. The input layer had 7 groups of 29 units. The hidden layer was comprised of 80 units and attempted to improve the feature detection needed for the input/output transformation. The output layer had 26 units that encoded phonemes and stresses and drove the sound synthesizer. NETtalk examined a window of seven character at a time. Errors were corrected and backpropagated. At first the talk was gibberish. After a night, it achieved a 95-percent performance level, reading with a few mistakes. People are capable of handling the inconsistencies in spelling and pronunciation of the English language. An inflexible rule-based system would encounter significant problems, but NETtalk was capable of representing these complex rules.

2.6. Related Work

Xavier [21] is a robot navigation architecture based on a partially observable Markov decision process model. It is a multi-layer system that includes the partially observable Markov decision process-based navigation architecture, a servo-control layer that controls the motors of the robot, an obstacle avoidance layer that keeps the robot moving smoothly in a goal direction while avoiding static and dynamic obstacles, a path planning layer that reasons about uncertainty to choose paths that

have high expected utility and a multiple-task planning layer that uses PRODIGY, a symbolic, non-linear planner, to integrate and schedule delivery requests that arrive asynchronously. The layers are implemented as a number of distributed, concurrent processes operating on several processors, are integrated using the Task Control Architecture, that provides facilities for interprocess communication, task decomposition and sequencing, execution monitoring and exception handling, and resource management.

The partially observable Markov decision process planning problem is transformed into a planning problem for a completely observable Markov decision process model, where the belief state comprises the state of the model. GROW-BW [40, 41], an extension of the Baum-Welch algorithm, is used to tune the initial sensor and actuator models to better match the environment of the robot while it learns the distances. The algorithm learns a probability distribution over the maximal and minimal bounds on the length of each corridor segment. Application of the algorithm to an initial partially observable Markov decision process and a history of actions and observations, allows for the generation of a partially observable Markov decision process that better fits the trace by updating the observation and transition probabilities. Application of the Viterbi [42] algorithm enables determination of the most likely continuous sequence of states encountered by the partially observable Markov decision process.

The office-navigating robot DERVISH [43] won the Office Delivery event of the 1994 Robot Competition and Exhibition, held as part of the 13th National Conference on Artificial Intelligence. DERVISH navigates in real office environments via a topological map. The topological map contains no distance information, only approximate hallway and doorway widths. DERVISH uses a partially observable Markov decision process to maintain a sense of position. This state set is updated whenever the robot discerns a new percept. Planning is based on the assumption that DERVISH is in the

most likely state. The resultant path is executed, while continually updating the state set until either the goal is reached or the current most likely state is no longer on the path, at which point it replans. Incorrect or missing percepts can cause DERVISH to temporarily assume the wrong state, but the correct state is retained in the state set and becomes the most likely state after one or more correctly detected percepts. Updates are based on a model of observational error. One weakness in DERVISH is that it does not incorporate metric information.

Planning in Xavier and DERVISH is based on collapsing the partially observable Markov decision process into a completely observable "belief" Markov decision process. Cassandra *et. al.*[22] also utilize partially observable Markov decision processes in constructing a layered navigation system for their mobile robot. In constructing the state space, the office environment is discretized into one square meter segments. The focus of their research is heuristic approximations to the optimal control strategy which is generated through the *most likely state*, *voting*, Q_{MDP}, *action entropy* and *entropy weighting* methods.

Although it does not utilize partially observable Markov decision processes, the approach used in RHINO[44, 18] also integrates both metric and landmark based approaches. Occupancy grid based maps are learned using artificial neural networks and Bayesian integration. Topological maps are then generated from these maps by using Voronoi diagrams to partition them into coherent regions. This approach take advantage of the accuracy and consistency of a metric-based approach and the efficiency of a landmark-based approach. Localization is performed via wheel encoders, map correlation and wall orientation. Maps are autonomously acquired through greedy exploration that directs the robot to move on a minimum-cost path to the nearest grid cell, where the cost for traversing a grid cell is determined by its occupancy value. Path planning is performed on the abstract topological map by using a value iteration approach.

CHAPTER 3

A ROBOT NAVIGATION ARCHITECTURE

This chapter describes the navigation architecture implemented on the robot PAVLOV. The POSMDP planning layer uses an abstract Bayesian model of the environment, and the robot's actions and observations. The planner receives observations, maintains a belief state and chooses actions. The reactive behavior level is responsible for executing the abstract actions from the planner on the robot, while the ANN is responsible for determining observations for the planner from local occupancy grids.

3.1. The POSMDP Planning Layer

The planning layer assumes an office-type environment, made up of corridors and intersections oriented along two orthogonal axes. It is provided with approximate metric information about the office environment via a *topological map* (Figure 3). A topological map is a graph of nodes and edges that contain information about the connectivity of the environment. Nodes represent junctions between corridors and rooms. Edges, containing approximate length information, represent corridors and entrances into rooms.

A state map is generated from this map. The continuous state space is discretized into locations of one square meter, and an orientation in one of four compass directions. A *Markov state* consists of a location and an allowed orientation. Figure 8 shows a *Markov node* which is a collection of the four Markov states at one location. The location represented by a Markov node corresponds to an area of one square

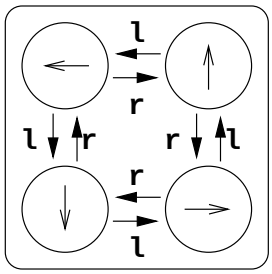


Figure 8. A Markov Node. This a a group of four Markov states modeling a one square meter location. Each state at a node is oriented along one of four compass directions. An `l` indicates a *turn-left* action, while an `r` signifies a *turn-right* action.

meter. The connections between states at one location show the states resulting from deterministic turn actions. If the robot is facing North, a *turn-left* action will move it to the West state, while a *turn-right* action will move in to the South state, both in the same Markov node.

The map in Figure 9 shows the corresponding discretized environment, represented by Markov nodes. The state map describes the underlying connectivity of the states, and helps in defining the ideal observations for each state. For example, if the robot is facing North in a corridor oriented along the East-West axis, it should see, in an ideal environment, walls in the front and at the back, and openings on either side.

Figure 10 represents the general architecture of the planner. For all of our experiments, the distribution was initialized by setting the probability of the chosen starting location to 1. Goals can then be specified and correspond to any topological node specified in Figure 3.

The POSMDP model extends the SMDP model through the incorporation of an observational model. The robot is unable to perceive the true states of the

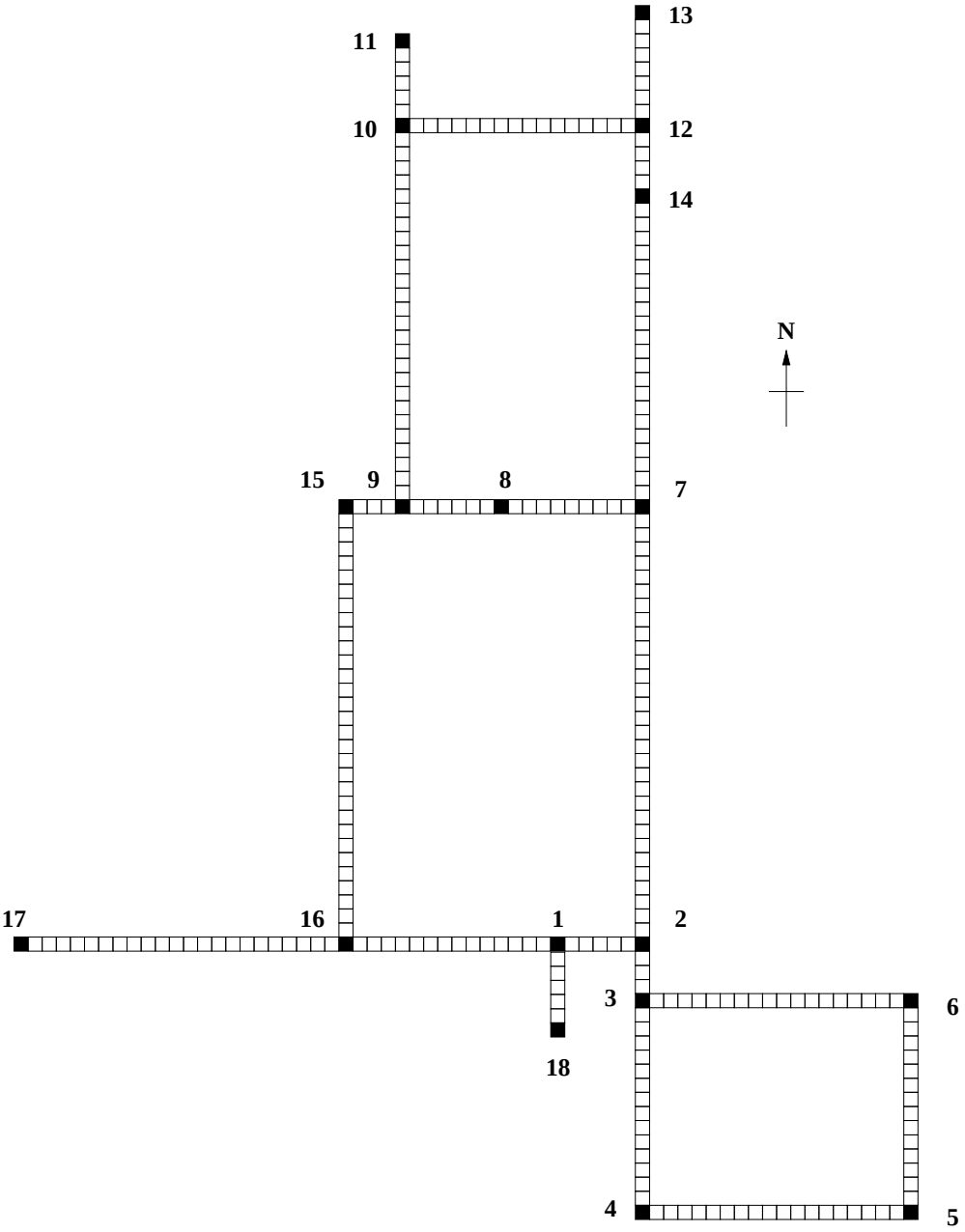


Figure 9. The Markov Map. The Markov map is generated from the topological map (Figure 3). This is a discretization of the physical space into the Markov state space. Each square represents a Markov node, a collection of four Markov states. The topological map is segmented into 288 Markov nodes, corresponding to 1152 Markov states

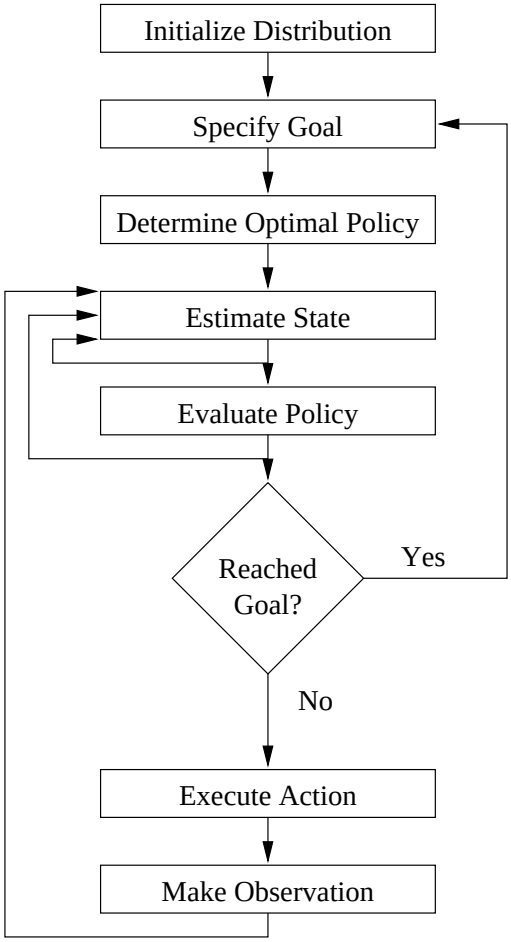


Figure 10. Flow Diagram for the Planning Layer.

environment. It is only able to make observations. The model is partially observable because the size of $\mathcal{S}$ is much larger than that of the observations $\mathcal{O}$. For example, in this thesis $|\mathcal{S}| = 1152$ and $|\mathcal{O}| = 64$.

If the current state distribution is α_{prior}, the state distribution α_{post}, after the execution of an abstract action a, is given by

$$\alpha_{post}(s) = \frac{1}{scale} \sum_{s' \in S | a \in A(s')} P(s|s', a)\alpha_{prior}(s'), \quad \forall s \in \mathcal{S} \tag{3.1}$$

This updated state distribution now serves as α_{prior} when the state distribution is updated to α_{post}, after an abstract observation o

$$\alpha_{post}(s) = \frac{1}{scale} O(o|s)\alpha_{prior}(s), \quad \forall s \in \mathcal{S} \tag{3.2}$$

In both updates, $scale$ is a normalization constant that ensures that

$$\sum_{s \in S} \alpha_{post}(s) = 1$$

This is necessary since not every action is defined in every state (for example, the action *go-forward* is not defined in states where the robot is facing a wall).

3.1.1. Abstract Actions

The robot has four abstract actions: *go-forward*, *turn-left*, *turn-right* and *no-action*. Table 1 shows the action transition probabilities used in experiments. These probabilities were obtained empirically. The actions are defined in terms of possible outcomes. For example, assume that the robot is situated in the middle of a corridor oriented along the North-South axis, facing South. A *go-forward* action has three

possible outcomes: there will be no change of state with probability 0.05 (N), it will advance South by one meter with probability 0.90 (F), or it will advance South by two meters with probability 0.05 (F-F). A *turn-left* action from this state also has three possible outcomes: there will be no change of state with probability 0.05 (N), the robot will end up in the West facing state at the same location with probability 0.90 (L), or it will end up in the North facing state at the same location with probability 0.05 (L-L). A *turn-right* state moves in a counter-clockwise direction, with similar results. If a *no-action* directive is issued, then the robot will stay in the same state with probability 1.00. The robot is deemed to have reached the goal node if it receives a *no-action* directive.

Table 1. Transition Probabilities for Abstract Actions

Action	State		
	Same	Next	Next-Next
go-forward	0.05 (N)	0.90 (F)	0.05 (F-F)
turn-left	0.05 (N)	0.90 (L)	0.05 (L-L)
turn-right	0.05 (N)	0.90 (R)	0.05 (R-R)
no-action	1.00 (N)	0.00	0.00

3.1.2. Abstract Observations

In each state, the robot is able to make an abstract observation. This is facilitated through the modeling of four virtual sensors that can perceive features in the nominal directions front, left, back and right. Each sensor is capable of determining if a percept is a *wall*, an *opening*, a *door* or if it is *undefined*. An abstract observation is a combination of the percepts in each direction, and thus there are 64 possible abstract observations. Table 2 shows the conditional probabilities for the abstract

features, obtained empirically. The observation model specifies, for each state and action, the probability that a particular observation will be made.

Table 2. Conditional Observational Probabilities

Percept	Feature			
	wall	*opening*	*door*	*undefined*
wall	0.75	0.20	0.15	0.00
opening	0.20	0.70	0.15	0.00
door	0.00	0.00	0.69	0.00
undefined	0.05	0.10	0.01	1.00

Denote the set of virtual sensors by I and the set of features that sensor $i \in I$ can report on by $Q(i)$. The sensor model is specified by the probabilities $v_i(f|s)$ for all $i \in I$, $f \in Q(i)$, and $s \in \mathcal{S}$, encoding the sensor uncertainty. $v_i(s)$ is the probability with which sensor i reports feature f in state s. An observation o is the aggregate of the reports from each sensor. This is not explicitly represented. We calculate only the observation probability. Thus, if sensor i reports feature f, then

$$O(o|s) = \prod_{i \in I} v_i(f|s) \tag{3.3}$$

Given the state, this assumes sensor reports from different sensors are independent. Assume that the robot somewhere in North-South corridor, oriented North. In the ideal case, the sensor report should be:

$$(front\ opening)\ (left\ wall)\ (back\ opening)\ (right\ wall)$$

However, the actual sensor report might read:

$$(front\ wall)\ (left\ undefined)\ (back\ opening)\ (right\ wall)$$

The individual sensor probabilities are then:

$$v_{front}(wall|opening) = 0.20$$
$$v_{left}(undefined|wall) = 0.05$$
$$v_{right}(opening|opening) = 0.70$$
$$v_{back}(wall|wall) = 0.75$$

The aggregate of these probabilities produces the observation probability.

3.1.3. Probabilistic Planning in the POSMDP

It is well known that computing exact optimal policies for POMDPs is intractable [32, 45]. There is ongoing research on computing approximately optimal policies [46], but most of the previous POMDP-based robot navigation systems assume that the underlying MDP is completely observable. Then, the optimal policy for the MDP is computed, and a variety of heuristic approximations are used to obtain the policy for the POMDP. For example, the DERVISH system [43] uses the policy associated with the most likely state, that is $\pi(\alpha(s)) = \pi^*(\text{argmax}_{s \in S}(\alpha(s)))$. Here, π is the POMDP policy, and π^* is the optimal policy of the underlying MDP. In the XAVIER system, a voting strategy is used, whereby the action selected is the one with the highest mass, namely $\text{argmax}_{a \in A} \sum_{s \in S | \pi^*(s) = a} \alpha(s)$. A detailed comparison of these heuristic strategies is given in [22].

For the results reported in this thesis, we used the *most likely state* strategy to determine the status of the POSMDP since this strategy was found to work well in noisy environments [22]. This reduces to finding the world state with the highest

probability and executing the action that would be optimal for that state in the SMDP:

$$\pi_{mls}(b) = \pi^*(argmax_s b(s)) \tag{3.4}$$

1. Let k be the time step, initialized to zero
2. Initialize the value function, $V_k(s) = 0$, $\forall s \in \mathcal{S}$
3. Select $\epsilon > 0$
4. Repeat
   (a) Increment k
   (b) Compute
   $$V_k(s) = \max_a \left(R(s,a) + \sum_{s' \in \mathcal{S}} P(s|a,s') V_{k-1}(s') \int_0^\infty e^{-\beta t} F(dt|a,s') \right), \ \forall s \in \mathcal{S}$$
5. Until $sp\,|V_k(s) - V_{k-1}(s)| \leq \epsilon \left(\frac{1-e^{-\beta}}{e^{-\beta}} \right)$

Figure 11. The Modified Value Iteration Algorithm.

Figure 11 describes the value iteration procedure for discounted SMDPs, which is used to determine an ϵ-optimal policy. Value iteration [47] is based on the Bellman optimality equations [24, 26, 48]. It iteratively computes a better approximation of the optimal value function, based on the sum of the immediate reward received over a decision epoch and the discounted value of the next state. We use an improved stopping criterion based on the span semi-norm [24], which usually improves performance by about 25 percent. The span is defined as $sp(f(s)) = max_{s \in \mathcal{S}} f(s) - min_{s \in \mathcal{S}} f(s)$, where $f(s) = |V_k(s) - V_{k-1}(s)|$. The constant $e^{-\beta}$ is essentially the discount factor γ for a discrete-time MDP.

3.1.4. Temporal Models of Robot Actions

An important characteristic of the POSMDP model is that actions are modeled as taking random amounts of time. Different distributional models for actions can be specified. The generic uniform distribution model used in this thesis is shown in Figure 12.

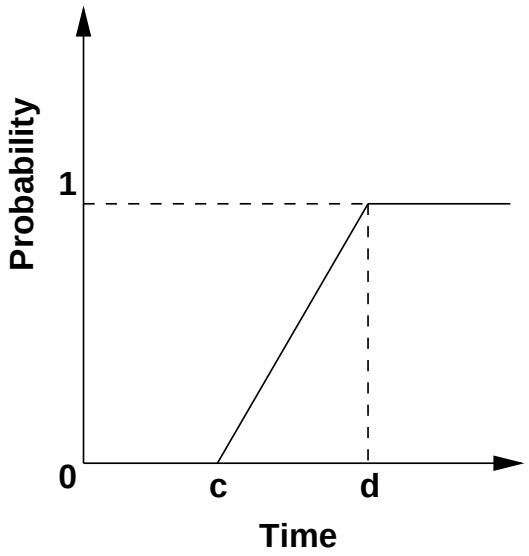


Figure 12. Temporal Model of Robot Actions. The temporal model of an action gives the probability of the next decision epoch occurring in a given interval. The action takes time uniformly distributed in the range [c, d]

Here, the transition time for an action is modeled as taking a random time between c and d units of time. This model can be used for different actions by specifying c and d. For the uniform distribution model, the distribution function

$F(t|a,s)$ can be written as

$$F(t|s,a) = \begin{cases} 0 & (0 \leq t < c) \\ \frac{t-c}{d-c} & (c \leq t \leq d) \\ 1 & (t > d) \end{cases} \tag{3.5}$$

The discounted transition function $m(s'|s,a)$ can by computed by

$$\begin{aligned} m(s'|s,a) &= \int_0^\infty e^{-\beta t} Q(dt, s'|s,a) \\ &= P(s'|a,s) \int_0^\infty e^{-\beta t} F(dt, s'|s,a) \\ &= \frac{P(s'|a,s)(e^{-\beta c} - e^{-\beta d})}{\beta(d-c)} \end{aligned}$$

and is used in value iteration (Figure 11) to determine an ϵ-optimal policy.

Table 3. Idealized Transition Times for Abstract Actions

Action	c (sec)	d (sec)
go-forward (Interior)	5	10
go-forward (Intersection)	10	25
turn-left, turn-right (Everywhere)	5	10
go-forward (Cluttered Corridor)	20	100

The upper and lower bound values used in PAVLOV depend upon what action is performed, and where it is performed (see Table 3). For example, if the robot believes itself to be in the middle of a corridor, a *go-forward* action takes anywhere from 10 to 20 seconds (depending on the speed of the robot and encountered obstacles). The same action takes twice as long near intersections, since the robot usually slows down there. Turns take the same time everywhere. Busy corridors are modeled by increasing the variance on the transition time (e.g. anywhere from 20 to a 100 seconds).

Another generic model is the truncated exponential model, defined as

$$F(t|s,a) = \begin{cases} 0 & (0 \leq t < c) \\ \frac{e^{-\delta c} - e^{-\delta t}}{e^{-\delta c} - e^{-\delta d}} & (c \leq t \leq d) \\ 1 & (t > d) \end{cases} \tag{3.6}$$

This model can accommodate a variety of temporal profiles by suitably choosing δ. Low values (e.g. $\delta = 0.01$) will closely approximate a linear increase, and higher values will generate a nonlinear profile.

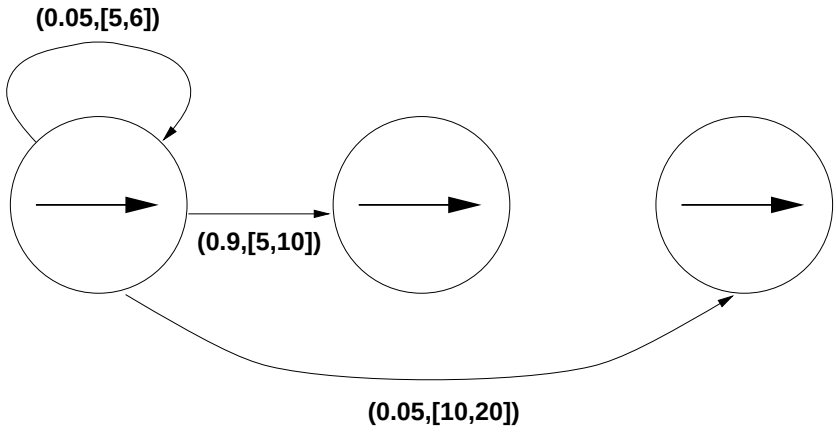


Figure 13. Example of an SMDP Model of the Forward Action. The figure displays the transition probabilities and the transition times. With high probability, the robot moves one grid cell with a transition time distributed uniformly between 5 and 10 seconds. With low probability, the robot stays in the same cell, or moves two cells, in a shorter or longer time interval, respectively.

One immediate question that arises is whether the transition times can be learned, say from execution traces. We use the Viterbi algorithm [42] (Figure 14) to determine the most likely sequence of states from the execution trace. An execution trace is a snapshot of the POSMDP at a decision epoch. It is a record of the probability distribution, the observation, the action and the time required to complete the

1. Set $scale'_1 = \sum_{s \in \mathcal{S}} O(o_1|s)\alpha_1(s)$
2. Set $\alpha'_1(s) = O(o_1|s)\alpha_1(s)/scale'_1, \ \forall s \in \mathcal{S}$
3. For $k = 1$ to $K - 1$
   (a) Set $scale'_{k+1} = \sum_{s \in \mathcal{S}} O(o_{k+1}|s) \max_{s' \in \mathcal{S} \wedge a_k \in \mathcal{A}(s')}[P(s|s', a)\alpha'_k(s')]$
   (b) Set $\alpha'_{k+1}(s) = O(o_{k+1}|s) \max_{s' \in \mathcal{S} \wedge a_k \in \mathcal{A}(s')}[P(s|s', a)\alpha'_k(s')]/scale'_{k+1}, \ \forall s \in \mathcal{S}$
4. Set $\bar{s}_K = \max_{s \in \mathcal{S}} \alpha'_K(s)$
5. For $k = K - 1$ to 1
   (a) Set $\bar{s}_k = \text{argmax}_{s' \in \mathcal{S} \wedge a_k \in \mathcal{A}} \alpha'_k(s') P(\bar{s}_{k+1}|s', a_k)$

Figure 14. The Viterbi Algorithm.

action. The observation at the first decision epoch is used to normalize the initial probability distribution. Moving forward in time until the penultimate decision epoch, scaling factors are computed for each probability distribution in the execution trace. The scaling factors, used in normalizing the probability distribution, and the updated state probabilities are dependent on the current observational probability and state probabilities from the previous decision epoch. Once all the probability distributions in the execution trace have been updated, the robot is estimated to be in the maximum probability state, in the current distribution. Starting from the penultimate decision epoch, and moving backward in time, the most likely state at each decision epoch is determined by taking the *argmax*[2] over all states, and all actions allowed at each state, in an execution trace. Moving forward in time, the observed transition times are then associated with the action that caused the transition from the current state to the next state in the Viterbi sequence.

[2] $\text{argmax}_x(f(x))$ signifies the x that maximizes $f(x)$

3.2. The Reactive Behavior Layer

The reactive behavior layer is implemented on PAVLOV (Figure 1), a Nomadic Technologies Nomad 200 mobile robot based on the approach described by Connell [49]. PAVLOV has a ring of 16 Polaroid 6500 sonar ranging modules and 16 infra-red sensors evenly distributed about its circumference at regular spacings of 22.5 degrees. The infra-red sensors are fairly reliable short range proximity sensors, returning distance values ranging from 0 inches to 15 inches. The ultrasonic sensors give longer range proximity information, returning distance values ranging from 5 inches to 255 inches, but are highly prone to specular reflections. A set of 20 bump sensors encircle the robot, enabling it to determine if it is in physical contact with an object. Odometric and angular information is also maintained, but this becomes inaccurate over long runs.

This layer supports the abstract actions *go-forward*, *turn-left*, *turn-right* and *no-action*. This is accomplished by an action vector that controls the translation and rotational speed of the robot. Multiple behaviors modify this action vector.

A *no-action* directive causes the robot to stop all motions, and is only issued by a goal state.

The *turn-left* and *turn-right* directive reorient the robot to the axis orthogonal to the current one. The `heading_align` behavior forces the robot to turn to an axis orthogonal to the current one, while the `avoid_bump` behavior causes the robot to back up and turn, if the bump sensors indicate the robot is in physical contact with an obstacle.

A *go-forward* directive attempts to move the robot approximately 1 meter forward along its current axis. Movement along the orthogonal axis, resulting when the robot is attempting to avoid obstacles, is not used in determining the distance

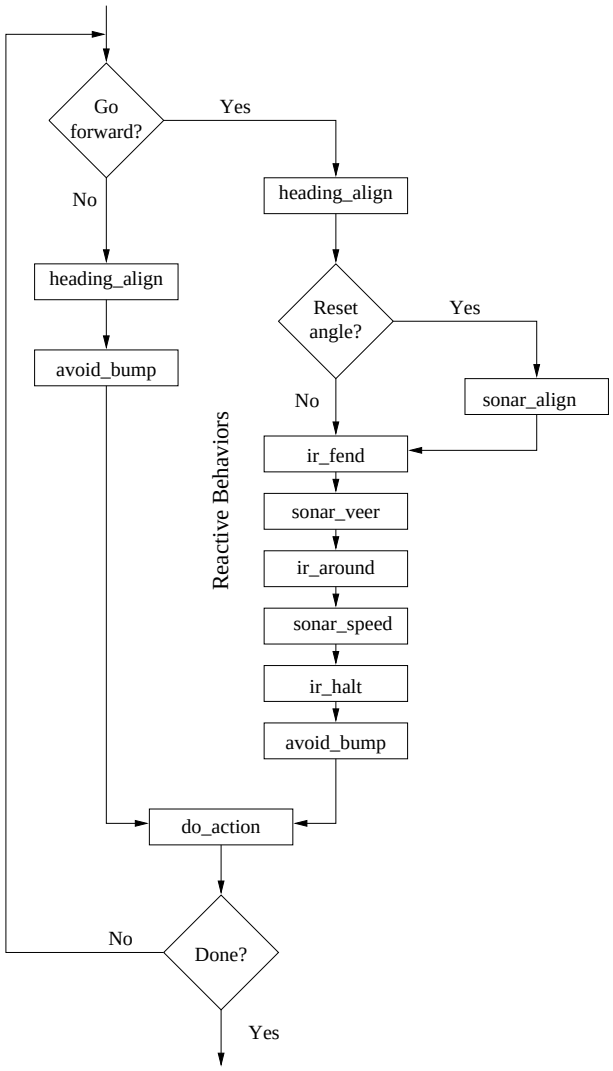


Figure 15. Flow Diagram for the Reactive Behavior Layer. For turns (*turn-left*, *turn-right*), only two behaviors are activated. For a *go-forward* directive, all behaviors are initially active. Once the robot is aligned, the `sonar_align` behavior is deactivated. The action vector is repeatedly modified until an action is deemed to have been completed

traveled. The `heading_align` behavior attempts to keep the robot oriented in approximately the current direction, as it moves forward. At the beginning of every *go-forward* action the `sonar_align` behavior forces the robot to stay parallel to walls. This is accomplished by attempting to fit sonar data to lines, and altering the action vector to align parallel to observed lines, if the fit is good enough. Once it judges the robot to be parallel to a wall, it resets the internal angle to the current heading and is deactivated, until the next *go-forward* action. This is necessary to compensate for angular drifts, which becomes very noticeable over long runs. Responsibility for orienting the robot to the desired heading is now the sole responsibility of `heading_align`. Short range infra-red sensor readings are interpreted by `ir_fend` (prevents the robot from side swiping wall), `ir_around` (forces the robot to avoid obstacles directly in front by turning in the direction of least resistance), and `ir_halt` (forces the robot to stop moving if the front infra-red sensors judge an obstacle to be closer than a threshold). Longer range sonar information is interpreted by `sonar_veer` (steers the robot to avoid obstacles in front) and `sonar_speed` (causes the robot to slow down or stop, depending on whether an obstacle is detected in front, and the judged distance). `avoid_bump` interprets data from the ring of bump sensors.

The action vector is interpreted by `do_action`, which continuously sets the rotation and translation speeds of the robot. This is repeated until the desired rotation or translation is accomplished, indicating the end of the abstract action. Control returns to the planning layer, from which the next abstract action is issued.

3.3. Feature Detectors with ANNs

A four square meter robo-centric local occupancy grid of size 32 x 32 is continually updated while the abstract actions are executed. This provides the robot with a spatial representation of the environment, from which features can be extracted.

In our office environment, most of the walls are very smooth and non-textured. Specular reflections become very prevalent. Konolige [50] use a sensor model that is a mixture of specular and diffuse reflections to update occupancy grids. Yamauchi [35] recommends rotating the sonar sensors through a range of angles equivalent to the width of the sonar arc while constructing the occupancy grids. If both specular and non-specular reflections are possible from a given viewpoint, then both will be incorporated into the evidence grid.

Since the elimination of specularities is not the focus of this thesis, we use a simple approach of incorporating long sonar returns from sensors either parallel or orthogonal to the walls in updating the grid. For all other sonars, only ranges below 4 meters were incorporated into the occupancy, since out widest corridors are about 3 meters wide. Even after these standard efforts, effects of the specular readings are very noticeable in the occupancy grids. These specularities make feature detection from occupancy grids a challenging task.

In our experiments, the occupancy grids were updated according to the methods defined by [20, 51, 19]. This approach assumes independent sensor readings in updating occupancy cells. Thrun [44] trains an artificial neural network using back-propagation to map sonar measurements to occupancy values. This approach does not make a conditional independence assumption between adjacent sensors measurements and could possibly do a better job in eliminating specularities.

Each local grid of 1024 values is divided into 4 regions - left and right halves, and top and bottom halves. Consequently each region has 512 inputs, which are directly applied to a neural net. We used a single hidden layer feed-forward neural net (Figure 16) comprised of asymmetrical sigmoid units to detect features from the local occupancy grids. Each of the 512 input units is fully connected to a hidden layer of 32 units, which is, in turn, fully connected to an output layer of 4 units. Each of the outputs indicates the confidence in the corresponding feature - *door*, *opening*,

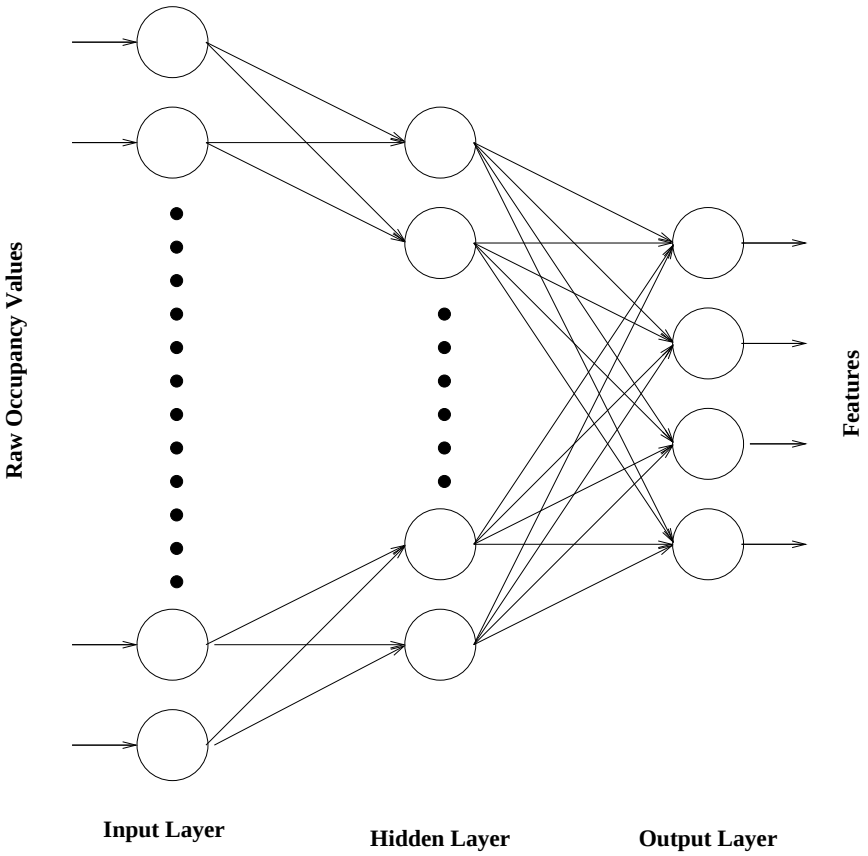


Figure 16. The Artificial Neural Network. 512 raw occupancy cell probabilities serve as input to the net and connect up to 32 hidden units. There are 4 output units, corresponding to the four features.

wall and *undefined*. Currently we use the output with the maximum activation to determine the feature detected.

Data for the nets was obtained by running the robot in the corridors. Initially, occupancy grids were collected by issuing abstract actions to the robot (*go-forward, turn-left, turn-right*) and recording the data. Each local occupancy grid was used to generate 4 training examples, each of which was hand-labeled. It is possible to perform rotations on the occupancy grids and generate many more examples. Since data collection is a relatively easy process, this was not deemed to be necessary. The artificial neural network was trained on this initial data set.

Subsequent training data was autonomously obtained as PAVLOV performed navigation tasks. The final network weights, used in all experiments, was learned from 872 labeled examples, generated by 218 occupancy grids. The data was obtained from a run around the Electrical Engineering department and a run in which the robot was sent from node 1 to node 9 and back. The numerous tasks, performed by PAVLOV, have generated over four thousand of occupancy grids. Generation of more than sixteen thousand labeled example from this data would require a large time commitment. To avoid this, we take the uniformity of the environment into account. Doors and most walls have the same textures, anywhere in the environment. Random visual inspection of the occupancy data indicate that the 218 occupancy grids, used in generating the training data, is an encompassing representation of all possible situations.

Simmons [52], Cassandra [22] and Schiele [53] use hard-coded feature detectors. One advantage to using neural nets to compute high level features is that the robot can be easily trained to work in different environments, where walls have a different texture. This is facilitated by a short training time and the fact that data collection and labeling is a relatively quick process. It is also possible to add new high level features. In this case an easier approach would be to have a neural net for each

feature, and then train a net to predict the new feature. We choose to use one net with four outputs because this requires only one forward-propagation, as opposed to four for individual nets.

CHAPTER 4

RESULTS

In this chapter we provide a discussion of the results obtained from experiments, divided into two sections. A discussion of the neural net used in feature detection is provided. To demonstrate the effectiveness of this approach to deal with specularities, example occupancy grids are provided. This is followed by a discussion of the POSMDP approach. This includes a presentation of results which demonstrate the ability of POSMDPs to estimate location, and model and learn transition times.

4.1. Feature Detection

Through experimentations, we discovered that the sonars were prone to specular reflections in a majority of the environment. We attempted to create a hard-coded feature detector, but found that an artificial neural network could be trained to produce more accurate and consistent results. Not only was it easy to implement and train, but it is also possible to port it to other environments and add new features.

The ANN used in feature detection was trained using Fahlman's Quickprop program [54]. This variation of the backpropagation algorithm has been highly optimized to quickly learn the network weights. The sum of the weighted inputs are used with an asymmetrical sigmoid function to produce an activation that is in the range 0 to 1. Initially, the weights between the input layer and the hidden layer and the hidden layer and the output layer are set to random values. The ANN is presented with each input-output pair of the training set. The activation from the input is

propagated through the network to the output. The ANNs response is compared to the desired output. If the predicted response does not match the actual response, an error for each output unit is computed and stored. The process is repeated for every member of the training set, and is called a single *training epoch*. At the end of a training epoch, the errors for all training examples are used, in a batch update, to adjust the weights of the interconnections. The goal is to produce outputs that can provide an overall better match to the desired outputs in the next training epoch.

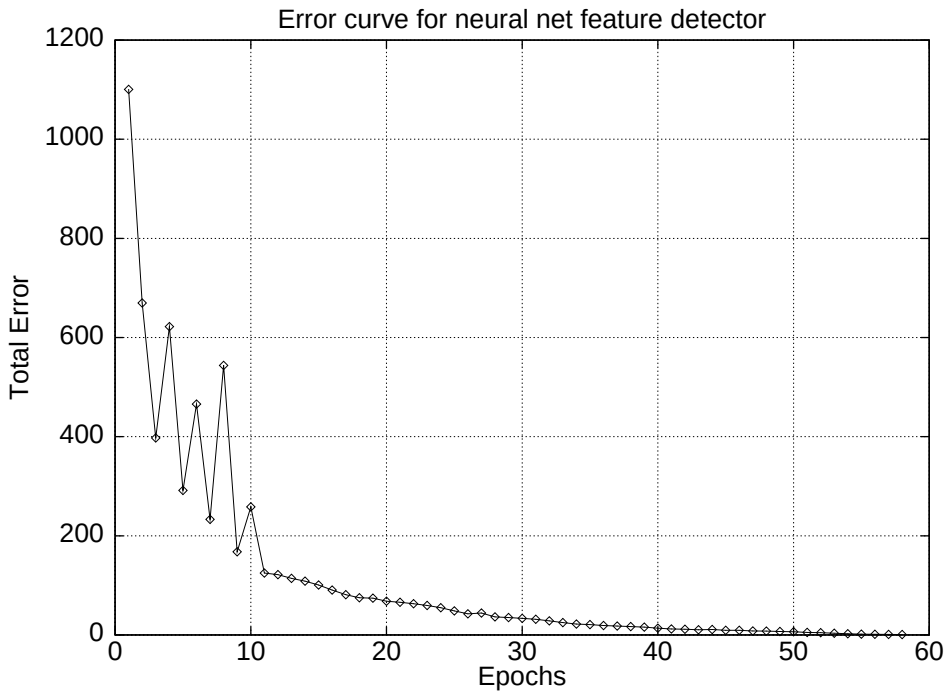


Figure 17. Learning Curve for ANN. The ANN for feature detection trained on 872 hand labeled examples using Quickprop.

Sample local occupancy grids were collected by running the robot through the hallways. Each local occupancy grid was then used to produce 4 training patterns. The ANN was trained on 872 hand labeled examples. Since all sensors predict the same set of features, it was only necessary to learn one set of weights. Figure 17

shows the learning curve for the ANN, using batch update. Starting off with a set of random weights, the total error over all training examples converged to an acceptable range (< 1) within about 60 training epochs.

A separate set of data, with 380 labeled patterns, was used to test the net. This would be the approximately the number of examples encountered by the robot, as it navigated the loop in the Electrical Engineering department (nodes 3-4-5-6 in Figure 3). Feature prediction is accomplished by using the output with the maximum value. Out of the 380 test examples, the ANN correctly predicts features for 322, leading to an accuracy of 85%.

Figure 18 illustrates the variation in observation data, using examples of the robo-centric local occupancy grids. In these occupancy grids, free space is represented by white, while black represents occupied space. Gray areas indicate that occupancy information is unknown. The figures are labeled with the virtual sensors and corresponding features, as predicted by the ANN.

Specular reflections occur when a sonar pulse hits a smooth, flat surface angled obliquely to the transducer. The possibility exists that the sonar pulse will reflect away from the sensor, and undergo multiple reflections before it is received by the sensor. As a result, the sensor registers a range that is substantially larger than the actual range. In the occupancy grids, this results in a physically occupied region having a low occupancy probability. In Figure 18(a) where the specularities are relatively insignificant, the ANN does an accurate job of predicting the features. Effects of the specularities are noticeable in Figure 18(b) and Figure 18(c). In Figure 18(b) the ANN is able to predict a wall on the left, although it has been almost totally obscured by specular reflections. The occupancy grid in Figure 18(c) shows some bleed-through of the sonars. In both examples, the ANN correctly predicts the high level features. Figure 18(e) and Figure 18(f) are examples of occupancy grids where the effects of the specularities become very noticeable. In these examples specularities

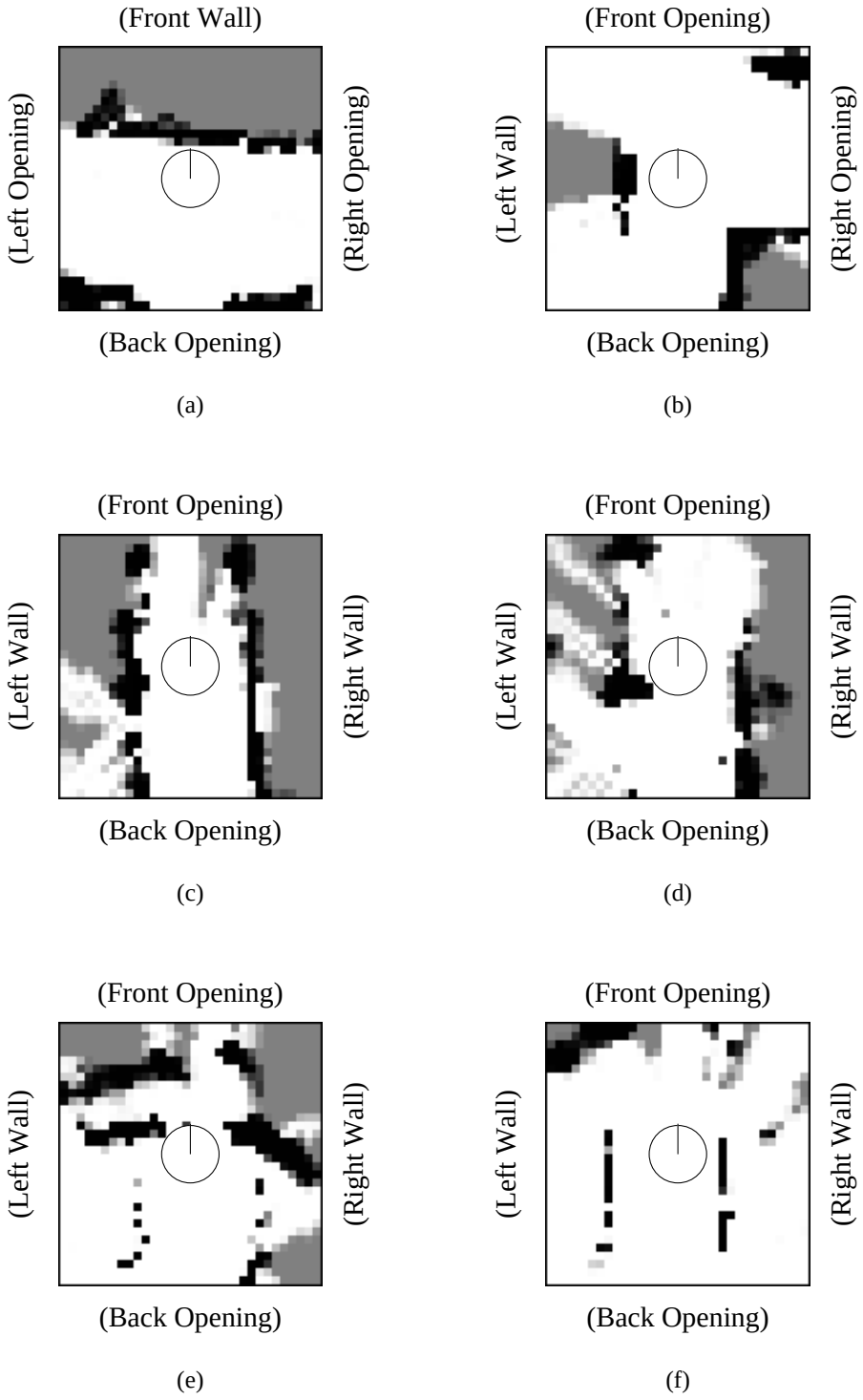


Figure 18. Example Occupancy Grids.

dominate, almost totally wiping out any useful information, yet the ANN is still able to correctly predict features.

From the presented examples, it is apparent that the ANN can robustly predict features in a highly specular environment. Testing the ANN on an unseen set of labeled data reveals that it is able to correctly predict 85% of the features. In addition, although examples have not been presented, the ANN is able to accurately predict features even when the robot is not approximately oriented along one of the allowed compass directions.

4.2. Navigation Results

PAVLOV is able to successfully navigate a dynamically changing environment while traveling to goal locations. In the process of navigation, PAVLOV is able to avoid static and moving objects. By maintaining a distribution over states, PAVLOV is able to compensate for the different sources of uncertainty and can relocalize itself, in the event that it gets lost. We also demonstrate that PAVLOV can learn to avoid heavily populated corridors.

Table 4. Summary of Sample Runs on PAVLOV

Task	Trials	Avg. Time	Std. Dev	Total Distance
1-5-1	18	15.09 min.	0.97 min.	1655 m
1-16-15-1	5	11.61 min.	1.56 min.	485 m
1-7-1	5	15.15 min.	0.86 min.	644 m

Table 4 summarizes some of the experimental results, in terms of the number of tasks completed, total distance traveled, and the average time taken to do each task. A task corresponds to PAVLOV traveling to specified goal nodes. For example,

in Table 4, task 1-7-1 indicates that PAVLOV was instructed to go from node 1 to node 7, and then back to node 1. All tasks were performed on the third floor of the Engineering building at the University of South Florida. In addition to randomly placed static obstacles, PAVLOV also had to deal with enthralled and inquisitive onlookers. In all cases, PAVLOV is able to successfully complete the specified task. The table does not show all the runs that have been performed on PAVLOV, which exceed many kilometers over a period of several weeks.

4.2.1. Odometric Uncertainty

One of the attractions of using the POSMDP approach is the fact that we can combine the advantages of using a geometric and topological based navigation system. While odometry is very accurate over small distances, errors accumulates and becomes very noticeable with the passage of operating time. It is not possible to rely solely on observations, since numerous locations in the environment can have the same set of features, and sensors can occasionally be unreliable. The POSMDP is able to take advantage of using both odometry and sensors in navigation. While performing a task, PAVLOV might overshoot a node at which it was supposed to turn. By maintaining a distribution over states, observations eventually localize the robot. Once PAVLOV realizes that it has overshot the node, it can follow the pre-computed policy to retrace its steps.

Figure 19 is an odometric plot of three successive runs on PAVLOV. Starting from node 1, the specified goals are 4-5-6-1 (see Figure 3). The loop 3-4-5-6 corresponds to the Electrical Engineering department, where corridors are at most 2 meters wide. The severe odometric error is immediately apparent. From the figure, and taking the diameter of the robot into account (0.275 meters), we should interpret the corridors to more than 3 meters wide. Each run places intersections and corridors

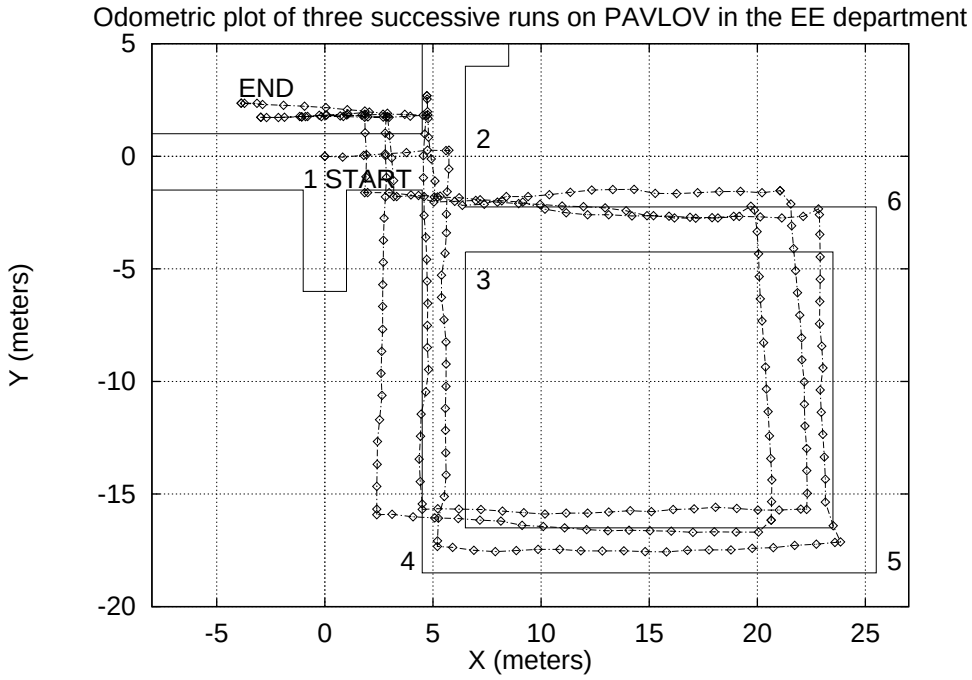


Figure 19. Odometric Trace in the Electrical Engineering Department. This shows the robot starting at node 1, doing the loop (3-4-5-6) and returning to node 1. The robot repeated this task three times, and succeeded despite significant odometric errors

at vastly different absolute (X,Y) coordinates. This is definitely not the case, since the members of the reactive behavior layer force the robot to stay approximately equidistant between walls. Yet the robot successfully completes this task three times. This is clearly a demonstration of the power and robustness of the probabilistic approach to compensate for navigation uncertainty.

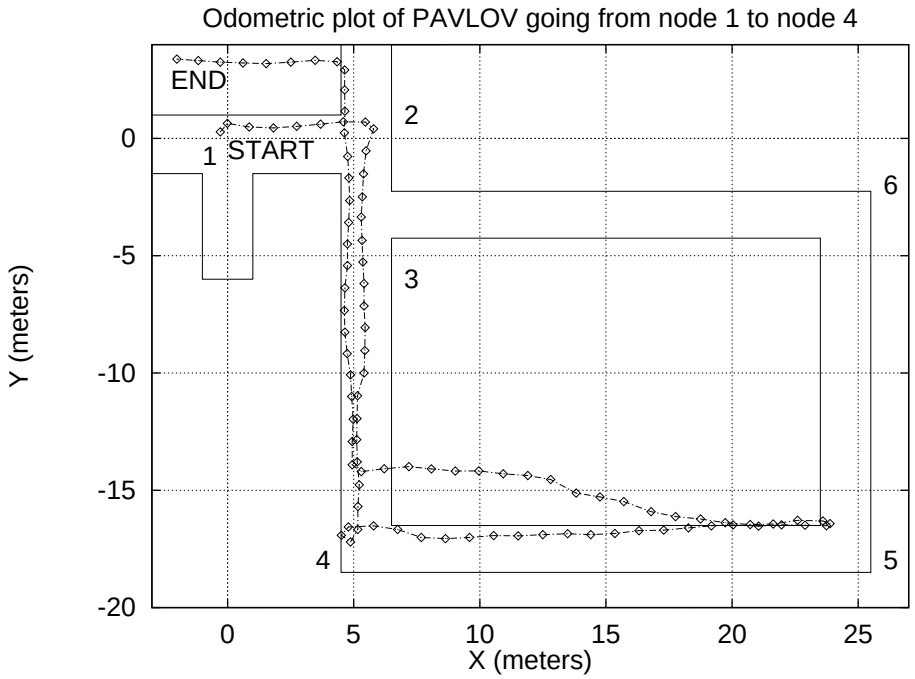


Figure 20. Odometric Trace Demonstrating Localization. PAVLOV was instructed to go to node 4 and return to node 1. After successfully reaching node 4, PAVLOV becomes lost when attempting to head back to node 1. PAVLOV assumes the East-West corridor connecting node 4 to node 5 is the North-South corridor connecting node 3 to node 4. However, upon reaching node 5, PAVLOV is able to relocalize itself.

In addition to modeling uncertainty, the POSMDP approach is capable of relocalizing the robot, in the event that it gets lost. In Figure 20, PAVLOV was instructed to go from node 1 to node 4 and back. PAVLOV successfully reaches node 4, but

becomes lost when attempting to return to node 1. At node 4, PAVLOV turns and believes itself to be oriented North, when it is actually facing West. Since PAVLOV cannot physically distinguish between two different corridors, it moves forward, maintaining the belief that it is traveling towards node 3 from node 4 in the connecting North-South corridor. In actuality, it is traveling from node 4 to node 5 in the connecting East-West corridor. In getting to node 1 from node 4, PAVLOV travels North. Ideally, it should first encounter the observation at node 3 as

$$(front\ opening)\ (left\ wall)\ (back\ opening)\ (right\ opening)$$

and then at node 2 as

$$(front\ opening)\ (left\ opening)\ (back\ opening)\ (right\ wall)$$

Since PAVLOV is traveling in the corridor connecting node 4 and 5, it does not encounter either of the observations. Instead, the observation

$$(front\ wall)\ (left\ opening)\ (back\ opening)\ (right\ wall)$$

is received when it reaches the end of the corridor. By maintaining a distribution over states, PAVLOV is able to determine its true location to be at node 5. The precomputed policy is followed in reaching node 1. This demonstrates the relocalization ability of the POSMDP approach, even when the robot is traveling in an orthogonally different direction.

4.2.2. Temporal Modeling

By using the POSMDP approach, not only are we able to account for navigation uncertainty, but we can also account for varying actions times in planning a path. The standard POMDP approach is unable to deal with continuous time. The semi-Markov approach allows us to model arbitrary time distributions. In addition, we can also learn transition times that could result from PAVLOV traversing a crowded corridor.

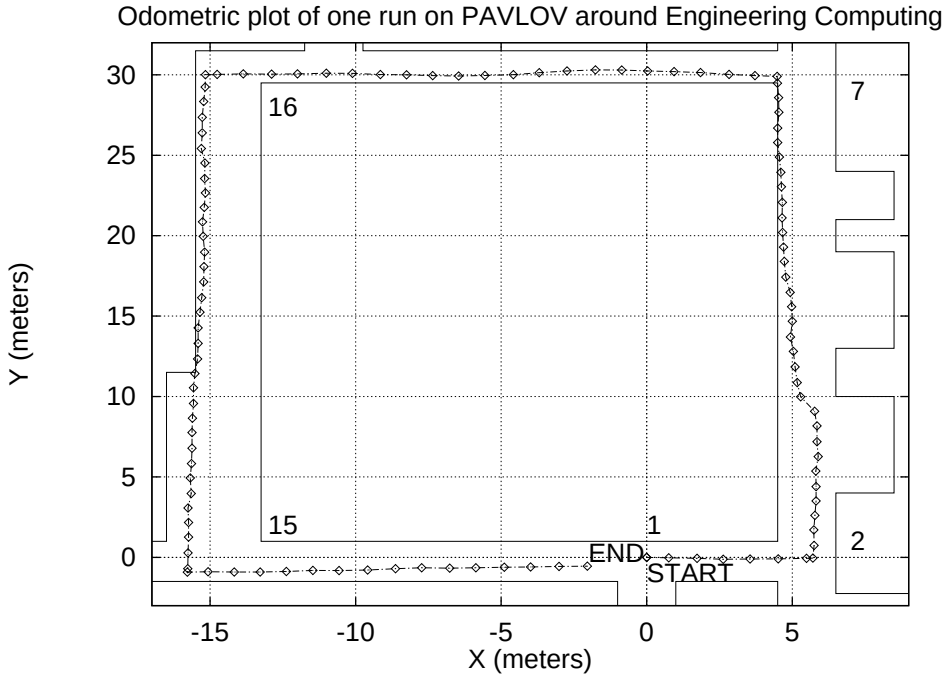


Figure 21. Odometric Trace around Engineering Computing. This is an odometric trace of one run on PAVLOV around Engineering Computing. In this plot, we are not taking advantage of the POSMDP approach, and the robot chooses the shortest path in reaching the goals (node 7, 15, 16 and 1). The numbers on the plot correspond to the topological nodes in Figure 3.

To demonstrate the effectiveness of the POSMDP approach to deal with clut-

tered corridors, refer to Figure 21 and Figure 22. Figure 21 is an odometric trace of the path followed by PAVLOV, starting from node 1 and specifying nodes 7, 15 and 1 as goals. PAVLOV chooses the path covering the least distance, visiting, in sequence nodes 2, 7, 8, 9, 15, 16 and 1. This is the path followed if the standard POMDP approach is used.

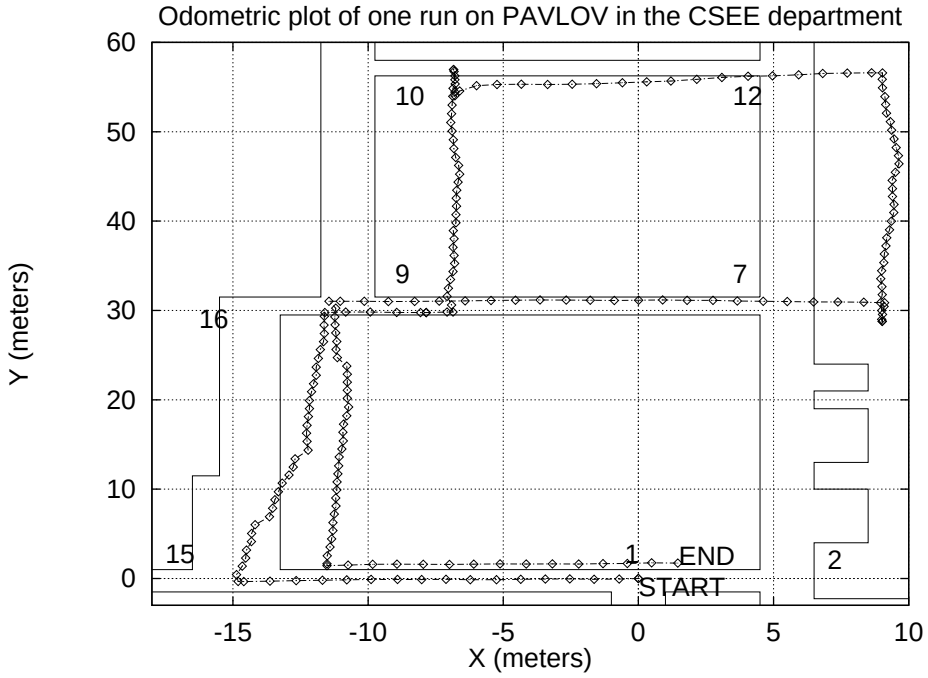


Figure 22. Odometric Trace in the Computer Science Department. This is an odometric trace of one run on PAVLOV around the Engineering Computing and the Computer Science Department. The corridor connecting nodes 2 and 7 has been marked cluttered by modeling actions as having a high variation in transition times. The specified goals were nodes 12, 10, 15 and 1. In getting from node 1 to 12, traversing the corridor connecting node 2 and 7 would cover the shortest distance. However, since we are now using the POSMDP approach for planning, the robot chooses the longer route. The numbers on the plot correspond to the topological nodes in Figure 3.

The corridor connecting node 2 and node 7 is now modeled as "cluttered".

This is facilitated by setting a high variance in the transition times for actions in this corridor. Starting from node 1, the specified goals are nodes 12, 10, 15 and 1. If the transition times were not modeled, PAVLOV would take the shortest path through node 7 to get to node 12. Incorporation of the transition times for actions causes PAVLOV to avoid the "cluttered" corridor. In getting to node 12 from node 1, PAVLOV chooses the longer path (Figure 22).

4.2.3. Learning Transition Times

Rather than arbitrarily modeling the time difference in various parts of the environment, it would be advantageous and realistic for PAVLOV to learn the true variance in the transition times. With this in mind, Figure 23 and Figure 24 show an experiment to test the algorithm for inferring transition times from the Viterbi sequence of most likely states visited.

PAVLOV was asked to go from node 1 to node 5. In the plot in Figure 23 the route taken is via node 4. In this run, a crowded corridor was simulated by forcing the robot to go around many obstacles on the way to node 5 and back. As the robot avoids the obstacles, it slows down, resulting in a longer time to complete actions. At the end of each action, PAVLOV saves the *execution trace*[3]. Transition times are learned off-line by the Viterbi algorithm, which is used to correctly associate transition times with states. The second time around, PAVLOV utilizes the learned transition times and replans a route to avoid the crowded corridor, by going to node 5 via node 6 (Figure 24).

In Figure 25 and Figure 26, Markov states that lie at topological nodes are colored black, while non-topological Markov states are colored white. The transition times, in seconds, are atop the arcs connecting states. The table in each figure

[3]An execution trace is a record of the probability distribution, the observation, the action and the time required to complete the action

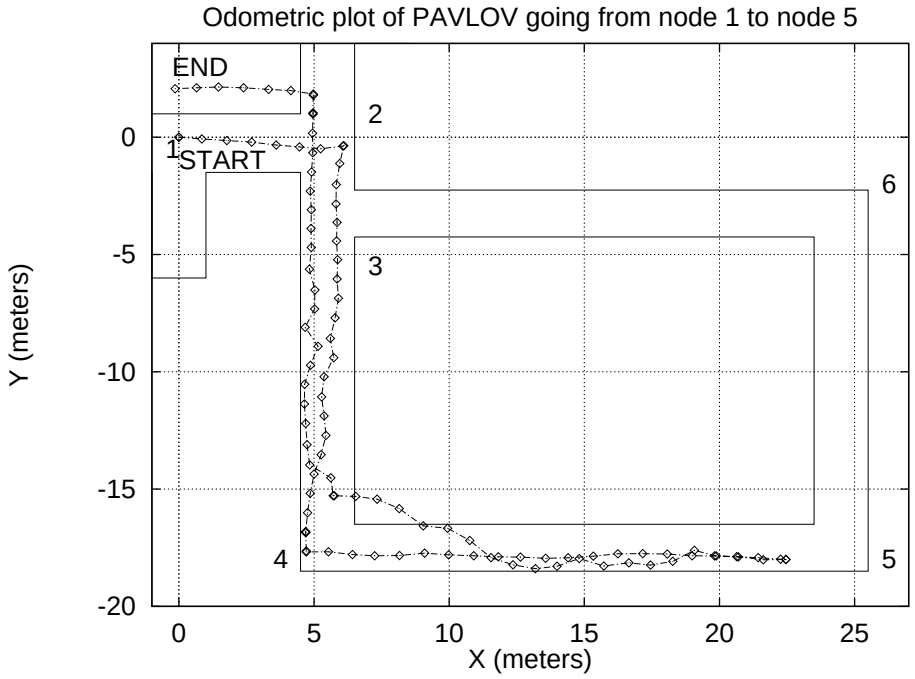


Figure 23. Odometric Trace before Learning Transition Times. PAVLOV is instructed to go to node 5. The default path is through the corridor connecting node 3 and 4. This corridor has a number of obstacles, and hence there is a high variance in the transition times for actions in the corridor. The numbers on the plot correspond to the topological nodes in Figure 3

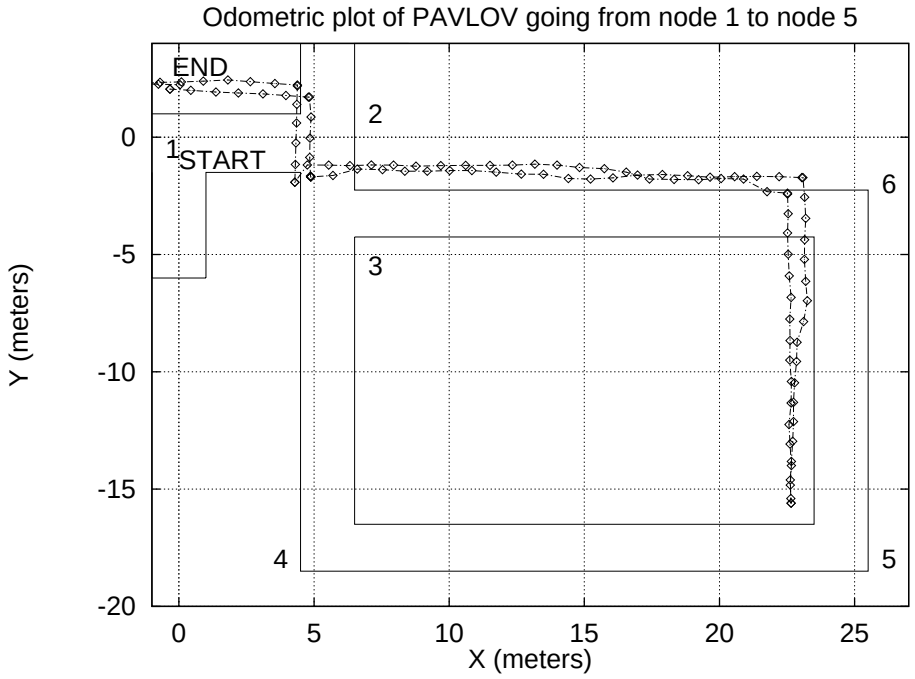
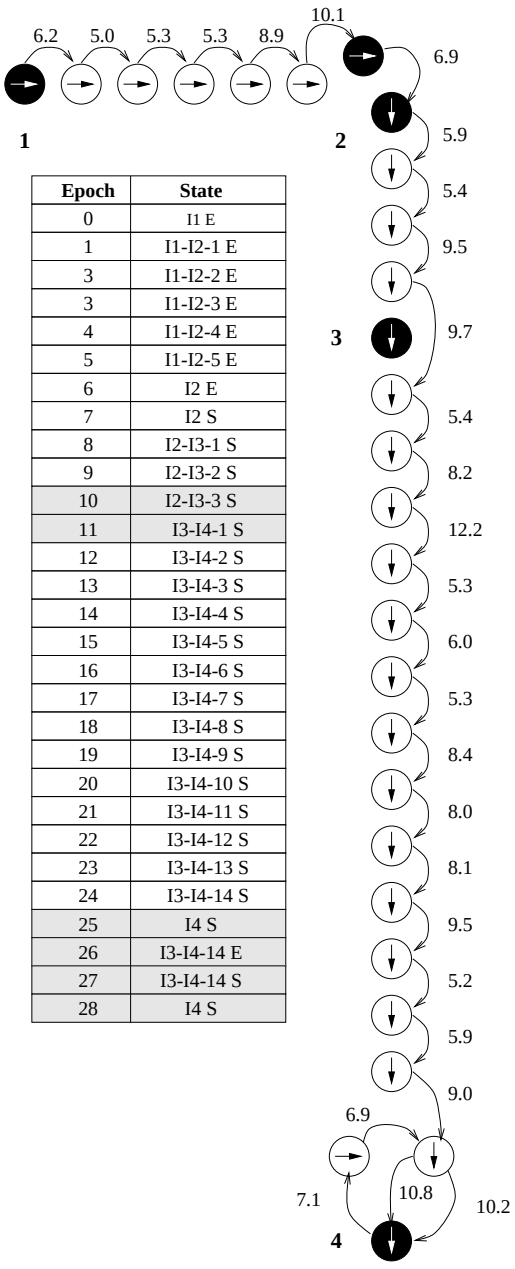


Figure 24. Odometric Trace after Learning Transition Times. Using the Viterbi algorithm to extract the most likely sequence of states and correctly associate transition times, PAVLOV now decides to avoid the corridor connecting node 3 and 4. In getting to node 5, it now chooses to go through node 6. The numbers on the plot correspond to the topological nodes in Figure 3

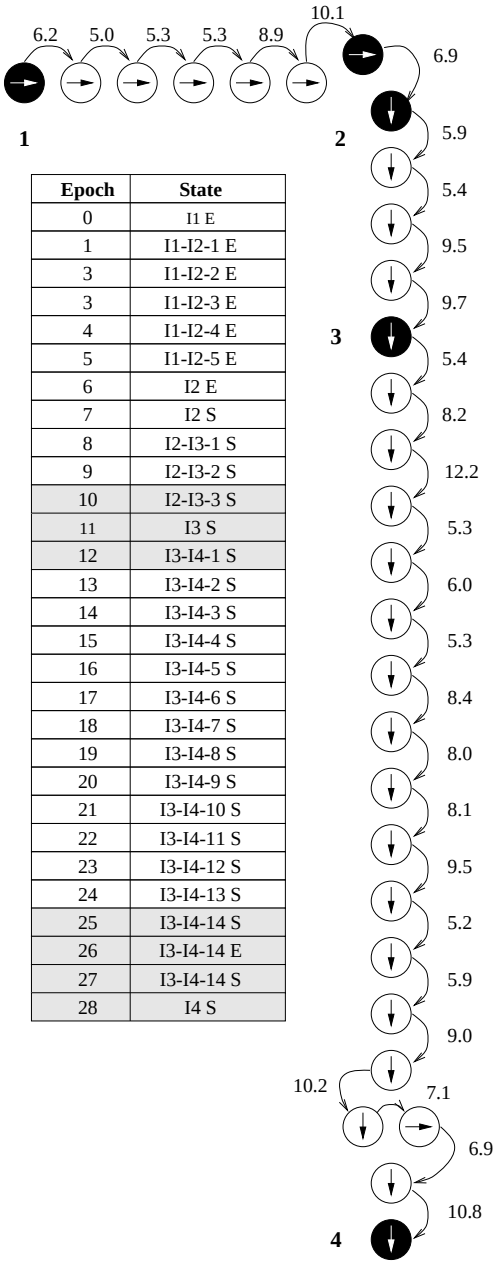
represents the sequence of encountered states. A Markov node is identified either by the topological node at which it lies, or by the pair of topological nodes in between which it lies. Markov states are identified a Markov node label and a direction. For example, PAVLOV starts off at node 1, facing east. This corresponds to the Markov node `I1` and Markov state `I1 E`. After the first decision epoch, PAVLOV is in the first East facing Markov state in the corridor connecting node 1 and 2, which is labeled `I1-I2-1 E`. Labels can be interpreted in a similar fashion. For example, `I3-I4-2 S` corresponds to the second south facing Markov state in the corridor connecting node 3 and 4. Corridors are discretized into one set of Markov nodes. For example, the corridor connecting node 2 and 3 is approximately 3 meters long, resulting in three Markov nodes in between. The Markov node `I2-I3-1` corresponds both to the first set of states encountered when heading from node 2 to 3, and the third set of states when heading from node 3 to 2.

Using a most likely state estimation technique with the execution traces does not guarantee a connected sequence of states. Application of such a technique to determine the sequence of states encountered by PAVLOV can result in physically disparate states being introduced. It has been observed that the POSMDP will occasionally predict the robot to be in a state that is a couple of meters away from its true location. This arises partly because of perceptual aliasing. PAVLOV is usually able to localize itself after the next observation has been received. However, this still results in the incorrect state being introduced into the sequence. Figure 25 displays the believed sequence of states, as predicted by the POSMDP, when going from node 1 to 4 in Figure 23. This interpretation indicates that PAVLOV never encountered node 3, and reached node 4 before it actually did. Since this is obviously not the case, assumption of this sequence would cause an incorrect association of transition times between states.



Epoch	State
0	I1 E
1	I1-I2-1 E
3	I1-I2-2 E
3	I1-I2-3 E
4	I1-I2-4 E
5	I1-I2-5 E
6	I2 E
7	I2 S
8	I2-I3-1 S
9	I2-I3-2 S
10	I2-I3-3 S
11	I3-I4-1 S
12	I3-I4-2 S
13	I3-I4-3 S
14	I3-I4-4 S
15	I3-I4-5 S
16	I3-I4-6 S
17	I3-I4-7 S
18	I3-I4-8 S
19	I3-I4-9 S
20	I3-I4-10 S
21	I3-I4-11 S
22	I3-I4-12 S
23	I3-I4-13 S
24	I3-I4-14 S
25	I4 S
26	I3-I4-14 E
27	I3-I4-14 S
28	I4 S

Figure 25. A POSMDP Sequence. This shows the most likely sequence of states, as observed by the POSMDP. A comparison with Figure 26 reveals the disparity. Assuming the sequence of most likely states predicted by the POSMDP would result in an incorrect association of transition times. The table displays the most likely state at each decision epoch



Epoch	State
0	I1 E
1	I1-I2-1 E
3	I1-I2-2 E
3	I1-I2-3 E
4	I1-I2-4 E
5	I1-I2-5 E
6	I2 E
7	I2 S
8	I2-I3-1 S
9	I2-I3-2 S
10	I2-I3-3 S
11	I3 S
12	I3-I4-1 S
13	I3-I4-2 S
14	I3-I4-3 S
15	I3-I4-4 S
16	I3-I4-5 S
17	I3-I4-6 S
18	I3-I4-7 S
19	I3-I4-8 S
20	I3-I4-9 S
21	I3-I4-10 S
22	I3-I4-11 S
23	I3-I4-12 S
24	I3-I4-13 S
25	I3-I4-14 S
26	I3-I4-14 E
27	I3-I4-14 S
28	I4 S

Figure 26. A Viterbi Sequence. The Viterbi algorithm is applied to the sequence of execution traces to determine the most likely path followed by PAVLOV. The transition time for performing an action was then associated with two successive nodes. The table displays the most likely state at each decision epoch

To correctly associate transition times with states, it is necessary to determine the most likely sequence of states followed. The Viterbi algorithm (Figure 14) uses the history of execution traces in reconstructing the most likely sequence of states encountered. Figure 26 shows the most likely sequence of states encountered in going from node 1 to 4 in Figure 23, when using an execution trace history. The highlighted decision epochs in the tables in Figures 25 and 26 indicate the inconsistencies in the encountered state sequence. The first inconsistency arises when PAVLOV is traveling through node 3. The POSMDP-determined state sequence indicates that state `I3 S` was never encountered when going from state `I2-I3-3 S` to `I3-I4-1 S`. The resulting transition time will be associated with the transition from `I2-I3-3 S` to `I3-I4-1 S`. Not only would this be incorrect, but the net result is that all future transition times will also be incorrectly associated. The Viterbi algorithm is able to determine that PAVLOV traveled from `I2-I3-3 S` to `I3 S`, thereby performing a correct transition time to state association for this and subsequent states. Another inconsistency arises when PAVLOV believes itself to have reached `I4 S`, when it is actually in `I3-I4-14 S`. By using the execution trace, the Viterbi algorithm is also able to handle this second inconsistency.

CHAPTER 5

CONCLUSION

5.1. Contributions

This thesis has developed a novel POSMDP-based navigation system which was implemented on the real robot PAVLOV. Incorporation of time into the standard POMDP-based architecture allows a more realistic approach in planning routes to avoid crowded corridors. By determining actual transition times, PAVLOV is able to learn to avoid crowded corridors. By learning the transition times in different corridors, this approach can better determine a minimum time path. PAVLOV averages twenty centimeters per second, and control is reactive during navigation. In the presence of unreliable actuators and sensors, as well as uncertainty in the metric information, this architecture has performed successfully through experiments that required the robot to navigate a total of several kilometers. Errors resulting from odometric uncertainty, incorrect sensor interpretation and perceptual aliasing are minimized, while combining the advantages of geometric and topological map-based approaches. Maintaining a probability distribution over states prevents PAVLOV from becoming irrevocably lost. This probabilistic navigation technique allows PAVLOV to operate unattended in a complex and uncertain environment. The navigation system will serve as a basis of any future work, since to perform any task a robot must first be able navigate reliably.

In addition we have shown that it is possible to utilize ANNs for reliable feature detection from local occupancy grids in a highly specular environment. Results from a test data set show that the ANN is able to correctly classify 85% of the labeled features. It is easy to obtain and label examples, and training time is short. This makes addition of other features and portability of the ANN to different environments a realistic possibility.

5.2. Future Work

The main focus of this thesis is to show the representational power of the SMDP model in modeling non-constant action times. However, there in one inherent difficulty in using the Viterbi algorithm to learn transition times. The Viterbi sequence, extracted from the execution trace, specifies only the most likely set of states, which could have been occupied by the robot. There is much more to be investigated, because transition can depend on many factors, such as time of day, trash in the corridor, dynamic conditions and other random events.

More accurate observations will prevent PAVLOV from getting lost. Therefore, research into methods of making feature detection more robust is necessary. At the present time, specular reflections are the main source of error in feature detection from local occupancy grids. The advantage of using the occupancy grids is that they can be quickly updated, and are a useful means for sensing the environment in a reactive system. Various researchers [50, 35, 55] have described methods for dealing with the specularities, that could be adapted to our use. Thrun [44] describes a neural net to predict occupancy values that forces dependence between adjacent sonars. However, there is a limit to the usefulness of occupancy grids in feature detection. Occupancy grids cannot be used to determine fine grained features. Specularities often prevent

detection of large features, such as doors. Other means of observing the environment must be considered. One obvious choice is vision. Stemm [56] uses ANNs with image segmentation to recognize objects and recycling bins. In our environment, where doors are a distinctive color, such an approach would be practical. Sensor fusion is supported by the POMDP framework. Therefore any future work should attempt to maximize the information collected from both occupancy grids and vision. Since image processing can often be computationally expensive, PAVLOV can resort to using vision when other means of sensing do not provide enough information.

Currently we are using a static set of observation and transition probabilities. It would advantageous to generate a POSMDP that best fits the empirical data. Chrisman [57] uses a variation of Q-learning to approximate the optimal POMDP, while Koenig [21] uses the Baum-Welch algorithm to update the transition and observation probabilities, and cause the improved POMDP to one that locally best fits the execution trace.

LIST OF REFERENCES

[1] I. J. Cox, "Blanche: An experiment in guidance and navigation of an autonomous mobile robot," *IEEE Transactions Robotics and Automations*, vol. 7, no. 3, pp. 193–204, 1991.

[2] J. Borenstein, H. Everett, and L. Feng, *Navigating Mobile Robots*. Wellesley, Massachusetts: A. K. Peters, Ltd., 1996.

[3] I. A. Getting, "The global positioning system," *IEE Spectrum*, pp. 46–57, December 1993.

[4] L. J. Duchnowski, "Vehicle and driver analysis with real-time precision location techniques," *Sensors*, pp. 40–47, May 1992.

[5] K. Jenkin, E. Milios, P. Jasiobedzki, N. Bains, and K. Tran, "Global navigation for ARK," in *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, (Yokohama, Japan), pp. 2165–2171, July 1993.

[6] H. R. Everett, D. W. Gage, G. A. Gilbreth, R. T. Laird, and R. Smurlo, "Real-world issues in warehouse navigation," in *Proceedings SPIE Mobile Robots IX*, pp. 2–4, November 1994.

[7] C. DeCorte, "Robots train for security surveillance," *Access Control*, pp. 37–38, June 1994.

[8] L. Gould, "Is off-wire guidance alive or dead?," *Managing Automation*, pp. 38–40, May 1990.

[9] R. H. Byrne, P. R. Klarer, and J. Pletta, "Techniques for autonomous navigation," Tech. Rep. SAND920457, Sandia National Laboratories, Albuquerque, NM, March 1992.

[10] L. Kleeman, "Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning," in *Proceedings of IEEE International Conference on Robotics and Automations*, (Nice, France), pp. 2582–2587, May 1992.

[11] L. Kleeman and R. Russell, "Thermal path following robot vehicle: Sensor design and motion control," in *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, (Yokohama, Japan), pp. 1319–1323, July 1993.

[12] R. A. Russell, "Mobile robot guidance using a short-lived heat trail," *Robotica*, vol. 11, no. 5, pp. 427–431, 1993.

[13] R. Deveza, D. Thiel, R. A. Russell, and A. Mackay-Sim, "Odor sensing for robot guidance," *The International Journal of Robotics Research*, vol. 13, pp. 232–239, June 1994.

[14] R. A. Russell, "A practical demonstration of the application of olfactory sensing to robot navigation," in *Proceedings of the International Advanced Robotics Programme (IARP)*, (Sydney, Australia), pp. 35–43, May 1995.

[15] R. A. Russell, "Laying and sensing odor markings as a strategy for assisting mobile robot navigation tasks," *IEEE Robotics and Automation Magazine*, vol. 2, pp. 3–9, September 1995.

[16] B. Kuipers and Y. Byun, "A qualitative approach to robot exploration and map-learning," in *Proceedings of the Spatial Resoning and Multi-Sensor Fusion Workshop*, (Chicago, IL), 1987.

[17] S. Thrun and A. Bücken, "Learning maps for indoor mobile robot navigation," Tech. Rep. CMU-CS-96-121, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996.

[18] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Frohlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt, "Map learning and high-speed navigation in RHINO," in *AI-based Mobile Robots: Case-studies of Successful Robot Systems* (D. Kortenkamp, P. Bonasso, and M. R., eds.), MIT Press, 1997.

[19] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 116–121, 1985.

[20] H. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Magazine*, vol. 9, no. 2, pp. 61–74, 1988.

[21] S. Koenig and R. Simmons, "Xavier: A robot navigation architecture based on partially observable Markov decision process models," in *AI-based Mobile Robots: Case-studies of Successful Robot Systems* (D. Kortenkamp, P. Bonasso, and R. Murphy, eds.), MIT Press, 1997.

[22] T. Cassandra, L. Kaelbling, and J. Kurien, "Acting under uncertainty: Discrete bayesian models for mobile-robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 963–972, 1996.

[23] D. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, Massachusetts: Athena Scientific, 1995.

[24] M. Puterman, *Markov Decision Processes: Discrete Dynamic Stochastic Programming*. New York, USA: John Wiley, 1994.

[25] C. Watkins, *Learning from Delayed Rewards.* PhD thesis, Kings College, Cambridge, England, May 1989.

[26] S. Ross, *Stochastic Process.* New York, NY: Wiley, 1996.

[27] A. Cassandra, L. Kaelbling, and M. Littman, "Acting optimally in partially observable stochastic domains," in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, (Seattle, WA), 1994.

[28] W. Lovejoy, "A survey of algorithmic methods for partially observable Markov decision processes," *Annals of Operations Research*, vol. 28, pp. 47–66, 1991.

[29] C. Papadimitriou and J. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[30] A. Cassandra, "Optimal policies for partially observable Markov decision processes," Tech. Rep. CS-94-14, Brown University, Department of Computer Science, Providence, RI, 1994.

[31] M. Littman, "The Witness algorithm: Solving partially observable Markov decision processes," Tech. Rep. CS-94-04, Brown University, Department of Computer Science, Providence, RI, 1994.

[32] M. Littman, T. Cassandra, and L. Kaelbling, "Learning policies for partially-observable environments: scaling up," in *Proceedings of the 12th International Conference on Machine Learning*, pp. 362–370, Morgan Kaufmann, 1995.

[33] E. J. Sondik, "The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 2, pp. 282–304, 1978.

[34] K. J. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[35] B. Yamauchi and P. Langley, "Place learning in dynamic real-world environments," in *Proceedings of ROBOLEARN-96: International Workshop for Learning in Autonomous Robots*, (Key West, FL), pp. 123–129, May 1996.

[36] M. McCord-Nelson and W. T. Illingworth, *A Practical Guide to Neural Nets.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1990.

[37] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard, Cambridge, MA, August 1974.

[38] J. McClelland and D. Rumelhart, *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

[39] T. J. Sejnowski and C. R. Rosenberg, "NETtalk: a parallel network that learns to read aloud," in *Neurocomputing: Foundations of Research* (J. Anderson and E. Rosenfield, eds.), MIT Press, 1988.

[40] S. Koenig and R. Simmons, "Passive distance learning for robot navigation," in *Proceedings of the 13th International Conference on Machine Learning*, pp. 266–274, Morgan Kaufmann, 1996.

[41] S. Koenig and R. Simmons, "Unsupervised learning of probabilistic models for robot navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[42] A. Viterbi, "Error bounds for convolutional codes and as asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. IT-13, no. 2, pp. 260–269, 1967.

[43] I. Nourbakhsh, R. Powers, and S. Birchfield, "Dervish: An office-navigating robot," *AI Magazine*, vol. 16, no. 2, pp. 53–60, 1995.

[44] S. Thrun and A. Bücken, "Integrating grid-based and topological maps for mobile robot navigation," *AAAI National Conference on Artificial Intelligence*, 1996.

[45] M. Littman, *Algorithms for Sequential Decision Making.* PhD thesis, Brown University, 1996.

[46] R. Parr and S. Russell, "Approximating optimal policies for partially observable stochastic domains," in *Proceedings of the Fourteenth IJCAI*, pp. 1088–1094, 1995.

[47] D. White, "Dynamic programming, Markov chains, and the method of successive approximation," *Journal of Mathematical Analysis and Applications*, vol. 6, pp. 373–376, 1963.

[48] S. Mahadevan, "Average reward reinforcement learning: Foundations, algorithms, and empirical results," *Machine Learning*, vol. 22, no. 1, pp. 159–95, 1996.

[49] J. Connell, "SSS: A hybrid architecture applied to robot navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2719–2724, 1992.

[50] K. Konolige, "A refined method for occupancy grid interpretation," in *Proceedings of the International Workshop on Uncertainty in Robotics*, (Amsterdam, Netherlands), 1995.

[51] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, pp. 46–57, June 1989.

[52] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," in *Proceedings of the IJCAI*, pp. 1080–1087, 1995.

[53] B. Schiele and J. Crowley, "A comparison of position estimation techniques using occupancy grids," *IEEE Conference on Robotics and Autonomous Systems*, pp. 1628–1634, May 1994.

[54] S. Fahlman, "Faster-learning variations on back-propagation: An empirical study," in *Proceedings of 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1988.

[55] A. Howard and L. Kitchen, "Generating sonar maps in highly specular environments," in *Proceedings of the 4th International Conference on Control, Automation, Robotics and Vision*, December 1996. To appear.

[56] M. Stemm, *Using Artificial Neural Networks and Image Segmentation to Assist in Mobile Robot Navigation.* Bachelors Thesis, School of Computer Science, Carnegie Mellon University, 1994.

[57] L. Chrisman, "Reinforcement learning with perceptual aliasing," in *Proceedings of AAAI*, 1992.