Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models

Sven Koenig and Reid G. Simmons
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213-3890
{skoenig, reids}@cs.cmu.edu

Abstract

Autonomous mobile robots need very reliable navigation capabilities in order to operate unattended for long periods of time. We present a technique for achieving this goal that uses partially observable Markov decision process models (POMDPs) to explicitly model navigation uncertainty, including actuator and sensor uncertainty and approximate knowledge of the environment. This allows the robot to maintain a probability distribution over its current pose. Thus, while the robot rarely knows exactly where it is, it always has some belief as to what its true pose is, and is never completely lost. We present a navigation architecture based on POMDPs that provides a uniform framework with an established theoretical foundation for pose estimation, path planning, robot control during navigation, and learning. Our experiments show that this architecture indeed leads to robust corridor navigation for an actual indoor mobile robot.

1 Introduction

We are interested in autonomous mobile robots that perform delivery tasks in office environments. It is crucial that such robots be able to navigate corridors robustly in order to reliably reach their destinations. While the state of the art in autonomous office navigation is fairly advanced, it is not generally good enough to permit robots to traverse corridors for long periods of time without eventually getting lost.

Our approach to this problem involves using partially observable Markov decision process models (POMDPs). POMDPs explicitly account for various forms of uncertainty: uncertainty in actuation, sensing and sensor data interpretation, uncertainty in the initial *pose* (position and orientation) of the robot, and uncertainty about the environment, such as corridor distances and blockages (including closed doors). Instead of maintaining a single estimate of its current pose, the robot uses a POMDP to maintain a probability distribution over its current pose at all times. Thus, while it rarely knows exactly where it is, the robot always has some belief as to what its true pose is, and thus is never completely lost. To update the belief in its current pose, the robot can utilize all available sensor information, including landmarks sensed and distance traveled.

We use the POMDP-based navigation architecture on a daily basis on our robot, Xavier. Our experiments show that the architecture leads to robust long-term autonomous navigation in office environments (with corridors, foyers, and rooms) for an actual indoor mobile robot, significantly outperforming the landmark-based navigation technique that we used previously [27].

The POMDP-based navigation architecture uses a compiler that automatically produces POMDPs from topological maps, actuator and sensor models, and uncertain knowledge of the environment. The resulting POMDPs seamlessly integrate topological and metric information, and enable the robots to utilize as much, or as little, metric information as they have available. Pose estimation easily deals with

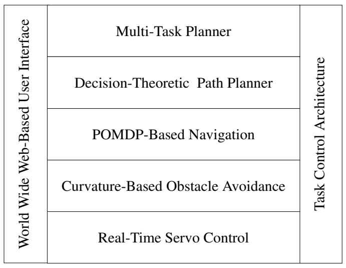


Figure 1: A Layered Architecture for Office Delivery Robots

metric map uncertainty, and deteriorates gracefully with the quality of the models. Finally, learning algorithms can be used to improve the models, while the robots are carrying out their delivery tasks.

We show how to use POMDPs to build a whole architecture for mobile robot navigation that provides a uniform framework with an established theoretical foundation for pose estimation, path planning (including planning when, where, and what to sense), control during navigation, and learning. In addition, both the POMDP and the generated information (such as the pose information and the plans) can be utilized by higher-level planning modules, such as task planners.

The POMDP-based navigation architecture is one layer of our office delivery system (Figure 1) [30]. Besides the navigation layer described here, the layers of the system include 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 [29], a path planning layer that reasons about uncertainty to choose paths that have high expected utility [13], and a multiple-task planning layer that uses PRODIGY, a symbolic, non-linear planner, to integrate and schedule delivery requests that arrive asynchronously [9]. The layers, which are implemented as a number of distributed, concurrent processes operating on several processors, are integrated using the Task Control Architecture. The Task Control Architecture provides facilities for interprocess communication, task decomposition and sequencing, execution monitoring and exception handling, and resource management [28]. Finally, interaction with the robot is via the World Wide Web, which provides pages for both commanding the robot and monitoring its progress.

In the following, Section 2 contrasts our POMDP-based navigation architecture with more traditional approaches to robot navigation. Section 3 discusses POMDPs and algorithms for pose estimation, planning, and learning in the abstract, and Section 4 applies the models and algorithms to our navigation problem. Finally, Section 5 presents experiments that we performed both with our robot simulator and the actual robot.

2 Traditional Approaches to Robot Navigation

Two common approaches to robot navigation are metric-based and landmark-based navigation:

Metric-based navigation relies on metric maps of the environment, resulting in navigation plans such as: move forward ten meters, turn right ninety degrees, move forward another ten meters, and stop. Metric-based navigation methods are able to take advantage of information about the motion of the robot, which we call *motion reports*, such as the translation and rotation derived from the wheel encoders. They are, however, vulnerable to inaccuracies in both the map making and dead-reckoning abilities of the robot. Such approaches are often used where the robot has good absolute position estimates, such as for outdoor robots using GPS.

Landmark-based navigation, on the other hand, relies on topological maps whose nodes correspond to *landmarks* (locally distinctive places), such as corridor junctions or doors. Map edges indicate how the landmarks connect and how the robot should navigate between them. A typical landmark-based navigation plan might be to move forward to the second corridor on the right, turn into that corridor, move to its end, and stop. Landmark-based approaches are attractive because they are able to take advantage of information about the sensed features of the environment, which we term *sensor reports*, such as data from the sonar sensors that indicate whether the robot is in a corridor junction. Thus they do not depend on geometric accuracy. They suffer, however, from problems of unreliable sensors occasionally not detecting landmarks and problems of *sensor aliasing* (sensors not being able to distinguish between similar landmarks, such as different doorways of the same size). This can lead to both inefficiencies and mistakes. For example, it takes the robot longer to notice that it overshot its destination when it does not use metric information, and it might even turn into the wrong one of two adjacent corridors.

To maximize reliability in navigation, it makes sense to utilize all information that is available to the robot (that is, both motion and sensor reports). While some landmark-based approaches use motion reports, mostly to resolve topological ambiguities, and some metric-based approaches use sensor reports to continuously realign the robot with the map [17, 21], the two sources of information are treated differently. We want an approach that seamlessly integrates both sources of information, and is amenable to adding new sources such as a-priori information about which doorways are likely to be open or closed.

Another problem with the metric- and landmark-based approaches is that they typically represent only a single pose that is believed to be the current pose of the robot. If this pose proves to be incorrect, the robot is lost and has to re-localize itself, an expensive operation. This can be avoided by representing all possible poses of the robot. Since some poses will be more likely than others, this suggests explicitly maintaining a probability distribution over possible poses. Then, Bayes rule can be used to update this pose distribution after each motion and sensor report. To achieve this, we have to explicitly represent the various forms of uncertainty present in the navigation task:

- **Actuator Uncertainty**
  For example, motion commands are not always carried out perfectly due to wheel slippage and mechanical tolerances, resulting in *dead-reckoning error.*
- **Sensor Uncertainty**
  Unreliable sensors produce *false positives* (features that are not present) or *false negatives.* For example, sonar can bounce around several surfaces before returning to the robot and therefore does not necessarily give correct information on the distance to the closest surface.
- **Uncertainty in the Interpretation of the Sensor Data**
  For example, sonar sensor data often do not allow the robot to clearly distinguish between walls, closed doors, and lines of people blocking a corridor junction.
- **Map Uncertainty**
  For example, the lengths of corridors might not be known exactly.
- **Uncertainty about the Initial Pose of the Robot**
- **Uncertainty about the Dynamic State of the Environment**
  For example, blockages can change over time as people close and open doors and block and unblock corridors.

Previously reported approaches that maintain pose distributions often use either Kalman filters [16, 32] or temporal Bayesian networks [5]. Both approaches can utilize motion and sensor reports to update the pose distribution.

Kalman filters model only restricted pose distributions in continuous pose space. In the simplest case, these are Gaussians. While Gaussians are efficient to encode and update, they are not ideally suited for office navigation. In particular, due to sensor aliasing, one often wants to encode the belief that the robot might be in one of a number of non-contiguous (but similar looking) locations, such as at either of two adjacent doorways. In this case, since Gaussians are unimodal, the Kalman filter would estimate that the most likely location is right in between the two doorways – precisely the wrong result. Although Kalman filters can be used to represent more complex pose distributions than Gaussians (at the cost of an increased complexity), they cannot be used to model arbitrary pose distributions.

Temporal Bayesian networks unwind time. With such methods, the size of the models grows linearly with the amount of temporal lookahead, which limits their use for planning to rather small lookaheads. They usually tessellate the possible poses into discrete states, which allows them to represent arbitrary pose distributions.

To summarize, temporal Bayesian networks are typically able to represent arbitrary pose distributions, but discretize the pose space. Kalman filters, on the other hand, do not discretize the pose space, but cannot represent arbitrary pose distributions. Consequently, there is a tradeoff between the precision and expressiveness of the models. We contend that, for office navigation, the added expressiveness of being able to model arbitrary pose distributions outweighs the decrease in precision from discretization, especially since a coarse-grained representation of uncertainty is often sufficient. When a more fine-grained representation is required, one can use low-level control routines to overcome the discretization problem. For example, we use such control routines to keep the robot centered in corridors and aligned along the main corridor axis. Similarly, we use vision and a neural network to align the robot exactly with doorways when it has reached its destination.

POMDPs compare to the traditional robot navigation approaches as follows: They seamlessly integrate topological information and approximate metric information, and therefore utilize both motion and sensor reports to determine the pose distribution. They are similar to Bayesian networks in that they discretize the possible poses, and thus allow one to represent arbitrary pose distributions, but do not suffer from the limited horizon problem for planning, since their lookahead is unlimited.

There have been several other approaches that use Markov models for robot navigation: Dean et. al. [6] use Markov models, but, different from our approach, assume that the location of the robot is always known precisely. Nourbakhsh et. al. [23] use Markov models that do not assume that the location of the robot is known with certainty, but do not utilize any metric information (the states of the robot are either at a topological node or somewhere in a connecting corridor). Cassandra et. al. [2] build on our work, and consequently use Markov models similar to ours, including modeling distance information, but assume that the distances are known with certainty.

3 An Introduction to POMDPs

This section provides a general, high-level overview of partially observable Markov decision process models and common algorithms that operate on such models. These models and algorithms form the basis of our POMDP-based navigation architecture, which is described in Section 4.

POMDPs consist of a finite set of states S; a finite set of observations O; and an initial state distribution π (a probability distribution over S), where $\pi(s)$ denotes the probability that the initial state of the POMDP process is s. Each state $s \in S$ has a finite set of actions $A(s)$ that can be executed in s. The POMDP further consists of a transition function p (a function from $S \times A$ to probability distributions over S), where $p(s'|s,a)$ denotes the probability ("transition probability") that the system transitions from state s to state s' when action a is executed; an observation function q (a function from S to probability distributions over O), where $q(o|s)$ denotes the probability ("observation probability") of observing o in state s; and an immediate reward function r (a function from $S \times A$ to the real numbers), where $r(s,a)$ denotes the immediate reward resulting from the execution of action a in state s.

A *POMDP process* is a stream of (state, observation, action, immediate reward) quadruples: The process is always in exactly one state and makes state transitions at discrete time steps. Assume that at time t, the POMDP process is in state $s_t \in S$. Initially, $p(s_1 = s) = \pi(s)$. Then, an observation

$o_t \in O$ is generated according to the probabilities $p(o_t = o) = q(o|s_t)$. Next, a decision maker chooses an action a_t from $A(s_t)$ for execution. This results in the decision maker receiving immediate reward $r_t = r(s_t, a_t)$ and the process changing state. The successor state $s_{t+1} \in S$ is selected according to the probabilities $p(s_{t+1} = s) = p(s|s_t, a_t)$. This process repeats forever. Note that the probabilities with which the observation and successor state are generated depend only on s_t and a_t, but not, for example, on how the current state s_t was reached. This is called the *Markov property*.

An *observer* of the POMDP process is someone who knows the specification of the POMDP (as stated above) and observes the actions $a_1 \dots a_{T-1}$ and observations $o_1 \dots o_T$, but not the current states $s_1 \dots s_T$ or immediate rewards $r_1 \dots r_{T-1}$. A *decision maker* is an observer who also determines which actions to execute. Consequently, observers and decision makers usually cannot be sure exactly which state the POMDP process is in. This is the main difference between a POMDP and a completely observable Markov decision process model, where observers and decision makers always know exactly which state the process is in, although they will usually not be able to predict the state that results from an action execution, since actions can have non-deterministic effects.

Properties of POMDPs have been studied extensively in Operations Research. In Artificial Intelligence and Robotics, POMDPs have been applied to speech and handwriting recognition [11] and the interpretation of tele-operation commands [10, 37]. They have also gained popularity in the Artificial Intelligence community as a formal model for planning under uncertainty [3, 12]. Consequently, standard algorithms are available to solve tasks that are typically encountered by observers and decision makers. In the following, we describe some of these algorithms.

3.1 State Estimation: Determining the Current State

Assume that an observer wants to determine the current state of a POMDP process. This corresponds to estimating where the robot currently is. Observers do not have access to this information, but can maintain a belief in the form of a *state distribution* (a probability distribution α over S) since they know which actions have been executed and which observations resulted. We write $\alpha(s)$ to denote the probability that the current state is s. Under the Markov property, α summarizes everything known about the current state. This probability distribution can be computed incrementally using Bayes rule. To begin, the probability of the initial state of the POMDP process is $\alpha(s) = \pi(s)$. Subsequently, if the current state distribution is α_{prior}, the state distribution after the execution of action a is α_{post}:

$$\alpha_{post}(s) = \frac{1}{scale} \sum_{s' \in S | a \in A(s')} [p(s|s', a)\alpha_{prior}(s')], \tag{1}$$

where *scale* is a normalization constant that ensures that $\sum_{s \in S} \alpha_{post}(s) = 1$. This normalization is necessary only if not every action is defined in every state.

Updating the state distribution after an observation is even simpler. If the current state distribution is α_{prior}, then the state distribution after observing o is α_{post}:

$$\alpha_{post}(s) = \frac{1}{scale} q(o|s)\alpha_{prior}(s), \tag{2}$$

where *scale*, again, is a normalization constant that ensures that $\sum_{s \in S} \alpha_{post}(s) = 1$.

3.2 POMDP Planning: Determining which Actions to Execute

Assume that a decision maker wants to select actions so as to maximize the expected total reward received over an infinite planning horizon, which is defined as $E(\sum_{t=1}^{\infty}[\gamma^{t-1}r_t])$, where $\gamma \in (0, 1]$ is the discount factor. This corresponds to navigating the robot to its destination in a way that minimizes its expected travel time. The *discount factor* specifies the relative value of an immediate reward received after t action executions compared to the same reward received one action execution earlier. If $\gamma = 1$, the reward is called undiscounted. To simplify mathematics, we assume here that $\gamma < 1$, because this ensures that the expected total reward is finite, no matter which actions are chosen.

Consider a completely observable Markov decision process model. A fundamental result of Operations Research is that, in this case, there always exists a *policy* (a mapping from states to actions) so that the decision maker maximizes the expected total reward by always executing the action that the policy assigns to the current state of the process. Such an *optimal policy* can be determined by solving the following system of $|S|$ equations for the variables $v(s)$, that is known as Bellman's Equation [1]:

$$v(s) = \max_{a \in A(s)}[r(s,a) + \gamma \sum_{s' \in S}[p(s'|s,a)v(s')]] \qquad \text{for all } s \in S. \tag{3}$$

$v(s)$ is the expected total reward if the process starts in state s and the decision maker acts optimally. The optimal action to execute in state s is $a(s) = \arg\max_{a \in A(s)}[r(s,a) + \gamma \sum_{s' \in S}[p(s'|s,a)v(s')]]$. The system of equations can be solved in polynomial time using dynamic programming methods [20]. A popular dynamic programming method is value iteration [1] (we leave the termination criterion unspecified):

1. Set $v_1(s) := 0$ for all $s \in S$. Set $t := 1$.
2. Set $v_{t+1}(s) := \max_{a \in A(s)}[r(s,a) + \gamma \sum_{s' \in S}[p(s'|s,a)v_t(s')]]$ for all $s \in S$. Set $t := t + 1$.
3. Go to 2.

Then, for all $s \in S$, $v(s) = \lim_{t \to \infty} v_t(s)$.

The POMDP planning problem can be transformed into a planning problem for a completely observable Markov decision process model. First, since the decision maker can never be sure which state the POMDP process is in, the set of executable actions A must be the same for every state s, that is, $A(s) = A$. The completely observable Markov decision process model can then be constructed as follows: The states of the model are the state distributions α ("belief states"), with the initial state being α_o with probability $\sum_{s \in S}[q(o|s)\pi(s)]$ for all $o \in O$, where $\alpha_o(s) = q(o|s)\pi(s)/\sum_{s \in S}[q(o|s)\pi(s)]$ for all $s \in S$. Its actions are the same as the actions of the POMDP. The execution of action a in state α results in an immediate reward of $\sum_{s \in S}[\alpha(s)r(s,a)]$ and the Markov process changing state. There are at most $|O|$ possible successor states, one for each $o \in O$. The successor state α'_o is characterized by:

$$\alpha'_o(s) = \frac{q(o|s)\sum_{s' \in S}[p(s|s',a)\alpha(s')]}{\sum_{s \in S}[q(o|s)\sum_{s' \in S}[p(s|s',a)\alpha(s')]]} \qquad \text{for all } s \in S.$$

The transition probabilities are

$$p(\alpha'_o|\alpha,a) = \sum_{s \in S}[q(o|s)\sum_{s' \in S}[p(s|s',a)\alpha(s')]].$$

Any mapping from states to actions that is optimal for this completely observable Markov decision process model is also optimal for the POMDP (under reasonable assumptions) [33]. This means that for POMDPs, there always exists a *POMDP policy* (a mapping from state distributions to actions) that maximizes the expected total reward. This policy can be pre-computed. During action selection, the decision maker only needs to calculate the current state distribution α (as shown in Section 3.1) and can then look up which action to execute.

Unfortunately, the number of belief states α is infinite. Thus, the completely observable Markov decision process model is infinite, and an optimal policy cannot be found efficiently. In fact, the POMDP planning problem is PSPACE-complete in general [24]. However, there are POMDP planning algorithms that trade off solution quality for speed, but that usually do not provide quality guarantees [19, 25]. The SPOVA-RL algorithm [25], for example, can determine approximate policies for POMDPs with about a hundred states in a reasonable amount of time. Further performance improvements are anticipated, since POMDP planning algorithms are the object of current research [18] and researchers are starting to investigate, for example, how to exploit the restricted topology of some POMDPs.

We describe here three greedy POMDP planning approaches that can find policies for large POMDPs fast, but still yield reasonable robot navigation behavior. These three approaches share the property that they pretend that the POMDP is completely observable and, under this assumption, use Equation (3) to determine an optimal policy. Then, they transform this policy into a POMDP policy. We call this transformation "completing" the policy because the mapping from states to actions is completed to a mapping from state *distributions* to actions. Given the current state distribution α, the approaches greedily complete the policy as follows:

- The **"Most Likely State" Strategy** [23] executes the action that is assigned to the most likely state, that is, the action $a(\arg\max_{s \in S} \alpha(s))$.
- The **"Voting" Strategy** [31] executes the action with the highest probability mass according to α, that is, the action $\arg\max_{a \in A} \sum_{s \in S | a(s)=a} \alpha(s)$.
- The **"Completely Observable after the First Step" Strategy** [4, 35] executes the action $\arg\max_{a \in A} \sum_{s \in S} [\alpha(s)(r(s,a) + \gamma \sum_{s' \in S} [p(s'|s,a)v(s')])]$. This approach allows one, for example, to choose the second best action if all states disagree on the best action but agree on the second best action. The selected action is optimal if the POMDP becomes completely observable after the action execution.

3.3 POMDP Learning: Determining the POMDP from Observations

Assume that an observer wants to determine the POMDP that maximizes the probability of generating the observations for the given actions. This POMDP is the one that best fits the empirical data. In our application, this corresponds to fine-tuning the navigation model from experience, including the map, actuator, and sensor models. While there is no known technique for doing this efficiently, there exist efficient algorithms that approximate the optimal POMDP [4, 22, 34]. The Baum-Welch algorithm [26] is one such algorithm. This iterative expectation-maximization algorithm does not require control of the POMDP process and thus can be used by an observer to learn the POMDP. It overcomes the problem that observers can never be sure about the current state of the POMDP process, because they cannot observe the current state directly and are not allowed to execute actions to reduce their uncertainty. Given an initial POMDP and a training trace (a sequence of actions and observations), the Baum-Welch algorithm generates a POMDP that better fits the trace, in the sense that it increases the probability $p(o_{1\ldots T}|a_{1\ldots T-1})$ with which observations are generated given the actions times the probability that the action sequence $a_{1\ldots T-1}$ can be executed (the Baum-Welch algorithm changes only the transition and observation probabilities, but not the number of states or the possible transitions between them). By repeating this procedure with the same training trace and the improved POMDP, we get a hill-climbing algorithm that eventually converges to a POMDP which locally, but not necessarily globally, best fits the trace.

The Baum-Welch algorithm estimates the improved POMDP in three steps. First Step: A dynamic programming approach ("forward-backward algorithm") is used that applies Bayes rule repeatedly [7].

The forward phase calculates scaling factors $scale_t$ and alpha values $\alpha_t(s) = p(s_t = s|o_{1\dots t}, a_{1\dots t-1})$ for all $s \in S$ and $t = 1 \dots T$. The alpha values are the state distributions calculated in Section 3.1.

A1. Set $scale_1 := \sum_{s\in S}[q(o_1|s)\pi(s)]$.

A2. Set $\alpha_1(s) := q(o_1|s)\pi(s)/scale_1$ for all $s \in S$.

A3. For $t := 1$ to $T - 1$

  (a) Set $scale_{t+1} := \sum_{s\in S}[q(o_{t+1}|s) \sum_{s'\in S|a_t\in A(s')}[p(s|s', a_t)\alpha_t(s')]]$.

  (b) Set $\alpha_{t+1}(s) := (q(o_{t+1}|s) \sum_{s'\in S|a_t\in A(s')}[p(s|s', a_t)\alpha_t(s')])/scale_{t+1}$ for all $s \in S$.

The backward phase of the forward-backward algorithm calculates beta values $\beta_t(s)$ for all $s \in S$ and $t = 1 \dots T$:

A4. Set $\beta_T(s) := 1/scale_T$ for all $s \in S$.

A5. For $t := T - 1$ downto 1

  (a) Set $\beta_t(s) := \begin{cases} \sum_{s'\in S}[p(s'|s, a_t)p(o_{t+1}|s')\beta_{t+1}(s')]/scale_t & \text{for all } s \in S \text{ with } a_t \in A(s) \\ 0 & \text{for all } s \in S \text{ with } a_t \notin A(s) \end{cases}$.

Second Step: The Baum-Welch algorithm calculates the gamma values $\gamma_t(s, s') = p(s_t = s, s_{t+1} = s'|o_{1\dots T}, a_{1\dots T-1})$ for all $t = 1 \dots T - 1$ and $s, s' \in S$, and $\gamma_t(s) = p(s_t = s|o_{1\dots T}, a_{1\dots T-1})$ for all $t = 1 \dots T$ and $s \in S$. The gamma values $\gamma_t(s)$ are more precise estimates of the state at time t than the alpha values $\alpha_t(s)$, because they also utilize information (represented by the beta values) that became available only after time t.

A6. Set $\gamma_t(s, s') := \alpha_t(s)p(s'|s, a_t)p(o_{t+1}|s')\beta_{t+1}(s')$ for all $t = 1 \dots T - 1$ and $s, s' \in S$ with $a_t \in A(s)$.

A7. Set $\gamma_t(s) := scale_t\alpha_t(s)\beta_t(s)$ for all $t = 1 \dots T$ and $s \in S$.

Third Step: The Baum-Welch algorithm uses the following frequency-counting re-estimation formulae to calculate the improved initial state distribution, transition probabilities, and observation probabilities (the over-lined symbols represent the probabilities that constitute the improved POMDP):

A8. Set $\bar{\pi}(s) := \gamma_1(s)$.

A9. Set $\bar{p}(s'|s, a) := \sum_{t=1\dots T-1|a_t=a} \gamma_t(s, s')/ \sum_{t=1\dots T-1|a_t=a} \gamma_t(s)$ for all $s, s' \in S$ and $a \in A(s)$.

A10. Set $\bar{p}(o|s) := \sum_{t=1\dots T|o_t=o} \gamma_t(s)/ \sum_{t=1\dots T} \gamma_t(s)$ for all $s \in S$ and all $o \in O$.

To apply the Baum-Welch algorithm to real-world problems, there exist standard techniques for dealing with the following issues [15]: when to stop iterating the algorithm, with which initial POMDP to start the algorithm and how often to apply it to different initial POMDPs, how to handle transition or observation probabilities that are zero or one (the Baum-Welch algorithm does not change these probabilities), and how to deal with short training traces. In addition, we have extended the Baum-Welch algorithm to address the issues of limited memory and the cost of collecting training data [15] and augmented it so that it is able to change the structure of the POMDP [14].

3.4 Most Likely Path: Determining the State Sequence from Observations

Assume that an observer wants to determine the most likely sequence of states that the POMDP process was in. This corresponds to determining the path that the robot most likely took to get to its destination. While the techniques from Section 3.3 can determine the most likely state of the POMDP process at each point in time, merely connecting these states might not result in a continuous path. The Viterbi algorithm uses dynamic programming to compute the most likely path efficiently [36]. Its first three steps are Steps A1. to A3. in Section 3.3, except that the summations on Lines A3.(a) and A3.(b) are replaced by maximizations:

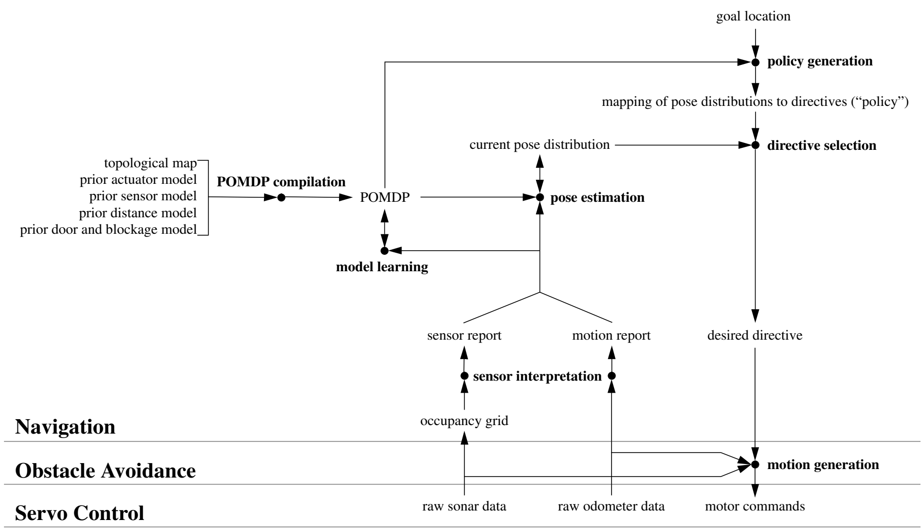


Figure 2: The POMDP-Based Navigation Architecture

B1. Set $scale'_1 := \sum_{s \in S}[q(o_1|s)\pi(s)]$.

B2. Set $\alpha'_1(s) := q(o_1|s)\pi(s)/scale'_1$ for all $s \in S$.

B3. For $t := 1$ to $T-1$

- (a) Set $scale'_{t+1} := \sum_{s \in S}[q(o_{t+1}|s) \max_{s' \in S | a_t \in A(s')}[p(s|s', a_t)\alpha'_t(s')]]$.
- (b) Set $\alpha'_{t+1}(s) := [q(o_{t+1}|s) \max_{s' \in S | a_t \in A(s')}[p(s|s', a_t)\alpha'_t(s')]]/scale'_{t+1}$ for all $s \in S$.

B4. Set $\bar{s}_T := \max_{s \in S} \alpha'_T(s)$.

B5. For $t := T-1$ to 1

- (a) Set $\bar{s}_t := \arg\max_{s' \in S | a_t \in A(s')}[p(\bar{s}_{t+1}|s', a_t)\alpha'_t(s')]$.

$\bar{s}_t$ is the state at time t that is part of the most likely state sequence.

4 The POMDP-Based Navigation Architecture

This section describes the architecture of a navigation system that applies the general models and algorithms presented in Section 3 to the corridor navigation problem. The POMDP-based navigation architecture is one layer of our autonomous mobile robot system for office delivery (Figure 1). Given a goal location for the robot, the role of the navigation layer is to send directives to the obstacle avoidance layer that move the robot towards its goal, where a *directive* is either a change in desired heading or a "stop" command. The obstacle avoidance layer then heads in that direction while using sensor information to avoid obstacles.

The POMDP-based navigation architecture consists of several components (Figure 2). The **sensor interpretation** component converts the continual motion of the robot into discrete *motion reports* (heading changes and distance traveled) and uses the raw sonar sensor data to produce *sensor reports* of high-level features, such as walls and openings of various sizes, observed in front of the robot and to its immediate left and right. The **pose estimation** component uses the motion and sensor reports to maintain a belief about the current pose of the robot by updating the state distribution of the POMDP.

The **policy generation** component, in conjunction with the path planning layer, generates a *policy* that maps state distributions to directives. Whenever the pose estimation component generates a new belief in the current pose of the robot, the **directive selection** component uses this state distribution to index the policy and send the resulting directive to the obstacle avoidance layer, which then generates the robot motion commands. Thus, directive selection is fast and very reactive to the motion and sensor reports received.

The initial POMDP is generated once for each environment by the **POMDP compilation** component. It uses a topological map, initial approximate actuator and sensor models, and uncertain knowledge of the environment. As the robot gains more experience while it performs its tasks, the unsupervised and passive **model learning** component uses the Baum-Welch algorithm to automatically adapt the initial POMDP to the environment of the robot, which improves the accuracy of the actuator and sensor models and reduces the uncertainty about the environment. This increases the precision of the pose estimation component, which ultimately leads to improved navigation performance.

In the following, we first describe the interface between the POMDP-based navigation architecture and the obstacle avoidance layer, including the sensor interpretation component. Then, we describe the POMDP and the POMDP compilation component in detail. Finally, we explain how the POMDP is used by the pose estimation, policy generation, and directive selection components. The model learning component is described in [14, 15].

4.1 Interface to the Obstacle Avoidance Layer

An advantage of our layered robot system is that the navigation layer is insulated from many details of the actuators, sensors, and the environment (such as stationary and moving obstacles). The navigation layer itself provides further abstractions in the form of discrete motion and sensor reports that further insulate the POMDP from details of the robot control.

This abstraction also has the advantage of enabling us to discretize the possible poses of the robot into a finite, relatively small number of states, and keep the number of possible motion and sensor reports small. In particular, we discretize the location with a precision of one meter, and discretize the orientation into the four compass directions. While more fine-grained discretizations yield more precise models, they also result in larger POMDPs and thus in larger memory requirements, more time consuming computations, and a larger amount of training data needed to learn good POMDPs by the model learning component.

4.1.1 Directives

The main role of the obstacle avoidance layer is to make the robot head in a given direction while avoiding obstacles. The task of the navigation layer is to supply a series of changes in desired headings, which we call *directives*, to the obstacle avoidance layer to make the robot reach its goal location.

The directives issued by the navigation layer are: change the desired heading by 90 degrees (turn right), -90 degrees (turn left), 0 degrees (go forward), and stop. Directives are cumulative, so that, for example, two successive "right" directives result in a smooth 180 degree turn. If the robot is already moving, a new directive does not cause it to stop and turn, but merely to change the desired heading, which the obstacle avoidance layer then tries to follow. This results in the robot making smooth turns at corridor junctions.

The robot uses the Curvature Velocity Method [29] to do local obstacle avoidance. The Curvature Velocity Method formulates the problem as one of constrained optimization in velocity space. Constraints are placed on the translational and rotational velocities of the robot that stem from physical limitations (velocities and accelerations) and the environment (the configuration of obstacles). The robot chooses velocity commands that satisfy all the constraints and maximize an objective function that trades off speed, safety, and goal directedness. These commands are then sent to the motors and, if necessary, change the actual heading of the robot.

The obstacle avoidance layer also keeps the robot centered along the main corridor axis by correcting

Sensor	Features that the Sensor Reports on
front	`unknown`, `wall`
left	`unknown`, `wall`, `small_opening`, `medium_opening`, `large_opening`
right	`unknown`, `wall`, `small_opening`, `medium_opening`, `large_opening`

Table 1: Sensors and their Features

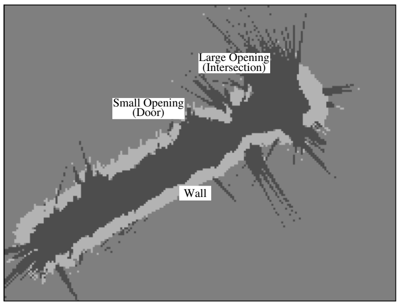


Figure 3: Occupancy Grid with Corridor Features

for angular dead-reckoning error. It tries to fit lines to the sonar data and, if the fit is good enough, uses the angle between the line (for example, a wall) and the desired heading to compensate for angular drift.

4.1.2 Sensor Interpretation: Motion and Sensor Reports

The sensor interpretation component asynchronously generates discrete motion and sensor reports, that are abstractions of the continuous stream of information provided by the sensors on-board the robot (wheel encoders for the motion reports, and sonar sensors for the sensor reports).

Motion reports are generated relative to the desired heading. They are discretized in accordance with the resolution of the POMPD (ninety degrees, one meter). Thus, there are motion reports for when the robot has turned left or right by ninety degrees, gone forward or backward one meter (in a direction parallel to the desired heading), and slid left or right one meter (orthogonal to the desired heading). Slide motions often occur in open spaces (such as foyers) where the robot can move a significant distance orthogonally to the desired heading during obstacle avoidance.

The motion reports are derived by discretizing the smooth motion of the robot. The sensor interpretation component periodically receives reports from the odometer on-board the robot. It combines this information with the robot's commanded heading to produce a virtual odometer that keeps track of the distance traveled along, and orthogonal to, that heading. This ensures that the distance the robot travels in avoiding obstacles is not counted in determining how far it has traveled along a corridor. The odometer reports are integrated over time. A "forward" motion report is generated after each meter of cumulative travel in the desired heading. Similarly, the sensor interpretation component reports when the heading of the robot has changed relative to the desired heading, and this is reported in units of ninety degree turns. This assumes that corridors are straight and perpendicular to each other.

Sensor reports are generated for three virtual (high-level) sensors, that report features in front, to the left, and to the right of the robot. New sensors (such as vision-based sensors) are easily added by specifying their observation probabilities (see Section 4.2.2). Table 1 lists the sensors that we currently use, together with the features that they report on. The sensor reports are derived from the raw sensor data by using a small occupancy grid [8] in the coordinates of the robot that is centered around the robot (Figure 3). The occupancy grid combines the raw data from all sonar sensors and integrates them over the recent past. The occupancy grid is then processed by projecting a sequence of rays perpendicular to the robot heading until they intersect an occupied grid cell. If the end points

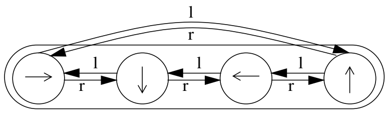


Figure 4: Group of Four States Modeling One Location

of the rays can be fit to a line with a small chi-squared statistic, a wall has been detected with high probability. Similarly, a contiguous sequence of long rays indicates an opening.

The sensor reports filter some noise out of the raw sensor data by integrating them over time and over different raw sensors. Raw data from sonar sensors, for example, can be very noisy due to specular reflections. The sensor reports also approximate the Markov property better than raw sensor data. Two sonar sensors, for example, that point in approximately the same direction produce highly correlated data. By bundling sets of features into one virtual sensor we attempt to make sensor reports more independent of each other, given the current pose of the robot.

4.2 POMDP Compilation

A POMDP is defined by its states and its initial state distribution, its observations and observation probabilities, and its actions, transition probabilities, and immediate rewards. In our POMDP-based navigation architecture, the states of the POMDP encode the pose of the robot. The initial state distribution encodes the available knowledge about the initial pose. The observations are probability distributions over the features, one for each virtual sensor, and the observation probabilities encode the sensor model. The actions are the motion reports (for pose estimation) and the directives (for policy generation), and the transition probabilities encode the topological map, the map uncertainty, and the actuator model. Finally, the immediate rewards (here, costs) express the expected travel times needed to complete the actions, and the objective is to minimize the expected total cost.

The map information is initially encoded as a *topological map*, a graph of nodes and edges that contains information about the connectivity of the environment. Nodes represent junctions between corridors, doorways, or foyers. They are connected by edges, which are augmented with approximate length information in the form of probability distributions over the possible lengths of the edges. We assume that the topological map of the environment can easily be obtained. The approximate corridor lengths can then be obtained from either rough measurements, general knowledge, or the model learning component. The rest of this section describes how this information is used to create the POMDP automatically.

4.2.1 States and Initial State Distribution

Since we discretize the orientation of the robot into the four compass directions, a group of four states, together with "left turn" (l) and "right turn" (r) actions, is necessary to fully represent the possible robot poses at each spatial location (Figure 4). Since we discretize space with a resolution of one meter, each group of four nodes represents one square meter of free space. The initial state distribution of the POMDP process then encodes the, possibly uncertain, knowledge of the initial robot pose.

4.2.2 Observations and Observation Probabilities

We denote the set of virtual sensors by I and the set of features that sensor $i \in I$ reports on by $F(i)$ (Table 1). The sensor model is then specified by the probabilities $q_i(f|s)$ for all $i \in I$, $f \in F(i)$, and $s \in S$, which encode the sensor uncertainty. $q_i(f|s)$ is the probability with which sensor i reports feature f in state s. An observation of the POMDP is the aggregate of one report from each sensor.

Class	Explanation
`wall`	a wall about one meter away
`near-wall`	a wall about two meters away
`open`	a wall three or more meters away (for example, a corridor)
`closed-door`	a closed door
`open-door`	an open door
`door`	a door with unknown door state (open or closed)

Table 2: Classes of States

We do not represent it explicitly, but calculate only its observation probability: If sensor i reports feature f, then $q(o|s) = \prod_{i \in I} q_i(f|s)$. This formula assumes that the sensor reports of different sensors are independent given the state.

A sensor that has not issued a report (for example, because the sensor interpretation component has made no determination which feature is present or was not able to issue a report in time) is assumed to have reported the feature `unknown` with probability one. The probabilities $q_i(\texttt{unknown}|s)$ are chosen so that the state distribution remains unaffected in this case, that is, $q_i(\texttt{unknown}|s) = q_i(\texttt{unknown}|s')$ for all $i \in I$ and $s, s' \in S$. Learning can change these probabilities later because even the sensor report `unknown` can carry information.

To simplify the specification of the sensor model, rather than characterizing $q_i(f|s)$ for each state individually, we characterize classes of states (Table 2). New classes can easily be added, for example for walls adjacent to openings so that a sensor could pick up some of the opening (an example is the wall marked X in Figure 6). The sensor model is then specified by the probabilities that a sensor reports a given feature when the robot is in that particular class of states. For example, the "left" sensor is partially characterized by:

$$
\begin{aligned}
q_{\text{left sensor}}(\texttt{wall}|\texttt{open}) &= 0.05\\
q_{\text{left sensor}}(\texttt{small\_opening}|\texttt{open}) &= 0.20\\
q_{\text{left sensor}}(\texttt{medium\_opening}|\texttt{open}) &= 0.40\\
q_{\text{left sensor}}(\texttt{large\_opening}|\texttt{open}) &= 0.30\\
q_{\text{left sensor}}(\texttt{unknown}|\texttt{open}) &= 0.05
\end{aligned}
$$

These probabilities indicate that corridors are most commonly detected as medium-sized openings, but can often be seen as either large or small openings although they are hardly ever confused for walls.

4.2.3 Actions, Transition Probabilities, and Immediate Rewards

We first discuss how we model actions in general, then how we encode corridors, and finally how we encode junctions, doorways, rooms, and foyers.

4.2.3.1 Modeling Actions The actions of the POMDP encode the motion reports (for pose estimation) and the directives (for policy generation). For the most part, the transition probabilities of a motion report and its corresponding directive are identical. A motion report "turned left" and a directive "turn left," for example, both lead with high probability to a state at the same location whose orientation is ninety degrees counterclockwise. The transition probabilities of actions encode the actuator uncertainty, and their immediate costs encode how long it takes to complete them.

Only the semantics of "forward" motion reports and "forward" directives differ slightly. If the robot is able to move forward one meter, it is unlikely that it was facing a wall. Thus, for dealing with motion reports, the self-transition probability of "forward" actions are set very low in states that face walls. On the other hand, for planning purposes, the same self-transition probabilities are set high, since we know that low-level control routines prevent the robot from moving into walls.

In general, all actions have their intended effect with high probability. However, there is a small chance that the robot ends up in an unintended pose, such as an unintended orientation (not shown

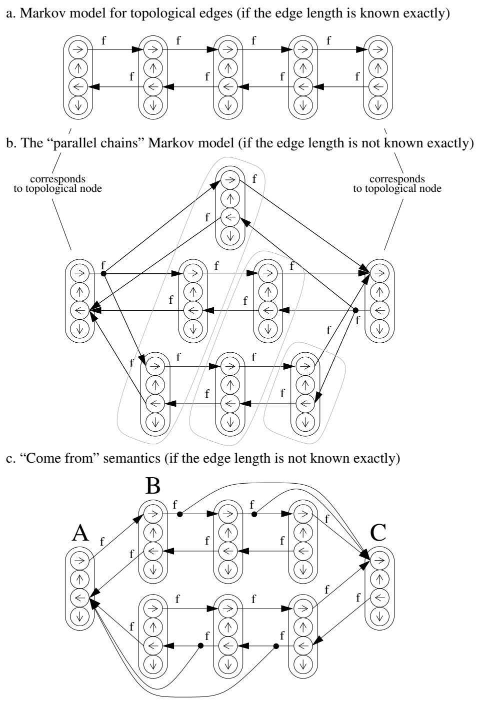


Figure 5: Representations of Topological Edges

in the figures). Exactly which poses these are is determined by the actuator model. Dead-reckoning uncertainty, for example, usually results in the robot overestimating its travel distance in corridors (due to wheel slippage). This can be modeled with self transitions, that is, "forward" actions do not change the state with small probability.

There is a trade-off in which possible transitions to model and which ones to leave out. Fewer transitions result in smaller POMDPs and thus in smaller memory requirements, less time consuming computations, and a smaller amount of training data needed to learn good POMDPs. On the other hand, if one does not model all possible action outcomes, the pose estimates can become inconsistent (every possible pose is ruled out). This is undesirable, because the robot can only recover from an inconsistency by re-localizing itself. Therefore, while we found that most actions have fairly deterministic outcomes, we introduce probabilistic outcomes where appropriate.

4.2.3.2 Modeling Corridors The representation of topological edges is a key to our approach. If the edge lengths are known exactly, it is simple to model the ability to traverse a corridor with a Markov chain that has "forward" (f) actions between those states whose orientations are parallel to the main corridor axis (Figure 5a).

When only approximate edge lengths are known, an edge is modeled as a set of parallel Markov chains, each corresponding to one of the possible lengths of the edge (Figure 5b). The transition probabilities into the first state of each chain model the map uncertainty. They are the same as the probability distribution over the possible edge lengths. Each "forward" transition after that is deterministic (modulo actuator uncertainty). While this representation best captures the actual structure of the environment, it is relatively inefficient: The number of states is quadratic in the difference between the maximum and minimum length to consider.

As a compromise between fidelity and efficiency, one can model edges by collapsing the parallel chains in a way that we call the "come from" semantics (Figure 5c). The states that we collapse into one are framed in Figure 5b. Each topological edge is then represented using two chains, one for each of the

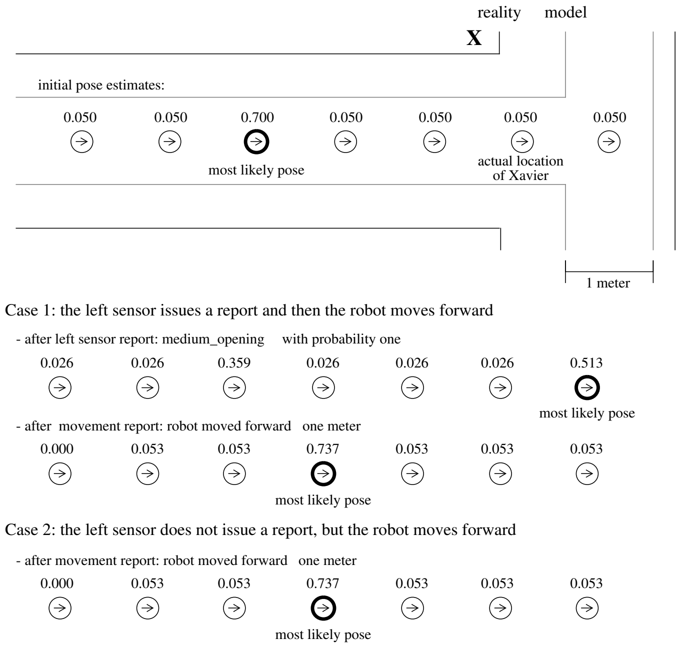
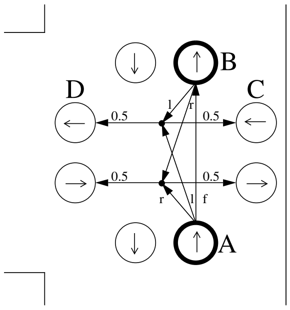


Figure 6: The Effect of Pretending that Corridors are One Meter Wide

corridor directions. In the "come from" semantics, the spatial location of a state is known relative to the topological node from which the robot comes, but its location relative to the end of the chain is uncertain (for example, state B is 1 meter away from A, but is between 1 and 3 meters away from C). An alternative representation is the "go to" semantics, in which the location of a state is specified relative to the topological node towards which the robot is heading, but the distance from the start node is uncertain.

When the length uncertainty is large, the "come from" or "go to" semantics can save significant space over the "parallel chains" representation. For example, for a topological edge between 2 and 10 meters long, they need only 80 states to encode the edge, compared to 188 for the "parallel chains." Since the length uncertainty in our maps is not that large, we actually use the "parallel chains" representation in the POMDP-based navigation architecture that is implemented on Xavier.

4.2.3.3 Modeling Junctions, Doorways, Rooms, and Foyers While we could represent corridor junctions simply with a single group of four states, our experience with the real robot has shown this representation to be inadequate, since the spatial resolution of a state is one meter, but our corridors are two meters wide. To understand why this can lead to problems, consider the scenario shown in Figure 6: The robot is one state away from the center of a T-junction (facing the wall), but due to distance uncertainty still believes it to be further away from the junction. The "left" sensor picks up the opening and reports it. This increases the belief that the robot is already in the junction. Now assume that, due to communication delays, the robot continues to move forward, which is possible because the junction is two meters wide. According to the model, however, this rules out that the robot was in the junction (the robot cannot move forward in a junction that is one meter wide, except for a small probability of self-transitioning), and the most likely state jumps back into the corridor. The resulting state distribution is approximately the same as if the "left" sensor had not issued its report. Thus, the sensor report contributes little to the current pose estimate and its information is essentially lost. Figure 6 illustrates this effect under the simplifying assumption that there is no distance uncertainty, the actuator model is completely deterministic, and the "left" sensor is characterized by $q_{\text{left sensor}}(\texttt{medium\_opening}|\texttt{open}) = 0.40$ and $q_{\text{left sensor}}(\texttt{medium\_opening}|\texttt{wall}) = 0.02$. However, the effect is independent of the sensor model or the initial state distribution.



(for clarity, only actions from the highlighted nodes are shown)

Figure 7: Representation of Corridor Junctions

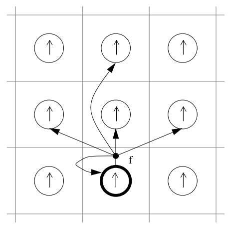


Figure 8: Representation of Two-Dimensional Space

While one remedy is to represent junctions using four (that is, two by two) groups of four states each, we achieve nearly the same result with four groups of two states each, which both saves space and makes the model simpler (Figure 7). The basic idea is that turns within a junction are non-deterministic and transition with equal probability to one of the two states of the appropriate orientation in the junction. For example, in entering the junction of Figure 7 from the South, the robot would first encounter state A, then state B if it continued to move forward. If it then turned left, it would be facing West, and would transition to either states C or D with equal probability. This model agrees with how the robot actually behaves in junctions. In particular, it corresponds to the fact that it is very difficult to pin down the robot's location exactly while it is turning in the middle of an intersection.

Doorways can be modeled more simply, since the width of our doors is approximately the resolution of the POMDP. A single exact-length edge, as in Figure 5a, leads through a door into a room. The state of a door (open or closed) can typically change and is thus often not known in advance. We therefore associate with doorways a probability p that the door is open. This probability encodes the uncertainty about the dynamic state of the environment. The observation probabilities associated with seeing a doorway are:

$$q(o|\texttt{door}) = p \times q(o|\texttt{open-door}) + (1-p) \times q(o|\texttt{closed-door}).$$

While we model corridors as one-dimensional chains of states, we represent foyers and rooms by tessellating two-dimensional space into a matrix of locations. From each location, the "forward" action has some probability of transitioning straight ahead, but also some probability of self-transitioning and moving to diagonally adjacent states which represents the robot drifting sideways without noticing it (Figure 8). Currently, we do not have a good method for efficiently representing distance uncertainty in rooms and foyers.

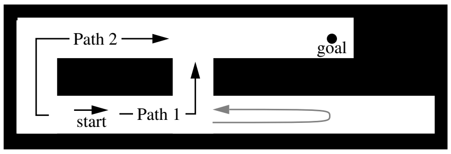


Figure 9: An Office Corridor Environment

4.3 Using the POMDP

In this section, we explain how our navigation architecture uses POMDP algorithms for estimating the robot's pose and for directing its behavior.

4.3.1 Pose Estimation

The pose estimation component uses the motion and sensor reports to update the state distribution (that is, the belief in the current pose of the robot) using Equations (1) and (2) from Section 3.1. The updates are fast, since they require only one iteration over all states, plus an additional iteration for the subsequent normalization. While the amount of computation can be decreased by performing the normalization only once in a while (to prevent rounding errors from dominating), it is fast enough that we actually do it every time.

Since reports by the same sensor at the same location are clearly dependent (they depend on the same cells of the occupancy grid), aggregating multiple reports without a motion report in between would violate the Markov assumption. The pose estimation component therefore uses only the latest report from each sensor between motion reports to update the state distribution.

Motion reports tend to increase the pose uncertainty, due to non-deterministic transitions (actuator and distance uncertainty), while sensor reports tend to decrease it. An exception is "forward" motion reports that can decrease uncertainty if some of the states with non-zero probability are "wall facing". In practice, this effect can be seen when the robot turns at an intersection: Before the turn, there is often some probability that the robot has not yet reached the intersection. After the robot has turned and successfully moved forward a bit, the probability that it is still in the original corridor quickly drops. This is a major factor in keeping the positional uncertainty low, even when the robot travels long distances.

4.3.2 Policy Generation and Directive Selection

The policy generation component has to compute a policy that minimizes the expected travel time of the robot. The directive selection component then simply indexes this policy repeatedly with the current state distribution to determine which directives to execute.

For policy generation, we need to handle "stop" directives. This is done by adding a "stop" action to the set $A(s)$ of each state s. The immediate cost of the "stop" action is zero in states that correspond to the goal location, and is high otherwise. In all cases, the "stop" action leads with certainty to a distinguished POMDP state from which no future immediate costs are possible. The policy generation component then has to determine a policy that minimizes the expected total cost received, since the immediate costs of all actions reflect their execution times.

Since our POMDPs typically have thousands of states, we need to use the greedy POMDP planning approaches from Section 3.2. These approaches pretend that the POMDP is completely observable. Under this assumption, they determine an optimal mapping from states to directives (a policy) and then transform it into a mapping from state *distributions* to directives (a POMDP policy). This, however, can lead to very suboptimal results. In Figure 9, for example, Path 1 is shorter than

Path 2 and thus requires less travel time if the states are completely observable. Because of sensing uncertainty, however, a robot can miss the first turn on Path 1 and overshoot. It then has to turn around and look for the corridor opening again, which increases its travel time along Path 1. On the other hand, when the robot follows Path 2 it cannot miss turns. Thus, it might actually be faster to follow Path 2 than Path 1, but the greedy POMDP planning approaches would always recommend Path 1.

We address this problem by using a different approach for generating the policy, but continue to use the approaches from Section 3.2 to complete it. To generate the policy, we use the decision-theoretic path planning layer of our robot system, that takes into account that the robot can miss turns and corridors can be blocked. It uses a generate, evaluate, and refine strategy to determine a path in the topological map that minimizes the expected travel time of the robot [13]. The navigation layer then converts this path to a complete policy. For states corresponding to a topological node on the path, directives are assigned to head the robot towards the next node (except for the states corresponding to the last node on the path, which are assigned "stop" directives). Similarly, for all states corresponding to the topological edges between two nodes on the path, directives are assigned to head the robot towards the next node. Finally, for states not on the path, directives are assigned that move the robot back towards the planned path. In this way, if the robot strays from the nominal (optimal) path, it will automatically execute corrective directives once it realizes its mistake. Thus, re-planning is only necessary when the robot detects that the nominal path is blocked. While this planning method is still suboptimal, it is a reasonable compromise between planning efficiency and plan quality.

As described in Section 3.2, there are several greedy approaches for choosing which directive to issue, given a policy and the current state distribution. It turns out that the "Most Likely State" strategy does not work well with our models, because topological entities are encoded in multiple states. For example, since corridor junctions are modeled using several states for each orientation, it is reasonable to consider all of their recommendations when deciding which directive to issue. The "Voting" and "Completely Observable after the First Step" strategies both work well in our environment if the initial positional uncertainty is not overly large. Both strategies are relatively immune to the limited positional uncertainty that arises during navigation.

These greedy approaches also have disadvantages. Since they operate on a policy and account for positional uncertainty only greedily, they make the assumption that the robot collects sensor data on the fly as it moves closer to its destination. As opposed to other POMDP planning methods, they do not plan when, where, and what to sense. This property fits sonar sensors well, since sonar sensors produce a continuous stream of data and do not need to be pointed. A disadvantage of this property is, however, that the robot does not handle well situations in which localization is necessary, and we thus recommend to use more sophisticated POMDP planning techniques as they become more efficient. For example, it is often more effective to actively gather information that helps the robot to reduce its pose uncertainty, even if this requires the robot to move away from the goal temporarily.

On the other hand, the greedy approaches often lead to an optimal behavior of the robot. For example, even if the robot does not know for certain which of two parallel corridors it is traversing, it does not need to stop and re-plan, as long as the directives associated with both corridors are the same. In this way, the robot can continue making progress towards its desired goal, while at the same time collecting evidence, in the form of sensor readings, that can help to disambiguate its true location. This behavior takes advantage of the fact that buildings are usually constructed in a way that allows people to obtain sufficient clues about their current location from their local environment – otherwise they would easily become confused (of course, some landmarks that can be observed by people, such as signs or door labels, cannot be detected by sonar sensors).

5 Experiments

The POMDP-based navigation architecture described in this article is implemented in C and runs on-board the robot on Intel 486 computers under the Linux operating system. It is a separate, concurrent

Figure 10: Xavier

process and communicates with the other layers of the robot system via message passing, supported by the Task Control Architecture [28].

In this section, we report on experiments that we performed in two environments for which the Markov property is only an approximation: an actual mobile robot navigating in our building, and a realistic simulation of the robot. The experiments show that the Markov property is satisfied well enough for the POMDP-based navigation architecture to yield a reliable navigation performance. We use the same navigation code for both sets of experiments, since the robot and its simulator have the exact same interfaces, down to the level of the servo control layer (Figure 1).

In all experiments, we modeled the length uncertainty of each topological edge as a uniform distribution over the interval ranging from 80 to 150 percent of its true length and kept the initial positional uncertainty minimal: The initial probability for the robot's actual location was about 70 percent. The remaining probability mass was distributed in the vicinity of the actual location.

We report results for the "Voting Strategy" of policy generation and directive selection. The experiments demonstrate that for the office navigation problems considered here, the efficient voting strategy performs very well. For an empirical comparison of several greedy policy generation and directive selection strategies in more complex environments, but using simpler POMDPs than we use here, see [2].

5.1 Experiments with the Robot

The robot experiments were performed on Xavier (Figure 10). Xavier, which was designed and built by our group at Carnegie Mellon University, is built on top of a 24 inch diameter RWI B24 base, which is a four-wheeled synchro-drive mechanism that allows for independent control of the translational and rotational velocities. The sensors on Xavier include bump panels, wheel encoders, a Denning sonar ring with 24 ultrasonic sensors, a front-pointing Nomadics laser light striper with a 30 degree field of view, and a Sony color camera that is mounted on a pan-tilt head from Directed Perception (the current navigation system does not use the laser light striper or the camera).

Control, perception, and planning are carried out on two on-board 66 Megahertz Intel 486 computers. An on-board color Intel 486 lap-top computer is used to monitor Xavier's status with a graphical interface, a screen shot of which is shown in Figure 11 (the sizes of the small circles in the right corridor are proportional to the probability mass in each corridor segment; the amount of positional uncertainty shown is typical). The computers are connected to each other via Ethernet and to the outside world via a Wavelan radio connection.

Xavier is controlled via a World Wide Web interface (http://www.cs.cmu.edu/∼Xavier) that allows users worldwide to specify goal locations for Xavier on one floor of our building, half of which is shown

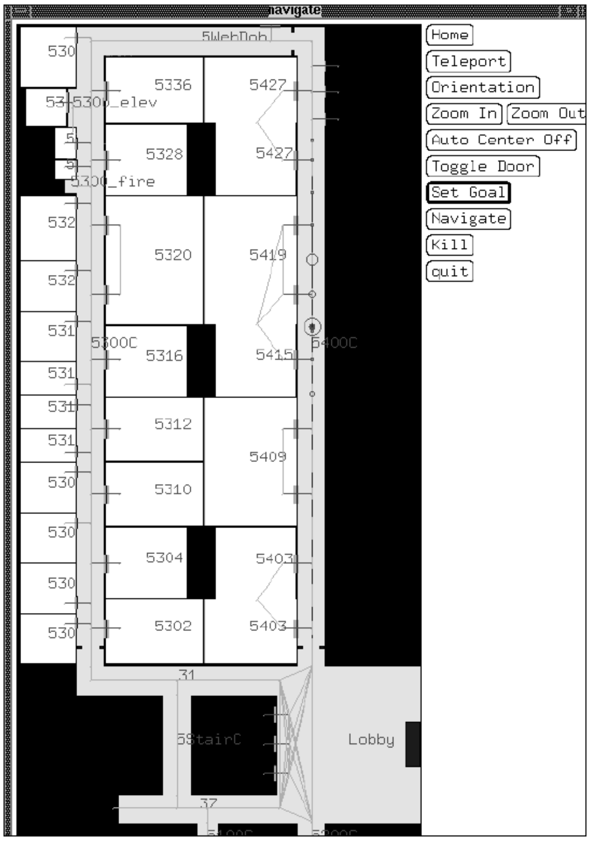


Figure 11: Graphical Interface

Month	Days in Use	Jobs Attempted	Jobs Completed	Completion Rate	Distance Traveled
December 1995	13	262	250	95 %	7.7 km
January 1996	16	344	310	90 %	11.4 km
February 1996	15	245	229	93 %	11.6 km
March 1996	13	209	194	93 %	10.0 km
April 1996	18	319	304	95 %	14.1 km
May 1996	12	192	180	94 %	7.9 km
Total	87	1,571	1,467	93 %	62.7 km

Table 3: Performance Data (all numbers are approximate)

in Figure 11. The part shown has 95 nodes and 180 directed edges, and the POMDP has 3,348 states. In the period from December 1, 1995 to May 31, 1996 Xavier attempted 1,571 navigation requests and reached its intended destination in 1,467 cases, where each job required it to move 40 meters on average (see Table 3 and [30] for more details). Most failures are due to problems with our hardware (boards shaking loose) and the wireless communication between the on-board robot system and the off-board user interface (which includes the statistics-gathering software), and thus are unrelated to the POMDP-based navigation architecture. This success rate of 93 percent compares favorably with the 80 percent success rate that we obtained when using a landmark-based navigation technique in place of the POMDP-based navigation layer on the same robot with an otherwise unchanged robot system [27]. Thus, the difference in performance can be directly attributed to the different navigation techniques.

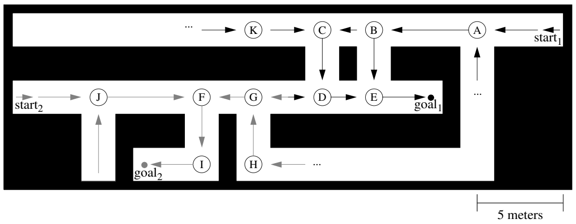


Figure 12: Another Office Corridor Environment

Path	Frequency	Time	Speed
ABE	12	68.2 s	25.7 cm/s
ABCDE	3	79.7 s	29.5 cm/s

Table 4: Experiment 1

5.2 Experiments with the Simulator

To show the performance of the POMDP-based navigation architecture in an environment that is more complex than what we have available in our building, we also performed two navigation experiments with the Xavier simulator in the office corridor environment shown in Figure 12. Its topological map has 17 nodes and 36 directed edges, and the POMDP has 1184 states.

In Experiment 1, the task was to navigate from $start_1$ to $goal_1$. The preferred headings are shown with solid arrows. Note that the preferred heading between B and C is towards C because this way the robot does not need to turn around if it overshoots B (which minimizes its travel time, even though the goal distance is a bit longer). We ran a total of 15 trials (Table 4), all of which were completed successfully. The robot has to travel a rather long distance from $start_1$ before its first turn. Since this distance is uncertain and corridor openings are occasionally missed, the robot occasionally overshoots B, and then becomes uncertain whether it is really at C or B. However, since the same directive is assigned to both nodes, this ambiguity does not need to be resolved; the robot turns left in both cases and then goes straight. The same thing happens when it gets to D, since it thinks it may be at either D or E. The robot eventually corrects its beliefs when, after turning left and traveling forward, it detects an opening to its left. At this point, the robot becomes fairly certain that it is at E. A purely landmark-based navigation technique can easily get confused in this situation, since it has no expectations about seeing this opening, and can only attribute it to sensor error (which, in this case, is incorrect).

In Experiment 2, the robot had to navigate from $start_2$ to $goal_2$. The preferred headings for this task are shown with dashed arrows. Again, we ran 15 trials (Table 5). For reasons that are similar to those in the first experiment, the robot can confuse G with F. If it is at G but thinks it is probably at F, it turns right and goes forward. However, when it detects the end of the corridor but does not detect a right corridor opening, it realizes that it must be at H rather than I. Since the probability mass has now shifted, it turns around and goes over G, F, and I to the goal. This shows that our navigation technique can gracefully recover from misjudgments based on wrong sensor reports – even if it takes some time to correct its beliefs. It is important to realize that this behavior is not triggered by any explicit exception mechanism, but results automatically from the way the pose estimation and directive selection interact.

6 Conclusions

This paper has presented a navigation architecture that uses partially observable Markov decision process models (POMDPs) for autonomous indoor navigation. The navigation architecture, which is one layer in a larger autonomous mobile robot system, provides for reliable and efficient navigation in

Path	Frequency	Time	Speed
JFI	11	60.6 s	28.9 cm/s
JFGFI	2	91.5 s	25.7 cm/s
JFGHGFI	1	116.0 s	23.7 cm/s
JFGFGFI	1	133.0 s	22.2 cm/s

Table 5: Experiment 2

office environments.

The implemented POMDP-based navigation architecture has demonstrated its reliability, even in the presence of unreliable actuators and sensors, as well as uncertainty in the metric map information. Robot control is fast and reactive during navigation (Xavier averages fifty centimeters per second), and robust, as demonstrated by experiments that required the robot to navigate over 60 kilometers in total.

We believe that such probabilistic navigation techniques hold great promise for getting robots reliable enough to operate unattended for long periods of time in complex and uncertain environments. Applying POMDPs to robot navigation also opens up new application areas for more theoretical results in the area of planning and learning with Markov models.

Acknowledgements

Thanks to Lonnie Chrisman, Richard Goodwin, Karen Haigh, and Joseph O'Sullivan for helping to implement parts of the robot system and for many valuable discussions. Swantje Willms helped to perform some of the experiments with the simulator. This research was supported in part by NASA under contract NAGW-1175 and by the Wright Laboratory and ARPA under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies.

References

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

[2] A. Cassandra, L. Kaelbling, and J. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 963–972, 1996.

[3] A. Cassandra, L. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1023–1028, 1994.

[4] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 183–188, 1992.

[5] T. Dean, K. Basye, R. Chekaluk, S. Hyun, M. Lejter, and M. Randazza. Coping with uncertainty in a control system for navigation and exploration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1010–1015, 1990.

[6] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 574–579, 1993.

[7] P. Devijver. Baum's forward backward algorithm revisited. *Pattern Recognition Letters*, 3:369–373, 1985.

[8] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, pages 46–57, 6 1989.

[9] K. Haigh and M. Veloso. Interleaving planning and robot execution for asynchronous user requests. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 148–155, 1996.

[10] B. Hannaford and P. Lee. Hidden Markov model analysis of force/torque information in telemanipulation. *The International Journal of Robotics Research*, 10(5):528–539, 1991.

[11] X. Huang, Y. Ariki, and M. Jack. *Hidden Markov models for speech recognition.* Edinburgh University Press, 1990.

[12] S. Koenig. Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Master's thesis, Computer Science Department, University of California at Berkeley, Berkeley (California), 1991. (Available as Technical Report UCB/CSD 92/685).

[13] S. Koenig, R. Goodwin, and R. Simmons. Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In L. Dorst, M. van Lambalgen, and R. Voorbraak, editors, *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Artificial Intelligence*, pages 322–337. Springer, 1996.

[14] S. Koenig and R. Simmons. Passive distance learning for robot navigation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 266–274, 1996.

[15] S. Koenig and R. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2301–2308, 1996.

[16] A. Kosaka and A. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2177–2186, 1992.

[17] B. Kuipers and Y.-T. Byun. A robust, qualitative method for robot spatial learning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 774–779, 1988.

[18] M. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, Providence (Rhode Island), 1996. (Available as Technical Report CS-96-09).

[19] M. Littman, A. Cassandra, and L. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 362–370, 1995.

[20] M. Littman, T. Dean, and L. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 394–402, 1995.

[21] M. Mataric. Environment learning using a distributed representation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 402–406, 1990.

[22] A. McCallum. Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems 7*, pages 377–384, 1995.

[23] I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish: An office-navigating robot. *AI Magazine*, 16(2):53–60, 1995.

[24] C. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[25] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1088–1094, 1995.

[26] L. Rabiner. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, 1 1986.

[27] R. Simmons. Becoming increasingly reliable. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 152–157, 1994.

[28] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[29] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3375–3382, 1996.

[30] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O'Sullivan. A layered architecture for office delivery robots. In *Proceedings of the International Conference on Autonomous Agents*, 1997.

[31] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1080–1087, 1995.

[32] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5:56–68, 1986.

[33] E. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

[34] A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In *Advances in Neural Information Processing Systems 5*, pages 11–18, 1993.

[35] J. Tenenberg, J. Karlsson, and S. Whitehead. Learning via task decomposition. In *Proceedings of the "From Animals to Animats" Conference*, pages 337–343, 1992.

[36] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, 1967.

[37] J. Yang, Y. Xu, and C. Chen. Hidden Markov model approach to skill learning and its application to telerobotics. *IEEE Transactions on Robotics and Automation*, 10(5):621–631, 1994.