Acting under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation

Anthony R. Cassandra
arc@cs.brown.edu

Leslie Pack Kaelbling
lpk@cs.brown.edu

James A. Kurien
jmk@cs.brown.edu

Department of Computer Science
Brown University

Abstract

Discrete Bayesian models have been used to model uncertainty for mobile-robot navigation, but the question of how actions should be chosen remains largely unexplored. This paper presents the optimal solution to the problem, formulated as a partially observable Markov decision process. Since solving for the optimal control policy is intractable, in general, it goes on to explore a variety of heuristic control strategies. The control strategies are compared experimentally, both in simulation and in runs on a robot.

1 Introduction

A robot that delivers items and performs errands in an office environment needs to be able to navigate robustly. It should be able to overcome errors in perception and action, at worst getting lost for some period of time, but then being able to recover by re-localizing itself and continuing with its task.

The Bayesian framework is particularly appropriate for modeling the robot's belief about its location (or, more generally, the state of the world). It supplies a well-founded method for taking the robot's current beliefs and combining them with uncertain information gained from sensing and acting. The Bayesian framework has been used for a long time in robotics with good results [7]. In this paper, we develop a two-level architecture, with Bayesian modeling done at the top level only. In addition, we explore sub-optimal control strategies given the Bayesian belief state.

The standard process for applying the Bayesian framework to modeling mobile-robot uncertainty is:

- build, typically manually, a detailed, metric, Cartesian map of the robot's environment, including positions of perceptible features;
- represent the true state of the robot as a position and orientation (pose) in global coordinates;
- represent the belief state of the robot using a continuous parametric probability distribution (typically Gaussian) over poses;
- model uncertainty in the robot's actions and observations using continuous parametric distributions (typically Gaussians);
- predict which features will be observed and match them to actual observations; then use a Kalman filter to update belief state based on actions and matched observations; and
- choose actions most appropriate for the mean or mode of the belief state.

For many tasks in many environments, it is not necessary to know the robot's pose in detail. Given robust low-level routines that can, for example, use local sensors to drive through a door, it is only necessary to know that the robot is in some region outside the door that serves as a precondition to the low-level operator. In such cases, a more coarse-grained uncertainty model may be appropriate. We pursue an approach to navigation in which the process is to:

- develop a set of robust low-level perception and action routines (such as driving around corners and detecting doors);
- build, manually, a topological model of the environment, with nodes corresponding to regions of pose space that can be grouped as preconditions or postconditions of the action routines;
- represent the true state of the robot as a node in the topological map;
- represent the belief state of the robot using a discrete probability distribution over these nodes;
- model the uncertainty in the robot's actions and observations using discrete distributions;
- use Bayes' rule to update the belief distribution based on its actions and observations;
- use instantaneous rewards and a discounted optimization criterion to specify the objective; and
- take control actions given the robot's belief state.

One advantage of this approach is that it can be applied just as easily to situations in which the robot initially knows its location as to situations in which the robot's initial belief state is uniform (it has no idea where it is). As the robot begins to take actions and make observations, the belief distribution will change to reflect its new information; there is no need for this distribution to be continuous or unimodal. It may be the case, for example, that the robot will know that it is in a corner of the building, but not which one.

This is by no means the first project to use discrete belief models. The Dervish project at Stanford University [13] used a topological map combined with robust low-level behaviors. Their belief-state update was heuristic, based only on a model of observational error (but not error due to actions). The Xavier project at Carnegie-Mellon University [17] used a full belief-state update. Both of these projects used fairly *ad hoc* action strategies, based on planning paths as if the domain were deterministic. The major contribution of this paper is to outline the optimal control strategy, then to present a number of heuristic approximations to it, comparing their performance in simulation and in runs on a real robot.

We begin by describing the architecture of the system. Next, we introduce the optimal belief state update and control strategies, and describe a number of heuristic control strategies. Finally, we present experimental results comparing these control strategies both in simulated experiments and on our mobile robot.

2 System Architecture

The system described in this paper has a two-level software architecture. The higher level, referred to as the *navigator*, uses an abstract Bayesian model of the environment and the robot's actions and observations. In the context of this model, the navigator receives observations, maintains a belief distribution, and chooses actions. The role of the lower level software, called the *pilot*, is to bridge the gap between the actions and observations of the abstract model and the motions and percepts of the underlying robot.

2.1 Navigator Layer

The navigator, as we have implemented it, assumes that it is running in an office-type environment, made up of corridors and rooms oriented along two approximately orthogonal axes. The general approach described here could be applied to any other environment that is easily characterized as a network of states with abstract operations that move between them. *e.g.*, street networks and underground sewer systems.

We provide the navigator with rough metric information for the office environment, from which it generates a state map. A state consists of a location and an orientation (one of four compass directions). The locations can be discrete square areas of any size in the environment, though for all of our experiments we used a one-square-meter discretization. The map describes the underlying connectivity of these states and specifies a type, room or corridor, for each location. The location types help define the ideal observations for each state. For example, if the robot is in a corridor-type location and looks in the direction of an adjacent location, which happens to be a room-type location, then ideally it would see a doorway.

Abstract Actions The robot has five abstract actions: *move-forward*, *turn-left*, *turn-right*, *no-op* and *declare-goal*. The *declare-goal* action is used by the robot to indicate that it has achieved its objective. The action model specifies, for each state and action, the probability that each state will result. Due to the regularities in the environment, we can compute the action model from the connectivity of the map and an action-error model. These probabilities were obtained through informal experimentation, although it would be a fairly simple extension to learn from experiences.

Table 1 shows the action probabilities used in our experiments. We define these abstract actions in terms of possible actual outcomes. For example, in the *turn-left* action, F-L (0.1) means that, with probability 0.1, the the robot would actually move forward (F) and then rotate 90 degrees to the left (L). Rotating 90 degrees to the right is notated by (R) and no movement by (N). If an outcome cannot occur in a particular state of the world (for example, going forward when there is wall in front), then the robot is left in the last state before the impossible outcome. The "standard" outcomes were used in simulations and on the robot and the "noisy" column were used in simulation experiments to explore the effects of increasing noise.

Action	Standard outcomes (probabilities)
move-forward	N (0.11), F (0.88), F-F (0.01)
turn-left	N (0.05), L (0.9), L-L (0.05)
turn-right	N (0.05), R (0.9), R-R (0.05)
no-op	N (1.0)
declare-goal	N (1.0)

Action	Noisy outcomes (probabilities)
move-forward	N (0.05), F (0.7), F-F (0.05), L (0.1), R (0.1)
turn-left	N (0.1), L (0.7), L-L (0.1), F-L (0.1)
turn-right	N (0.1), R (0.7), R-R (0.1), F-R (0.1)
no-op	N (1.0)
declare-goal	N (1.0)

Table 1: Action probabilities for abstract actions.

Abstract Observations In each state, the robot is able to make an abstract observation. It can perceive, in each of three nominal directions (front, left, and right) whether there is a *doorway*, *wall*, *open*, or it is *undetermined*. An abstract observation is the combination of the percepts in each direction. Thus, there are 64 possible abstract observations. The observation model specifies, for each state and action, the probability that a particular observation will be made. As with the abstract action model, the entire table need not be specified: the observation probabilities are computed from the ideal observations, as specified in the map, and an error model.

Table 2 shows the conditional probabilities for the abstract observations. The observations depend on the action taken, since the robot has to move to get additional useful sensor readings. In our implementation, the observation probabilities after a *move-forward*, *turn-left* or *turn-right* action are the same, but *no-op* results in no new observational information.

Modeling the observations this way makes some strong independence assumptions. Not only do we assume that the individual directional percepts at a given time are independent, we also assume that the observations at one instant are independent of the observations at another instant. In the actual implementation of the observations in the pilot layer, these assumptions are not justified. However, this does not seem to adversely affect the results.

2.2 Pilot Layer

The pilot layer supports the navigator's model on a real robot, in this case a Real World Interface, Inc.

O_i	O_o	$P(O_o \mid O_i)$	
Ideal	Observed	Standard	Noisy
wall	wall	0.90	0.70
wall	open	0.04	0.19
wall	doorway	0.04	0.09
wall	undetermined	0.02	0.02
open	wall	0.02	0.19
open	open	0.90	0.70
open	doorway	0.06	0.09
open	undetermined	0.02	0.02
doorway	wall	0.15	0.15
doorway	open	0.15	0.15
doorway	doorway	0.69	0.69
doorway	undetermined	0.01	0.01
undetermined	undetermined	1.00	1.00

Table 2: Conditional observation probabilities used to construct the abstract observation noise model.

B21 mobile robot. The B21 is a four-wheeled cylindrical synchro-drive base with 24 ultrasonic sensors and 24 infrared sensors evenly distributed about its circumference. The infra-red sensors are fairly reliable short-range proximity sensors used only for emergency obstacle detection and avoidance. The ultrasonic sensors give longer-range proximity information, but, due to higher-order specular reflections, their readings must be combined in order to get a true picture of surfaces in the world. They are combined in a local occupancy grid, which is the basis of high-level feature detection.

Occupancy Grid The pilot fuses ultrasonic measurements in an occupancy grid using a simplified variant of the algorithm of Moravec and Elfes [12]. Since the pilot is responsible solely for local navigation, the occupancy grid only maps features whose distance from the robot is within a small range. As the robot moves forward, the occupancy grid maintains a small, robot-centric map, with information derived from old ultrasonic data constantly scrolling off the grid's trailing boundary. Since only short-range ultrasonic readings are considered, a simple certainty model is used in place of a more complex probabilistic model involving the spread of the ultrasonic wave over distance.

Restricting the grid to a local region allows numerous simplifications over global occupancy-grid methods such as those used in RHINO [1]. First, even when the robot is operating in very narrow corridors there is no preprocessing of ultrasonic data to eliminate higher-order specular reflection. Any ultrasonic reading greater than 1.5 meters is not relevant to the grid and is discarded, eliminating the majority of troublesome readings. Second, the grid is robot-centric and data only persists over a few meters of motion. There is no requirement that the robot's dead reckoning be accurate except over very small intervals, eliminating the need for error estimates and correction. Third, even using as much resolution as the ultrasonic sensors can deliver, our small map requires relatively little time and space to maintain. This resolution has enabled robust feature detection in the grid.

Supporting Abstract Observations The pilot's percepts and actions are highly dependent upon detecting features of the environment such as doors or the direction of a world axis. These features are determined by processing the local occupancy grid with several specialized feature detection algorithms. The orientation of the closest world axis is estimated by integrating the world axis estimates of several local occupancy grids as the robot travels. Within the current grid, a local estimate of wall direction is found by looking for one or more long parallel line segments in the occupancy grid image. If the robot turns, the existing world axis estimate is rotated ninety degrees and serves as a seed in the new estimate of the axis in the current direction of travel.

Once a world axis is estimated, a simple search for doors, openings and walls in the robot's vicinity is made to the left, front and right of the robot. If there is a rough line segment (using a Hough transform [4]) in the occupancy grid that is parallel or perpendicular to the estimated axis, then a *wall* is observed. Similarly, if the occupancy grid is largely clear along an axis an *open* observation results. A *door* is observed by an *ad hoc* detector that searches for door-like indentations or openings in what would otherwise be classified as a wall segment. Percepts for the left, front and right of the robot are returned by the pilot after each action and combined into a single observation.

Supporting Abstract Actions The pilot layer supports four of the abstract actions: *move-forward*, *turn-left*, *turn-right*, and *no-op*. Note that in the pilot layer the *declare-goal* action has the same effect as the *no-op* action. The *turn-left* and *turn-right* actions attempt to reorient the robot (independent of the current orientation) to the axis orthogonal to the current one.

To simplify modeling of location, the navigator discretizes the world into locations a uniform distance apart. As a result, its model specifies that each *move-forward* takes the robot one distance unit (currently a meter) and that all interesting features (doors, start of an opening, etc) are separated by even multiples of the distance unit. The resulting semantics of the *move-forward* action are slightly complicated. If the pilot does not detect a new feature, it simply attempts to move forward one unit along the estimated axis of travel. If obstacles are encountered, the pilot will attempt to guide the robot past the objects until approximately one unit of travel in the requested direction has been accomplished. If a new feature is encountered, such as a door or a new opening, the forward motion is completed as if one unit had been traveled. Thus, the robot always appears to travel a whole number of distance units between features, supporting the discretized model the navigator requires.

2.3 Reliability

It is important to note that the pilot system is not completely reliable. For example, it occasionally mistakes a hallway for an open door and it never knows which of the four world axes it is following. It occasionally even mis-estimates an axis, traveling diagonally in the world for short distances. In informal experiments using the pilot as if it were deterministic, these slight fallibilities proved disastrous. However, since the navigator explicitly models the fallibility of the abstraction the pilot provides, the current level of competence is adequate for the navigation tasks we have attempted.

3 POMDP Model

In this section, we briefly describe the class of models known as *partially observable Markov decision processes* (POMDPs). This model was developed in the operations research community [10, 20] and has been recently introduced to artificial intelligence researchers [2]. We start by describing the simpler class of Markov decision processes.

3.1 Markov Decision Processes

An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is a finite set of environment states that can be reliably identified by the robot; $\mathcal{A}$ is a finite set of actions; T is a state transition model of the environment, which is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$; and R is a *reward function* mapping $\mathcal{S} \times \mathcal{A}$ to $\Re$ that specifies the instantaneous reward that the robot derives from taking each action in each state. We write $T(s, a, s')$ for the probability that the environment will make a transition from state s to state s' when action a is taken and we write $R(s, a)$ for the immediate reward to the robot for taking action a in state s. For a process to be Markov, the current state and action must provide all of the information available for predicting the next state. A *policy* π is a mapping from $\mathcal{S}$ to $\mathcal{A}$, specifying an action to be taken in each situation.

Given a policy π and a reward function R, the *value* of state $s \in \mathcal{S}$, $V_\pi(s)$, is the expected value of sums of the rewards to be received at each future time step, discounted by how far into the future they occur. That is, $V_\pi(s) = E(\sum_{t=0}^{\infty} \gamma^t R_t)$, where R_t is the reward received on the tth step of executing policy π after starting in state s. A closely related quantity $Q_\pi(s, a)$, is the value of state s given action a is executed first and policy π followed thereafter.

The *discount factor*, $0 \leq \gamma < 1$, controls the influence of rewards in the distant future. When $\gamma = 0$, the value of a state is determined entirely by rewards received on the next step; we are generally interested in problems with a longer horizon and set γ to be near 1. Due to properties of the exponential, the definition of V can be rewritten as

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \Pr(s, \pi(s), s') V_\pi(s') \ .$$

We say that policy π *dominates* (is better than) π' if, for all $s \in \mathcal{S}$, $V_\pi(s) \geq V_{\pi'}(s)$, and for at least one $s \in \mathcal{S}$, $V_\pi(s) > V_{\pi'}(s)$. A policy is optimal if it is not dominated by any other policy. Given a Markov decision process and a value for γ, it is possible to compute the optimal policy fairly efficiently [16]. We shall use $\pi^*(s)$ to refer to an optimal policy for an MDP, and express the the optimal value and Q functions as $V(s)$ and $Q(s, a)$.

3.2 Adding Partial Observability

When the state is not completely observable, we must add a model of observations. This includes a finite set $\mathcal{O}$ of possible observations and an observation function O, mapping $\mathcal{A} \times \mathcal{S}$ into discrete probability distributions over $\mathcal{O}$. We write $O(a, s, o)$ for the probability of making observation o from state s after having taken action a.

A *belief state* is a discrete probability distribution over the set of environment states, $\mathcal{S}$, representing for each state the robot's belief that it is currently occupying that state. Let $\mathcal{B}$ be the set of belief states. We write $b(s)$ for the probability value assigned to environment state s in belief state b. Now, we can decompose the problem of acting in a partially observable environment into two components: a *state estimator*, which takes as input the last belief state, the most recent action and the most recent observation, and returns an updated belief state; and a policy, which now maps belief states into actions.

The state estimator can be constructed out of T and O by straightforward application of Bayes' rule. The output of the state estimator is a belief state, which can be represented as a vector of probabilities, one for each environmental state, that sums to 1. The component corresponding to state s', written $\mathrm{SE}_{s'}(b, a, o)$, can be determined from the previous belief state b, the previous action a, and the current observation o as follows:

$$\mathrm{SE}_{s'}(b, a, o) \quad = \quad \frac{O(a, s', o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s)}{\Pr(o \mid a, b)}$$

where $Pr(o \mid a, b)$ is a normalizing factor. The resulting function ensures that the current belief state accurately summarizes all available knowledge.

4 Constructing Optimal Policies

The key to finding optimal policies in the partially observable case is that the problem can be cast as a *completely observable* continuous-space MDP. The state set of this "belief MDP" is $\mathcal{B}$ and the action set is $\mathcal{A}$. Given a current belief state b and action a, there are only $|\mathcal{O}|$ possible successor belief states b', so the new state transition function, τ, can be defined as

$$\tau(b, a, b') = \sum_{o \in \mathcal{O} \text{s.t.} \mathrm{SE}(b,a,o) = b'} \Pr(o \mid a, b) \ ,$$

where $\Pr(o \mid a, b)$ is defined above. The reward function, ρ, is constructed from R by taking expectations according to the belief state; that is,

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

The belief MDP is Markov [18], that is, having information about previous belief states cannot improve the choice of action. Most importantly, if an agent adopts the optimal policy for the belief MDP, the resulting behavior will be optimal for the partially observable process. The remaining difficulty is that the belief process is continuous; the established algorithms for finding optimal policies in MDPs work only in finite state spaces and the existing exact POMDP solution procedures are computationally intractable [14]. A number of algorithms exist for solving the belief

MDP [19, 3, 11, 9], but even the most efficient of these would do well to solve problems with 10 states and 10 observations.

5 Heuristic Control Strategies

Since it is computationally intractable to compute the optimal POMDP control strategy for all but the simplest environments, we consider a number of simple heuristic control strategies, some of which are fairly *ad hoc*, and others with more principled motivations.

5.1 Belief Replanning

We have implemented a strategy, *replan*, that is a slight variation on the strategy used in Dervish [13]. The algorithm starts by assuming it is in the most likely world state and planning a path to the goal in a deterministic idealization of the domain. In addition, it generates a sequence of predicted world states that will be traversed if the nominal trajectory is followed. The robot then embarks upon this plan, updating its belief state and checking, at each step, that the most likely world state (according to the belief distribution) is equal to the predicted state. If it is not, the cycle begins again by planning from the current most likely state.

The main difference between our implementation and the original approach is that, rather than computing the Bayesian belief state, the original approach used a model of observational uncertainty only (no action uncertainty) to compute certainty factors on the world states.

5.2 MDP-Based Algorithms

A POMDP model is an MDP model with probabilistic observations. In the MDP-based algorithms of this section, we make use of the solution to the underlying MDP, which can be obtained quickly for even very large models.

The *most likely state* (MLS) policy finds the world state with the highest probability and executes the action that would be optimal for that state in the MDP: $\pi_{\text{mls}}(b) = \pi^*(\text{argmax}_s\, b(s))$. The only real difference between this method and the belief-replan method is the use of the optimal MDP policy instead of a plan based upon a simplified deterministic version of the environment. These may be very similar in the current domain, but if there are potentially dangerous consequences to some actions (such as falling down a staircase) the MDP will take them into account.

In the *voting* method, we start by computing the probability (according to the belief distribution) that each action is optimal:

$$w_a(b) = \sum_{s \in \mathcal{S}} b(s) I(\pi^*(s) = a) \enspace ,$$

where I is an indicator function, with value 1 if the argument is true and 0 otherwise. Then, we choose the action that is most likely to be optimal: $\pi_{\text{vote}}(b) = \text{argmax}_a\, w_a(b)$. This is similar to the action-choice method used in Xavier [17]. The only difference is that they used a deterministic planning algorithm to choose the best action for each world state, rather than finding the best action in the underlying MDP.

The Q_{MDP} method [8] is a more refined version of the voting method, in which the votes of each state are apportioned among the actions according to their Q value:

$$\pi_{Q_{\text{MDP}}}(b) = \underset{a}{\text{argmax}}(\sum_s b(s) \cdot Q(s,a)) \enspace .$$

This is in contrast to the "winner take all" behavior of the voting method. This technique would be optimal if the agent's uncertainty existed for only a single step.

5.3 Using the Degree of Uncertainty

In general, it is useful for a robot's actions to depend on its degree of uncertainty. The optimal policy for a POMDP model seamlessly integrates the two concerns of acting in order to reduce uncertainty and acting in order to achieve the goal. Since it is infeasible to solve the POMDP directly, the algorithms of this section explicitly trade off these two concerns.

Both of our methods consider a myopic strategy for taking actions to gain information. The entropy of a probability distribution b is $H(b) = -\sum_{s \in \mathcal{S}} b_s \log b_s$, where $b_s \log b_s = 0$ when $b_s = 0$. The lower the value, the more certain the distribution. When the robot is confused, the entropy is high, and it will be reasonable to take actions to reduce the entropy of the belief state. We define the expected entropy of a belief state, b, and action, a, as

$$EE(a,b) = \sum_{b'} \tau(b,a,b') H(b') \enspace ,$$

where the summation is over all possible next belief states. It is possible to consider the expected entropy resulting from longer sequences of action, but it quickly becomes quite expensive to compute.

A simple strategy would be to follow one of the MDP-based policies as long as the entropy of the belief state is not too high, then act explicitly to reduce the entropy using the myopic strategy of taking the action that most reduces expected entropy of the next belief state. In some cases, however, there is benign entropy in the belief state; that is, there is confusion among states that require the same action. In such cases, there is no reason to try to reduce the entropy. For this reason, we compute the entropy of the action-optimality distribution, $w_a(b)$, instead. If this entropy is above some threshold ϕ, then there is a genuine confusion about which action to take and we seek to reduce it. Thus, the *action entropy* policy is

$$\pi_{\text{AE}}(b) = \begin{cases} \text{argmax}_a\, w_a(b) & \text{if } H(w(b)) < \phi \\ \text{argmin}_a\, EE(a,b) & \text{otherwise} \end{cases} \enspace .$$

In the *entropy-weighting* (EW) approach, rather than exclusively pursuing the goal or exclusively trying to reduce entropy, the policy weights these two aims using a normalized form of the entropy. Define the *normalized entropy* of a belief state as $\tilde{H}(b) = (H(b)/H(u))^k$, where u is the uniform belief state (which has maximum entropy) and k is a factor used to control the relative weighting of the entropy. This yields a measure between 0 and 1. If the robot had certain knowledge that it was in world state s and would maintain its certainty into the future, then the

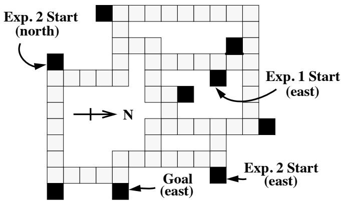

Figure 1: Synthetic office environment A.

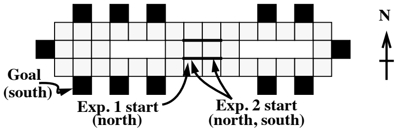

Figure 2: Synthetic office environment B.

long-term value would be $V(s)$, because it could take appropriate actions. Note that $V \cdot b$ is an upper bound on the value of being in belief state b. On the other hand, if the robot were completely confused, then its value would be considerably lower. One way to estimate this is to consider the value of a state, $V^L(s)$ resulting from performing some fixed sequence of actions designed to disambiguate the state, then performing optimally thereafter. While these values may still be too optimistic, they provide a truer picture of the value of being confused. We used a sequence of length 20, which consisted of a repetition of 5 *move-forward* actions followed by a single *turn-left* action. This sequence was based upon the homing sequence [6] for a determinized version of our domain. We define the weighted entropy value as

$$EV(b) = \tilde{H}(b')(b' \cdot V^L) + (1 - \tilde{H}(b'))(b' \cdot V) \ ,$$

and the associated Q value as

$$EQ(b,a) = \rho(b,a) + \gamma \sum_{b'} \tau(b,a,b') EV(b') \ .$$

Finally, the policy is $\pi_{\text{EW}}(b) = \text{argmax}_a \, EQ(b,a)$.

6 Simulation Experiments

We first tested these control strategies in simulation on a variety of synthetic office environments, where it is easier to run extensive experiments. We then confirmed these results by running experiments in a model of our real office environment. For the AE algorithm $\phi = 1$, and for the EW algorithm $k = 2$.

Synthetic Environments Figures 1 through 4 show the layouts of four synthetic environments. Each location is represented as four states in the POMDP model, one for each major orientation. The dark locations are rooms, connected to corridors by doorways.

We conducted three separate sets of experiments in each synthetic environment, with different start and goal-state sets (see figures). In the first set of experiments, the robot knows which state it has started in, whereas in the second set there is some small set of similar-looking states where the robot could have started. The last set of experiments have the robot starting off with no information about its starting state. When there is more than one start state for a given experiment, the robot starts randomly in one of the states, with its belief state initialized to the uniform distribution over the set of possible start states.

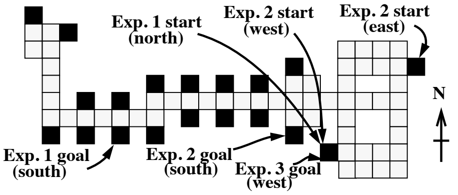

Figure 3: Synthetic office environment C.

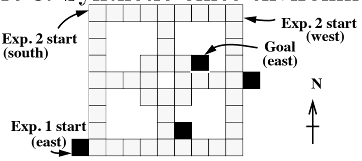

Figure 4: Synthetic office environment D.

In these domains, a reward of 1 is given for performing the action *declare-goal* in the goal state, and 0 for all other state-action pairs. An alternative would be to give reward for entering the goal state. However, navigation is rarely useful for its own sake; the robot will have to perform some further actions when it arrives at the goal and, therefore, has only succeeded if it has arrived at the goal and knows it (or believes it to a high enough degree). Trials are terminated when the *declare-goal* action has been executed or after 300 steps; performance is measured as the discounted sum of rewards ($\gamma = 0.99$) starting from the initial state. The results reported are averages over 250 trials.

Table 3 shows the results of the different methods on the four synthetic office environments when the starting state is known. In addition to the heuristics, we include the results of an *omniscient* agent, which knows the true state at all times. This gives some coarse measure of how hard the problem is, though it should not be confused with what is achievable, even for an optimal policy. We performed a T-test to determine whether or not there were significant differences between the algorithms. The best performance is outlined darkly and those that are not significantly worse are lightly outlined.

Although there is some variability between the methods, for the most part all of them do well. Since each is an approximation method, there are particular circumstances where they can be made to fail. The data points where one of the methods appear significantly worse than the others are examples of such circumstances. However, the MLS method is fairly robust across the environments, showing that not much more than the information in the belief state is needed

Algorithm	A	B	C	D
MLS	0.642	$\boxed{0.749}$	$\boxed{0.662}$	0.791
Voting	0.639	0.704	0.612	0.800
Q-MDP	$\boxed{0.662}$	0.743	0.452	$\boxed{0.825}$
Replan	0.625	$\boxed{0.768}$	$\boxed{0.660}$	0.773
EW	0.650	$\boxed{0.777}$	0.464	$\boxed{0.821}$
AE	0.642	0.742	0.605	0.802
Omniscient	0.677	0.846	0.756	0.836

Table 3: Experiment 1: Known starting state.

Algorithm	A	B	C	D
MLS	0.695	$\boxed{0.639}$	$\boxed{0.779}$	0.791
Voting	0.669	0.000	0.737	0.791
Q-MDP	$\boxed{0.704}$	0.000	$\boxed{0.788}$	$\boxed{0.853}$
Replan	0.654	$\boxed{0.649}$	$\boxed{0.773}$	0.787
EW	$\boxed{0.720}$	0.000	0.742	0.811
AE	$\boxed{0.712}$	0.000	0.737	0.809
Omniscient	0.728	0.848	0.845	0.878

Table 4: Experiment 2: Multiple possible start states.

Algorithm	A	B	C	D
MLS	$\boxed{0.630}$	0.615	$\boxed{0.601}$	$\boxed{0.729}$
Voting	$\boxed{0.599}$	0.570	0.257	0.648
Q-MDP	0.405	0.502	0.308	0.574
Replan	$\boxed{0.618}$	0.602	$\boxed{0.626}$	0.690
EW	0.465	0.473	0.252	0.546
AE	$\boxed{0.608}$	$\boxed{0.679}$	0.373	0.671
Omniscient	0.750	0.868	0.853	0.898

Table 5: Experiment 3: Uniform starting belief.

to do exceedingly well in this situation.

The next situation we address is when the agent is not certain of its starting state. In the experiments shown in Table 4, there are two possible starting states (four possible states for office B) that are similar in their immediate surroundings.

This is the first case where we see some of the methods performing poorly. The Q-MDP and voting methods are never able to reach the goal in the office B experiment. This is because they cycle through the same set of actions without making any progress toward the goal. This cycling behavior is not always present, as can be seen by their performance in the other environments. In fact, the Q-MDP method is significantly better than the others in the office D experiment, where that particular configuration of starting states and goal state allows it to behave nearly as well as the omniscient method.

A problem with selecting the voting or Q-MDP methods is that it is not easy to know *a priori* if the particular environment will bring out the best or the worst in these algorithms. The MLS method performs well across all of these situations and would be the prefered choice unless something more was known about the particular problem instance.

The final situation we explore is the most difficult: what if the robot is equally likely to start in any of the states? In this situation, its initial belief distribution is uniform over all states. Table 5 shows the results of applying the algorithms to this situation. These results show that the Q-MDP and voting methods do not usually perform well when the uncertainty is high. The MLS method is still quite robust and suggests that for moderately noisy environments, it is the best choice across different types of environment layouts and starting beliefs.

These results also show that the AE method is a useful enhancement to the voting method. It is never significantly worse than the voting method and it can sometimes outperform all of the other methods. We believe that entropy-based methods have even more merit when the action set contains explicit information gathering actions. Unfortunately, the models we used for these experiments did not have a very rich set of actions, so no one action would provide significantly more information than another.

Noisier Synthetic Environments The navigation problem becomes harder as the noise in action and observation increases. In order to gauge the effects of noise on the various methods, we repeated the experiments above using the more noisy action and observation probabilities shown previously in Tables 1 and 2. Space precludes the inclusion of the full data set, so we merely touch upon the results.

The Q-MDP method was universally bad across all configurations and starting beliefs. In none of the trials did it ever declare itself to be in the goal. This is a direct result of Q-MDP's assumption that it will be completely disambiguated on the next step. Since the noise is high, it will never have a very confident belief that it is in the goal. It would prefer to delay declaring the goal by doing one more action in hopes of knowing where it will be after that action.

The belief replan method, though not as bad as Q-MDP, was inferior to the MLS and voting algorithms. Again, the MLS method was the best choice in these environments. Note that the entropy-based methods are not of much help in these noisy environments. Since all actions and observations have a fair amount of noise, no action is likely to significantly reduce the entropy.

Real Office Environment Figure 5 shows the layout of our robot's environment, which has 1052 POMDP states. This layout with the standard noise model was used both in simulation and on the actual robot. In these experiments we used 7 different starting belief state configurations. For all of these experiments, the goal state was the same, location G facing east. The first three experiments were for a known starting location, (A-east, B-east and C-south), roughly corresponding to differing physical distances from the goal. The next three experiments used a starting belief over two locations; for each of the previous starting states we added a state, (D-east, E-east and F-north respectively), roughly the same distance and with similar immediate observations. The final experiment was conducted with a uniform starting belief over all states.

Table 6 shows the simulation results for 100 trials. These results are consistent with the synthetic office A layout, which is a simplified version of this real environment. Since the entropy-based methods were not likely to be significantly better than the other methods, and since runs on the robot are quite time consuming, we did not use the EW or AE methods in either the simulated or actual robot experiments.

7 Robot Experiments

In order to verify the simulation results, we duplicated the experiments from Table 6 (with the exception of experiment 7) on a real robot running in the unmodified Computer Science department (Figure 5). Due to the time required for the robot to complete an experimental trial, only 108 trials were run. Three trials per algorithm were run for experiments one, two

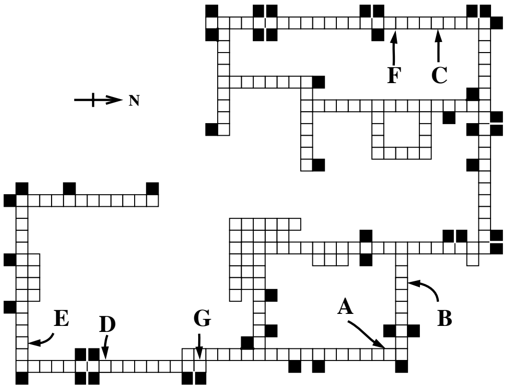

Figure 5: Real office environment.

	Experiment						
Alg.	1	2	3	4	5	6.	7
MLS	.77	.72	.45	.78	.76	.39	.50
Voting	.69	.72	.34	.71	.78	.21	.54
Q-MDP	.80	.73	.45	.82	.76	.43	.41
Replan	.74	.70	.44	.81	.71	.37	.48
Omnisc.	.81	.75	.49	.86	.78	.48	.65

Table 6: Simulations of robot office environment.

and three while three trials per starting location were run for each algorithm in experiments four, five and six. The average discounted reward for each algorithm is shown in Table 7.

As in simulation, all four algorithms proved adequate when the starting state was known (1,2,3). Each reached the goal rather directly on every trial despite occasional errors in observation or action. In experiments where the starting belief state had a bimodal distribution (4,5,6) only MLS reached the goal on every trial. In these experiments both Q-MDP and Replan at least once issued the *declare-goal* action in a state which was near and similar to the goal, but in fact not the goal. A larger number of trials might reveal the same fallibility in MLS. In experiment 6 the appropriate initial actions of the possible starting states are at odds, and all algorithms spend up to the first half of the required actions collapsing the belief state to the actual state then making a direct run to the goal. In simulation, experiment 7 tested performance from a uniform belief state and 100 randomly chosen start states for each algorithm. Since this was not practical with the robot, a few informal runs were made to ensure that MLS could succeed in reaching the goal state when given a uniform initial belief state.

Voting performed significantly worse in reality than our simulation suggested it might. Voting can lead to cycles of belief states and actions that do not make progress toward the goal. This behavior was not as pronounced in simulation because persistent cycles are less likely. The simulator models the pilot as the navigator does, with action and observation errors determined by a stationary probability distribution which is independent of the robot's real pose in the world. Most cycles of belief states and actions are soon disrupted by a failed action or erroneous observation. In reality, the likelihood of errors by the robot is dependent upon the its current pose in the environment. For some configurations and sets of actions the robot is extremely reliable, allowing an action/observation cycle and the associated belief state cycle to persist indefinitely. The initial configurations of experiments 4 and 6 reliably cause the voting algorithm to enter a cycle of in-place turns which persists for at least hundreds of actions. The simulator's strict adherence to the simplified probability model can also cause the simulated omniscient controller to encounter more errors, leading to worse performance for simple tasks than other algorithms running on the robot.

	Experiment					
Alg.	1	2	3	4	5	6
MLS	.83	.77	.49	.84	.78	.31
Voting	.86	.77	.50	.00	.77	.00
Q-MDP	.83	.78	.46	.78	.76	.25
Replan	.84	.75	.30	.70	.39	.30

Table 7: Experiments on robot.

8 Conclusions

By accepting the inevitable errors in low-level piloting, we can account for them and recover using a robust high-level navigation strategy. Although the optimal control strategy is not easily computed, some well motivated heuristic strategies perform better than *ad hoc* ones. This is only a very preliminary study and a great deal of work remains to be done.

All of the control strategies we have explored here are essentially myopic with respect to uncertainty. None of them is able to take a long string of actions in order to disambiguate its belief state. We are beginning to apply some more sophisticated methods from the machine learning literature [8, 15] that find approximations to the true value function of the POMDP. We expect that they will improve performance when the starting location is extremely uncertain.

The simple domains explored in this paper do not exercise the abilities of the POMDP models to control active perception. A domain in which the robot has "perceptual" actions that perhaps have an associated cost would be appropriately controlled with this model. In future work, we will extend the set of actions to include some visual operations that will provide better information.

One of the big shortcomings of modeling the environment with POMDP models is that there is too much dependence on the world being static. While the system can cope with dynamic events by incorporating these events into the noise model, it can only deal with transient changes. If a fundamental change in the layout occurs (e.g., a door closes), the robot's performance could deteriorate rapidly, especially if there are many such changes. Another line of future work is to learn the world model from experience using techniques adapted from hidden Markov models; this is currently being pursued (for learning the probabilities, but not the topology) by Koenig and Simmons [5].

Acknowledgments

Hagit Shatkay implemented an algorithm to find homing sequences. This work was supported in part by NSF grants IRI-9453383 and IRI-9312395.

References

[1] J. Buhmann, W. Burgard, Cremers A., D. Fox, T. Hofmann, F. Scheider, J. Strikos, and S. Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–37, Summer 1995.

[2] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

[3] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, British Columbia, Canada, 1988.

[4] Richard Duda and Peter Hart. Use of the Hough transform to detect lines and curves in pictures. *Graphics and Image Processing*, 15(1):11–15, 1972.

[5] Sven Koenig and Reid Simmons. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[6] Zvi Kohavi. *Switching and finite automata theory.* McGraw-Hill, New York, N.Y., 1978.

[7] J.J. Leonard and Hugh Durrant-Whyte. Localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(6), 1991.

[8] Michael Littman, Anthony Cassandra, and Leslie Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 362–370, San Francisco, CA, 1995. Morgan Kaufmann.

[9] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. An efficient algorithm for dynamic programming in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, Providence, Rhode Island, 1995.

[10] William S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[11] George E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[12] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 19–24, 1985.

[13] Illah Nourbakhsh, Rob Powers, and Stan Birchfield. Dervish: An office-navigating robot. *AI Magazine*, pages 53–60, Summer 1995.

[14] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[15] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1088–1094. Morgan Kaufmann, 1995.

[16] Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, New York, 1994.

[17] Reid Simmons and Sven Koenig. Probabilistic navigation in partially observable environments. In *Fourteenth International Joint Conference on Artificial Intelligence*, pages 1080–1087, Montreal, Canada, 1995. Morgan Kaufmann.

[18] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[19] Edward J. Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, Stanford, California, 1971.

[20] Chelsea C. White, III. Partially observed Markov decision processes: A survey. *Annals of Operations Research*, 32, 1991.