Robust Navigation Execution by Planning in Belief Space

Bhaskara Marthi
Willow Garage, Inc.
Menlo Park, CA 94025
Email: bhaskara@willowgarage.com

***Abstract*—We consider robot navigation in environments given a known static map, but where dynamic obstacles of varying and unknown lifespans appear and disappear over time. We describe a roadmap-based formulation of the problem that takes the sensing and transition uncertainty into account, and an efficient online planner for this problem. The planner displays behaviors such as persistence and obstacle timeouts that would normally be hardcoded into an executive. It is also able to make inferences about obstacle types even with impoverished sensors. We present empirical results on simulated domains and on a PR2 robot.**

I. Introduction

Navigation given a known map is a fundamental capability for mobile robots, and there exist practical and efficient methods for subproblems such as localization and path-planning. Less well-understood is how to put these pieces together such that the overall system acts robustly in the world. Define a navigation system to be a program that takes in a known static map, localization information, sensor data, and navigation goals (represented as positions in space), and controls lower level systems (say via velocity commands of the form $(\dot{x}, \dot{y}, \dot{\theta})$). Given a correct map of a static indoor environment (i.e., with no new obstacles besides the ones in the map), a navigation system is easy to write. For example, we could take localization input from the AMCL algorithm [24], call a planner such as A* [21] on a discretized grid to produce a path, then use a trajectory following algorithm such as DWA [1] to generate velocity commands that will follow that path. An autonomous robot in an unconstrained real-world environment such as an office must, however, also deal with various sorts of changes to the map. A person might be standing in a doorway, a couch may have temporarily been moved into a corridor, rendering it impassable, and so on. Navigation systems should behave efficiently and sensibly given that such dynamic obstacles appear (and disappear) over time.

A standard approach is to maintain a map that is updated in some manner given obstacles. Getting this to work robustly is surprisingly tricky. Consider the prototypical example shown in Figure 1. Here the shortest way to the goal is to go down the narrow hallway. The next best alternative is to go around the building, which takes ten times as long. Now suppose a set of obstacle points appear in the hallway at position A, making it impossible to traverse. Certainly, the robot should not stand and wait forever. The map must therefore be updated with this obstacle, causing the robot to eventually choose the long path.

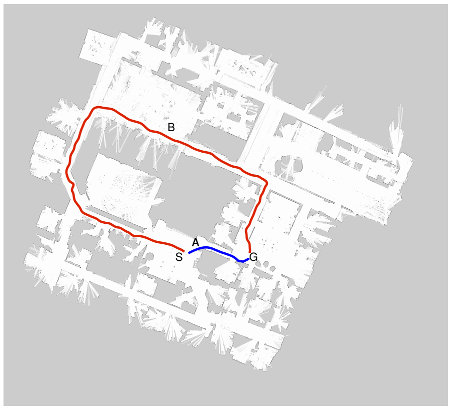

Fig. 1. An example navigation problem. The goal is to get to G from S.

Also, the robot must not suddenly change its mind when it is, say, at position B, and turn back around. This requires either making obstacles fairly long-lived, or hardcoding "persistence" of some sort into the system. In the first case, there needs to be some scheme for eventually timing out obstacles, to avoid the possibility, e.g., of a corridor being considered permanently out of bounds due to seeing a person blocking it once. Finally, the algorithm should deal intelligently with different types of obstacles. It might, for example, make sense to wait for a person to move, but not for a couch. Ideally, perception would give us this information. But even if, as is currently the norm, perception is fairly impoverished/noisy, there is still the possibility of a kind of *implicit sensing* of the obstacle type: in the example, it might make sense to just wait for a few seconds. If the obstacle disappears, the robot can take the short path after all. If it stays where it is, it is likely to be static and the robot should take the long route.

The usual way to achieve robust and efficient behavior in such cases is to have an program known as an executive sitting above the planners [16]. The executive maintains its own local state and contains various hand-coded procedures, which are intended to avoid infinite loops and dead-ends and to appear intelligent and goal-directed. As a typical example, we consider the executive from the ROS navigation stack [11], a popular open source navigation system. Table I shows the set of local state variables maintained by this executive. The

State variable	Meaning
`mode`	Overall mode: planning, controlling, or clearing
`time`	The current time
`last-valid-plan`	Timestamp when planning last succeeded
`last-valid-control`	Timestamp when a valid velocity last found
`oscillation-pose`	Reference pose used to detect oscillations
`oscillation-timestamp`	Timestamp for oscillation pose
`recovery-index`	Which recovery behavior was last tried
`recovery-cause`	Cause of last failure. One of `no-valid-plan`, `no-valid-control`, or `oscillation`

TABLE I
STATE VARIABLES OF THE EXECUTIVE IN THE ROS NAVIGATION STATCK

idea is that the overall system can be in one of three modes. In "planning" mode, the robot is stationary and waiting for a plan. In "controlling" mode, the robot has a plan and is sending velocity commands at some rate. Finally, in "clearing" mode, something has gone wrong, and one of a user-specified set of recovery behaviors is being executed. These may include removing obstacles from the map beyond some distance, or rotating in place. Several other variables govern the transitions between these modes, as shown in Table I.

The ROS navigation system has achieved impressive performance in real-world demonstrations, such as navigating with no human intervention for eleven days in an office environment. But there are a few disadvantages to using a complicated and stateful executive. First, there is significant programmer effort involved in coming up with the execution strategy, which is usually done in a trial-and-error fashion. Second, understanding the system's behavior requires knowing all the state variables in Table I and their interactions with each other and with the recovery behaviors. This makes it hard to debug or modify. It also means the planning algorithms have an inaccurate model of how their plans will be executed. Finally, the executive is not capable of certain types of behavior, such as information-seeking actions.

We propose an alternative way of structuring navigation systems. The idea is to explicitly model and plan for the uncertainty in the world, then use a simple stateless executive that just follows the resulting plan. There are many ways to formulate the planning problem, some of which we discuss in Section III. One possibility is to model the transition uncertainty with costs or probabilities, while still pretending that the state is fully observable. These approaches are systematically suboptimal, though, as they will not take information-gathering actions. Alternatively, we can explicitly model the sensing uncertainty, using a partially observable Markov decision process (POMDP). POMDPs are notoriously difficult to solve, however, and the POMDP in our case has size exponential in the number of possible obstacle locations.

In Section III, we present a particular formulation of the problem as a POMDP, and an algorithm for solving it efficiently enough to run online on a mobile robot. The formulation is based on a roadmap over the static map, where nodes represent particular locations and edges are local paths that can become blocked and unblocked over time. Obstacles may belong to different classes, with varying lifetimes. As we will show, the optimal policy for this POMDP automatically exhibits the various behaviors discussed above, such as persistence, patience, and implicit sensing, without having to hardcode them into an executive.

The scientific contributions include efficient algorithms for state-updates (Section IV) and planning (Section V) in this POMDP. At the systems level, the main contribution is showing by example that it is possible to build a robust navigation system with failure recovery using a principled decision-theoretic planning approach, rather than scripted behaviors. Section VI describes the implemented system and empirical results on both simulated and real-world domains.

II. BACKGROUND

A. MDPs

An undiscounted Markov decision process [17], or MDP, consists of a state space S, action set A, transition model $P(s'|s,a)$, reward function $R(s,a,s')$, and terminal states $T \subset S$. A stationary policy is a function $\pi : S \to A$. A policy induces a distribution over state-action trajectories that continue until reaching a terminal state. The value function of a policy $V^{\pi}(s)$ is the expected total reward for following π starting at s, and the Q-function $Q^{\pi}(s,a)$ is the expected total reward for doing a in s, then following π. The optimal policy π^* maximizes the value at all states, and we write V and Q for its value and Q-functions. In our examples, the set of actions varies depending on the state, but this can be represented by making nonapplicable actions have reward $-\infty$.

B. POMDPs

A partially observable Markov decision process [6], or POMDP, is like an MDP except that states are not directly observed. Instead, there is an observation distribution $Z(o|s,a,s')$ over the observation that's received when making the transition from s to s' via a.

In the context of POMDPs, we call a distribution over the state space a *belief state*. Given a belief state b about the current state s, if we do an action a, the marginal distribution over the next state is the result of the projection operator $b' = \mathcal{P}(b,a)$ where:

$$b'(s') = \sum_{s \in S} b(s) P(s'|s,a)$$

Also, given an observation o, the conditional distribution is $b'' = \mathcal{C}(b',s,a,o)$ where:

$$b''(s') = \frac{b'(s')Z(o|s,a,s')}{\sum_{s' \in S} b'(s')Z(o|s,a,s')}$$

The filtering operator $\mathcal{F}$ consists of projection followed by conditioning, and is used to update the distribution over the current state. A key fact about POMDPs is that the sequence of beliefs itself forms an MDP (with the same action space). To sample from the transition model of this MDP given belief b and action a, first sample s from b, then sample s' from $P(\cdot|s,a)$ and o from $Z(\cdot|s,a,s')$, and set $b' = \mathcal{F}(b,a,o)$ (s' is discarded). The reward function is $R(b,a,b') = \sum_{s,s'} b(s)P(s'|s,a)R(s,a,s')$. The belief state summarizes the relevant information about the action–observation history; in particular, any optimal policy for the belief state MDP is also optimal for the POMDP, assuming the belief state is maintained exactly.

C. Solution algorithms

The literature has tended to focus on the *offline planning* problem of finding a full policy π over the belief space. Due to the constraints of running on a robot in real-time, we consider instead the *online planning* problem [20] where the agent is repeatedly given an observation and just returns an action for the current belief state. Most online algorithms are based on *forward search*. A forward search tree for (belief state) MDPs consists of alternating layers of action nodes and chance nodes. The root of the tree is an action node labelled with the initial (belief) state. An action node has children corresponding to the possible actions. The chance node corresponding to a state-action has children corresponding to the possible successor states, labelled with the probability of that particular state. A search tree can be used to estimate the value of taking each action at the root by repeated *backups*. The leaf action nodes are given values according to some heuristic estimate of the value function at their state. The value of a chance node is the average of the child values weighted by the probabilities. The value of an action node is the maximum of the child values.

There are various choices in how to generate search trees. First, the children of a chance node can be generated either exhaustively, based on an explicitly given transition model of the MDP, or indirectly, by sampling repeatedly from it, which only requires a simulator. Second, there is the choice of which nodes to expand, given finite total computation time. Apart from simple fixed-depth strategies [7], there are various more sophisticated methods [20, 9]. Finally, the choice of evaluation function at the leaves is of key importance, especially when the tree depth is much lower than the expected time to termination.

III. Modeling the problem

Our goal is to build a robotic system that navigates between locations in an indoor environment efficiently. We now consider several ways to model this as a planning problem, in order to motivate our chosen formulation in Section III-E. All models will be defined with respect to a *roadmap* over the static map. This is generated as follows from a known static map: first, waypoints are sampled from the free space. This can be done using simple uniform tiling sampling, or more intelligently, e.g., based on the Voronoi graph of free space [3]. The main constraint is that from every point in free space, there is a path of length $l < R$ leading to a waypoint, where R is a fixed radius threshold. Next, given the waypoints, standard path planning is run offline on pairs of nearby waypoints to generate edges. If obstacles are not considered, this graph can be used for planning by adding the start and goal positions to the graph, then searching for a path between them. When the robot is at a waypoint, it does a quick local reachability check to the neighbors in the graph. This process acts as a

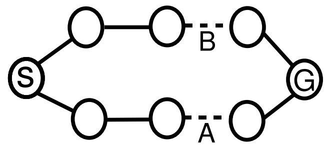

Fig. 2. Problem in which two paths are possible, but edge A was more recently observed blocked than edge B.

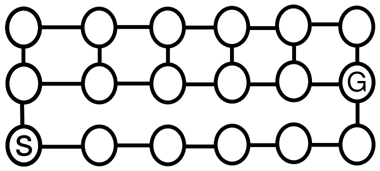

Fig. 3. Problem in which the top path is better because there is more scope to move around obstacles.

deterministic virtual sensor [10] that allows us to know the state of all edges incident to the node of the graph the robot is currently at.

A. Deterministic

A simple scheme for updating the roadmap is to maintain a list of blocked edges. An edge is added to this list whenever it is observed blocked and removed when it is observed free. An immediate problem is that the robot can eventually get into a situation where no path exists. To avoid getting stuck, whenever a path cannot be found, all edges that are not currently observed blocked are unblocked. If a path still cannot be found, a wait action is chosen.

This scheme does not take any account of the likelihood of an edge being blocked, or of the relative cost of alternative paths. It therefore makes various kinds of systematic errors.

Example 1: (Block probabilities) In Figure 2, there are two paths, both blocked, but edge A has been observed blocked much more recently than edge B. It therefore makes sense to take the top path.

Example 2: (Prediction) In Figure 3, no edges have been observed blocked. It nevertheless makes sense to take the top path, because a single blocked edge on that path can be circumvented, while a single blocked edge on the bottom path requires going back to the start.

Example 3: (Patience) In Figure 4, if an obstacle is observed on the edge between S and G, it might make sense (assuming a dynamic model of obstacles) to wait rather than taking the longer path, since the obstacle could disappear.

In each of these cases, the deterministic algorithm will choose the wrong action either always or often.

B. Deterministic with blocked-edge costs

Rather than viewing blocked edges as completely impassable, we could give them an extra cost proportional to their probability of being blocked. In other words, given an edge such that it was last observed T seconds ago, and it was

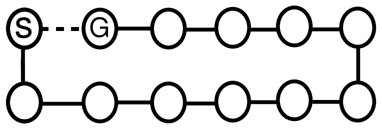

Fig. 4. Problem in which shortest path is currently blocked, but it might make sense to wait and see if it disappears.

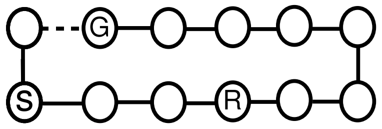

Fig. 5. Problem in which it may make sense to resense the obstacle before committing to the longer path.

blocked at that time, its cost is $c(e) + e^{-aT}B$. It therefore deals correctly with Example 1. It still fails on Examples 2 and 3. While this algorithm takes more account of uncertainty, it still neglects the fact that actions can add information.

Example 4: (Value of information) In Figure 5, suppose the robot is at S, and the short path to G has been recently observed blocked. Given the length of the long path, it might still make sense to check again if the path has become free before deciding on the long path. It is not possible to achieve this in general by adjusting the a and B parameter, for that would prevent the robot from ever considering the long path.

Having time-varying costs can also lead to another problem.

Example 5: (Persistence) In Figure 5, suppose now that the robot is at R, proceeding down the long path which was chosen because the short path to G from S had a high initial probability of being blocked. The probability of being blocked will decrease exponentially though, so it is possible that the edge's blocked cost decreases enough that the robot will stop in the middle of the path and go back down the other way, which leads to behavior that is suboptimal (and looks strange).

C. Most likely state

A related method is to maintain a probability distribution over the true state of the world and plan assuming the most likely graph. As above, this method fails to take sensing actions in Example 4.

D. MDP

Rather than using costs, we can model the uncertainty using an MDP. A straightforward way to do this would be to have each edge's status be sampled afresh each time the robot is adjacent to it. The problem this runs into is that, since the edges have no hidden state, a behavior such as "wait for 10 seconds, then choose another path if this edge is still blocked" would never be followed in cases like Example 4 — the robot would either leave immediately or wait forever.

E. POMDP

POMDPs model both the transition and sensing uncertainty in the domain. We use the following POMDP model:

- There is one state variable for the current position in the roadmap and, for each edge a status, which can either be free or blocked. In the latter case it belongs to one of a predefined set of classes. In our examples, the obstacle classes are `temporary`, `person`, and `static`.
- The actions at a state are to take one of the outgoing edges from that node, or to wait for 3 seconds at the current position. [1]
- The transition model is that a move succeeds iff the edge is not blocked (at the start of the move). If so, it has a duration depending on the edge length, and the robot ends up at the other node incident to the edge. Additionally, each edge status evolves according to an independent continuous time Markov chain. The parameters are:
  - the *block rate* (expected time till an obstacle appears)
  - the prior distribution over obstacle classes;
  - for each obstacle class, the *unblock rate*, or expected time till it disappears.
- The observation model is that the current position is known with certainty and, for each adjacent edge, the robot observes whether it is free or blocked (but not which class the obstacle belongs to).
- The cost of an action is the time it takes.

Optimal solutions to (instances of) this POMDP avoid the problems listed above. For example, in Figure 5, the belief will include a distribution on whether the obstacle is static, temporary, or a person. If the prior probability of being temporary or a person is high enough, the optimal plan will be to move to the blocked edge and wait for some amount of time. If the path clears, take it. If not, then over time the probability of being a static obstacle will increase. This falls out of the Bayesian updating formula: $P(E|h) \propto P(E)P(h|E)$ where h is the observation history and E is the event that the obstacle is static. Even if the prior probability $P(E)$ is low, the longer the robot waits without the obstacle moving, the higher the likelihood term $P(h|E)$ becomes, until it eventually becomes optimal to take an alternate path.

IV. State Estimation

A belief update given belief b, action a, and observation o, consists of a projection through a followed by conditioning on o. In our case, the transition model is:

$$P(s'|a, s) = I_{\{s'_p = f_p(s,a)\}} \prod_{(u,v)\in E} P(s'_{uv}|s_{uv}, t(s, a))$$

where:

- $f_p(s, a)$ is the deterministic transition function of position that results in moving to the other incident node of a if the move is legal, and staying at the current position for

[1]A 3 second interval was chosen as the minimum reasonable wait time in our environment; longer waits can be obtained by repeating the action.

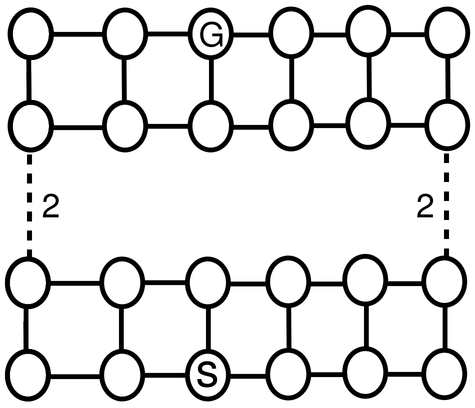

Fig. 6. Original problem. Dashed edges have length 2 and have block probability above the threshold. All other edges have length 1, and have block probability below the threshold.

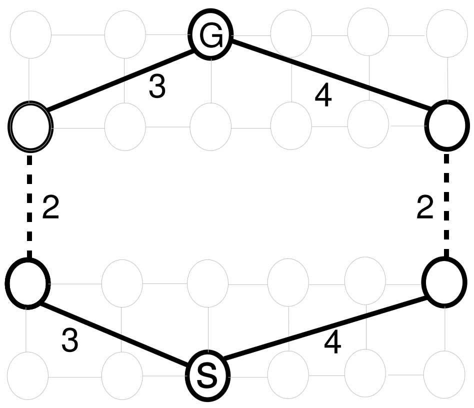

Fig. 7. Abstracted version of problem in Figure 6. In the initial belief state, dashed edges are possibly blocked, and all others are definitely free.

1 second otherwise. Wait actions also result in staying at the current position.

- $t(s,a)$ is the duration of the action, which is just the edge length.
- $P(s'_{uv}|s_{uv},t)$ is given by the continuous time Markov chain transition distribution

We represent our belief states in factored form, as consisting of a known position b_p, and a distribution b_{uv} over each edge's status. Given such a belief, since the transition model above factors over terms, each of which depend on one of the belief variables above, the projected distribution $\mathcal{P}(b,a)$ will also have this factored form. Similarly, the observation model is deterministic, and can be written:

$$Z(o|s,a,s') = \prod_{(u,v)\in E} I_{\{o_{uv}=g(s'_{u,v},s_p)\}}$$

where the function g returns `free` or `blocked` if s_p equals u or v, and `unobserved` otherwise. Since the position is known, the update can once again be done in factored form: for edges adjacent to the current position, if the edge is observed free the conditional distribution is `free` with probability 1, and if it is observed blocked, we make the probability of it being free 0 and reweight the remaining statuses to sum to 1.

V. Efficient Planning

The state space of the POMDP model is exponential in the number of edges, which is beyond the capabilities of current general-purpose offline planners. Online planners based on forward search do not directly depend on the state space size, but do have exponential dependence (for a fixed bound on error) on the search horizon, which can be large in our case, since plan sizes can be on the order of the graph diameter.

A. The Abstract Graph

We can take advantage of structure in the problem. Figure 6 shows an example instance. The dashed edges are two locations where the robot has recently observed obstacles. Intuitively, (the nodes incident to) these edges are the important ones to reason about during planning. The remaining nodes (apart from the start and goal) are just intermediate locations we pass through — assuming the block rate is not high, the optimal conditional plan will look something like 1) visit the left dashed edge; 2) if is free, take it; 3) otherwise wait for some number of seconds; 4) if the edge still is blocked, move to the right dashed edge; and so on. Figure 7 shows the an abstracted version of the problem based on this intuition. The edge lengths have been computed using Dijkstra's algorithm in the original graph, implicitly assuming no new obstacles will appear. Note that Figure 7 is still nontrivial to solve optimally because it is partially observable, and obstacles may still appear and disappear on the dashed edges. Nevertheless, it has a shorter effective horizon.

More precisely, define the *abstract graph* G_a given a problem and belief state. First, let B be the set of edges whose probability of being blocked exceeds some threshold (we use $\frac{1+p_\beta}{2}$ where p_β is the stationary probability of an unobserved edge being blocked). Let the *cut graph* be the original graph with edges in B removed. Given current position v_s and goal v_g, the abstract graph has vertices $\tilde{V} = V_B \cup \{v_s, v_g\}$ where V_B are the incident vertices to edges in B. For each edge in B, there is a corresponding edge in G_a with the same length. Also, for every pair of vertices $u,v \in \tilde{V}$, if the distance d_{uv} between them in the cut graph is finite, add an edge in G_a between u and v with length d_{uv}.

We can then construct the *reduced POMDP* based on this abstract graph. The initial belief b_0 of the reduced problem is the same as b except that edges not in B are not present, and the newly added abstract edges are included, and are considered free with probability 1. The unblocking rates are the same as the original problem, but the block rate is 0. The proposed solution algorithm is to plan in the reduced POMDP and follow the first concrete edge of the first action in that plan (this is also computed as part of the Dijkstra search to find the edge costs). The assumption made in the reduced POMDP, that new obstacles do not appear on abstract edges, may not hold. Therefore, we replan after traversing each edge of the original graph, using a new abstract graph constructed based on the latest observations.

B. Analysis

We can analyze the quality of this approximation as a function of the domain parameters. Specifically, let $p = (p_1, \ldots, p_K)$ be the stationary probability distribution for the edge obstacle Markov chains, where p_1 corresponds to no obstacle, and let $e = (e_1, \ldots, e_K)$ be the expected unblocking time for each obstacle type (and $e_1 = 0$).

Theorem 1: Let s and g be nodes such that the optimal expected travel time between them is T. Suppose the robot follows an optimal policy with respect to the abstract graph. Then, the expected time T' to travel from s to g satisfies $T' \leq (1 + p^T e)T$.

Intuitively, the approximation quality depends on how frequent obstacles, especially long-lived ones, are. The bound is loose because in fact, unforeseen obstacles mainly matter in cases like Example 2 where there are bottlenecks and narrow hallways. In practice, the approximation is unlikely to be substantial unless most of the domain consists of narrow paths.

C. Planning with the Abstract Graph

The reduced POMDP still has a large state space, but a shorter horizon, making it a good candidate for forward search. We use simple fixed-depth search with fixed-width sampling, as described in Section II-C. More intelligent strategies [9] would likely improve performance.

Leaves of the search tree are evaluated using the following heuristic function. Given a belief state b, sample some number of graphs; in each graph, compute the minimum of the shortest path distance to the goal and the graph diameter (caching within and across leaf nodes is used to make this efficient), and average the results. Intuitively, this corresponds to taking full account of partial observability until the search horizon, then making a full-observability approximation.

At state nodes of the lookahead tree, since the current position in the graph and neighboring edge statuses are known with certainty, the number of children is bounded by the out-degree of the roadmap. For action nodes, suppose we are at a node corresponding to doing action a after belief state b. A naive way to generate a child would be:

1) Sample a state s from the distribution b
2) Sample a next-state and observation from $P(s', o|s, a)$
3) Set $b' \leftarrow \mathcal{F}(b, a, o)$
4) If there is already a child node corresponding to b', increase its count; otherwise create a new node

This procedure results in a large number of belief state updates in the inner loop. Instead, we use a trick that we have not previously seen in the literature, based on the fact that $\mathcal{F}$ is deterministic given b, a, o: label each outgoing edge with an observation, and only compute $\mathcal{F}$ when the observation has not been seen before. This helps in our setting because the observation space is much smaller than the state space.

VI. EXPERIMENTS

We evaluated the various formulations and the ROS navigation stack on simulated graph instances as well as on a PR2 robot. The parameters for the models (e.g., block rates) were chosen based on the known domain model in the simulated case, and set by hand for the physical robot experiments.

A. Simulated experiments

We first compared several of the approaches described in Section III on a set of simulated instances. There are two metrics of interest: planning time and plan quality, which we model as total execution time [2]. In our evaluations, we bounded planning time and compared the total execution time across algorithms. The scale of our roadmaps is such that traversing each edge typically takes several seconds; we therefore placed a conservative limit of one second of planning time per action, and compared the following approaches:

- DA is the deterministic agent from Section III-A.
- BA1 and BA2 are the block-cost agents from Section III-B, using block costs of 10 and 1000.
- ALA1 through ALA3 are abstract POMDP agents from Section III-E, using search depths 1 to 3, and a sampling width of 100.

We used a set of instances with graph sizes ranging from 20 to 1000. The results are shown in Table II. The POMDP-based algorithms were the best by a significant margin on each domain. Interestingly, ALA2 occasionally did better than ALA3. This phenomenon is known in the search literature as a lookahead pathology [2]. A more intelligent search strategy than uniform-depth search should mitigate this problem [9].

B. Experiments on robotic platform

We also evaluated our algorithm on a Willow Garage PR2 robot. A roadmap was generated offline from a static map of our environment by starting with uniformly spaced nodes, which were then perturbed locally to move away from obstacles. The environment is about 40 by 40 meters, and the roadmap had about 200 nodes. We implemented a bridge ROS node to connect our graph-based system to the continuous state of the robot, as shown in Figure 8. Pose messages from the AMCL localization system are mapped to the nearest node on the roadmap, while actions (edges of the roadmap) are mapped to goal commands for a trajectory planner based on the elastic band algorithm [19]. A local occupancy grid is maintained in the odometric frame, based on laser range readings; when reaching a new node, a shortest path computation is done in this grid (with obstacles inflated by the robot radius) to determine the neighboring edge statuses. An action is run either until the trajectory follower reports success or failure, or the current nearest node changes. The planner runs asynchronously, using increasing tree depths, while the

[2]The framework can be straightforwardly extended to more complex cost functions including, e.g., proximity to obstacles.

Instance	DA	ALA1	ALA2	ALA3	BA1	BA2
1	161 ± 23	186 ± 36	$\mathbf{132 \pm 7}$	142 ± 12	272 ± 57	285 ± 94
2	758 ± 207	225 ± 59	228 ± 65	$\mathbf{149 \pm 38}$	522 ± 207	380 ± 123
3	1203 ± 64	936 ± 58	$\mathbf{907 \pm 54}$	1025 ± 48	1064 ± 73	1134 ± 63
4	353 ± 14	361 ± 33	191 ± 22	$\mathbf{178 \pm 12}$	259 ± 33	286 ± 40
5	897 ± 97	853 ± 122	639 ± 43	$\mathbf{625 \pm 37}$	914 ± 103	1321 ± 91
6	1115 ± 80	1026 ± 56	$\mathbf{915 \pm 54}$	924 ± 51	1079 ± 88	1226 ± 76
7	441 ± 39	493 ± 58	410 ± 44	$\mathbf{378 \pm 36}$	408 ± 37	461 ± 36
8	732 ± 101	857 ± 122	684 ± 55	$\mathbf{621 \pm 51}$	817 ± 62	808 ± 53

TABLE II
RESULTS ON EVALUATING DIFFERENT AGENTS ON A SET OF BENCHMARK ENVIRONMENTS. EACH CELL ENTRY REPORTS COST TILL REACHING THE GOAL AND SAMPLE STANDARD DEVIATION, GIVEN 30 INDEPENDENT TRIALS. PLANNING TIME WAS BOUNDED AT ONE SECOND.

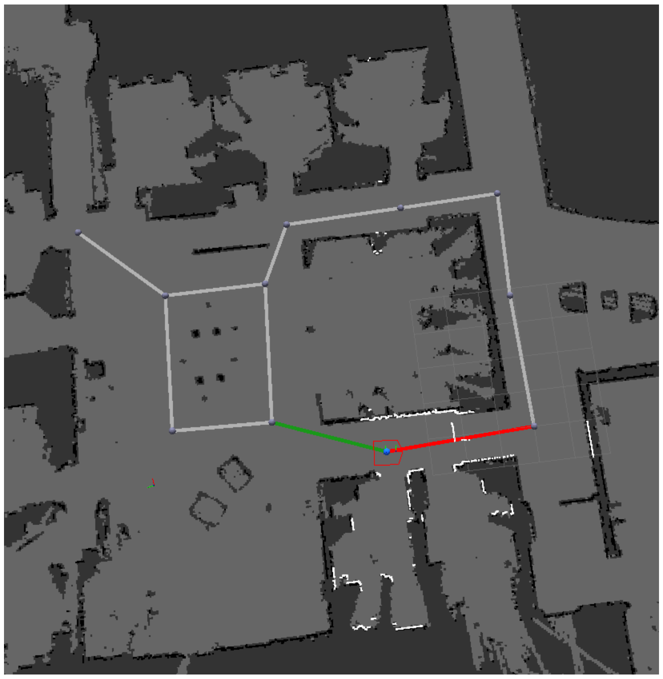
Fig. 8. Screenshot from visualization during execution on PR2 (on subset of overall environment used in experiments). The roadmap graph is overlaid on a static occupancy grid. The laser readings are also shown in white. The red polygon shows the robot's current position, which is mapped to a particular node of the roadmap. An obstacle is visible in the hallway, resulting in the red edge being blocked. Thus the only available action is to follow the green edge. This figure is best viewed in color.

robot is navigating, assuming the action succeeds, i.e., that we end up at the node on the other side of the given edge. If this does happen, the next action can be selected immediately without pausing. If the action terminates at an unexpected node (usually because a person was in the robot's way), the robot pauses and the planner is rerun for one second. This only happens occasionally, as the elastic band planner has some degree of reactivity to unforeseen obstacles.

We compared the performance of our navigation system against the ROS navigation stack [11] by measuring the time taken to navigate to a sequence of waypoints. Five trials were run for each system. The trials were conducted during the day, when there is a reasonably high density of people. Our environment is fairly typical of academic or office buildings, with a mixture of open spaces, rooms, and hallways of varying width. The people in the building are familiar with navigating robots; they do not try to interact with or move out of the way for them.

Table III shows navigation times. In the best case, the two systems performed similarly. This corresponds to situations where no unforeseen obstacles were encountered in hallways or doorways. However, the worst case cost of the ROS navigation stack was much higher. This seems to mainly be because it does not model the probability of different observation types or take the length of the second-best path into account when deciding how long to wait given an unforeseen obstacle; it therefore often gives up prematurely in favour of a significantly longer path.

VII. RELATED WORK

Navigation using roadmaps has been well studied. Most existing work has assumed the graph is known. [13] is an exception, in which a PRM planner is modified to accept or reject sampled points based on the probability of collision, and then looks for an unconditional plan with a high chance of success. In contrast, since our approach has an observation model, it will return conditional plans that base future actions on the results of observations.

Ong et al [15] studied *mixed observability Markov decision processes* (MOMDPs), in which part of the state is perfectly observed, and showed how to speed up offline dynamic programming algorithms in this case. Our problem is a MOMDP, since the robot position in the roadmap is perfectly observed. Unfortunately, the unobserved portion of the state space is still exponentially large.

Closely related to this research is that of Kneebone and Dearden [8]. Like us, they consider navigation in a partially observed graph, and represent the uncertainty using a POMDP. A difference is that in their framework, obstacles do not appear and disappear over time. This makes a qualitative difference in the kinds of policies that are found: there is no longer any utility in taking wait actions, nor are there useful inferences to be made about obstacle classes (effectively, all obstacles belong to a single class whose unblocking rate is 0). Also related is the literature on the Canadian Traveller's Problem [14]. Again, the obstacles here are either static or Markovian, so that a bounded wait will never be optimal.

Trial	ROS Navigation Stack	ALA
1	436	460
2	844	605
3	1118	549
4	491	534
5	520	502

TABLE III
EXECUTION TIME (SECONDS) FOR PR2 USING THE ROS NAVIGATION STACK AND OUR SYSTEM ON A FIXED SET OF WAYPOINTS IN OUR ENVIRONMENT.

The idea of using an abstract graph to solve the planning problem is reminiscent of approaches based on subgoals and macroactions [4]. Subgoal choice tends to be a challenging question for these approaches [22]. We make use of the structure of the problem to choose subgoals as places where obstacles are likely to be.

Theocharous and Kaelbling [23] also describe a macro-action-based POMDP planning algorithm for robot navigation, but they were concerned with uncertainty about the robots position rather than the obstacles. Another complementary line of research is on motion planning among movable obstacles [5]. Unlike our work, which considers an unknown and changing set of static obstacles, this work considers a known, fixed set of moving obstacles.

Execution systems for robotics have been widely studied [16] but many implemented systems still use finite state machines or scripts in a general-purpose programming language. TREX [18] is a sophisticated executive based on temporal planning, that was recently used in a navigation task involving failure recovery [12]. The executive in that case still required significant procedural knowledge to be prespecified in the form of temporal constraints, and planning was mainly used to ensure that the constraints were met.

VIII. CONCLUSION

We view the main contribution of this work as showing by example that it is possible to build robotic systems that reason about and react to execution failures using planning. Our formulation deals with uncertainty in a principled way, and optimal solutions to it exhibit various behaviors that are normally hardcoded into an executive, such as persistence and inference about obstacles. From a software engineering point of view, the resulting systems are less stateful, hence easier to understand and modify. We also described an efficient approximate solution algorithm that can feasibly be run online on a robot, and showed improved performance compared with traditional deterministic planners.

REFERENCES

[1] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *ICRA*, 1999.

[2] Vadim Bulitko and Mitja Lustrek. Lookahead pathology in real-time path-finding. In *AAAI*. AAAI Press, 2006.

[3] Howie Choset and Joel W. Burdick. Sensor-based exploration: The hierarchical generalized Voronoi graph. *IJRR*, 19(2), 2000.

[4] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res. (JAIR)*, 13, 2000.

[5] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *IJRR*, 21(3), 2002.

[6] Leslie Kaelbling, Michael Littman, and Anthony Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101, 1998.

[7] Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, 1999.

[8] Michael Kneebone and Richard Dearden. Navigation planning in probabilistic roadmaps with uncertainty. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *ICAPS*. AAAI, 2009.

[9] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, 2006.

[10] Steve Lavalle. Filtering and planning in information spaces. *IROS tutorial notes*, 2009.

[11] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *ICRA*, 2010.

[12] Wim Meeussen, Melonee Wise, Stuart Glaser, Sachin Chitta, Conor McGann, Patrick Mihelich, Eitan Marder-Eppstein, Marius Muja, Victor Eruhimov, Tully Foote, John Hsu, Radu Rusu, Bhaskara Marthi, Gary Bradski, Kurt Konolige, Brian Gerkey, and Eric Berger. Autonomous door opening and plugging in with a personal robot. In *ICRA*, 2010.

[13] Patrycja Missiuro and Nicholas Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *ICRA*, 2006.

[14] Evdokia Nikolova and David R. Karger. Route planning under uncertainty: The Canadian traveller problem. In *AAAI*, 2008.

[15] S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee. POMDPs for robotic tasks with mixed observability. In *Proc. Robotics: Science and Systems*, 2009.

[16] Ola Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2), 2005.

[17] M.L. Puterman. *Markov decision processes*. Wiley-Interscience, 2005.

[18] Frederic Py, Kanna Rajan, and Conor McGann. A systematic agent framework for situated autonomous systems. In *AAMAS*, 2010.

[19] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *ICRA*, 1993.

[20] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *J. Artif. Intell. Res. (JAIR)*, 32, 2008.

[21] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Pearson, 2010.

[22] Özgür Simsek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *ICML*, 2005.

[23] Georgios Theocharous and Leslie Kaelbling. Approximate planning in POMDPs with macro-actions. In *NIPS*, 2003.

[24] Sebastian Thrun. Probabilistic robotics. *Commun. ACM*, 45(3), 2002.