Motion Planning under Uncertainty for Robotic Tasks with Long Time Horizons

Hanna Kurniawati
Singapore–MIT Alliance for Research Technology
Singapore 117543, Singapore
hannakur@smart.mit.edu

Yanzhu Du
Department of Computer Science, Stanford University
Stanford, CA 94305, USA
yanzhudu@stanford.edu

David Hsu
Department of Computer Science, National University of Singapore
Singapore 117590, Singapore
dyhsu@comp.nus.edu.sg

Wee Sun Lee
Department of Computer Science, National University of Singapore
Singapore 117590, Singapore
leews@comp.nus.edu.sg

Abstract

Motion planning with imperfect state information is a crucial capability for autonomous robots to operate reliably in uncertain and dynamic environments. Partially observable Markov decision processes (POMDPs) provide a principled general framework for planning under uncertainty. Using probabilistic sampling, point-based POMDP solvers have drastically improved the speed of POMDP planning, enabling us to handle moderately complex robotic tasks. However, robot motion planning tasks with long time horizons remains a severe obstacle for even the fastest point-based POMDP solvers today. This paper proposes *Milestone Guided Sampling* (MiGS), a new point-based POMDP solver, which exploits state space information to reduce effective planning horizons. MiGS samples a set of points, called *milestones*, from a robot's state space and constructs a simplified representation of the state space from the sampled milestones. It then uses this representation of the state space to guide sampling in the belief space and tries to capture the essential features of the belief space with a small number of sampled points. Preliminary results are very promising. We tested MiGS in simulation on several difficult POMDPs that model distinct robotic tasks with long time horizons in both 2-D and 3-D environments. These POMDPs are impossible to solve with the fastest point-based solvers today, but MiGS solved them in a few minutes.

1 Introduction

Motion planning with imperfect state information is a crucial capability for autonomous robots to operate reliably in uncertain and dynamic environments. With imperfect state information, a robot cannot decide the best actions on the basis of a single known state; instead, the best action depend on the set of all possible states consistent with the available information, resulting in much higher computational complexity for planning the best actions. Partially observable Markov decision processes (POMDPs) [9, 21] provide a principled general framework for such planning tasks. In a POMDP, we represent a set of possible states as a *belief*, which is a probability distribution over a

robot's state space. We systematically reason over the *belief space* $\mathcal{B}$, the space of all beliefs, by taking into account uncertainty in robot control, sensor measurements, and environment changes, in order to choose the best actions and achieve robust performance. By incorporating uncertainty into planning, the POMDP approach has led to improved performance in a number of robotic tasks, including localization, coastal navigation, grasping, and target tracking [5, 8, 14, 18].

Despite its solid mathematical foundation, POMDP planning faces two major computational challenges. The first one is the "curse of dimensionality": a complex robotic task generates a high-dimensional belief space. If a robotic task is modeled with a discrete state space, its belief space has dimensionality equal to the number of states. Thus, a task with 1,000 states has a 1,000-dimensional belief space! In recent years, point-based POMDP solvers [11, 14, 22, 23] have made dramatic progress in overcoming this challenge by sampling the belief space and computing approximate solutions. Today, the fastest point-based POMDP solvers, such as HSVI2 [22] and SARSOP [11], can handle moderately complex robotic tasks modeled as POMDPs with up to 100,000 states in reasonable time. The success of point-based solvers can be largely attributed to *probabilistic sampling*, which allows us to use a small number of sampled points as an approximate representation of a high-dimensional belief space. The approximate representation substantially reduces computational complexity. The same reason underlies the success of probabilistic sampling in other related problems and approaches, *e.g.*, probabilistic roadmap (PRM) algorithms [2] for geometric motion planning (without uncertainty).

The second major challenge is the "curse of history". In a motion planning task, a robot often needs to take many actions to reach the goal, resulting in a long planning horizon. The complexity of planning often grows exponentially with the horizon. Together, a long planning horizon and a high-dimensional belief space compound the difficulty of planning under uncertainty. For this reason, even the best point-based POMDP solvers today have significant difficulty with robotic tasks requiring long horizons (see Section 6 for examples).

To overcome this second challenge and scale up POMDP solvers for realistic robot motion planning tasks, we have developed a new point-based POMDP solver called *Milestone Guided Sampling* (MiGS). It is known from earlier work on related problems that the most important component of a sampling-based planning algorithm is its sampling strategy [6]. MiGS reduces the planning horizon by constructing a more effective sampling strategy. It samples a set of points, called *milestones*, from a robot's *state space* and constructs a *roadmap* graph. Each node of the roadmap corresponds to a sampled milestone, and each edge is labeled with a sequence of actions that brings the robot from one milestone to another. MiGS then uses the roadmap to guide sampling in the *belief space*. Using the roadmap as a simplified representation of the state space, MiGS avoids exploring many of the similar belief space paths and thus improves computational efficiency.

MiGS' approach to belief space sampling is related to earlier POMDP solvers that use macro-actions [15, 24]. However, the earlier works define macro-actions as policies. They manually decompose the POMDP into several smaller POMDPs, solve the smaller POMDPs, and then use the resulting policies as macro-actions to solve the original POMDP. Instead of constraining the resulting policy in terms of such pre-defined macro-actions, MiGS automatically generates long action sequences, which can be regarded as open-loop policies, and use them to guide sampling in the belief space. Only a small number of action sequences are initially allowed, restricting the resolution of sampling. The number of action sequences is increased as required, allowing the sampling resolution to scale up gradually with the complexity of the problem.

We tested MiGS in simulation on several difficult POMDPs that model distinct robotic tasks with long time horizons. These POMDPs are impossible to solve with the fastest point-based POMDP solvers today, but MiGS solved them in a few minutes.

2 Background

2.1 Motion Planning under Uncertainty

Despite its importance and more than three decades of active research [12, 25], motion planning under uncertainty remains a challenge in robotics. Several recent successful algorithms are based on

the probabilistic sampling approach. Stochastic Motion Roadmap [1] combines PRM with the Markov decision process (MDP) framework to handle uncertainty in robot control, but it does not take into account uncertainty in sensing. Belief Roadmap [17] handles uncertainty in both robot control and sensing, but one major limitation is the assumption that the uncertainties can be modeled as Gaussian distributions. Unimodal distributions such as the Gaussian are inadequate when robots operate in complex geometric environments. Using a mixture of Gausssians can partially address this weakness, but at a significant computational cost.

POMDPs are a principled general framework that can overcome the above limitations. By tackling the difficulty of long planning horizons, MiGS brings POMDPs a step closer to being practical for complex robotics tasks.

2.2 POMDPs

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its reward. Formally, it is specified as a tuple $(S, A, O, T, Z, R, \gamma)$, where S is a set of states describing the agent and the environment, A is the set of actions that the agent may take, and O is the set of observations that the agent may receive. The remaining elements of the tuple are explained below.

In each time step, the agent lies in a state $s \in S$, takes an action $a \in A$, and moves from a start state s to an end state s'. Due to the uncertainty in action, the end state s' is modeled as a conditional probability function $T(s, a, s') = p(s'|s, a)$, which gives the probability that the agent lies in s', after taking action a in state s. The agent then receives an observation that provides information on its new state s'. Due to the uncertainty in observation, the observation result $o \in O$ is again modeled as a conditional probability function $Z(s', a, o) = p(o|s', a)$.

To elicit desirable agent behavior, we define a suitable reward function $R(s, a)$. In each step, the agent receives a real-valued reward $R(s, a)$, if it takes action a in state s. The goal of the agent is to maximize its expected total reward by choosing a suitable sequence of actions. When the sequence of actions has infinite length, we typically specify a discount factor $\gamma \in (0, 1)$ so that the total reward is finite and the problem is well defined. In this case, the expected total reward is given by $\mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where s_t and a_t denote the agent's state and action at time t.

POMDP planning means computing an *optimal policy* that maximizes the agent's expected total reward. In the more familiar case where the agent's state is fully observable, a policy prescribes an action, given the agent's current state. However, a POMDP agent's state is partially observable and not known exactly. So we rely on the concept of a belief. A POMDP policy $\pi\colon \mathcal{B} \to A$ is a mapping from $\mathcal{B}$ to A, which prescribes an action a, given the agent's belief b.

A policy π induces a value function $V_\pi(b)$. The value function $V_\pi(b) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)|b, \pi\right]$ specifies the expected total reward of executing policy π: the agent has initial belief b, and its action a_t at time t is chosen according to π. It is known that V^*, the value function associated with an optimal policy π^*, can be approximated arbitrarily closely by a convex, piecewise-linear function,

$$V(b) = \max_{\alpha \in \Gamma} (\alpha \cdot b) = \max_{\alpha \in \Gamma} \Big(\sum_{s \in S} \alpha(s) b(s) \Big) \qquad (1)$$

where Γ is a finite set of vectors called α-vectors and b is a belief. Each α-vector is associated with an action a. The policy can be executed by selecting the action corresponding to the best α-vector at the current belief. So a policy can be represented as a set Γ of α-vectors. For each $\alpha \in \Gamma$, $\alpha(s)$ is the expected total reward that the agent receives when it starts in s, takes the action a associated with α, and acts according to the policy defined by Γ afterwards. Policy computation, which, in this case, involves the construction of Γ, is usually performed offline.

Given a policy, represented as a set Γ of α-vectors, the control of the agent's actions, also called policy execution, is performed online in real time. It consists of two steps executed repeatedly. The first step is action selection. If the agent's current belief is b, it finds the action a that maximizes $V(b)$ by evaluating (1). The second step is belief update. After the agent takes an action a and receives an observation o, its new belief b' is given by

$$b'(s') = \tau(b, a, o) = \eta Z(s', a, o) \sum_s T(s, a, s') b(s) \qquad (2)$$

where η is a normalization constant.

More information on POMDPs is available in [9, 25].

2.3 Related POMDP Solvers

Large state spaces and long time horizons, typical of many robot motion planning tasks, are two main obstacles that hinder the use of POMDPs in practice. Many approaches have been proposed to alleviate these difficulties.

Point-based POMDP solvers are currently one of the most successful approaches. Point-based solvers reduce the complexity of planning in the belief space by representing $\mathcal{B}$ as a set of sampled beliefs. Interestingly, probabilistic sampling, a key idea of point-based solvers, is also what makes PRM planning successful in geometric motion planning. For computational efficiency, most of the recent point-based solvers [11, 14, 20, 22, 23] sample only from a subset $\mathcal{R} \subseteq \mathcal{B}$ reachable from a given initial belief. They differ in how they sample $\mathcal{R}$. PBVI [14] and Perseus [23] spread sampled points over $\mathcal{R}$ to cover it well. HSVI2 [22] maintains upper and lower bounds on the optimal value function and samples $\mathcal{R}$ to close the gap between the upper and lower bounds. FSVI [20] uses only the upper bound to guide sampling. SARSOP [11] biases sampling towards optimally reachable spaces, the subset of beliefs reachable under optimal policies. Point-based solvers have made impressive progress in computing approximate solutions for POMDPs with large state spaces, and they have been successfully applied to a variety of non-trivial robotic tasks, including localization [16, 18], coastal navigation [16, 18], grasping [5], target tracking [14, 8], and exploration [11, 22]. Despite the progress, even the best point-based POMDP solvers today have significant difficulty with robotic tasks that require long planning horizons.

MiGS follows the approach of point-based POMDP solvers, but aims at overcoming the difficulty of long planning horizons. It uses sequences of actions, rather than single primitive actions, to explore the belief space and thus significantly reduces effective planning horizons.

3 Milestone Guided Sampling

A key idea of point-based POMDP solvers is to sample a set of points from $\mathcal{B}$ and use it as an approximate representation of $\mathcal{B}$. Let $\mathcal{R} \subseteq \mathcal{B}$ be the set of points reachable from a given initial belief point $b_0 \in \mathcal{B}$ under arbitrary sequences of actions and observations. Most of the recent point-based POMDP algorithms sample from $\mathcal{R}$ instead of $\mathcal{B}$ for computational efficiency. The sampled points form a belief tree $T_{\mathcal{R}}$ (Figure 1). Each node of $T_{\mathcal{R}}$ represents a sampled point $b \in \mathcal{B}$. The root of $T_{\mathcal{R}}$ is the initial belief point b_0. To sample a new point b', we pick a node b from $T_{\mathcal{R}}$ as well as an action $a \in A$ and an observation $o \in O$ according to suitable probability distributions or heuristics. We then compute $b' = \tau(b, a, o)$ using (2) and insert b' into $T_{\mathcal{R}}$ as a child of b. If a POMDP requires an effective planning horizon of h actions and observations, $T_{\mathcal{R}}$ may contain $\Theta\big((|A||O|)^h\big)$ nodes in the worst case, where $|A|$ is the number of actions and $|O|$ is the number of observations for the POMDP. Thus any point-based solvers trying to construct $T_{\mathcal{R}}$ exhaustively must have running time exponential in h and suffer from the "curse of history".

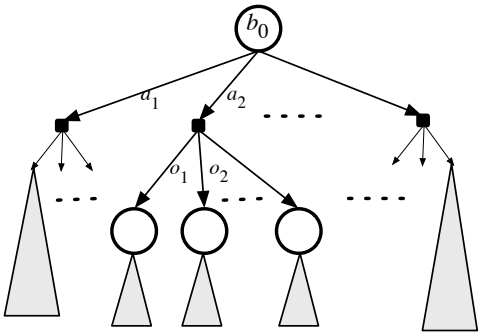


Figure 1: The belief tree rooted at b_0.

To overcome this difficulty, let us consider the space from which we must sample. Suppose that the effective planning horizon is h. We must sample from a subset of $\mathcal{R}$ that contains $\mathcal{R}_h$, the set of belief points reachable from b_0 with at most h actions and observations. Our difficulty is that the size of $\mathcal{R}_h$ grows exponentially with h. The basic idea of MiGS is to sample $\mathcal{R}_h$ hierarchically at multiple resolutions and avoid exhaustively sampling $\mathcal{R}_h$ unless necessary.

To do so, MiGS builds a roadmap graph M in a robot's state space S. The nodes of M are states sampled from S and are called milestones. An edge e between two milestones s and s' of M is annotated with a sequence of actions $(a_1, a_2, \ldots, a_\ell)$ that bring the robot from s to s'. The edge e is also annotated with a sequence of states $(s_0, s_1, \ldots, s_\ell)$ that the robot traverses under the actions $(a_1, a_2, \ldots, a_\ell)$, with $s_0 = s$ and $s_\ell = s'$. A path in M encodes a long action sequence obtained by concatenating the action se-

quences associated with the edges along the path. By traversing M suitably, we can generate a rich collection of action sequences for sampling $\mathcal{B}$. At a node b of $T_{\mathcal{R}}$, we can simply apply a *sequence of actions* associated with an edge of M, instead of a single action, and derive a child node b'.

The resulting belief tree $T_{\mathcal{R}}$ then becomes much smaller. Suppose, for example, that M has maximum degree d and the minimum length of an action sequence contained in each edge of M is ℓ. Then, for a POMDP with time horizon h, the number of nodes in $T_{\mathcal{R}}$ is $\mathcal{O}\big((d\,|O|^{\ell})^{h/\ell}\big) = \mathcal{O}(d^{h/\ell}|O|^{h})$ in the worst case. This indicates that the action sequences encoded in M help in reducing the effect of long planning horizons due to actions. Since the size of $T_{\mathcal{R}}$ grows exponentially with h, the improvement is significant. MiGS uses an additional technique for reducing the long-horizon effect due to observations, but it is secondary and will be described later.

Now it should be clear that MiGS is potentially faster, as the belief tree $T_{\mathcal{R}}$ is smaller. However, a more fundamental question remains. The roadmap M is a simplified representation of the state space S. In general, M contains only a subset of sampled states and not all states in S. Do the belief points sampled with the help of M cover the reachable belief space well and is it likely to lead to a good approximation to the optimal value function and the optimal policy? The answer is yes, if we sample S adequately in a sense which we now explain.

Our main intuition is that the underlying state space $\underline{S}$ of a robotic system is typically continuous. The POMDP state space S is a discretization of $\underline{S}$. States that are close together in $\underline{S}$ should have similar properties. Formally, denote by Γ^* a set of α-vectors representing the optimal value function. Given a constant $\epsilon > 0$, we partition the state space S into a collection $\mathcal{K}_\epsilon$ of disjoint subsets so that for any $\alpha \in \Gamma^*$ and any two states s and s' in the same subset $K \in \mathcal{K}_\epsilon$, we have $|\alpha(s) - \alpha(s')| \leq \epsilon$. Intuitively, the partitioning condition means that any two states in the same subset K are similar in terms of their significance in the optimal value function. The constant ϵ controls the resolution of partitioning. We sometimes omit the subscript in $\mathcal{K}_\epsilon$ to simplify the notation. The partitioning $\mathcal{K}$ induces a distance metric on the belief space $\mathcal{B}$:

Definition 1 *Let $\mathcal{K}$ be an ϵ-partitioning of the state space S. The distance between any two beliefs b and b' in $\mathcal{B}$ with respect to $\mathcal{K}$ is*

$$d_{\mathcal{K}}(b, b') = \sum_{K \in \mathcal{K}} \Big| \sum_{s \in K} b(s) - \sum_{s \in K} b'(s) \Big|. \qquad (3)$$

This new metric is weaker than the usual L_1 metric and is upper-bounded by the L_1 metric. It measures the difference in probability mass for subsets of states rather than individual states. This is desirable, because the states within a single subset $K \in \mathcal{K}$ are similar under our assumption and there is no need to distinguish them. Now we can derive a Lipschitz condition on the optimal value function $V^*(b)$ under $d_{\mathcal{K}}$.

Theorem 1 *Let $\mathcal{K}$ be an ϵ-partitioning of the state space S, and let $\mathcal{B}$ be the corresponding belief space over S. For any $b, b' \in \mathcal{B}$, if $d_{\mathcal{K}}(b, b') \leq \delta$, then $|V^*(b) - V^*(b')| \leq \frac{R_{\max}}{1-\gamma}\delta + 2\epsilon$, where $R_{\max} = \max_{s \in S, a \in A} |R(s, a)|$.*

The proof is given in the appendix. Theorem 1 provides a sampling criterion for approximating V^* well. Suppose that we sample a set B of beliefs that *covers* $\mathcal{B}$ in the sense that for any $b \in \mathcal{B}$, there is a point b' in B with $d_{\mathcal{K}}(b, b') \leq \delta$. Then Theorem 1 implies that we can approximate $V^*(b)$ for any $b \in \mathcal{B}$ by $V^*(b')$ for some $b' \in B$ with a small error that depends on δ. What is even more interesting is that the distance function $d_{\mathcal{K}}(b, b')$ relaxes the covering requirement by averaging out variations of $b(s)$ and $b'(s)$ within the same partition $K \in \mathcal{K}$, at the cost of introducing an additional error ϵ in the value function approximation. This suggests that it is probably sufficient to have only one representative state—a *milestone*—from each partition $K \in \mathcal{K}$. These milestones can be used to construct an approximate partitioning, as done by MiGS during roadmap construction.

Of course, MiGS does not know the partitioning $\mathcal{K}$ in advance. One simple way of ensuring one milestone in each partition $K \in \mathcal{K}$ is to sample a large number of milestones from S uniformly at random. If each $K \in \mathcal{K}$ is sufficiently large in size, then we can guarantee that uniform sampling generates at least one milestone from each $K \in \mathcal{K}$ with high probability. To improve efficiency, observe that under an optimal policy, the robot likely goes through states that provide high rewards or good observations. So we bias the sampling towards such states. See Section 4 for details.

We now give a sketch of the overall algorithm. Suppose that we are given as input a POMDP

model $(S, A, O, T, Z, R, \gamma)$. In addition, we are given a set of *goal states* $S_{\mathrm{g}} \subseteq S$, which is common for robotic tasks. Each goal state $g \in S_{\mathrm{g}}$ is self-absorbing with $T(g, a, g) = 1$ for all $a \in A$, which indicates that once a robot reaches a goal state, the task ends. The robot receives zero reward in a goal state and incurs a negative cost for actions in all other states. Formally, if $g \in S_{\mathrm{g}}$, $R(s, a) = 0$ for all $a \in A$; otherwise, $R(s, a) \leq 0$. Such a reward function motivates the robot to reach the goal as efficiently as possible.

MiGS iterates over two stages. In the first stage, we sample a set of new milestones from S and use it to construct and refine a roadmap M. In the second stage, we use M to guide sampling in $\mathcal{B}$. Following the approach of point-based POMDP planning, we perform *value iteration* [19] on a set Γ of α-vectors, which represents a piecewise-linear lower-bound approximation to the optimal value function V^*. Exploiting the fact that V^* must satisfy the Bellman equation, value iteration starts with an initial approximation to V^* and performs backup operations on the approximation by iterating on the Bellman equation until the iteration converges.

What is different in value iteration for point-based POMDP planning is that backup operations are performed only at a set of sampled points from $\mathcal{B}$ rather than the entire $\mathcal{B}$. We sample this set from $\mathcal{B}$ by constructing a belief tree $T_{\mathcal{R}}$ rooted at an initial belief point b_0. To add a new node to $T_{\mathcal{R}}$, we first choose an existing node b in $T_{\mathcal{R}}$ from less densely sampled regions of $\mathcal{B}$, as this likely leads to sampled points that cover $\mathcal{B}$ well. We then choose a suitable edge e from the roadmap M and use the associated action sequence $(a_1, a_2, \ldots, a_\ell)$ and state sequence $(s_0, s_1, s_2, \ldots, s_\ell)$ to generate a new node.

Specifically, we first sample an observation sequence $(o_1, o_2, \ldots, o_\ell)$ consistent with the state sequence $(s_0, s_1, \ldots, s_\ell)$ in the sense that $Z(s_i, a_i, o_i) = p(o_i | s_i, a_i) > 0$, for $1 \leq i \leq \ell$. The consistency requirement limits the number of possible observations. The main motivation for sampling observation sequences is similar to that of using action sequences contained in M. Often, many observation sequences provide similar information. Sampling observation sequences avoids exploring many similar belief space paths and helps to reduce the long-horizon effect due to observations.

After obtaining the observation sequence, we then apply the action-observation sequence $(a_1, o_1, a_2, o_2, \ldots, a_\ell, o_\ell)$ to b and generate a sequence of new beliefs $(b_1, b_2, \ldots, b_\ell)$, where

$$b_1 = \tau(b, a_1, o_1) \tag{4}$$

$$b_i = \tau(b_{i-1}, a_{i-1}, o_{i-1}) \text{ for } 2 \leq i \leq \ell. \tag{5}$$

Finally, b_ℓ is inserted to $T_{\mathcal{R}}$ as a new child node of b, while $(b_1, b_2, \ldots, b_{\ell-1})$ is associated with the edge from b to b_ℓ for backup operations. After creating the node b_ℓ, we perform backup operations for every belief associated with the nodes and edges of $T_{\mathcal{R}}$ along the path from b_ℓ to the root b_0. A backup operation at a node b improves the approximation of $V^*(b)$ by looking ahead one step. We perform the standard α-vector backup (Algorithm 1). Each backup operation creates a new α-vector, which is added to Γ to improve the approximation of V^*.

After a round of backup operations, we check the improvement in value function approximation at the root node b_0. If the value no longer changes after several rounds of backup, it likely indicates that all the information in the current roadmap has been exploited and a more refined one is necessary. We then repeat the two stages of MiGS to refine M and improve the value function approximation by sampling additional new beliefs. The details for roadmap construction and belief space sampling are described in Sections 4 and 5, respectively.

4 Roadmap Construction

To construct a roadmap M, MiGS samples a set of points from the state space S. They become the milestones, *i.e.*, the nodes of M. The set Q of milestones induces a partitioning $\mathcal{K}_Q$ of S into disjoint subsets. The adjacency relationship between the partitions $\mathcal{K}_Q$ provides the edge connectivity of M. The partitioning $\mathcal{K}_Q$ is an approximation of $\mathcal{K}_\epsilon$, which is unknown in advance. To improve the approximation, MiGS incrementally refines $\mathcal{K}_Q$ by adding more and more milestones to Q.

4.1 Sampling Milestones

Ultimately we would like to sample a set Q of milestones so that each partition in $\mathcal{K}_\epsilon$ contains a milestone. For efficiency, we bias the sampling towards states that provides higher rewards or informative observations. Specifically, we sample a point s from S with probability

$$p(s) \propto e^{\lambda h(s)},$$

Algorithm 1 Perform α-vector backup at a belief b.

BACKUP(b, Γ)

1: For all $a \in A$, $o \in O$, $\alpha_{a,o} \leftarrow \text{argmax}_{\alpha \in \Gamma}(\alpha \cdot \tau(b, a, o))$.
2: For all $a \in A, s \in S$, $\alpha_a(s) \leftarrow R(s, a) + \gamma \sum_{o,s'} T(s, a, s')Z(s', a, o)\alpha_{a,o}(s')$.
3: $\alpha' \leftarrow \text{argmax}_{a \in \mathcal{A}}(\alpha_a \cdot b)$
4: Insert α' into Γ.

where λ is a pre-specified positive constant that controls the smoothness of the sampling distribution and $h(s)$ indicates the importance of a state s. We define the heuristic importance function as

$$h(s) = (\alpha_h(s) + R_{\max})b_h(s),$$

where $R_{\max} = \max_{s \in S, a \in A} |R(s, a)|$. If we ignore the $R_{\max}$ term for the moment, $h(s)$ becomes simply a product $\alpha_h(s)b_h(s)$. This is motivated by the form of the optimal value function $V^*(b) = \max_{\alpha \in \Gamma^*} \sum_{s \in S} \alpha(s)b(s)$, which indicates that under an optimal policy, the robot likely goes through states with large $\alpha(s)b(s)$ values. The term $\alpha_h(s)$ in $h(s)$ tries to capture those states with high expected total reward. Suppose that the robot starts in s and receives the immediate reward $R(s, a)$ for taking action a in s. We assume that all actions occur with equal probability and that the robot receives the maximum reward, which is 0 by our assumption, for all future actions. This gives an optimistic estimate of the expected total reward that the robot can actually achieve, starting from s:

$$\alpha_h(s) = \sum_{a \in A} p(a|s)R(s, a) = \frac{1}{|A|} \sum_{a \in A} R(s, a).$$

Since $R(s, a) \leq 0$ by our assumption, we need to insert the additional term $R_{\max}$ in $h(s)$ so that $\alpha_h(s) + R_{\max} \geq 0$ for all $s \in S$ and a large positive value of $h(s)$ indicates an important state. The second component $b_h(s)$ tries to capture those states with informative observations. We identify such states by considering the probability $p(s|a, o)$. A large $p(s|a, o)$ value implies that if the robot arrives in s by taking action a and receives observation o, then it recognizes the state s with high probability. We assume again that all actions occur with equal probability, but that given an action a, the observations occur according to the observation function $Z(s, a, o)$. We then define $b_h(s)$ as the weighted average of $p(s|a, o)$ over all actions and observations:

$$\begin{aligned} b_h(s) &= \sum_{a \in A} p(a|s) \sum_{o \in O} Z(s, a, o)p(s|a, o) \\ &= \frac{1}{|A|} \sum_{a \in A} \sum_{o \in O} Z(s, a, o)p(s|a, o). \end{aligned}$$

To calculate $p(s|a, o)$, we apply the Bayes rule. Assuming that the prior probabilities for all states to occur given a are equal, we get $p(s|a, o) = p(o|s, a)/\sum_{s \in S} p(o|s, a) = Z(s, a, o)/\big(|S| \sum_{s \in S} Z(s, a, o)\big)$.

We choose to define the heuristic importance function $h(s)$ using local information only. Other more sophisticated definitions are possible, but we would like to keep $h(s)$ as simple as possible.

4.2 State Space Partitioning

The milestone set Q induces a partitioning $\mathcal{K}_Q$ of S. Each milestone $q \in Q$ spans a partition $K \in \mathcal{K}_Q$ such that each state in K is "closer" to q than any other milestones in Q. This is similar to a Voronoi diagram for a point set under the Euclidean distance. However, Euclidean distance is unsuitable here. The proximity measure should reflect how the states differ in terms of their importance in the optimal value function.

MiGS captures the proximity between states in a state graph. A state graph P is a weighted, directed multi-graph. The nodes of P are all the states in S. There is an edge (s, s', a) between two nodes s and s' and labeled with a, whenever $T(s, a, s') > 0$. The edge weight $w(s, s', a)$ tries to capture the difference in expected total reward between s and s'. If the robot is in s, it can go towards the goal directly, or it can take action a to reach s' and then go towards the goal from s'. In the latter case, after taking action a, the robot may end in a state $s'' \neq s'$, due to uncertainty in robot control. So we must take into account the difference in expected total reward between s' and

s''. We define $w(s, s', a)$ by calculating

$$\begin{aligned}&\alpha_w(s) - \alpha_w(s') \\ &= R(s,a) + \gamma \sum_{s'' \in S \setminus \{s'\}} T(s,a,s'') \\ &\qquad \times \sum_{o \in O} Z(s'',a,o)(\alpha_w(s'') - \alpha_w(s')), \qquad (6)\end{aligned}$$

where $\alpha_w(s)$ approximates the robot's expected total reward from a state $s \in S$. The first term $R(s,a)$ in (6) is the immediate reward of taking action a in s. The second term accounts for the outcomes after taking the action. We can write one such equation for each unknown $\alpha_w(s)$ and solve the resulting linear system of equations using standard numerical methods. However, this could be computationally expensive, if the state space S is large. We thus make a further approximation. Since s'' is reachable from s with a single action, we expect $\alpha_w(s)$ and $\alpha_w(s'')$ to have similar values if the reward function $R(s,a)$ is smooth. Now replacing $\alpha_w(s'')$ with $\alpha_w(s)$ in the right hand side of (6) and after some simple algebra, we get $\alpha_w(s) - \alpha_w(s') = R(s,a)/(1-\gamma+\gamma T(s,a,s'))$. Finally, since $R(s,a) \leq 0$ by our assumption, we define

$$w(s,s',a) = \frac{-R(s,a)}{1-\gamma+\gamma T(s,a,s')}$$

so that all edges weights are non-negative, and say that a path is *short* if it has low weight.

The state graph P defines a proximity measure d_P, where $d_P(s,s')$ is the minimum weight over all paths from s to s'. Although d_P is not necessarily symmetric, it can still be used to partition S. Specifically, the partition $K \in \mathcal{K}_Q$ for a milestone $q \in Q$ is a subset of states such that each state in K has $d_P(s,q) \leq d_P(s,q')$ for all $q' \in Q$. Such a partitioning $\mathcal{K}_Q$ is in fact an *inward Voronoi diagram* of Q over the state space S [3].

4.3 Connecting Milestones

To complete the construction of roadmap M, we need to insert edges between milestones and label them with action and state sequences. For this, we use the adjacency relationship between the partitions in $\mathcal{K}_Q$. Let q and q' be two milestones in Q. Let K and K' be the corresponding partitions in $\mathcal{K}_Q$ for q and q', respectively. The states in $K \cup K'$ induce a subgraph $P_{KK'}$ of P. We insert an edge (q,q') from q to q' into M if there is a path from q to q' in $P_{KK'}$. One may allow self-loops in the roadmap M if the POMDP model contains pure information-gathering actions, which change a robot's belief, but not its state. We then find a shortest path from q to q' in $P_{KK'}$ and use the edges along this path to label (q,q') with action and state sequences accordingly. The edge (q,q') is also assigned a weight, which is the weight of the shortest path.

4.4 Roadmap Refinement

To refine a roadmap M, we sample an additional set Q' of milestones. We then add Q' to M and construct a new roadmap over $Q \cup Q'$ incrementally, where Q denotes the milestones in M.

In a discrete POMDP, S is finite. So we can add in all states in S as milestones. The resulting roadmap M would still be insufficient for generating all possible sequences of actions, as M does not contain all the actions between a pair of states. As a final refinement, we can use the state graph P as a roadmap. This in principle makes planning complete, but is probably not necessary or desirable in practice.

5 Belief space sampling

MiGS samples from $\mathcal{B}$ by incrementally constructing a belief tree $T_{\mathcal{R}}$ rooted at an initial belief point b_0 (see Section 3). Adding a new node to $T_{\mathcal{R}}$ consists of two steps. In the first step, we choose an existing node b from $T_{\mathcal{R}}$. In the second step, we use the roadmap M as a guide to choose an action and an observation sequence, apply the action and the observation sequence to b, and generate a new node b' as a child of b.

We want to spread the newly sampled points evenly over $\mathcal{B}$ in order to improve the coverage of $\mathcal{B}$. A similar strategy is employed by PBVI [14], one of the early point-based POMDP solvers. However, we cover $\mathcal{B}$ under the distance $d_{\mathcal{K}}$, which provides a more relaxed covering requirement and makes it easier to cover $\mathcal{B}$. According to Theorem 1, this is sufficient for obtaining a good approximation of the optimal value function. We implement this idea by assigning a weight $w_{\mathrm{c}}(b)$ to each node b in $T_{\mathcal{R}}$. The weight $w_{\mathrm{c}}(b)$ is the number of beliefs in $T_{\mathcal{R}}$ within a small pre-specified distance of b according to $d_{\mathcal{K}}$. It gives an estimate of how densely the small region near b is sampled. We then choose an existing node b from $T_{\mathcal{R}}$ with probability proportional to $1/w_{\mathrm{c}}(b)$,

in order to improve coverage in the less densely sampled regions.

To generate a new node b', we use M as a guide to choose an action sequence and an observation sequence first. The roadmap M contains many paths. Each path represents a long action sequence obtained by concatenating the action sequences associated with the edges along the path. If these action sequences are optimal, we can simulate these actions and generate a set of beliefs on the optimally reachable space $\mathcal{R}^* \subseteq \mathcal{B}$, which contains the subset of beliefs reachable from b_0 under an optimal policy. It is known that given a sufficiently large set of beliefs forming a cover of $\mathcal{R}^*$, we can compute the corresponding optimal policy efficiently [7]. Of course, we do not know which roadmap paths represent action sequences that are optimal. So we try many different paths, but bias towards those which likely result from an optimal policy. To do so, we order the outgoing edges at each node s in M by increasing edge weights and try the action sequences associated with the edges in this order. This helps us to explore different paths and avoid getting stuck in a local region. The details of generating a new node b' are slightly different, depending on whether the node b chosen in the first step is b_0. If $b = b_0$, conceptually we are starting a new path in M. So we sample a state s according to the belief b. We find the node s in M and choose the next outgoing edge (s, s') that has not been tried yet. If all edge have been tried, we restart from the edge with the lowest weight. The state sequence associated with (s, s') (see Section 3) is used to extract a valid observation sequence. We then apply the action sequence associated with (s, s') and the observation sequence to b and generate a new belief b', using (4) and (5). The resulting new belief b' is inserted into $T_{\mathcal{R}}$ as a child node of b and labeled with s' so that the roadmap path can be continued from s' later on. If the node b chosen in the first step is not b_0, then b must have an associated state s. We find the node s in M, and the remaining steps are the same as those for the case $b = b_0$.

6 Experiments

We tested MiGS on four robotic tasks that require long planning horizons. Below, we first describe these tasks and examine the robot's behavior under policies computed by MiGS (Section 6.1). Next, we compare MiGS with other point-based POMDP solvers and alternative approaches to motion planning under uncertainty (Section 6.2). Finally, we look at the robustness of MiGS policies (Section 6.3).

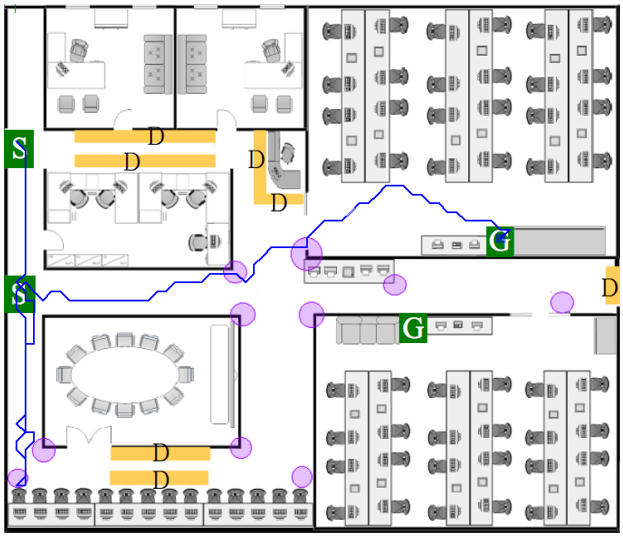
6.1 Tasks and Computed Policies

2-D Navigation. A mobile robot navigates in a laboratory with obstacles and danger zones that must be avoided (Figure 2*a*). It starts in one of the two entrances to the lab, but does not know exactly which one. It must reach one of the goal positions. We represent the robot's position on a uniform grid of size 60×70. In each step, the robot moves from the current grid cell to one of the eight adjacent ones. Due to imperfect control, the robot reaches its intended cell only 90% of the time. For the rest of the time, the robot either remains in the current cell or drifts to the left or the right of the intended cell. The robot can localize itself only in regions with landmarks. Despite imperfect control and localization, the robot must reach the goal as quickly as possible while avoiding obstacles and danger zones.

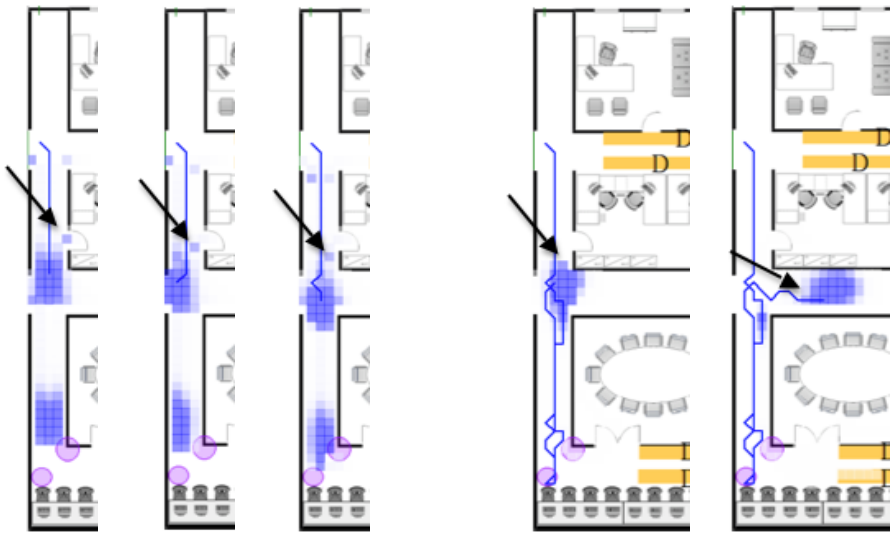
A simulation run under a policy computed by MiGS is shown in Figure 2*a*. The robot starts at the upper entrance. It first moves all the way down to localize itself. It then moves back up and turns right into a corridor towards one of the goals. Since the robot does not know its starting position, it must move down to localize itself first, despite the additional distance traveled. Otherwise, it may get stranded in a danger zone.

Interestingly, although jagged paths are undesirable in general, some parts of the jagged path in Figure 2*a* are in fact deliberate and improve the robustness of the robot's motion in the presence of uncertainty. For example, Figure 2*b* shows the robot makes a zigzag motion first down-left and then down-right. Before the zigzag motion, there is some probability that the robot may enter the wrong room. So it moves down-left to shift this probability mass out. Now the probability mass is concentrated at the lower entrance. To avoid getting stuck there, the robot then moves down-right to shift the probability mass towards the center of the corridor.

As another example (Figure 2*c*), the robot's motion as it turns right into the corridor is similar in nature to that of wiggling a peg and inserting it into a hole. By moving up and down at the mouth

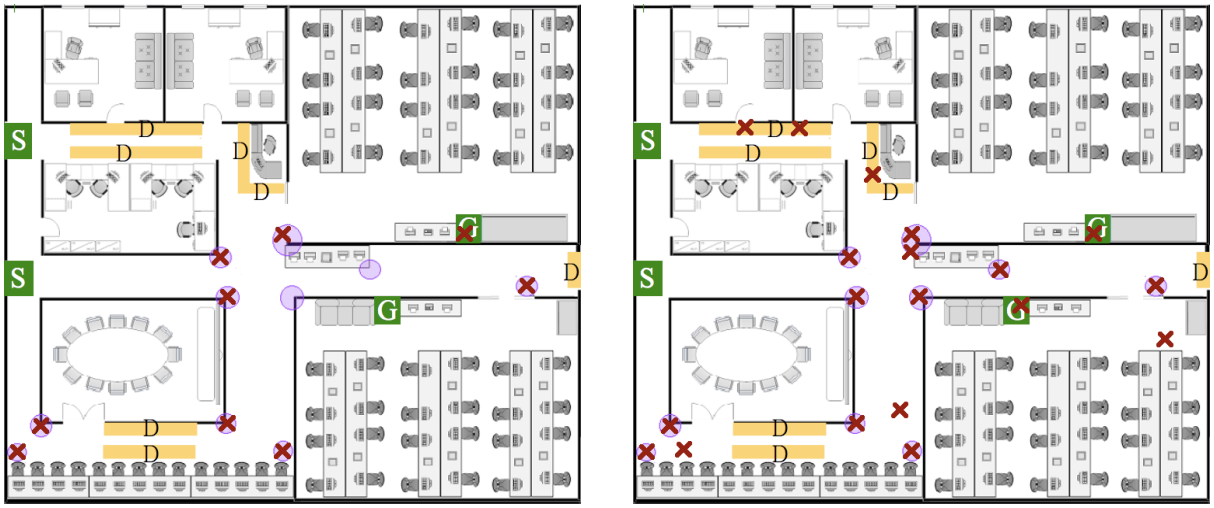


(a)



(b) (c)

Figure 2: 2-D Navigation. (a) The robot's path in a simulation run. (b) The robot makes jagged motion to reduce the probability of entering the wrong room. The arrows point to the probability mass of interest. (c) The robot "wiggles" its way into the corridor. "S" marks the robot's possible initial positions. "G" marks the goal positions. "D" marks danger zones that must be avoided. Shaded discs mark landmark regions for robot localization. The blue line indicates the robot's path in a simulation run. The shaded blue region marks the robot's belief on its own position. Darker colors indicate higher probabilities.



(a) (b)

Figure 3: Sampled milestones. (a) The initial roadmap. (b) The final roadmap. Each "×" marks a milestone.

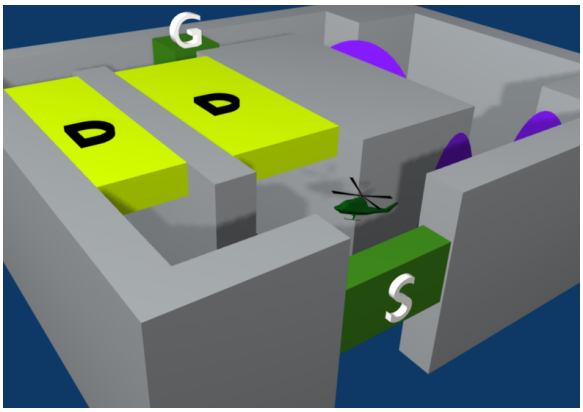


Figure 4: 3-D navigation. See the caption of Figure 2 for the meanings of labels.

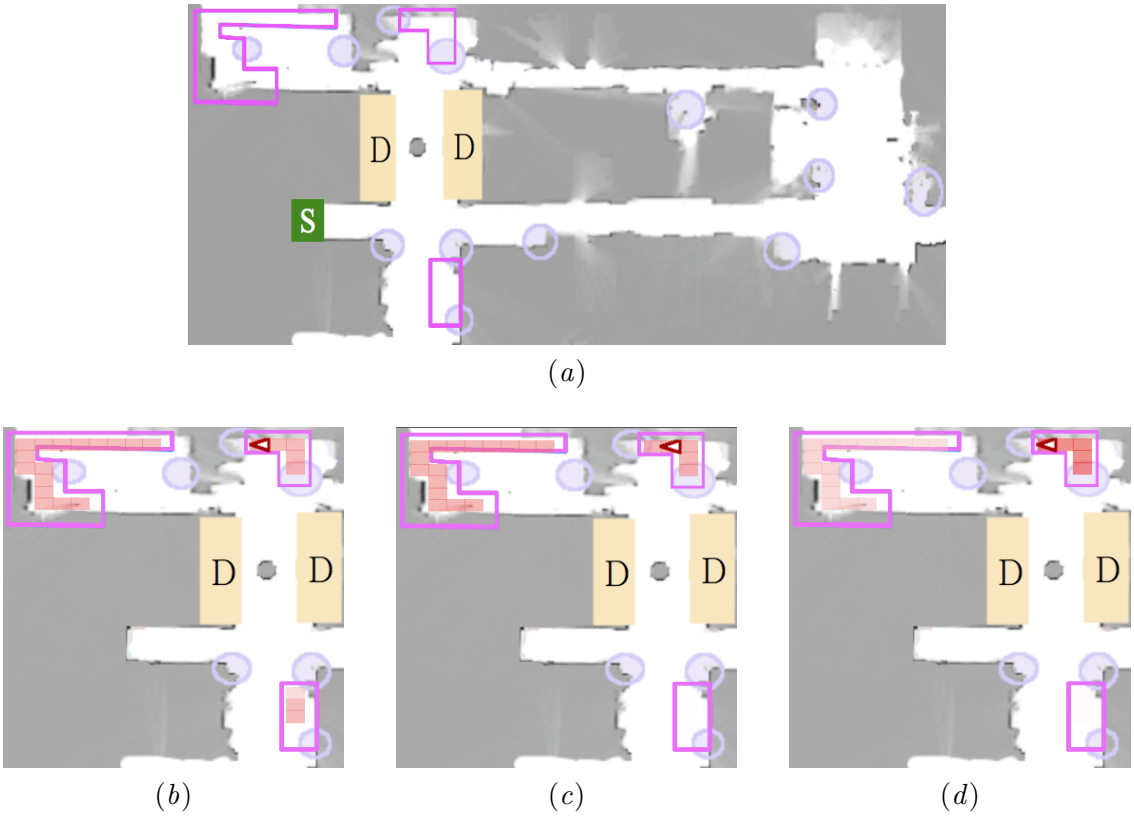
of the corridor, the robot tries to position itself near the center of the corridor and pushes all the probability mass into it.

Figure 3 shows the sampled milestones in the roadmaps constructed by MiGS. In the initial roadmap, the milestones fall in some of the landmark regions and one goal region, as they provide either good observations or high rewards. In the final roadmap, the milestones cover all the landmark regions and goal regions. Some danger zones also contain milestones, because the knowledge of being in a danger zone provides information on the robot's position.

3-D Navigation. An unmanned aerial vehicle (UAV) navigates in an indoor environment (Figure 4), where GPS signal is not available. The environment is populated with obstacles and danger zones. The robot's state is represented as $(x, y, z, \theta_{\mathrm{p}}, \theta_{\mathrm{y}})$, where (x, y, z) is the robot's position, θ_{p} is the pitch angle, and θ_{y} is the yaw angle. The 3-D environment is discretized into a grid with 18×14 horizontal positions and 5 height levels. The pitch angle is discretized into 3 values, and the yaw angle, 8 values. The robot starts at the entrance to the environment, but does not know its initial state exactly. In each step, the robot can either rotate in place or move forward to one of the adjacent cells according to its heading. Rotation motion is quite accurate. However, when the robot moves forward, the robot reaches its intended state only 90% of the time; the rest of the time, it may remain in the same state or drift to the left or right of the intended state. The robot can localize itself near the landmarks.

The main difficulty in this task is similar to that of 2-D Navigation, but the state space is much larger because of the 3-D environment. A typical policy computed by MiGS takes the UAV through the rightmost route, which is longer but safer, because no danger zones are present and there are several landmarks along the way to aid localization.

Target Finding. A robot wants to find a moving target as quickly as possible in an environment



(a) (b) (c) (d)

Figure 5: Target finding. (*a*) The environment. Courtesy of the Radish data set. (*b*) The belief on the target's position, before the robot clears the bottom target region. (*c*) The belief, after the robot clears the bottom target region. (*d*) The belief, after the robot clears the top-left target region. The pink lines mark the boundaries of target regions. The shaded red regions mark the robot's belief on the target's position. Darker colors indicate higher probabilities. The red hollow triangle marks the target's actual position. See the caption of Figure 2 for the meanings of other labels.

containing obstacles and danger zones (Figure 5). The environment is represented as a uniform grid of size 29×49. The state space is parametrized by two variables $(x_{\mathrm{r}}, x_{\mathrm{t}})$, where x_{r} is the robot's position and x_{t} is the target's position. The target may lie in one of three regions. It can move freely within a region, but can not jump to a different region. The target's behavior is unknown to the robot. The robot knows its starting position only roughly. In each step, the robot can move to one of its eight adjacent cells. However, due to imperfect control, the robot fails to reach the intended cell 15% of the time. The robot can localize itself near a landmark. Due to sensor limitations, the robot can detect the target only when they are in the same grid cell.

The strategy generated by MiGS balances the effort of searching the target in a particular target region with the cost of moving between different target regions. The robot first searches the bottom target region, which is close to its starting position. It dispenses significant effort there until the probability of finding the target in this region becomes very small (Figure 5*b*). The robot then moves towards the upper target regions along the long, but safer path to the right, in order to avoid accidentally entering the danger zones. Once there, the robot first searches the upper left target region, because the prior probability of finding the target there is higher than that in the upper right target region (Figure 5*b*). However, the robot does not search the upper left target region as hard as the bottom target region. When it leaves the upper left region, there is still some probability that the target may be there (Figure 5*c*). The cost of moving between the upper left and right target regions are low. It is advantageous to search the upper right target region and then return to the upper left target region when necessary. Luckily, the robot finds the target in the upper right region.

Cooperative Navigation. Autonomous underwater vehicle (AUV) navigation in deep sea explo-

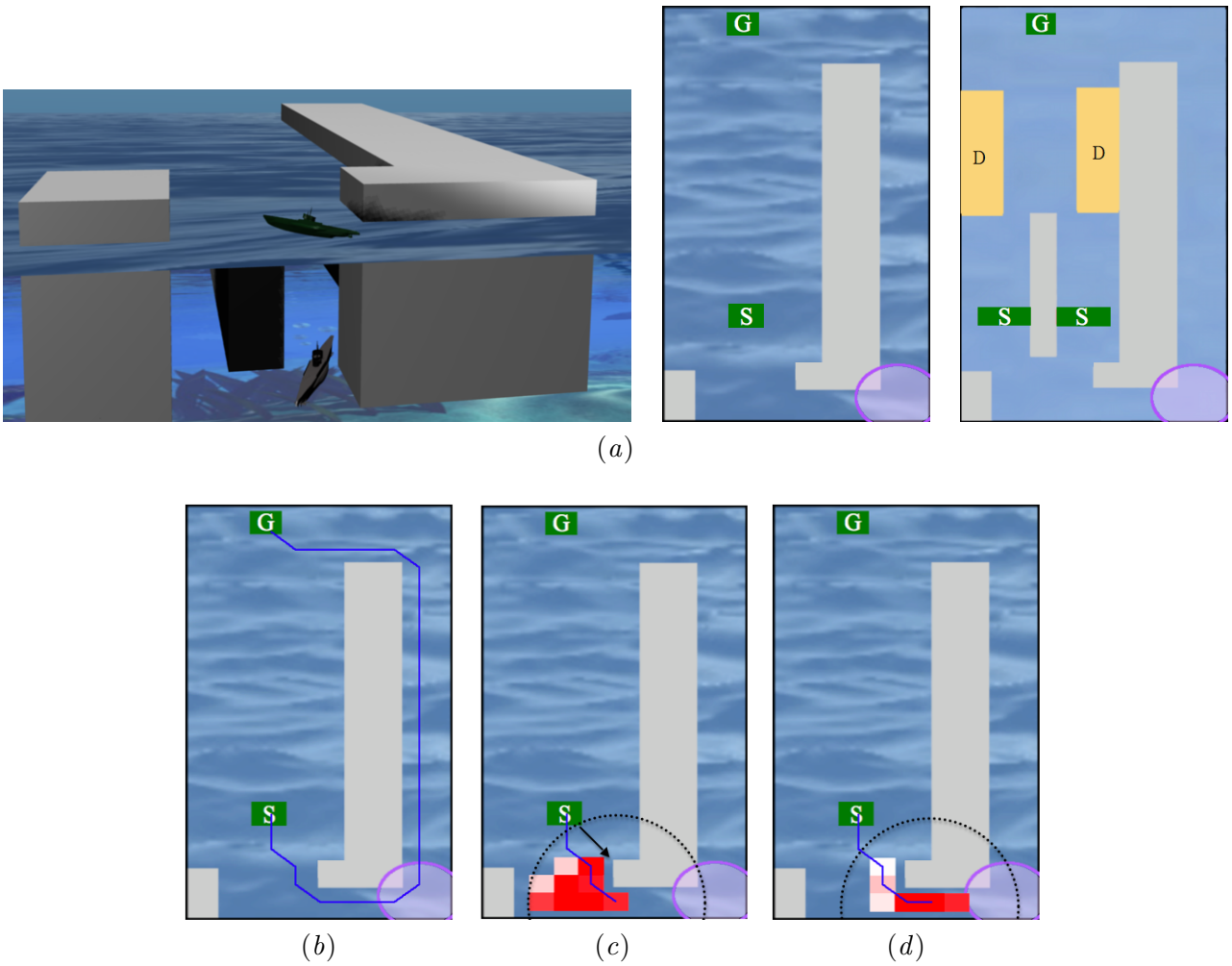


Figure 6: Cooperative Navigation. (a) The 3-D view (top), the top view (bottom left), and the bottom view (bottom right) of the environment. (b) The ASC's path in a simulation run. (c) Although the ASC is already at the southernmost position, it still moves southeast in order to reduce the chance that the AUV gets blocked by obstacles. The arrow points to the obstacle that potentially blocks the AUV. (d) The ASC moves straight east, when the chance of having the AUV blocked by obstacles is sufficiently small. The blue line indicates the ASC's path. The shaded red region indicates the ASC's belief on the AUV's position. Darker colors indicate higher probabilities. The dashed circle marks the boundary of the acoustic modem's effective range with respect to the current ASC's position. See the caption of Figure 2 for the meanings of other labels.

ration missions is a major challenge, because it is difficult to control the motion of AUVs accurately and AUVs often rely on dead-reckoning to estimate their positions. One interesting approach is to use autonomous surface crafts (ASCs) to guide AUV navigation [4]. Here, we model a simplified version of this cooperative navigation task. We want to find a motion strategy for an ASC to guide an AUV navigating from a start position to a goal position, even though the ASC does not know AUV's position exactly (Figure 6a).

The underwater environment where the AUV operates contains obstacles and danger zones. The surface environment where the ASC operates also contains obstacles. The obstacles layout and geometry in the underwater environment may differ from those in the surface environment. Both the surface and the underwater environment are represented as uniform grids of size 24×11.

The initial positions of the ASC and the AUV are not known exactly. In each step, the ASC moves to one of its eight adjacent cells. We assume that the ASC's control is sufficiently accurate and the error is negligible. In contrast, the AUV's con-

trol is quite noisy. In each step, it also moves to one of its eight adjacent cells, but reaches its intended cell only 60% of the time. For the rest of the time, it remains in its current cell or drifts to the left or the right of the intended cell.

To help the AUV navigates, the ASC is equipped with a GPS and an acoustic modem. The ASC transmits the GPS information and its own movement information continuously through the acoustic modem. When the AUV lies within a reliable transmission range of the ASC's acoustic modem, we assume that the AUV receives the GPS and movement information with perfect accuracy.

The AUV moves according to the following rules. When the AUV's estimated position is far from the boundary of the acoustic modem's range limit, the AUV replicates the ASC's action. When the AUV's estimated position is close to the acoustic modem's range limit, the AUV moves toward the ASC to ensure reliable signal reception. When the AUV lies beyond the reliable transmission range of the acoustic modem, we assume conservatively that the AUV receives no transmission and is lost. The AUV then moves in any direction with equal probability.

To help the ASC locates the AUV, the AUV uses an acoustic modem to transmit its own position. However, for efficiency, the AUV transmits only when it is able to localize itself well. The AUV localizes using GPS information from the ASC and from a few fixed transmission stations installed in the environment. Each fixed transmission station transmits its position continuously through an acoustic modem. Due to the acoustic modem's range limit, the AUV can only localize when it is sufficiently close to the ASC and the fixed transmission stations.

The policy generated by MiGS moves the ASC in such a way to help the AUV reaches the goal safely with high probability. To ensure the AUV's safety, the ASC deliberately takes a longer route (Figure 6b). The ASC also tries to ensure that the AUV is well within the reliable transmission range of the acoustic modem, based on its knowledge of the estimated AUV's position and AUV's behavior. For example, Figure 6c shows that the ASC keeps moving in the southeast direction, though it knows perfectly well from the GPS readings that it is already at the southernmost position. It would appear that moving east and making progress along the path towards the goal should be more effective. Closer examination of the ASC's belief on the AUV's position reveals why the ASC does not do so. If the ASC moves east, the AUV will replicate the action, but with substantial probability, it will get blocked by the obstacle and remain in the same position (Figure 6c). Now if the ASC continues moving east, the AUV will eventually fall out of the acoustic modem's transmission range and get lost. As a result, the ASC moves in the southeast direction for a number of steps. It moves east only after the probability of having the AUV blocked by the obstacle is sufficient small (Figure 6d).

6.2 Performance Comparison

We now compare MiGS quantitatively with other well-known approaches for motion planning and planning under uncertainty.

6.2.1 Experimental Setup

We ran three other planning algorithms: PRM [10], QMDP [25], and HSVI2 [22]. PRM is one of the most successful algorithms for motion planning without uncertainty. Results on PRM gives an indication of the importance of uncertainty planning in the tasks described in Section 6.1. We did not run PRM on the Target Finding task, because the classic PRM algorithm applies only to problems with static goals, while the target position, which serves as the goal in this task, moves. QMDP is a simple and yet practical approximate POMDP solver that is well known in robotics. Finally, HSVI2 is one of the fastest point-based POMDP solvers available.

All the algorithms were implemented in C++. For HSVI2, we used ZMDP v1.1.6., the latest software package released by HSVI2's original author. All the experiments were performed on a computer server with a 2.66GHz Intel processor, 2GB memory, and the Linux operating system.

For each task, we estimated the success rate of policies computed by MiGS. The success rate is the percentage of simulation runs in which the robot accomplishes a given task successfully within a specified time limit. Since MiGS is a randomized algorithm, we ran MiGS 30 times and performed 100 simulation runs for each resulting policy. The average success rate was then reported.

The performance of PRM was assessed in a similar way. For each task, we ran PRM 30 times. In each run, we first sampled a start state according

to the initial belief b_0 and sampled a goal state uniformly at random from the set of possible goal states. We then ran PRM to generate a path from the start to the goal state. Finally, we treated the generated path as an open loop policy and performed 100 simulation runs under the policy to estimate its average success rate.

QMDP and HSVI2 are both deterministic. We ran the algorithm until it either converged or reached a two hour limit. We then performed 3,000 simulation runs to estimate the success rate of the resulting policy.

6.2.2 Results

Table 1: The success rates of policies computed by various algorithms.

	PRM	QMDP	HSVI2	**MiGS**
2-D Navigation	0.0	0.0	0.0	**93.0**
3-D Navigation	14.4	0.1	24.0	**97.4**
Target Finding	—	11.7	12.2	**95.0**
Coop Navigation	0.0	0.0	0.0	**99.0**

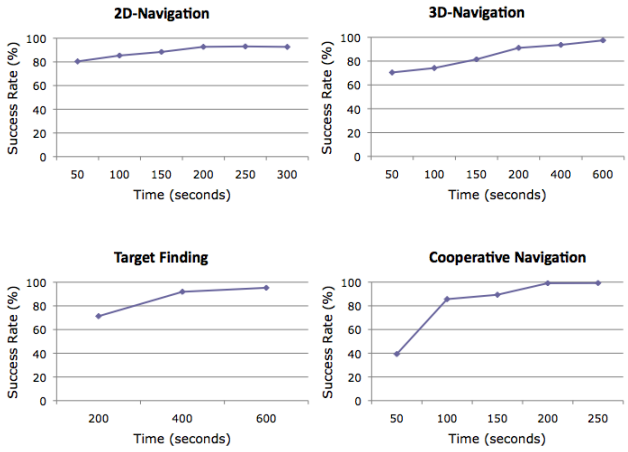


Figure 7: The success rate of MiGS policies with increasing running times.

The results for the four test tasks show that MiGS significantly outperforms the other algorithms under comparison (Table 1 and Figure 7). PRM performs badly, as it ignores uncertainty completely. Its failure indicates that handling uncertainty is a key issue in these tasks. QMDP essentially performs one-step lookahead search in the belief space. This is insufficient for tasks that require long planning horizons. While HSVI2 performs multi-step lookahead, it relies heavily on its heuristic to guide the search in the belief space. The heuristic is constructed from an MDP solution, which assumes that the system state is fully observable in the current and all future time steps. As the planning horizon increases, this assumption starts to fail, and the heuristic becomes less effective in guiding the belief space search. In contrast, using the action sequences in the roadmap, MiGS looks ahead much further in a hierarchical fashion.

It is interesting to note that in 3D-Navigation, even though PRM ignores uncertainty, it performs better than QMDP, out of "luck". If the UAV moves from the starting state directly towards the goal, it very likely passes through a danger zone, resulting in a failure. However, it may reach the goal successfully occasionally. PRM ignores all these and computes such a direct route, which leads to an average success rate of 14%. QMDP performs limited lookahead and recognizes the risk of this route. However, its one-step lookahead is not long enough to generate an alternative policy that brings the UAV to the goal consistently. Thus, the UAV ends up getting stuck in the start state, resulting in complete failure in terms of success rate. By performing longer lookahead, HSVI2 performs better than QMDP, and MiGS performs much better than both.

In Target Finding, both QMDP and HSVI2 produce policies that find the target successfully, whenever it is the lower target region. The lower target region is close to the robot's start position, and the planning horizon required to generate such a policy is relatively short. However, to generate policies capable of finding the target in the two upper target regions, we need a much longer planning horizon. As a result, the policies produced by QMDP and HSVI2 after 2 hours of computation only succeed around 12% of the time (Table 1), when the target is located in the lower target region. In contrast, the policies produced by MiGS achieve 95% success rate after 10 minutes of computation (Figure 7).

6.3 Robustness of MiGS Policies

For many real-world robotic tasks, it is not easy to build accurate models of robot control and sensing. For instance, the motion of an AUV depends heavily on the water current, whose effect is difficult to model accurately [13]. Thus it is desirable to have POMDP policies that are robust against

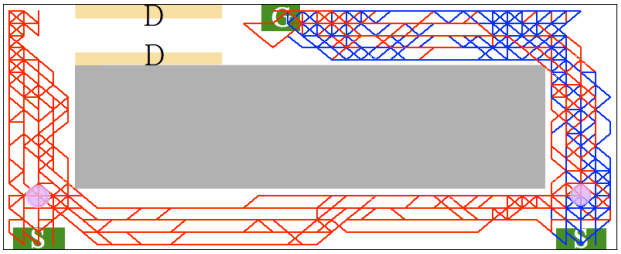


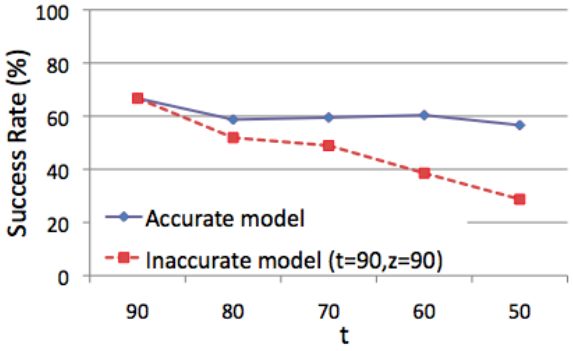
Figure 8: 2-D navigation in a simple geometric environment. The blue and red lines mark the robot's paths in simulation runs under $\pi_{90,90}$ from the two different initial positions, respectively. See the caption of Figure 2 for the meanings of labels.

such modeling errors.

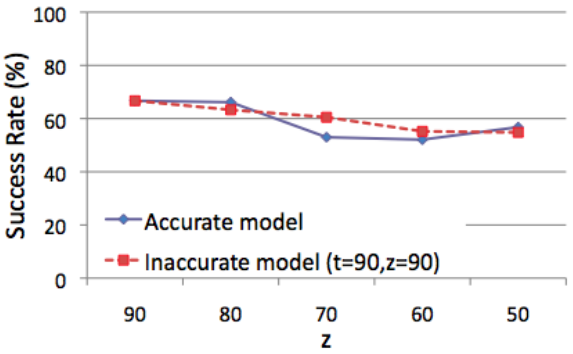
We performed preliminary tests on the robustness of policies computed by MiGS. We used a 2D-navigation task (Figure 8) similar to the one described in Section 6.1, but in a simpler geometric environment. The environment is represented as a uniform grid of size 20×42. In each step, the robot may move to one of eight adjacent cells and reaches the intended cell $t\%$ of the time. The robot can localize itself near landmarks, but it correctly identifies the landmark only $z\%$ of the time.

We applied MiGS to the POMDP model with $t = 90$ and $z = 90$. In other words, the model assumes quite accurate robot motion control and landmark identification. We simulated the resulting policy $\pi_{90,90}$, while systematically varying the values of t and z during the simulation, in order to examine the robustness of $\pi_{90,90}$, when $t \neq 90$ or $z \neq 90$. For comparison, we used MiGS to compute polices $\pi_{t,z}$ for POMDP models with correct values of t and z.

With decreasing t value, the uncertainty in robot control increases. It becomes more difficult for the robot to reach the goal reliably. A comparison of success rates for $\pi_{90,90}$ and $\pi_{t,z}$ suggests that small errors in the robot motion model do not have a significant effect here (Figure 9a). However, in relative terms, the performance of $\pi_{90,90}$ gradually degrades as the model errors increase. When the model errors are large, the reachable belief space under the model with $t = 90$ and $z = 90$ could be substantially different from that under the correct model. Thus, during the simulation, the robot may encounter a belief not close to any one considered during the planning. It then has to resort to a default action, which often causes poor performance. Figure 9b shows that $\pi_{90,90}$ is more tolerant of errors in the sensing model than those in the robot motion model. The reason is that the successful strategy commands the robot to move counter-clockwise along the hallway and approach the goal from the right, thus avoiding the danger zones. Localization using the landmarks makes this strategy more efficient to execute, but does not seem to play a critical role. Thus sensing model errors do not affect the success rate significantly. In summary, these results suggest that MiGS policies are robust against small model errors, though more tests are needed to confirm this.



(a)



(b)

Figure 9: The effect of model errors on the performance of MiGS policies. (a) The success rate with increasing error in the robot motion model. (b) The success rate with increasing error in the sensing model.

7 Conclusion

This paper proposes Milestone Guided Sampling (MiGS), a new point-based POMDP solver for robot motion planning tasks that require long time horizons. MiGS samples a set of milestones from a

robot's state space and constructs a roadmap as a simplified representation of the state space. Each edge of the roadmap represents an action sequence that brings the robot from one milestone to another. MiGS uses these action sequences in the roadmap, rather than single primitive actions, to sample the belief space, thus significantly reducing effective planning horizons while still capturing the important features of the belief space. We successfully tested MiGS in simulation on several difficult POMDPs modeling distinct robotic tasks in both 2-D and 3-D environments.

Two main obstacles have hindered the application of POMDPs to realistic robotic tasks: large state spaces and long time horizons. The recently introduced point-based algorithms have shown impressive progress in solving POMDPs with large state spaces. However, their performance degrades significantly when the planning task has a long time horizon. By alleviating the difficulty of long horizons, we hope that our work brings POMDPs one step closer to becoming a practical tool for robot motion planning in uncertain and dynamic environments.

Acknowledgments. We thank Sylvie Ong and Shao Wei Png for reading the first draft of this paper and helping with scripting a POMDP model. We also thank the anonymous reviewers whose many suggestions helped to improve the presentation of the paper. This work is supported in part by AcRF grant R-252-000-327-112 from the Singapore Ministry of Education. H. Kurniawati thanks N. M. Patrikalakis for his support.

References

[1] R. Alterovitz, T. Simeon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In *Proc. Robotics: Science and Systems*, 2007.

[2] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* The MIT Press, 2005.

[3] M. Erwig. The graph Voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.

[4] M.F. Fallon, G. Papadopoulos, and J.J. Leonard. Cooperative AUV navigation using a single surface craft. In *Proc. Field & Service Robotics*, 2009.

[5] K. Hsiao, L.P. Kaelbling, and T. Lozano-Perez. Grasping POMDPs. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4685–4692, 2007.

[6] D. Hsu, J.C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robotics Research*, 25(7):627–643, 2006.

[7] D. Hsu, W.S. Lee, and N. Rong. What makes some POMDP problems easy to approximate? In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[8] D. Hsu, W.S. Lee, and N. Rong. A point-based POMDP planner for target tracking. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2644–2650, 2008.

[9] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[10] L.E. Kavraki, P. Švestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. on Robotics & Automation*, 12(4):566–580, 1996.

[11] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*, 2008.

[12] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[13] B.H. Ooi, H. Zheng, H. Kurniawati, W. Cho, M.H. Dao, J. Wei, P. Zemskyy, P. Tkalich, P. Malanotte-Rizzoli, and N.M. Patrikalakis. Multi-vehicle oceanographic feature exploration. In *Proc. Int. Offshore & Polar Engineering Conf.*, 2009.

[14] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Int. Jnt. Conf. on Artificial Intelligence*, pages 1025–1032, August 2003.

[15] J. Pineau, G. Gordon, and S. Thrun. Policy-contingent abstraction for robust robot control. In *Proc. Uncertainty in Artificial Intelligence*, pages 477–484, 2003.

[16] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3–4):271–281, 2003.

[17] S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Proc. Int. Symp. on Robotics Research*, 2007.

[18] N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *J. Artificial Intelligence Research*, 23:1–40, 2005.

[19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[20] G. Shani, R.I. Brafman, and S.E. Shimony. Forward search value iteration for POMDPs. In *Int. Jnt. Conf. on Artificial Intelligence*, pages 2619–2624, 2007.

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

[22] T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Uncertainty in Artificial Intelligence*, July 2005.

[23] M.T.J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *J. Artificial Intelligence Research*, 24:195–220, 2005.

[24] G. Theocharous and L. P. Kaelbling. Approximate planning in POMDPs with macro-actions. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[25] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

A Proof of Theorem 1

Proof. The optimal value function V^* can be approximated arbitrarily closely by a piecewise-linear convex function and represented as $V^*(b) = \max_{\alpha\in\Gamma}(\alpha \cdot b)$ for a suitable set Γ of α-vectors. Let α and α' be the maximizing α-vectors at b and b', respectively. Without loss of generality, assume $V^*(b) \geq V^*(b')$. Thus $V^*(b) - V^*(b') \geq 0$. Since α' is a maximizer at b', we have $\alpha' \cdot b' \geq \alpha \cdot b'$ and $V^*(b) - V^*(b') = \alpha \cdot b - \alpha' \cdot b' \leq \alpha \cdot b - \alpha \cdot b' \leq \alpha \cdot (b - b')$. It then follows that

$$|V^*(b) - V^*(b')| \leq |\alpha \cdot (b - b')|. \tag{7}$$

Next, we calculate the inner product on the right-hand side of (7) over the partitioned state space:

$$\begin{aligned} |V^*(b) - V^*(b')| &\leq \Big|\sum_{s\in S} \alpha(s)(b(s) - b'(s))\Big| \\ &\leq \Big|\sum_{K\in\mathcal{K}} \sum_{s\in K} \alpha(s)(b(s) - b'(s))\Big| \end{aligned}$$

For any state s_K in the subset $K \in \mathcal{K}$. We have

$$\begin{aligned} &|V^*(b) - V^*(b')| \\ &\leq \Big|\sum_{K\in\mathcal{K}} \sum_{s\in K} (\alpha(s) - \alpha(s_K) + \alpha(s_K))(b(s) - b'(s))\Big| \\ &\leq \Big|\sum_{K\in\mathcal{K}} \sum_{s\in K} (\alpha(s) - \alpha(s_K))(b(s) - b'(s))\Big| \\ &\quad + \Big|\sum_{K\in\mathcal{K}} \sum_{s\in K} \alpha(s_K)(b(s) - b'(s))\Big|. \end{aligned} \tag{8}$$

Let e_1 and e_2 denote the two terms in (8), respectively. We now bound e_1 and e_2 separately. By the definition of ϵ-partitioning, $|\alpha(s) - \alpha(s_K)| \leq \epsilon$ for all s and s_K in $K \in \mathcal{K}$. Thus,

$$\begin{aligned} e_1 &\leq \sum_{K\in\mathcal{K}} \sum_{s\in K} \epsilon |b(s) - b'(s)| \\ &= \epsilon \sum_{s\in S} |b(s) - b'(s)| \leq 2\epsilon, \end{aligned} \tag{9}$$

where the last inequality holds because the L_1 distance between any two beliefs is no greater than 2. Let us now consider e_2. Since the absolute values of α-vector coefficients are no more than $R_{\max}/(1-\gamma)$, it follows that

$$\begin{aligned} e_2 &\leq \sum_{K\in\mathcal{K}} \Big|\alpha(s_K)\Big|\Big|\sum_{s\in K} (b(s) - b'(s))\Big| \\ &\leq \sum_{K\in\mathcal{K}} \frac{R_{\max}}{1-\gamma}\Big|\sum_{s\in K} b(s) - b'(s)\Big|. \end{aligned} \tag{10}$$

Using the condition $d_{\mathcal{K}}(b, b') \leq \delta$, we get $e_2 \leq \frac{R_{\max}}{1-\gamma}\delta$. Combining this with (8) and (9) gives the desired result. $\square$