Noname manuscript No.
(will be inserted by the editor)

Global Motion Planning under Uncertain Motion, Sensing, and Environment Map

Hanna Kurniawati ·
Tirthankar Bandyopadhyay ·
Nicholas M. Patrikalakis

Received: date / Accepted: date

Abstract Uncertainty in motion planning is often caused by three main sources: motion error, sensing error, and imperfect environment map. Despite the significant effect of all three sources of uncertainty to motion planning problems, most planners take into account only one or at most two of them. We propose a new motion planner, called *Guided Cluster Sampling (GCS)*, that takes into account all three sources of uncertainty for robots with active sensing capabilities. GCS uses the Partially Observable Markov Decision Process (POMDP) framework and the point-based POMDP approach. Although point-based POMDPs have shown impressive progress over the past few years, it performs poorly when the environment map is imperfect. This poor performance is due to the extremely high dimensional state space, which translates to the extremely large belief space B. We alleviate this problem by constructing a more suitable sampling distribution based on the observations that when the robot has active sensing capability, B can be partitioned into a collection of much smaller sub-spaces, and an optimal policy can often be generated by sufficient sampling of a small subset of the collection. Utilizing these observations, GCS samples B in two-stages, a subspace is sampled from the collection and then a belief is sampled from the subspace. It uses information from the set of sampled sub-spaces and sampled

Hanna Kurniawati
School of Information Technology & Electrical Engineering, University of Queensland
Queensland, Australia
E-mail: hannakur@uq.edu.au
Most of the work were done while the author was with the Center for Environmental Sensing and Modeling, Singapore–MIT Alliance for Research and Technology.

Tirthankar Bandyopadhyay
Future Urban Mobility, Singapore–MIT Alliance for Research and Technology
Singapore, Republic of Singapore
E-mail: tirtha@smart.mit.edu

Nicholas M. Patrikalakis
Department of Mechanical Engineering – Center for Ocean Engineering, MIT
Center for Environmental Sensing and Modeling, Singapore–MIT Alliance for Research and Technology
Massachusetts, USA
E-mail: nmp@mit.edu

beliefs to guide subsequent sampling. Simulation results on marine robotics scenarios suggest that GCS can generate reasonable policies for motion planning problems with uncertain motion, sensing, and environment map, that are unsolvable by the best point-based POMDPs today. Furthermore, GCS handles POMDPs with continuous state, action, and observation spaces. We show that for a class of POMDPs that often occur in robot motion planning, given enough time, GCS converges to the optimal policy. To the best of our knowledge, this is the first convergence result for point-based POMDPs with continuous action space.

1 Introduction

Uncertainty in robot motion planning are often caused by three main sources: motion error, sensing error, and imperfect environment map. These errors are no longer negligible when robots leave the highly structured environment, such as factory plants, to operate in our homes, offices, and outdoor environment. Robot motion and sensors are subject to various noise. Although nowadays system noise is often small and negligible, outside disturbances are beyond our control and often cause substantial noise. For instance, water currents highly accentuates motion errors of marine robots, snapping shrimps dominates noise in sonar measurement, etc. Furthermore, imperfect environment map is unavoidable as maps of our living spaces are acquired through sensors subject to various substantial noise. Although all three sources of uncertainty significantly affects motion planning problems, most planners take into account only one [5,8,16] or at most two sources of uncertainty [4,22]. This paper presents a new motion planner that takes into account all three sources of uncertainty for a robot with active sensing capability.

Our new planner uses the Partially Observable Markov Decision Process (POMDP) framework. POMDP is a mathematically principled and general framework for planning under uncertainty. Due to uncertainty, a POMDP agent never knows its exact state and hence can not decide the best action to perform based on a single state. Therefore, a POMDP agent decides its action based on a set of states that are consistent with the available information. It represents a set of possible states as a distribution over the state space, called a belief. A POMDP agent plans in the belief space B, which is the set of all possible beliefs.

Although solving a POMDP is computationally intractable in the worst case [17], recent development of point-based POMDPs [15,19,23] have drastically increased the speed of POMDP planning. Key to point-based POMDPs is to sample a set of representative beliefs from B and plan with respect to the set of sampled beliefs, instead of the entire B. By doing so, point-based POMDPs can generate a good approximation to the optimal solution for motion planning problems with moderate difficulty, within a few minutes [10,14,15,23].

However, even the best point-based POMDPs perform poorly in motion planning with imperfect environment map, due to these three major challenges. First is the curse of dimensionality. When both the environment map and robot configuration are not perfectly known, we need to define the state space S as a joint product between the configuration space and the space of possible environment maps. Setting the environment map as part of the state variables causes the dimension $dim(S)$ of

S to be extremely high. Consider a simplistic problem of a 2-DOFs robot operating in an environment containing two triangular obstacles where the positions of the vertices are not known perfectly. $Dim(S)$ is already 14 ! And 12 of them are contributed by the space of possible environment maps. This dimensionality issue is aggravated in POMDPs, as they plan in the belief space B whose size is doubly exponential in $dim(S)$. It is true that point-based POMDP plans with respect to a sampled representation of B. But, the sampled representation must be representative enough before a good approximation to the optimal policy can be generated. And in general, the larger B is, the more difficult it is to find the set of representative beliefs. As a result, even the best point-based POMDPs today [15,23] perform poorly on problems with uncertain environment map.

Second is the long planning horizon typical of motion planning problems. 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.

Third is continuous control space. Most motion planning problems have continuous action space, but most POMDP planners assume discrete action space. Although we can discretize the continuous control space by sampling, no convergence results is known, because a POMDP planner computes a *max* over the control space. While sampling has been shown to approximate the *average* operator well, almost no result is known on how well sampling approximates the *max* operator.

To alleviate the above three challenges for a robot that localizes itself through active sensing, we propose a new point-based POMDP planner, called *Guided Cluster Sampling (GCS)*.

To alleviate the dimensionality issues, GCS constructs a more suitable sampling distribution based on two observations. First, the optimal policy often consists of a small number of sensing actions. Second, only after a sensing action is performed, that the robot's understanding about its environment changes. Suppose $\mathscr{R}^*(b_0)$ is the set of beliefs that are reachable under an optimal policy from a given initial belief b_0. The above observations mean that if we partition B into a collection of sub-spaces, where each sub-space corresponds to the set of all beliefs with the same marginal distribution of environment maps, then $\mathscr{R}^*(b_0)$ lies in a small subset of the collections. Since we can generate a good approximation to the optimal policy from sufficient samples of $\mathscr{R}^*(b_0)$ [12], the above observations imply that although B is huge, a representative set of B is likely to be small and lie in a small set of much smaller subspaces of B.

Utilizing the above observations, GCS partitions B into subspaces based on the marginal distributions of environment map, and samples B by sampling a sub-space from the partition, and then sampling a belief from the sampled sub-space. Of course ideally, we would like to sample only from $\mathscr{R}^*(b_0)$. But since $\mathscr{R}^*(b_0)$ is not known a priori, we use heuristics based on information from the sampled sub-spaces and sampled beliefs to guide subsequent sampling.

To alleviate the long planning horizon issue, GCS adopts the strategy in [14]. It reduces the effective planning horizon by using action sequences, instead of a single primitive action, to guide sampling B.

GCS handles continuous state, control, and observation spaces by using sampled representations. We show that for a class of POMDPs that often occur in robot motion planning, given enough time, GCS converges to the optimal policy. To the best of our knowledge, this is the first convergence result for point-based POMDPs with continuous control space.

2 Background and Related Work

2.1 Motion planning under uncertainty

In motion planning, uncertainty arises from three main sources, i.e., imperfect control, imperfect sensing, and imperfect information about the environment. However, most motion planners take into account only one or two sources of uncertainty. For instance, Stochastic Motion Roadmap [1] takes into account only control uncertainty. The work in [5] takes into account only sensing uncertainty, while the work in [8, 16] take into account only imperfect information about the environment. LQG-MP [4] and Belief Roadmap [22] take into account two sources of uncertainty, i.e., control and sensing uncertainty, but restrict the uncertainty to be Gaussian. Our new planner takes into account all three sources of uncertainty and allows any type of distribution with bounded support for control, sensing, and environment map uncertainty.

A few work that take into account all three sources of uncertainty during planning [13,24] are designed specifically for exploration task. The work in [13] restricts control, sensing, and map uncertainty to be Gaussian, while [24] finds an approximate solution using a greedy one-step lookahead method, which is often inadequate for general motion planning problems as they often require long planning horizon.

Motion planning under uncertain control, sensing, and environment map is essentially a POMDP problem, and general POMDP planners can conceptually be used. However, despite the impressive progress of such planners [15,19,23], they face significant difficulties when the environment map is uncertain. Recently, [7] alleviates these difficulties by using online POMDP. However, this strategy does not perform global planning in the belief space. As a result, it may not converge to the optimal policy. Our planner performs global planning in the belief space by utilizing domain specific properties.

2.2 POMDP Background

Formally, a POMDP is specified as a tuple $\langle S,A,O,T,Z,R,\gamma\rangle$, where S is the set of states, A is the set of actions, and O is the set of observations. 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 density function $T(s,a,s') = f(s'|s,a)$. The agent may then receive an observation. Due to the uncertainty in observation, the observation result $o \in O$ is again modeled as a conditional probability density function $Z(s',a,o) = f(o|s',a)$. In each step, the agent receives a reward $R(s,a)$, if it takes action a in state s. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions. When the sequence of actions has infinite length, we specify a discount factor $\gamma \in (0,1)$ so that the total reward is finite and the problem is well defined.

A POMDP planner computes an *optimal policy* that maximizes the agent's expected total reward. A POMDP policy $\pi\colon B \to A$ is a mapping from 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 $V_\pi(b) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | b, \pi\right]$ specifies the expected total reward of executing policy π. A policy can be represented by various representations, e.g., policy graph [2], α-function [21], and pairs of a belief and an action [25]. GCS can use any policy representation for continuous S and O.

To execute a policy π, an agent executes action selection and belief update repeatedly. For example, if the agent's current belief is b, it selects the action referred to by $a = \pi(b)$. After the agent performs action a and receives an observation o according to the observation function Z, it updates b to a new belief b' given by

$$b'(s') = \tau(b,a,o) = \eta Z(s',a,o) \int_{s\in S} T(s,a,s')b(s)ds \quad (1)$$

where η is a normalization constant.

2.3 Point-based POMDP

Point-based POMDPs trade optimality with approximate optimality in exchange for speed. It reduces the complexity of planning in B by representing B as a set of sampled beliefs and planning with respect to this set only. To generate a policy, most point-based POMDPs use value iteration, utilizing the fact that the optimal value function satisfies Bellman equation. They start from an initial policy, represented as a value function V. And iteratively perform Bellman backup on V at the sampled beliefs, until the iteration converges. Over the past few years, impressive progress have been gained by improving the strategy for sampling B [15, 19, 23].

Despite the impressive progress, even the best POMDP planners face significant difficulties in solving motion planning under uncertain environment map. GCS alleviates these difficulties by constructing a more suitable sampling strategy based on domain specific properties.

Furthermore, most point-based POMDPs are designed for discrete state, action, and observation spaces. Although a few [9, 21] handle continuous state, action, and observation spaces, they do not guarantee convergence to the optimal policy when the action space is continuous. We guarantee convergence to the optimal policy for a class of POMDP problems with continuous state, action, and observation spaces, that often occur in robot motion planning.

3 Problem formulation

3.1 POMDP formulation

Let's first consider a robot operating in a 2D environment populated by polygonal obstacles. GCS can be extended directly to 3D environment, but for simplicity we only discuss 2D environment. We know the exact number of polygons, the exact number of vertices in each polygon, and the exact connectivity between the vertices in each polygon. However, we do not know the exact position of the vertices. We represent this uncertainty as probability distributions with bounded support (e.g., Figure 1).

We represent each possible environment map as a feature map, where each feature corresponds to an obstacle's vertex or a goal feature. A goal feature is a mark in the environment, indicating the robot's goal position. Suppose the features are numbered sequentially from 1 to n and suppose E_i is the set of all possible positions of feature-i. Then, the domain of each E_i is a bounded subset of $\mathbb{R}^2$ and the set E of all possible environment maps is defined as $E_1 \times \cdots \times E_n$. To cope with uncertainty in both the environment map and the robot's configuration, the POMDP state space S is defined as the joint product between the robot's configuration space Q and E, i.e., $S = Q \times E = Q \times E_1 \times \cdots \times E_n$.

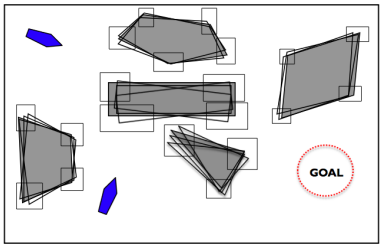


Fig. 1 The robot is a pentagon. The two blue pentagons represent the possible initial configurations of the robot. The grey polygons are obstacles. The position of each vertex is not perfectly known, and maybe anywhere within the rectangular region.

To model active sensing mechanism, the action space A consists of two subsets, i.e., control set U and $\{sensing\}$. A control action $u \in U$ moves the robot with some error. These errors are represented in the transition function as a probability distribution with bounded support. When a control action is performed, no observation is perceived. When a sensing action is performed, the robot does not move but perceives observations about the features positions, according to the observation function.

The observation space O is the set of all possible perceived positions of the features, i.e., $O = \prod_{i=1}^{n} O_i$, where O_i is the set of all possible perceived positions of feature-i and n is the number of features in the environment map. We set $O_i = \text{na}$ whenever the robot does not see feature-i. A perceived position of the features is the result of processing raw sensor data, which includes handling the data association problem. Any data association method can be used, including multiple hypothesis methods which generate multi-modal distribution for perceived features positions. Data association is a large domain in itself and is outside the scope of this paper. The perceived positions are defined with respect to the robot's local coordinate system. Notice that our observation space is exponential in the dimension of E. This large observation space significantly contributes to the difficulty of solving the POMDP problem, as we will discuss in Section 4.

The observation function Z is a conditional distribution function with bounded support. It represents two types of uncertainties. First is *visibility uncertainty*, which indicates the probability that the robot sees a feature that lies in its visibility region. It is represented as a discrete conditional distribution Z_{vis}, defined as $Z_{vis}(\langle q, e_i \rangle, sensing, see(o_i)) = P(see(o_i) \mid e_i \in Vis(q))$, where $see(o_i) = 1$ if the robot perceives feature-i and 0 otherwise, and $Vis(q)$ is the visibility region of the robot at configuration $q \in Q$. The second type of uncertainty is *measurement uncertainty*, that models the perceived position of the features given a state. It is represented as a continuous conditional density function $Z_{measure}$. The observation function Z is then defined as

$$\begin{aligned} Z(s, sensing, o) &= Z(\langle q, e\rangle, sensing, (o_1, \ldots, o_n)) \\ &= \prod_{i=1}^{n} Z_{measure}(\text{feature-}i \text{ is seen at } o_i \mid \langle q, e_i\rangle, see(o_i)) \times \\ &\qquad Z_{vis}(\langle q, e_i\rangle, sensing, see(o_i)) \end{aligned} \tag{2}$$

where n is the number of features. When feature-i is not seen, $o_i = \text{na}$. Various sensors constraints, such as the range and bearing limit, can be modelled easily in Z. And once Z models these constraints, GCS automatically takes them into account during planning. Now, since observations are defined with respect to the robot's local coordinate system, the function Z depends only on the relative configuration of the environment with respect to the robot. This relative dependency is used by GCS to construct a more effective sampling distribution (see Section 5.1).

We design the reward function R as an objective function for reaching a goal region with minimum cost. The goal region is the set of points within a small prespecified distance from a goal feature. The cost is the sum of moving/sensing cost and collision cost. For any state $s \in S$ and any action $a \in A$, we define the reward function as $R(s,a) = R_{goal}(s) + R_{action}(s,a) + R_{collision}(s,a)$, where $R_{goal}(s)$ is the reward for reaching a goal region, $R_{action}(s,a)$ is the cost of performing a from s, and $R_{collision}(s,a)$ is the expected collision cost of moving according to a from s. Notice that R depends only on the action performed and the relative position between the environment features and the robot. GCS uses this property to generate a more effective sampling guide (see Section 5.1).

3.2 Continuity properties

Unlike most work in POMDP planners, in this paper, S, U, and O are all continuous. Continuous spaces are more natural for modeling robotics problems. To represent these continuous spaces, GCS uses sampled representations. It represents beliefs using weighted particles. GCS can use any policy representations for POMDP with continuous state space, e.g., point representation [25] and policy graph [2].

Now, we define the continuity properties of R, Z, and T with respect to S and A, so that convergence to the optimal policy can be guaranteed even when A is approximated with its sampled representation. The approximation result is in Section 7. The properties with respect to S are,

Definition 1 Suppose $\langle S, A, O, T, Z, R, \gamma\rangle$ is a POMDP with continuous S. Let D_S be a metric in S. The POMDP is *LS*-continuous with parameter (K_{RS}, K_Z), $K_{RS}, K_Z \in \mathbb{R}$, when:

1. The motion uncertainty is the same everywhere. For any displacement vector d on S and any $s, s' \in S$, $T(s,a,s') = T(s+d,a,s'+d)$.
2. The observation function Z is Lipschitz continuous in S. For any $s, s' \in S$, any $a, a' \in A$, and any $o \in O$, $|Z(s,a,o) - Z(s',a,o)| \leq K_Z \cdot D_S(s,s')$.
3. The reward R is Lipschitz continuous in S. For any $s, s' \in S$ and any $a \in A$, $|R(s,a) - R(s',a)| \leq K_{RS} \cdot D_S(s,s')$.

Property-1 may seem odd as it means that the robot may go through an obstacle. However, notice that in motion planning, the modeling of actual physical dynamics during collision is essentially a way to generate motion strategy with low collision cost. We can generate the same strategy without modelling the effect of collision in T, by setting a high penalty for collision. This trick is similar to the use of potential function in deterministic motion planning. Property-1 simplifies proving the convergence result (Section 7), as transition from all states can be treated equally.
Property-2 means that the nearby states generates similar observations. This is a common assumption in robotics.
Property-3 means that the robot receives similar immediate reward when it performs the same action from nearby states. Since Lipschitz condition is closed under summation, to satisfy this property, we only need to ensure that each component of R is Lipschitz. This can be satisfied easily, for instance by setting R_{goal} to be linearly increasing as the robot becomes closer to the goal region, R_{action} for the same action to be the same everywhere, and $R_{collision}$ to be linearly increasing as the robot becomes closer to an obstacle. This reward function is similar to a potential function used in motion planning [6].

Now, we define the continuity properties of R, Z, and T on A.

Definition 2 Suppose $\langle S,A,O,T,Z,R,\gamma\rangle$ is a POMDP where S is continuous. And A can be partitioned into a finite collection $\mathscr{P}$ of disjoint continuous sets, where two actions $a,a' \in A$ are in the same set $P \in \mathscr{P}$ whenever $O_a \bigcap O_{a'} \neq \emptyset$, O_a is the set of observations that can be perceived when $a \in A$ is performed, i.e., $O_a = \{o \in O \mid \exists s \in S \quad Z(s,a,o) > 0\}$. Suppose D_S and D_P are metrics on S and on set $P \in \mathscr{P}$. Then, the POMDP is *LA*-continuous with action partition $\mathscr{P}$ and parameters (K_{RA},h) when the three properties below hold. The parameter $K_{RA} \in \mathbb{R}$, while h is an increasing function with $h(0) = 0$, that maps distance in A to distance in S. The properties are:

1. For any $P \in \mathscr{P}$, any $a,a' \in P$, and any $s,s' \in S$, $T(s,a,s') = T(s,a',s' + f(a,a'))$, where f is a function that maps a pair of actions in A to a displacement vector in S, such that $D_S(s,s+f(a,a')) \leq h(D_P(a,a'))$.
2. The observation function is the same for any action in the same set of $\mathscr{P}$. For any $o \in O$, any $s \in S$, any $P \in \mathscr{P}$, and any $a,a' \in P$, $Z(s,a,o) = Z(s,a',o)$.
3. The reward R is Lipschitz continuous on each set in $\mathscr{P}$. For any $s \in S$, any $P \in \mathscr{P}$, and any $a,a' \in P$, $|R(s,a) - R(s,a')| \leq K_{RA} \cdot D_P(a,a')$.

An example of $\mathscr{P}$ in our POMDP model is $\{U,\{sensing\}\}$. Property-1 means that when nearby actions in the same element of $\mathscr{P}$, are applied to the same state, the resulting states would be close too. Property-2 means that within the same action set $P \in \mathscr{P}$, the observation that can be perceived by the robot depends only on the robot's configuration and features' positions. Property-3 is similar to property-3 of Definition 1, but applied to A.

4 Overview of GCS

Key to point-based POMDPs is to sample a representative set of beliefs from the belief space B and use it as an approximate representation of B. For computational

efficiency, most recent point-based POMDPs sample from the set $\mathscr{R}(b_0)$ of points reachable from b_0, instead of the entire B. The sampled beliefs are represented as a belief tree T, where each node represents a sampled belief and the root is b_0. To sample a new belief, these planners sample a node $b \in T$, an action $a \in A$, and an observation $o \in O$ according to suitable probability distributions or heuristics. They then compute $b' = \tau(b,a,o)$ using (1) and inserts b' into T as a child of b in T. If a POMDP problem requires h action and observation steps as the effective planning horizon, the size of the complete belief tree —*the effective sampling domain*— is $O(|A|^h|O|^h)$ where $|A|$ and $|O|$ are the size of the action and observation spaces. This effective sampling domain is often much smaller than B, and hence significantly reduces the difficulty in sampling a representative set of beliefs. However in our case, the size of the effective sampling domain is still huge, as $|O|$ is exponential in the number of environment features and h tends to be large.

Having a huge effective sampling domain may pose two difficulties in solving the POMDP problem. One is in finding the set of representative beliefs. This is especially true when the set of representative beliefs is small. In this case, finding the set of representative beliefs is similar to the narrow passage problem that has often degraded the quality of sampling-based methods in deterministic path planning. However, once the set of representative beliefs is found, generating an optimal policy can be done fast. Another difficulty is in generating the optimal policy. The huge effective sampling domain can imply that the set of representative beliefs is large too. When this happens, point-based methods would face difficulties in generating the (approximately) optimal policy even after the set of representative beliefs is sampled, as they still need to perform backups —the most costly primitive operation in point-based POMDPs— on a large number of points.

Active sensing problems often pose only the first difficulty, which means that point-based method is still a viable option. Although the effective sampling domain is huge, a set of representative beliefs, i.e., one that enables the generation of an optimal policy, often lies in a much smaller subspace of B. The results in [12] shows that an optimal policy can be generated by sufficient sampling from the set $\mathscr{R}^*(b_0)$ of beliefs that are reachable under an optimal policy from the initial belief b_0. In active sensing problems, $\mathscr{R}^*(b_0)$ often lie in a much smaller subspace of B. Active sensing is often applied to problems with high sensing cost, because if the sensing cost is low, continuous sensing is acceptable and active sensing is not needed. Due to the high sensing cost, the optimal policy of many active sensing problems contains only a small number of sensing actions. Since a robot's belief over its environment changes only when a sensing action is performed, in-between two scanning actions, the robot's belief may change only to another belief that has the same marginal distribution of the environment space E. Therefore, if we partition B into sub-spaces, where each sub-space corresponds to the set of all beliefs $b \in B$ that have the same marginal distribution of E, then $\mathscr{R}^*(b_0)$ for active sensing problems often lie in a small set of these sub-spaces. More formally, if we define a partitioning $\mathscr{C}$ of B as,

Definition 3 The partition $\mathscr{C}$ of B is a collection $\{C(b_E^1), C(b_E^2), C(b_E^3), \dots\}$ of disjoint sub-spaces, where each $C(b_E) = \{b \in B \mid \int_{q \in Q} b(\langle q, e \rangle) dq = b_E, e \in E\}$.

Then, $\mathscr{R}^*(b_0)$ often lie in a small subset of $\mathscr{C}$. Since $dim(C(b_E))$ of any $C(b_E) \in \mathscr{C}$ is doubly exponential in $dim(Q)$, instead of the much larger $dim(S)$, $\mathscr{R}^*(b_0)$ lies in a much smaller subspace of B. Therefore, we can often find a representative set of beliefs by sampling from a small subset of $\mathscr{C}$.

Utilizing the above observations, GCS alleviates the difficulties due to the huge B and huge effective sampling domain by sampling a small number of relevant subspaces from $\mathscr{C}$ and then sampling beliefs only from the sampled subspaces. Of course ideally, the sampled beliefs are all in $\mathscr{R}^*(b_0)$. However, since $\mathscr{R}^*(b_0)$ is not known a priori, GCS samples B incrementally. It uses heuristics based on environment distributions in the sampled subspaces and the sampling history, to guide sampling subsequent subspaces and to guide sampling representative beliefs in each subspace. To further alleviate the planning complexity, GCS reduces the planning horizon by using action sequences, instead of primitive actions, to guide sampling B.

Similar to most point-based POMDPs, GCS constructs a belief tree T, but expands it using the above idea. In T, sampling a small number of sub-spaces in $\mathscr{C}$ is the same as limiting the number of sensing–observation branches to be small. For this purpose, GCS reduces the number of expansion using a sensing action, and reduces the number of observation branches after each sensing action. To reduce the number of expansion using a sensing action, GCS expands T using two types of action sequences, i.e., action sequences with a single sensing action located at the end and sequences with no sensing action. To reduce the number of observation branches after a sensing action, GCS samples observations that could be perceived by the robot and results in beliefs that are significantly different than other beliefs in T. GCS uses a more suitable metric (Section 5.3) to decide how different two beliefs are.

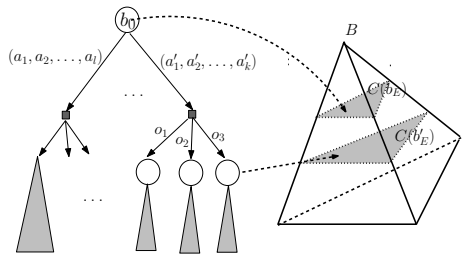


Fig. 2 The belief tree T (left) and the belief space B (right) it represents. Each node in T lies in an element of $\mathscr{C}$. Suppose $b \in T$ lies in $C(b_E) \in \mathscr{C}$. Each out-edge of a node b corresponds to a path in $C(b_E)$.

The overall algorithm on how GCS expands T is in Algorithm 1. Given a node $b \in T$ to expand, GCS finds a subspace $C(b_E) \in \mathscr{C}$ that contains b (line 4). It uses the environment map distribution b_E as a guide to generate an action sequence (line 5) that leads to the goal state with high probability and low collision cost, or to a belief where useful sensing data can be gained with high probability. The node b is then expanded using the generated action sequence. To expand $b \in T$ using an action sequence $(a_1, a_2, \ldots, a_l)$, GCS iteratively apply (1) to b. Specifically, $\tau(b, (a_1, a_2, \ldots, a_l), noObservation)$ (line 6) means that it computes a sequence of beliefs $(b_1, b_2, \ldots, b_l)$, where $b_1 = \tau(b, a_1, noObservation)$ and $b_i = \tau(b_{i-1}, a_i, noObservation)$ for $i \in [2, l]$, and returns the last belief b_l. A new belief is inserted into T only when the distance between the new belief and its nearest node in T is more than a given threshold. The action edge that connects b with the newly inserted belief is then annotated with the action sequence. Figure 2 shows an illustration of T. Line 7–16 presents the difference of expanding b using an action sequence that contains a sensing action at the end and a sequence that contains no sensing action.

Algorithm 1 Guided Cluster Sampling (N)

1: Initialize T by setting b_0 as the root.
2: **for** i = 1 to N **do**
3: b = Sample a node from T inversely proportional to the number of times b has been sampled.
4: $\langle \mathbf{0}, b_E \rangle$ = Transform(b).
5: $(a_1, a_2, \ldots, a_l)$ = GenerateAnActSeq(b_E, V(b), M).
6: Let $b_{l-1} = \tau(b_1, (a_1, \ldots, a_{l-1}), noObservation)$.
7: **if** a_l is a sensing action **then**
8: Let $O' \subset O$ be the set of sampled observations.
9: **for** each $o \in O'$ **do**
10: $b_l = \tau(b_{l-1}, sensing, o)$.
11: **if** $\min_{b' \in T} D_B(b_l, b') > \varepsilon$ **then**
12: Insert b_l into T as a child of b.
13: **else**
14: Let $b_l = \tau(b_{l-1}, a_l, noObservation)$
15: **if** $\min_{b' \in T} D_B(b_l, b') > \varepsilon$ **then**
16: Insert b_l into T as a child of b.
17: **for all** b' = children of b **do**
18: BACKUP(b').

5 Guided Cluster Sampling (GCS)

5.1 Finding a subspace in $\mathscr{C}$ that contains b

To find a subspace in $\mathscr{C}$ that contains b, GCS first transforms the environment map to the robot's local coordinate system (line-4 of Algorithm 1). More precisely, let $g : S \to S$ be a many to one function where $g(s) = g(\langle q, e \rangle) = \langle \mathbf{0}, \Phi(e) \rangle$, $\mathbf{0}$ is the origin of the robot's coordinate system, and $\Phi(e)$ is a function that transforms the position of each environment feature to the robot's local coordinate system. The transformation for beliefs is then defined as follows. Suppose $b' = Transform(b)$, then

$$b'(s') = \int_{s \in S} b(s) \cdot I(s, s') ds \quad \text{where } I(s, s') = \begin{cases} 1 \text{ if } g(s) = s'. \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

The transformed belief b' has probability one that the robot's configuration is at $\mathbf{0}$. This transformation transforms the robot's uncertainty to the environment map's uncertainty, such that the marginal distribution b_E of E in b' incorporates the uncertainty in both the robot's configuration and the environment map. As a result, the sampling guide constructed using information from $C(b_E) \in \mathscr{C}$ becomes more effective. Details on the sampling guide is in Section 5.2.

Now, the question is would $Transform$ change the computed optimal policy. The answer is no, more precisely,

Theorem 1 *A POMDP $\langle S, A, O, T, Z, R, \gamma \rangle$ that is LS-continuous, where R and Z depend only on the relative configuration of the robot with respect to the environment, $V^*(b) = V^*(Transform(b))$ for any belief $b \in B$.* [1]

This theorem means that both the belief and its transformation converge to the same optimal value, and therefore can be used interchangeably in computing the optimal solution.

[1] The proofs of all theorems are in Appendix.

5.2 Generating action sequences

Suppose $C(b_E) \in \mathscr{C}$ is the sub-space of B that contains the node $b \in T$ to be expanded. And suppose Q_{b_E} is the robot's configuration space Q, where the forbidden regions in Q are uncertain and are caused by obstacles whose positions in the workspace are distributed according to b_E. Then, the action sequences for expanding b are constructed by finding low collision cost paths from the initial configuration $\mathbf{0}$ to the sampled sub-goals, in Q_{b_E}. The details on how the action sequences are generated are in Section 6.

5.3 Belief space metric

To compute distance between beliefs in B (line 11 & 15 of Algorithm 1), GCS uses Wasserstein distance, a metric that is *dependent* on the underlying state space metric. The Wasserstein distance $W_D(b,b')$ between two beliefs $b,b' \in B$ is the minimum expected distance in S among all possible joint densities whose marginal densities are b and b'. More precisely we use the squared 2nd Wasserstein distance,

$$D_B(b,b') = W_D(b,b') = \inf_f \Bigg\{ \int_{s\in S}\int_{s'\in S} D_S(s,s')f(s,s')dsds' \\ \mid b = \int_{s'} f(s,s')ds', b' = \int_s f(s,s')ds \Bigg\} \tag{4}$$

where D_S is $L2$ norm in S and $f(s,s')$ is joint density function.

Compared to KL-divergence, which is commonly used in POMDPs with continuous S, Wasserstein distance is a true metric. This makes analysing approximation bound using Wasserstein distance easier than using KL-divergence.

Furthermore, Wasserstein distance is more suitable than KL-divergence, denoted as KL, for goal-reaching tasks, which is common in motion planning. As an illustration, consider a 1D navigation problem where the robot's position is not known exactly. The state space here is $\mathbb{R}$. Suppose the goal is g and the goal belief b_g is a belief where the support is all points in $[g-1, g+1]$. Now, suppose b is a belief where the support is all points in $[g-100, g-99]$ and b' is a belief where the support is all points in $[g-5, g-3]$. Then, regardless of the exact distribution, $KL(b,b_g)$ and $KL(b',b_g)$ are both undefined, even though the robot is actually much closer to g when it is at b' than when it is at b, assuming the robot's belief is a good estimate of the robot's true position. On the other hand, $W_D(b',b_g) < W_D(b,b_g)$, as desired.

The Wasserstein distance satisfies the Lipschitz condition,

Theorem 2 *In an LS-continuous POMDP with parameter (K_{RS}, K_Z) and normalized observation space, for any two beliefs $b,b' \in B$ and any policy π, if $W_D(b,b') \leq \delta$, then $|V_\pi(b) - V_\pi(b')| \leq 2\left(\frac{K_{RS}}{1-\gamma} + \frac{\gamma K_Z R_{max}}{(1-\gamma)^2}\right)\delta$, where R_{max} is the maximum possible immediate reward.* [1]

This property means that two nearby beliefs under this metric have similar values, and hence with a small error, one can be used to represent the other.

5.4 Backup

The function $BACKUP(b)$ for $b \in T$ (line 18 of Algorithm 1) finds a path from b to the root of T and performs point-based backup at each belief in the path, starting from b to the root node. It can use a slight modification of any backup computation for continuous S and O (e.g., [2,21,25]). The modification accommodates continuous control space, i.e.

$$\hat{H}_b V(b) = \max_{(a_1,\ldots,a_l)\in outEdge(b)} \left\{ R(b,(a_1,a_2,\ldots,a_l)) + \gamma^l \int_{o\in O} P(o\,|\,b_l,a_l) V(\tau(b_l,a_l,o)) \right\}$$

where $b_l = \tau(b,(a_1,\ldots,a_{l-1}),noObservation)$ and $R(b,(a_1,a_2,\ldots,a_l)) = R(b,a_1) + \sum_{i=1}^{l-1} \gamma^i T(b_i,a_i,b_{i+1}) R(b_{i+1},a_{i+1})$ with $b_1 = b$. The approximation results for using sampled representation for A is in Section 7.

6 Generating action sequences in detail

Algorithm 2 GenerateAnActSeq(b_E, L, M)

1: **if** path set Ψ has not been initialized **then**
2: Initialize path set Ψ to be an empty set.
3: Initialize sub-goals set G to be an empty set.
4: Set $\mathbf{0}$ as the initial configuration q_0.
5: **if** path set Ψ is empty **then**
6: Sample M sub-goals and add them to the sub-goals set G.
7: **for all** sub-goal $g \in G$ **do**
8: Generate a path from q_0 to g, and insert the path to Ψ.
9: Let b_p^1 be the belief over Q where $b_p^1(q_0) = 1$.
10: Choose a path $\psi \in \Psi$ with the highest estimated total discounted reward. And remove ψ from Ψ.
11: Let $(a_1,a_2,\ldots,a_l)$ be the action sequence that corresponds to ψ.
12: Let $i = 1$.
13: **while** $(i \le l)$ and $(\overline{Q}(b_p^1,(a_1,\ldots,a_i)) > L)$ **do**
14: $i = i+1$.
15: $i = max(i-1,0)$.
16: **if** $(i < l)$ **then**
17: Return $(a_1,\ldots,a_i,sensing)$.
18: **else**
19: Return $(a_1,\ldots,a_l)$.

The algorithm to generate action sequences for expanding a node of T is in Algorithm 2. Given a node $b \in T$ to be expanded, let's suppose b lies in $C(b_E) \in \mathscr{C}$, and Q_{b_E} is the robot's configuration space Q where the forbidden regions in Q are uncertain and are caused by obstacles whose positions in the workspace are distributed according to b_E. To construct action sequences for expanding b, GCS first samples a set of sub-goals in Q_{b_E} and constructs an uncertainty roadmap to generate paths from the initial configuration $\mathbf{0}$ to each sub-goal. The action sequences for expanding b is then the paths from $\mathbf{0}$ to the sub-goals in the uncertainty roadmap.

6.1 Sampling sub-goals

The sampled sub-goals consist of two types, i.e., sensing sub-goals and ending sub-goals. Recall from Section 4 that GCS constructs two types of action sequences, i.e.,

sequences with a single sensing action located at the end and sequences with no sensing action. GCS constructs the first type of action sequences based on paths from **0** to sensing sub-goals in Q_{b_E}. And constructs sequences with no sensing action based on paths from **0** to ending sub-goals in Q_{b_E}.

The sensing sub-goal in Q_{b_E} is used as a guide of where sensing action in B should be performed. To keep the size of T small, we would like to perform a sensing action only when sensing significantly reduces the robot's uncertainty about its state. Therefore, as a heuristic, GCS samples sensing sub-goals to be configurations in Q_{b_E} that have high probability of perceiving "useful" observations. More precisely, GCS samples sensing sub-goals using density function

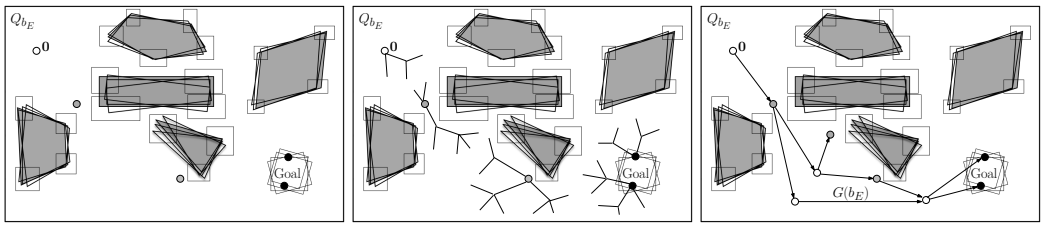
$$f_{sensing}(q) \propto \int_{o \in O} Z(\langle q, b_E \rangle, sensing, o) U(q,o) do \tag{5}$$

where $Z(\langle q, b_E \rangle, sensing, o)$ is the probability that the robot at configuration $q \in Q_{b_E}$ perceives o when the environment uncertainty is b_E, and is defined as $\int_{e \in E} Z(\langle q, e \rangle, sensing, o) b_E(e) de$. For computational efficiency, GCS computes the density based only on Z_{vis}, instead of the complete definition of observation function in (2). The function $U(q,o)$ is a utility function that indicates the use of perceiving observation o from configuration q, and is defined based on how much perceiving o from q reduces the entropy of b_E. Suppose $b' = \tau(\langle q, b_E \rangle, sensing, o)$, b'_E is the marginal distribution of E in b', and $H(.)$ is differential entropy. Then, $U(q,o) = H(b_E) - H(b'_E)$.

To sample ending sub-goals, GCS biases sampling towards configurations that have high expectation to be in the goal region. More precisely, GCS samples ending sub-goals using density function

$$f_{ending}(q) \propto \int_{e \in E} I_{goal}(q,e) \cdot b_E(e) de \quad \text{where } I_{goal}(q,e) = \begin{cases} 1 \text{ if } q \text{ is inside the goal region} \\ \quad \text{of environment } e \in E. \\ 0 \text{ otherwise.} \end{cases} \tag{6}$$

6.2 Path construction



(a) (b) (c)

Fig. 3 The uncertainty roadmap $G(b_E)$ in Q_{b_E}. The grey circles are the sensing sub-goals. The black circles are the ending sub-goals. (a) The initial roadmap. It contains only the initial configuration **0** and the sampled sub-goals. These nodes are the root of the trees. (b) The trees are expanded until there is at least one path from **0** to each sub-goal. (c) The constructed paths. When a tree with a sensing sub-goal as root is connected to a tree with an ending sub-goal as root, GCS generates two paths related to the sensing sub-goal. One is a path from **0** to the sensing sub-goal, which translates to a sequence of actions with a sensing action at the end. Another is a path from **0** to the ending sub-goal via the sensing sub-goal, which translates to a sequence of actions with no sensing action.

To construct low collision-cost paths from the initial configuration $\mathbf{0}$ to the sub-goals, GCS constructs an uncertainty roadmap $G(b_E)$ in Q_{b_E}. An uncertainty roadmap is a roadmap where each milestone and edge has collision cost lower than a given threshold col_{th}. Each edge $\overline{vv'}$ in $G(b_E)$ corresponds to a control sequence that moves the robot from v to v', when the robot's motion is deterministic.

GCS constructs $G(b_E)$ using a slight modification of the Sampling-based Roadmaps of Trees (SRT) [20] strategy. To adapt to the environment map uncertainty, GCS uses the method in Bounded Uncertainty Roadmap (BURM) [8] for collision check. GCS starts by initializing the roadmap $G(b_E)$ with $|G|+1$ trees. The initial configuration $\mathbf{0}$ and each sub-goal in G becomes the root of a tree in $G(b_E)$. Each tree is expanded using the Expansive Space Tree (EST) [11] strategy. GCS alternates expanding each tree until there is a path with collision cost lower than the given threshold col_{th} from $\mathbf{0}$ to each sub-goal in $G(b_E)$. The threshold is set adaptively, starting from zero, which means only collision-free paths in Q_{b_E} is used. When GCS fails to find a path after a large number of samples, it increases the threshold, allowing paths with low collision probability to be used. This process is repeated whenever needed until a path can be found. GCS stops expanding a tree whose root is a sub-goal $g \in G$, once a path from $\mathbf{0}$ to g is found, so that more resources can be used to connect $\mathbf{0}$ to other sub-goals. Each path from the initial configuration $\mathbf{0}$ to a sub-goal in $G(b_E)$ is inserted to the path set Ψ. Figure 3 shows an illustration of $G(b_E)$ and its construction.

Every time the belief $b \in T$ is chosen to be expanded, a path from Ψ is used for expansion and then deleted from Ψ. When Ψ becomes empty, GCS inserts new paths to Ψ. For this purpose, GCS samples additional sub-goals following the sampling criteria in Section 6.1 and inserts the newly sampled sub-goals to the sub-goals set G. It then constructs and inserts a path from $\mathbf{0}$ to all sub-goals in G. For each new sub-goal $g \in G$, GCS adds a new tree with root g to the uncertainty roadmap $G(b_E)$ and constructs a path from $\mathbf{0}$ to g using the above strategy. For a sub-goal g that has been in G before, GCS constructs a new path from $\mathbf{0}$ to g that are different from the existing paths in $G(b_E)$. These paths are constructed using the same strategy as above. GCS inserts all new paths, from $\mathbf{0}$ to the sub-goals, in $G(b_E)$ to the paths set Ψ, and ignores paths that have been used for expanding the belief node b.

Each path $\psi \in \Psi$ corresponds to a control sequence. Suppose $(u_1, u_2, \ldots, u_k)$, where $u_i \in U$ for $i \in [1,k]$, is the control sequence that corresponds to ψ. Then, the action sequence constructed from ψ, is the same as the control sequence $(u_1, u_2, \ldots, u_k)$ if ψ ends at an ending sub-goal, and the same as the control sequence appended with a sensing action, i.e., $(u_1, u_2, \ldots, u_k, \mathit{sensing})$, if ψ ends at a sensing sub-goal.

6.3 Which path to use first?

Ideally, we want to prioritize using action sequences that are more likely to sample beliefs in the optimally reachable set, to enable faster construction of the optimal policy. For this purpose, to each path $\psi \in \Psi$, GCS assigns an estimate $\widetilde{V}(\psi)$ of the total discounted reward, if we follow the path and continue with an optimal strategy afterwards. The path with highest estimated total discounted reward is used first.

For computational efficiency, GCS computes the estimated total discounted reward $\widetilde{V}(\psi)$ by assuming that the robot's motion is deterministic. Suppose the action

sequence constructed from ψ is $(a_1, a_2, \ldots, a_l)$. The estimated value is computed as $\widetilde{V}(\psi) = totCost(\psi) + \overline{h}(\psi)$. The function $totCost(\psi)$ is the total discounted reward for performing $(a_1, a_2, \ldots, a_l)$. More precisely,

$$totCost(\psi) = \int_{e \in E} b_E(e) \sum_{i=0}^{l-1} \gamma^i \left(R_{action}(\langle q_i, e\rangle, a_{i+1}) + R_{collision}(\langle q_i, e\rangle, a_{i+1})\right) de$$

where q_i for $i \in [1, l]$ is the configuration reached by the robot after performing a_i from configuration q_{i-1}, assuming that the robot's motion is deterministic, and $q_0 = \mathbf{0}$. The function $\overline{h}(\psi)$ is an upper bound on the total expected discounted reward for reaching a goal feature from q_l in Q_{b_E}.

6.4 From a path in Q_{b_E} to an action sequence

For computational efficiency, before expanding b using an action sequence $(a_1, a_2, \ldots, a_l)$ constructed from a path $\psi \in \Psi$, GCS revises the action sequence based on whether expanding b using $(a_1, a_2, \ldots, a_l)$ would improve the current policy or not (line 13–17 of Algorithm 2). For this purpose, GCS finds the longest sub-sequence $(a_1, a_2, \ldots, a_m)$ $(m \leq l)$ of $(a_1, a_2, \ldots, a_l)$ that may still improve the current policy and use this sub-sequence, instead of the full sequence, for expanding b. To find such a sub-sequence, GCS finds the longest sub-sequence whose Q-value's upper bound $\overline{Q}(b, (a_1, a_2, \ldots, a_m))$ is larger than the value $V_\pi(b)$ of b under the current policy π.

7 Approximation bound

Our main concern is how sampled representation of the action space A affects the quality of the generated policy. Approximation results are available when sampled representation of S and O are used [2,21]. However, no results are available when sampled representation of A is used. Here we show that when the POMDP problem satisfies *LS*-continuous and *LA*-continuous properties, the optimal policy can be approximated proportional to the sampling dispersion of A.

For clarity, we analyze a simplified version of GCS. We assume that the belief tree T is expanded using a primitive action, instead of a sequence of actions. In this case, the partition $\mathscr{P}$ of A (Definition 2) is $\{U, \{sensing\}\}$.

Theorem 3 *Consider a POMDP that is LS-continuous with parameters (K_{RS}, K_Z), and LA-continuous with parameters (K_{RA}, h) and action partition $\mathscr{P}$. Suppose the distance between two nearest sampled actions in any $P \in \mathscr{P}$ is $\leq \delta_A$, the distance between two nearest sampled beliefs is $\leq \delta_B$, and the maximum immediate reward is R_{max}. Then,* [1]

$$|V^*(b') - V_t(b')| \leq \frac{1}{1-\gamma}\left(K_{RA}\delta_A + \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2}\right)(4\delta_B + \gamma h(\delta_A))\right)$$

Since h is an increasing function with $h(0) = 0$, $|V^*(b') - V_t(b')|$ converges to zero as δ_B and δ_A goes to zero. Similar results hold for GCS as described in Section 4 and Section 5. However, the terms in the approximation bound becomes much more complex, and hinders understanding.

8 Experimental setup and results

8.1 Scenarios

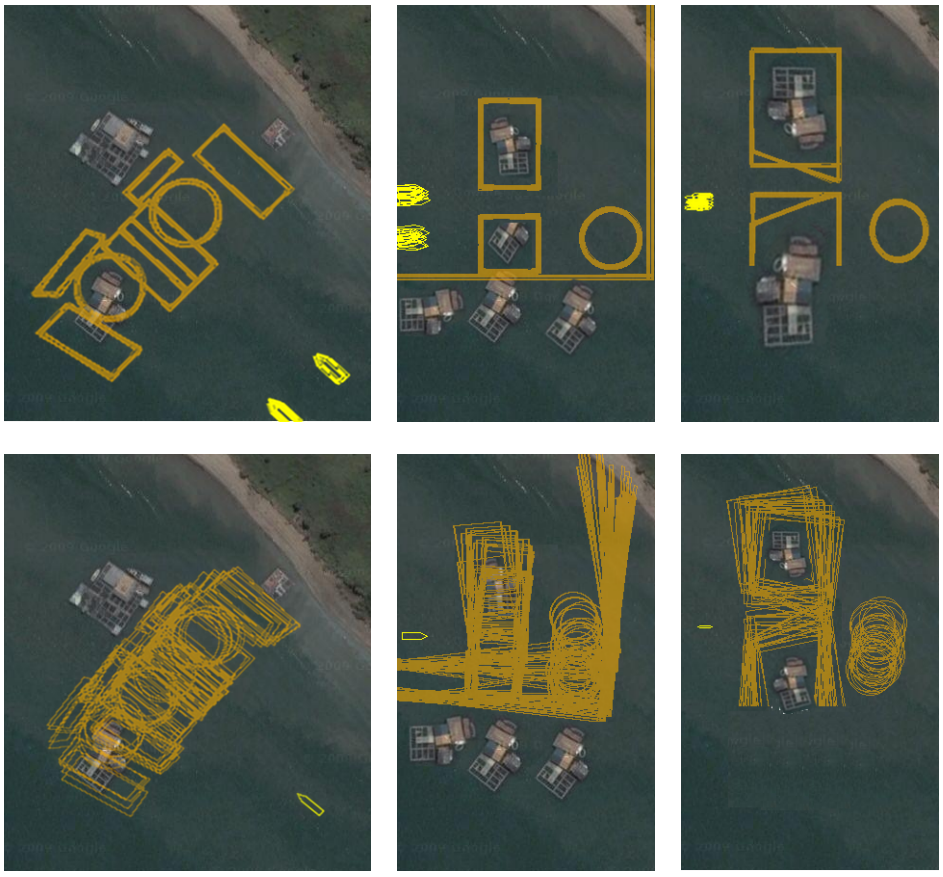
Fig. 4 Traditional fishing farms, called kelong, is quite common in coastal zones of Singapore, Indonesia, and Malaysia.

We tested GCS on simulations of three scenarios (Figure 5) of an Autonomous Surface Vehicle (ASV) navigating in a coastal zone populated by traditional fishing farms, called Kelong (e.g., Figure 4). The motivation for these scenarios comes from the mass death of fishes that has happened several times in Singapore water. To understand and prevent this phenomena from happening again, marine scientists need physical, biological, and chemical data of certain positions around the fishing farms, at various different times of the year. For this purpose, we plan to deploy an ASV to automatically gather these data. Now, the ASV uses sonar to help avoid fishing traps that lies underwater near the fishing farms, but each beam generated by the sonar disturbs marine life, including the fishes in the fishing farms. Therefore, the ASV uses an active sensing mechanism, so that it can perform as little sensing as possible, while reaching the goal region as fast as possible without colliding with any of the structures and fishing traps in the farms.

In all three scenarios, an imperfect environment map is available a priori. The main challenge in operating near Kelongs is avoiding the fishing traps that are attached underneath the structures at the Kelongs. In general, these traps surround the underwater part of the structures and are at varying distance from the structures. Our map models the position of these fishing traps.

Our map uses bounding polygons to represent the fishing traps. The positions of the polygons' vertices are uncertain and are represented as probability distribution with a bounded support. Our map can be constructed a priori by scanning the environment using a sonar mounted underneath an ASV (similar to the ASV setting in [18]), and applying a SLAM algorithm. However, the error in the constructed map needs to be adjusted to take into account the effect of water currents on the structures in these fishing farms. These structures are made from floating platforms with weights tied underneath. Due to water currents, the structures may move, but due to the weights, they can only move up to a certain limit and their movement is relatively slow. Therefore, it is safe to assume that the structures and traps are stationary during the ASV's operational period, but we can not assume that their positions remain the same as when the map was generated. To incorporate the effect of water currents to the error in environment map, we can use water currents prediction, which also has uncertainties and is represented as probability distribution. As a result of this error adjustment, the error distribution in the environment map may become multi-modal even when the result of SLAM algorithm is uni-modal. However, the support of the error can be bounded, as the structures can only move within a certain limit.

The ASV's configuration is defined by its position in a 2D plane and heading. It has holonomic control and its maximum speed is 2m/s. We discretize the time, with each time step is 0.5s. Due to control error and water currents, whenever a control u is



Parking. Navigation-1. Navigation-2.

Fig. 5 Initial beliefs of the scenarios. Above: The possible initial robot and environment configurations in the global coordinate system. Below: The possible initial robot and environment configurations in the robot's local coordinate system. Circles marked the goal regions. The yellow pentagon and hexagon are the ASVs.

applied for one time step to a configuration q, the robot moves to a new configuration $q' = q + u + N$, where N is a uniform distribution over a subset of $\mathbb{R}^2 \times \mathbb{S}$ centered at $q+u$ with size 5cm $\times$ 5cm $\times$ 0.36^0.

The ASV is equipped with a sonar, attached underneath. In this paper, we use a simplified sensor model. We assume the sonar has 360^0 field of view and a range limit of 10m. We set Z_{vis} to be 0.9. Feature-i is seen at position o_i according to the following distribution,

$$o_i = \begin{pmatrix} (dist(q,e_i) + N_d) \times \cos(bearing(q,e_i) + N_b) \\ (dist(q,e_i) + N_d) \times \sin(bearing(q,e_i) + N_b) \end{pmatrix}$$

where $dist(q,e_i)$ is the distance between the actual position of feature-i e_i and the robot's centroid at configuration q, and $bearing(q,e_i)$ is the direction of e_i from the robot's centroid at configuration q. The notation N_d defines the noise in range mea-

surement. It is distributed uniformly over an interval of $[-50cm, 50cm]$. The notation N_b defines the noise in bearing measurement. It is distributed uniformly over $[-1.8^0, 1.8^0]$ interval. We assume that when a feature is located outside the visibility region of q, the feature is not visible. We assume that our operating environment is GPS-denied. Due to obstructions from piers and other structures near coastal zones, GPS error is around 10m, which is too large for navigation in cluttered marine environment.

As an objective function, we define the cost for each control action to be -1, while the cost for each sensing action to be -5,000. Since sonar waves disturb underwater life, including fishes in the fishing farms, sensing becomes undesirable. A high sensing cost is used to discourage unnecessary sensing. Each collision incurs a -100,000 penalty. A reward of 10,000,000 is given when the goal is reached.

8.2 Experimental setup

For each scenario, we ran GCS to generate 30 policies, as GCS uses random numbers. Each policy is generated for 10min. To estimate the expected total reward of each policy, we ran 100 simulation runs and computed the average total discounted rewards.

For comparison, we use BURM [8] and reactive greedy strategy [3]. BURM takes into account uncertainty in the environment map only. Comparison with BURM would show the importance of taking into account motion and sensing uncertainty. We ran BURM to generate 30 different paths. Each path is considered as an open-loop policy. To estimate the expected total reward of each path, we ran 100 simulation runs and compute the average total discounted rewards. Variants of reactive greedy strategy are often used in ASVs. Our ASV has used [3]. The strategy uses continuous sensing to handle motion, sensing, and environment uncertainty. Comparison with this strategy would show how we perform compared to the current practice in marine robotics.

We can not compare with existing POMDP planners because the state space is too large for even the best planners (HSVI2 [23] and SARSOP [15]). As an illustration, in an environment with only two squares as obstacles, if we discretize the possible position of each feature into only 4 cells, the possible robot's position into 10×10 cells, and the robot's possible heading into 4 cells, $|S| > 2.5 \times 10^7$, which is beyond the capability of HSVI2 and SARSOP.

All planners were implemented in C++, and ran in a PC with 2.27GHz Intel processor and 1.5GB RAM.

8.3 Experimental results

Table 1 shows GCS outperforms BURM and reactive greedy strategy. By performing global planning while taking into account all three sources of uncertainty, GCS generates motion strategies that reach the goal with much higher probability and much lower cost.

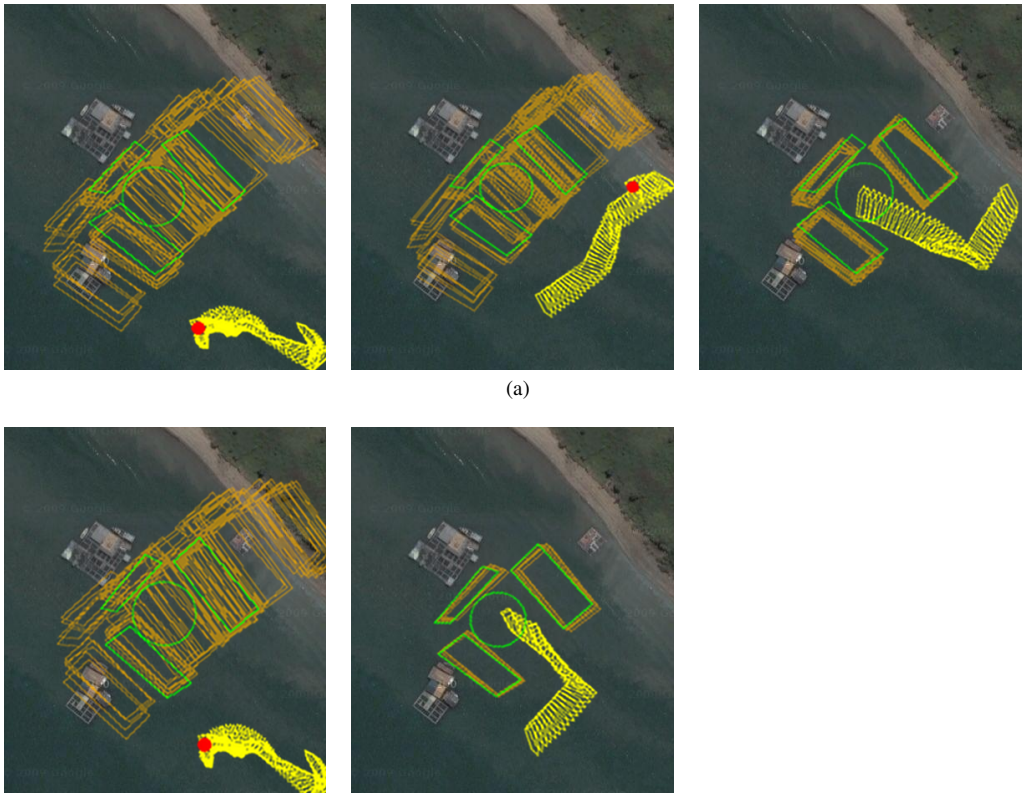
An underlying assumption of active sensing problems that GCS exploits is that sensing is relatively expensive compared to the cost of other actions. The high sensing cost implies that an optimal policy in active sensing problems tend to use a small

Expected Total Discounted Reward and Success Rate

Scenario	BURM		Reactive greedy		GCS (10 min)	
	E[total] disc. reward]	%success	E[total disc. reward]	%success	E[total] disc. reward]	%success
Parking	-395,632	0	3,346,221	45	8,748,667	80
Navigation-1	-1,242,902	0	-662,019	0	8,097,961	74
Navigation-2	-1,697,339	0	-761,415	0	8,642,707	81

Table 1 Comparison results

number of sensing. As discussed in Section 4, small number of sensing implies that the set of optimally reachable beliefs $\mathscr{R}^*(b_0)$ is small and has a nice structure that can help in generating a sufficient sampled representation of $\mathscr{R}^*(b_0)$, and hence a good approximation to the optimal policy. Careful exploitation of $\mathscr{R}^*(b_0)$ structure is key to the good performance of GCS.



(a)

(b)

Fig. 6 Typical simulation runs of the policies generated by GCS in the Parking scenario. Orange: Particles representing the robot's belief over its environment. Green: The actual environment with circles marking the goal regions. Yellow: the ASV path. Red dot: sensing position.

Figure 6 shows two typical runs of the policies generated by GCS in the Parking scenario. In this scenario, the marginal distribution of E in the initial belief is essentially bi-modal. In one modality, the parking region (represented as polygons) is

located near the left side of the environment, while in the other, it is located near the right side. Within each modality, the exact position of each vertex is also not known. Figure 6(a) shows that when the ASV runs a policy generated by GCS, it first moves to a position near the left side, where useful sensing data may be generated with high probability, and performs sensing from there. It tries to assess if the true environment is located around the left region. But due to sensing uncertainty, the ASV fails to get sensing data. Based on this event, the ASV updates its belief. The updated belief still has non-zero probability that the structures are located at the left region, because a feature may not be visible even though it is within the visibility region of the robot. However, the updated belief has higher probability that the true environment is near the right side. This higher probability can be seen from the fact that the particles near the right side of the environment in the middle picture of Figure 6(a) are denser compared to those in the left picture. Utilizing the information in the updated belief, the ASV moves to the right region where useful sensing data may be generated with high probability. This time, the ASV manages to get sensing data on the features and hence can assess the true environment much better. Although the exact environment is still not known, the ASV knows enough to park safely. So, instead of sensing the environment further, the ASV continues to park at the goal region and then terminates its operation. Figure 6(b) shows the case where the first sensing is a success. In this case, the ASV directly continues to park at the goal region and terminates its operation. These simulation runs show that GCS generates policies that enable an ASV to carefully assess its environment, taking into account sensing errors, before finally moving to accomplish the given task with lowest cost possible.

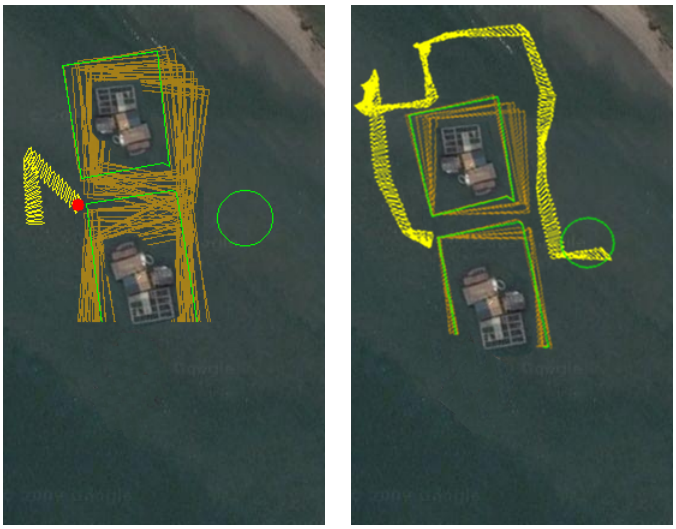


Fig. 7 Typical simulation runs of the policies generated by GCS in the Navigation-2 scenario. The meaning of colors is the same as those in Figure 6.

For Navigation-1 and Navigation-2, one tends to think that the optimal strategy would be to not do any sensing and directly go to the goal region via the wide open

passage on the top. But, this is not true. The accumulation of error in ASV's motion causes the ASV configuration to be quite uncertain after following a long path. Therefore, due to the limitation in operating area, when the environment map uncertainty is high, the ASV is likely to collide with the obstacles if it moves from the start to the goal via the wide open passage without ever sensing its environment.

When the ASV runs the policies generated by GCS, the results of Navigation-1 and Navigation-2 show similar trend. Here, we show a typical simulation run (Figure 7) for Navigation-2, which is the more complex environment. The ASV starts by moving to a region where sensing may generate useful data with high probability. Notice that it tries to sense very near to the mouth of the passage. This strategy reduces the error in the position of the seen features. Since sensing error is defined with respect to range and bearing, the position of the seen features would have smaller error when the ASV is close to the features. A more accurate information on the position of these two features results in better inference on the position of the entire structures, which then enables the ASV to reach the goal region safely without further sensing. These results indicate that GCS generates policies that enable the ASV to strategically decide where it should sense the environment, so that each sensing generates maximum benefit.

9 Conclusion

This paper proposes a new global motion planner under motion, sensing, and environment map uncertainty for robots with active sensing mechanism. We call our new planner Guided Cluster Sampling (GCS). GCS is a point-based POMDP planner that uses domain specific properties to construct a more suitable sampling strategy. Simulation results show that GCS successfully solves motion planning problems with uncertain motion, sensing, and environment map that are unsolvable by the best POMDP planners today, within reasonable time.

GCS is designed for POMDPs with continuous state, action, and observation spaces. We showed that for a class of POMDPs that often occur in motion planning problems, given enough time, GCS converges to the optimal policy. To the best of our knowledge, this is the first convergence result for point-based POMDPs with continuous action space.

Our simulation and theoretical results indicate that GCS provides an efficient and mathematically principled way for balancing sensing and acting to accomplish the given tasks, in the presence of various types of uncertainties.

Acknowledgement The authors thank D. Hsu for the discussion and cluster computing usage, L.P. Kaelbling and J.J. Leonard for the discussion, and S. Ong for reading the early drafts of this paper. This work is funded by the Singapore NRF through SMART, CENSAM.

References

1. R. Alterovitz, T. Simeon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In *RSS*, 2007.
2. H. Bai, D. Hsu, W.S. Lee, and A.V. Ngo. Monte Carlo Value Iteration for Continuous-State POMDPs. In *WAFR*, 2010.

3. T. Bandyopadhyay, L. Sarcione, and F. Hover. A simple reactive obstacle avoidance algorithm and its application in Singapore Harbour. In *FSR*, 2009.
4. J.V.D. Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information. In *RSS*, 2010.
5. B. Burns and O. Brock. Sampling-Based Motion Planning with Sensing Uncertainty. In *ICRA*, 2007.
6. 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.
7. A. Guez and J. Pineau. Multi-Tasking SLAM. In *ICRA*, 2010.
8. L. Guibas, D. Hsu, H. Kurniawati, and E. Rehman. Bounded Uncertainty Roadmaps for Path Planning. In *WAFR*, 2008.
9. K. Hauser. Randomized Belief-Space Replanning in Partially-Observable Continuous Spaces. In *WAFR*, 2010.
10. K. Hsiao, L.P. Kaelbling, and T. Lozano-Perez. Grasping POMDPs. In *ICRA*, pages 4685–4692, 2007.
11. D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *IJCGA*, 9(4–5):495–512, 1999.
12. D. Hsu, W.S. Lee, and N. Rong. What makes some POMDP problems easy to approximate? In *NIPS*, 2007.
13. T. Kollar and N. Roy. Efficient Optimization of Information-Theoretic Exploration in SLAM. In *AAAI*, pages 1369–1375, 2008.
14. H. Kurniawati, Y. Du, D. Hsu, and W.S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *IJRR*, 30(3):308–323, 2011.
15. H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS*, 2008.
16. P. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *ICRA*, 2006.
17. C.H. Papadimitriou and J.N. Tsitsiklis. The Complexity of Markov Decision Processes. *Math. of Operation Research*, 12(3):441–450, 1987.
18. G. Papadopoulos, H. Kurniawati, A.S.B.M. Shariff, L.J. Wong, and N.M. Patrikalakis. 3D-surface reconstruction for partially submerged marine structures using an Autonomous Surface Vehicle. In *IROS*, 2011.
19. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.
20. E. Plaku, K. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *TRO*, 21(4):597–608, 2005.
21. J.M. Porta, N. Vlassis, M.T.J. Spaan, and P. Poupart. Point-Based Value Iteration for Continuous POMDPs. *JMLR*, 7(Nov):2329–2367, 2006.
22. S. Prentice and N. Roy. The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance. In *ISRR*, 2007.
23. T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, July 2005.
24. C. Stachniss, G. Grisetti, and W. Burgard. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *RSS*, pages 65–72, 2005.
25. S. Thrun. Monte carlo POMDPs. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *NIPS 12*, pages 1064–1070. MIT Press, 2000.

Appendix

To prove Theorem 1 and Theorem 2, we use α-function as the policy representation. The policy π is represented by a set of α-functions Γ, where $\pi(b) = \arg\max_{\alpha\in\Gamma} \int_{s\in S} \alpha(s)\cdot b(s)ds$. Each α-function corresponds to a policy tree T_α. Each node in T_α corresponds to an action and each edge corresponds to an observation. The value $\alpha(s)$ is the expected total reward of executing T_α from s. Let a_0 denotes the root of T_α and let's use the same notation to denote a node and its corresponding action. Then, executing T_α starting from s means that the robot at state s starts execution by performing a_0. An arc from a_0 to a node at the next level of T_α is followed, based on the observation perceived. Suppose the arc points to node a_1,

then at the next step, the robot performs a_1. This process is repeated until a leaf node is reached. The value $\alpha(s)$ can be written as,

$$\alpha(s) = R(s,a_0) + \gamma \int_{s_1 \in S} \int_{o \in O} T(s,a_0,s_1) Z(s_1,a_0,o_1) \alpha_{a_0 o}(s_1) do ds_1 \quad (7)$$

where $\alpha_{a_0 o}$ is the α-function that corresponds to the sub-tree of T_α whose root is the child of a_0 via edge o.

A Proof of Theorem 1

To prove the theorem, we first show that for any α-vector, $\alpha(s) = \alpha(g(s))$. For this purpose, we show that for any function $f : S \to S$ that does not change the robot's relative configuration with respect to the environment, $\alpha(s) = \alpha(f(s))$. Since g is an instance of such function, $\alpha(s) = \alpha(g(s))$, too.

We prove $\alpha(s) = \alpha(f(s))$ by induction on the levels of T_α. When T_α has only one level, $\alpha(s) = R(s,a_0)$. Since the reward function depends only on the relative configuration of the robot and since applying f to s does not change this relative configuration, $\alpha(s) = R(s,a_0) = R(f(s),a_0) = \alpha(f(s))$. Assume that for any f, any α, and any $s \in S$, $\alpha(s) = \alpha(f(s))$ when T_α has i levels. Now, we show that $\alpha(s) = \alpha(f(s))$ when T_α has $(i+1)$ levels. The key is to show that applying f to s does not change the integration term in (7). Let's first look at the transition function. Based on property-1 of LS-continuous, we have $T(s,a_0,s_1) = T(f(s),a_0,s_1')$ where $s_1' = s_1 + (f(s) - s)$. Since, the displacement vector $(f(s) - s)$ does not change the relative robot's configuration with respect to the environment, the relative robot's configuration at s_1 is the same as that at s_1'. Since Z depends only on the relative robot's configuration, $Z(s_1,a_0,o) = Z(s_1',a_0,o)$ for any $o \in O$. Using the result from level-i, $\alpha(s_1) = \alpha(s_1')$. Hence, the integration term of (7) for $\alpha(s)$ and $\alpha(f(s))$ are the same. This result and the fact that the reward function depends only on the robot's relative configuration, gives us $\alpha(s) = \alpha(f(s))$. Now, we prove that for any policy π, $V_\pi(b) = V_\pi(Transform(b))$. Let $\mathscr{R}$ be a partition of S, such that each set $R_s \in \mathscr{R}$ consists of all states in S where the robot's relative configurations with respect to the environment, is the same as that of s. This means that each state in the same set of $\mathscr{R}$ has the same α-value. And hence we can write,

$$V_\pi(b) = \max_{\alpha \in \Gamma} \int_{s \in S} \alpha(s) \cdot b(s) = \max_{\alpha \in \Gamma} \int_{R_s \in \mathscr{R}} \alpha(s) \int_{s' \in R_s} b(s'). \quad (8)$$

From the definition of $Transform$ in (3), $(Transform(b))(s) = \int_{s' \in R_s} b(s')$. Therefore, (8) can be rewritten as $V_\pi(b) = \max_{\alpha \in \Gamma} \int_{R_s \in \mathscr{R}} \alpha(s) \cdot (Tranform(b))(s) = V_\pi(Transform(b))$, which is the result we want. □.

B Proof of Theorem 2

To prove Theorem 2, we first need the following lemma.

Lemma 1 *In an LS-continuous POMDP with parameter (K_{RS}, K_Z) and normalized observation space, for any α-function and any state $s, s' \in S$, $|\alpha(s) - \alpha(s')| \leq \left(\frac{K_{RS}}{1-\gamma} + \frac{\gamma K_Z R_{\max}}{(1-\gamma)^2} \right) D_S(s,s')$.*

Proof of Lemma 1. Using the definition of α value in (7) and the triangle inequality, we have

$$|\alpha(s) - \alpha(s')| \leq |R(s,a_0) - R(s',a_0)| + \gamma \left| \int_{s_1 \in S} \int_{o \in O} T(s,a_0,s_1) Z(s_1,a_0,o) \alpha_{a_0 o}(s_1) do ds_1 - \int_{s_1 \in S} \int_{o \in O} T(s',a_0,s_1) Z(s_1,a_0,o) \alpha_{a_0 o}(s_1) do ds_1 \right| \quad (9)$$

Based on property-3 of *LS*-continuous, we can bound the first absolute term on the right hand side of (9) as $|R(s,a_0) - R(s',a_0)| \leq K_{RS} D_S(s,s')$.
Now, we bound the second absolute term on the right hand side of (9). Let $d = s' - s$. Property-1 of *LS*-continuous gives us $T(s,a_0,s_1) = T(s',a_0,s_1+d)$. Hence, we can rewrite the last absolute term in (9) as,

$$\left| \int_{s_1 \in S} T(s,a_0,s_1) \int_{o \in O} \left(Z(s_1,a_0,o) \alpha_{a_0 o}(s_1) - Z(s_1+d,a_0,o) \alpha_{a_0 o}(s_1+d) \right) do ds_1 \right|.$$

Using property-2 of *LS*-continuous, we have $Z(s_1+d,a_0,o) \geq Z(s_1,a_0,o) - K_Z \cdot D_S(s_1,s_1+d)$. Substituting the above bounds to (9) gives

$$\begin{aligned}|\alpha(s)-\alpha(s')| \leq\ & K_{RS}\cdot D_S(s,s') + \gamma\Big|\int_{s_1\in S} T(s,a_0,s_1)\int_{o\in O} Z(s_1,a_0,o)\left(\alpha_{a_0o}(s_1)-\alpha_{a_0o}(s_1+d)\right)do + \\ & \int_{o\in O} K_Z\cdot D_S(s_1,s_1+d)\alpha_o(s_1+d)do\Big| \\ \leq\ & \left(\frac{K_{RS}}{1-\gamma}+\frac{\gamma K_Z R_{\max}}{(1-\gamma)^2}\right) D_S(s,s')\end{aligned}$$

The last inequality holds, after α_{a_0o} is expanded recursively and assuming that O is normalized. □.

Now, we proof Theorem 2. Let $V_\pi(b)=\alpha\cdot b$ and $V_\pi(b')=\alpha'\cdot b'$. Then, there must always be a point $b_c = ab+(1-a)b'$ such that $\alpha\cdot b_c = \alpha'\cdot b_c$, as $\alpha\cdot b \geq \alpha'\cdot b$ and $\alpha'\cdot b' \geq \alpha\cdot b'$

$$\begin{aligned}|V^*(b)-V^*(b')| &= |\alpha\cdot b-\alpha'\cdot b'| \\ &\leq |\alpha\cdot b-\alpha\cdot b_c| + |\alpha'\cdot b_c-\alpha'\cdot b'| \end{aligned} \tag{10}$$

Suppose f is the joint density function used in computing $W_D(b,b_c)$ with $b(s)=\int_{s'\in S} f(s,s')ds'$ and $b_c(s')=\int_{s\in S} f(s,s')ds$. And suppose g is the joint density function used in computing $W_D(b_c,b')$ with $b_c(s)=\int_{s'\in S} g(s,s')ds'$ and $b'(s')=\int_{s\in S} g(s,s')ds$. Then, we can rewrite (10) as,

$$\begin{aligned}|V^*(b)-V^*(b')| \leq\ & \left|\int_{s\in S}\alpha(s)\int_{s'\in S} f(s,s')ds'ds - \int_{s'\in S}\alpha(s')\int_{s\in S} f(s,s')dsds'\right| + \\ & \left|\int_{s\in S}\alpha'(s)\int_{s'\in S} g(s,s')ds'ds - \int_{s'\in S}\alpha'(s')\int_{s\in S} g(s,s')dsds'\right| \\ \leq\ & \int_{s\in S}\int_{s'\in S} f(s,s')\left|\alpha(s)-\alpha(s')\right|ds'ds + \int_{s\in S}\int_{s'\in S} g(s,s')\left|\alpha'(s)-\alpha'(s')\right|ds'ds\end{aligned}$$

Substituting the difference between α values in the above inequality with the result of Lemma 1, and using the definition of Wasserstein distance give us,

$$|V^*(b)-V^*(b')| \leq \left(\frac{K_{RS}}{1-\gamma}+\frac{\gamma K_Z R_{max}}{(1-\gamma)^2}\right)\left(W_D(b,b_c)+W_D(b_c,b')\right)$$

Using the convexity property of W_D, we get the desired result. □.

C Proof of Theorem 3

To proof Theorem 3, we first need the following lemma that bounds the error generated by a single backup operation.

Lemma 2 *Consider a POMDP that satisfies LS-continuous with parameter (K_{RS},K_Z) and LA-continuous with parameter (K_{RA},h). Suppose the sampling dispersion in each element of $\mathscr{P}$ is $\leq \delta_A$. Then, the error generated by a single simplified GCS backup at a belief b is bounded as, $|HV(b)-\hat{H}_bV(b)| \leq K_{RA}\delta_A + \gamma\left(\frac{K_{RS}}{1-\gamma}+\frac{K_Z R_{max}}{(1-\gamma)^2}\right)h(\delta_A)$.*

Proof of Lemma 2. To shorten the proof writing, let's use the Q-value notation. For any $b\in B$ and any $a\in A$,

$$Q(b,a) = \int_{s\in S} R(s,a)b(s)ds + \gamma\int_{s\in S}\int_{s'\in S}\int_{o\in O} T(s,a,s')Z(s',a,o)b(s)\alpha(s')dods'ds \tag{11}$$

The single backup error is then,

$$\begin{aligned}\left|HV(b)-\hat{H}_bV(b)\right| &= \max_{P\in\mathscr{P}}\max_{a\in P} Q(b,a) - \max_{P\in\mathscr{P}}\max_{a\in Samp(P)} Q(b,a) \\ &\leq \max_{P\in\mathscr{P}}\left(Q(b,a_P^*)-Q(b,\hat{a}_P^*)\right)\end{aligned} \tag{12}$$

where $Samp(P)$ is the sampled representation of $P \in \mathscr{P}$,
$a^*_P = \arg\max_{a \in P} Q(b,a)$, and $\hat{a}^*_P = \arg\min_{a \in Samp(P)} D_P(a, a^*_P)$.

Let's compute $Q(b,a^*_P) - Q(b,\hat{a}^*_P)$ for an element P of $\mathscr{P}$. For writing compactness, we drop the P subscript. Using (11) and triangle inequality, we get

$$Q(b,a^*) - Q(b,\hat{a}^*) \leq \int_{s\in S} |R(s,a^*) - R(s,a)|\, b(s) ds + \gamma \int_{s\in S} b(s) \left| \int_{s'\in S} \int_{o\in O} T(s,a^*,s') Z(s',a^*,o) b(s) \alpha(s') do ds' - \int_{s'\in S} \int_{o\in O} T(s,\hat{a}^*,s') Z(s',\hat{a}^*,o) \alpha(s') do ds' \right| ds \quad (13)$$

Using property-3 of *LA*-continuous, we bound the first term in the right hand side as $\int_{s\in S} |R(s,a^*) - R(s,a)|\, b(s) ds \leq K_{RA}\delta_A$. Using property-1 of *LA*-continuous, $T(s,a^*,s') = T(s,\hat{a}^*,s' + f(a^*,\hat{a}^*)$. Therefore, (13) can be rewritten as,

$$Q(b,a^*) - Q(b,\hat{a}^*) \leq K_{RA}\delta_A + \gamma \left| \int_{s\in S} \int_{s'\in S} b(s) T(s,a^*,s') \int_{o\in O} \left(Z(s',a^*,o)\alpha(s') - Z(s'+f(a^*,\hat{a}^*),\hat{a}^*,o)\alpha(s'+f(a^*,\hat{a}^*)) \right) do ds' ds \right|$$

Since a^* and $\hat{a}^*$ belong to the same element of $\mathscr{P}$, using property-2 of *LA*-continuous, we get $Z(s' + f(a^*,\hat{a}^*),\hat{a}^*,o) = Z(s'+f(a^*,\hat{a}^*),a^*,o)$. Using property-2 of *LS*-continuous, we get $Z(s'+f(a^*,\hat{a}^*),a^*,o) \geq Z(s',a^*,o) - K_Z D_S(s',s'+f(a^*,\hat{a}^*))$. Using these properties and the assumption that O is normalized, re-arranging the above inequality gives us

$$\begin{aligned} Q(b,a^*) - Q(b,\hat{a}^*) &\leq K_{RA}\delta_A + \gamma \left| \int_{s\in S} \int_{s'\in S} b(s) T(s,a^*,s') \left(K_Z D_S(s',s'+f(a^*,\hat{a}^*)) \alpha(s'+f(a^*,\hat{a}^*)) + \right.\right. \\ &\qquad \left.\left. \int_{o\in O} Z(s',a^*,o) \left(\alpha(s') - \alpha(s'+f(a^*,\hat{a}^*)) \right) do \right) ds' ds \right| \\ &\leq K_{RA}\delta_A + \gamma \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) h(D_P(a^*,\hat{a}^*)) \end{aligned}$$

The last inequality holds based on three properties: (1) any α value does not exceed $\frac{R_{max}}{1-\gamma}$, (2) property-1 of *LA*-continuous, i.e., $D_S(s',s'+f(a^*,\hat{a}^*)) \leq h(D_P(a^*,\hat{a}^*))$, and (3) Lemma 1.

Using the above inequality and the fact that for any $P \in \mathscr{P}$, $D_P(a^*_P,\hat{a}^*_P) \leq \delta_A$, and h is an increasing function, $\left|HV(b) - \hat{H}_b V(b)\right| \leq K_{RA}\delta_A + \gamma \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) h(\delta_A)$. □.

Now, we can prove Theorem 3. The difference between the optimal value V^* and the value V_t computed by the simplified GCS after t steps are,

$$|V^*(b) - V_t(b)| \leq \left|V^*(b) - V^*(b')\right| + \left|V^*(b') - V_t(b')\right| + \left|V_t(b') - V_t(b)\right|.$$

Applying Theorem 2 to V^* and V_t, and bounding $|V^*(b') - V_t(b')| \leq \varepsilon_t$, we get

$$|V^*(b) - V_t(b)| \leq 4 \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) \delta_B + \varepsilon_t. \quad (14)$$

To compute ε_t, notice that $V^*(b') = HV^*(b')$ and $V_t(b') \leq \hat{H}_b V_{t-1}(b')$. Hence,
$|V^*(b') - V_t(b')| \leq \left|HV^*(b') - \hat{H}_b V_{t-1}(b')\right|$ and the following holds

$$\left|V^*(b') - V_t(b')\right| \leq \left|HV^*(b') - HV_{t-1}(b')\right| + \left|HV_{t-1}(b') - \hat{H}_b V_{t-1}(b')\right|. \quad (15)$$

Using the contraction property of H and (14), we can bound the first absolute term on the right hand side of (15) as $|HV^*(b') - HV_{t-1}(b')| \leq \gamma \left(4 \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) \delta_B + \varepsilon_{t-1} \right)$. The last absolute term of (15) can be bounded using Lemma 2. As a result, we get

$$\left|V^*(b') - V_t(b')\right| \leq \gamma \left(4 \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) \delta_B + \varepsilon_{t-1} \right) + \left(K_{RA}\delta_A + \gamma \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) h(\delta_A) \right)$$

Expanding the recursion gives us,

$$\left|V^*(b') - V_t(b')\right| \leq \frac{1}{1-\gamma} \left(K_{RA}\delta_A + \left(\frac{K_{RS}}{1-\gamma} + \frac{K_Z R_{max}}{(1-\gamma)^2} \right) (4\delta_B + \gamma h(\delta_A)) \right),$$

which is the result we want. □.