Robotics: Science and Systems 2013
Berlin, Germany, June 24-28, 2013

Integrated Perception and Planning in the Continuous Space: A POMDP Approach

Haoyu Bai David Hsu Wee Sun Lee

Department of Computer Science
National University of Singapore
Singapore, 117417, Singapore

***Abstract*—The partially observable Markov decision process (POMDP) provides a principled mathematical model for integrating perception and planning, a major challenge in robotics. While there are reasonably efficient algorithms for discrete POMDPs, continuous models are often more natural for robotic tasks, and currently there are no practical algorithms that handle continuous POMDPs at an interesting scale. This paper presents an algorithm for continuous-state, continuous-observation POMDPs. We provide experimental results demonstrating its potential in robot planning and learning under uncertainty and a theoretical analysis of its performance. A direct benefit of the algorithm is to simplify model construction.**

I. Introduction

Integrated perception and planning is essential for reliable robot operation and poses a major challenge in robotics. The partially observable Markov decision process (POMDP) provides a mathematical model that connects perception and planning in a principled manner. It has been applied to a range of robotic tasks, including navigation [24], grasping [13], and aircraft collision avoidance [2]. However, efficient POMDP algorithms existing today typically assume discrete models (*e.g.*, [14], [17], [25]), in which an agent's states, actions, and observations are all discrete, while for robotic tasks, continuous models are often more natural. To our knowledge, there are currently no practical algorithms that handle continuous POMDPs at an interesting scale for robotic tasks. Here we aim to develop an efficient algorithm for *continuous-state*, *continuous-observation*, but discrete-action POMDPs and apply it to robot planning and learning under uncertainty.

Because of uncertainty inherent in robot control and sensing, a robot does not know its state perfectly. To choose an action, it must consider all possible states consistent with actions taken and observations received. In a POMDP, we capture the state uncertainty in a *belief*, which can be represented as a probability distribution over the robot's state space. The set of valid beliefs forms the belief space $\mathcal{B}$. An offline POMDP algorithm systematically reasons over $\mathcal{B}$ and tries to construct an optimal *policy* $\pi\colon \mathcal{B} \to A$, which prescribes to every belief in $\mathcal{B}$ a best action from an action set A.

To apply discrete POMDP algorithms to tasks with continuous states and observations, a common approach is to discretize state and observation spaces with a regular grid. This is difficult to scale up, as the computational cost increases exponentially for high-dimensional spaces. We avoid such fixed discretization. Conceptually we perform probabilistic sampling to "discretize" state and observation spaces *implicitly* during the policy computation. The sampling adapts to the accuracy of policy computation.

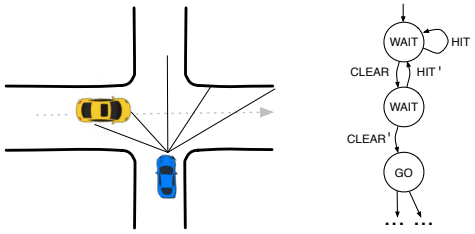


Fig. 1. Intersection navigation. (*a*) An autonomous vehicle, in blue, navigates through an uncontrolled road intersection. It is equipped with a sensor to measure proximity to obstacles along a set of beams. (*b*) An example GPG. CLEAR, CLEAR′, HIT, and HIT′ represent subsets of continuous observations.

One main challenge with continuous POMDPs lies in representing the belief and the policy: the belief is a continuous probability distribution, and the policy maps such a continuous probability function to an action. Developing finite representations for them is difficult. We introduce the *generalized policy graph* (GPG) as an alternative policy representation. Each node of a GPG is labeled with an action. Since the observation space is continuous, each node has an associated *edge classifier*, which is a function that maps an input observation to another GPG node. Intuitively we can think of a GPG as a finite-state controller with continuous input. Consider the example in Fig. 1. Under this policy, the stopped autonomous vehicle moves forward only after receiving two successive observations from CLEAR and CLEAR′, respectively.

We construct a GPG by iteratively applying the Bellman backup equation to an initial policy. This is the same basic idea of value iteration [28], but we perform backup on a GPG rather than a value function. To deal with continuous state and observation spaces, we evaluate the Bellman equation by Monte Carlo sampling. We also provide a performance bound on the number of samples required to compute an approximately optimal policy.

Our algorithm applies without modification to POMDPs with large discrete state or observation space as well.

One immediate benefit of the new algorithm is simplified model construction, as it removes the need to discretize states

and observations manually. Consider the example in Fig. 1 again. In a discrete POMDP model, we must choose discrete locations as the states for the vehicles. We must also construct observations. One way is to quantize the proximity sensor readings for each beam. This results in 2^K observations for K beams, even with a two-level quantization. A more sophisticated observation model calculates the maximum-likelihood vehicle location by preprocessing the sensor readings and uses the estimated location as the observation. This reduces the number of observations, but may lose information during the preprocessing and degrade the quality of the computed policy (see Section V-B). The new algorithm alleviates the difficulty of these modeling choices by sampling directly from the continuous state and observation spaces during the policy computation.

One way of dealing model uncertainty in a POMDP is to incorporate unknown model parameters into the state [8], thus performing planning and learning simultaneously. In this setting, one added benefit of the new algorithm is that it handles continuous model parameters directly, without the need to discretize them a priori [30].

II. Related Work

With continuous states, a main difficulty in POMDP planning is belief representation. One approach is to restrict to a parametric class of beliefs, *e.g.*, the Gaussian [6], [22] or the Gaussian mixture [19]. However, robotic tasks often involve beliefs with multiple modes and sharp edges, *e.g.*, when a robot navigates through long, narrow corridors with few features. In this case, the Gaussian mixture has difficulty in scaling up. Other approaches use sampled representations such as the particle filter [4], [19], [27], or aggregate sampled beliefs approximately [7]. Our algorithm uses a sampled belief representation for offline policy computation. For online policy execution, it exploits the policy graph representation and does not track the belief explicitly, thus avoiding the belief representation issue there.

Another difficulty with continuous states is policy representation. Instead of representing the policy directly, one may develop a hierarchical representation for the value function associated with the policy [5]. The hierarchical representation would be effective if the value function is sufficiently smooth, but may have difficulty in scaling up to high-dimensional state spaces. We thus choose to use the policy graph, which is more direct and simpler. However, the two approaches are complementary and may be combined (see Section VII).

Continuous observations cause difficulty in a different way. A POMDP policy must condition on all future observations. Clearly it is impossible to enumerate an infinite number of continuous observations. Earlier work addresses this issue by aggregating observations, but it requires discrete states for policy representation [12]. In contrast, our algorithm handles both continuous states and continuous observations.

Instead of restricting the belief space, a different approach is to search a restricted policy class, *e.g.*, finite-state controllers [16], [20] or memoryless reactive policies [1]. These methods, however, cannot guarantee the global optimality of the computed policy. We show in Section VI that under reasonable conditions, our algorithm is guaranteed to find an approximately optimal policy with high probability.

Our algorithm computes offline a policy conditioned on future observations. An orthogonal direction is to perform forward search online (*e.g.*, [11], [18], [23], [29]), which chooses a single best action for the current belief. It does not compute a policy and completely avoids the issue of policy representation for continuous state space. Online search and offline policy computation can be combined to solve difficult POMDPs, *e.g.*, by using approximate or partial policies computed offline as default policies for online search.

Our algorithm evaluates the Bellman equation by Monte Carlo sampling. This is the basic idea of approximate dynamic programming [21] and used in various MDP/POMDP planning and reinforcement learning algorithms (*e.g.*, [1], [15]).

Our algorithmic approach builds on the policy search algorithm in [9] and the MCVI algorithm [4]. The former is designed for discrete POMDPs. The latter deals with continuous states, but only discrete observations.

III. POMDPs with Continuous States and Observations

A. The Model

Formally, a POMDP is a tuple $(S, A, O, T, Z, R, \gamma)$, where S, A, and O denote a robot's state space, action space and observation space, respectively. At each time step, the robot takes an action $a \in A$ to move from a state $s \in S$ to $s' \in S$; it then receives an observation $o \in O$. The model for the system dynamics is specified by a conditional probability function $T(s, a, s') = p(s'|s, a)$, which accounts for uncertainty in robot control, unexpected environment changes, *etc.*. Similarly, the observation model is specified by a conditional probability function $Z(s', a, o) = p(o|s', a)$, which accounts for sensing uncertainty. The function $R(s, a)$ specifies a real-valued reward for the robot if it takes action a in state s. The robot's goal is to choose a sequence of actions that maximizes the expected total reward $\mathrm{E}\big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\big)$, where s_t and a_t denote the system's state and action at time t. The discount factor $\gamma \in [0, 1)$ ensures that the total reward is finite, even when a planning task has an infinite horizon.

As a modeling language, the POMDP is agnostic about whether S, A, and O are continuous or discrete. The difference is, however, significant for belief and policy representations. In our model, both S and O are continuous, but A is discrete.

B. Beliefs

In a POMDP, we capture the robot's state uncertainty in a belief, which is a probability distribution over S. Suppose that b is the current belief on the robot state. If the robot executes action a and receives observation o, the new belief b_{ao} is calculated according to the Bayes' rule:

$$b_a(s') = \int_{s \in S} p(s'|s, a) b(s) \, \mathrm{d}s, \tag{1}$$

$$b_{ao}(s) = \eta p(o|s, a) b_a(s), \tag{2}$$

where η is a normalizing constant. The update uses the system dynamics model and the observation model to integrate information from a and o into the new belief. Since S is continuous, we use a set of particles [28] as a finite belief representation in the offline policy computation.

C. Policies

One common POMDP policy representation is a *policy graph*, which is a directed graph. Each node of a policy graph is labeled with an action from A, and each edge is labeled with an observation from O. The policy graph representation is compact, as its size depends only on the complexity of a policy and not on the size of the state space S. It has been used successfully in various algorithms for POMDPs with large discrete or continuous state space (*e.g.*, [2], [20]).

However, each edge of a policy graph corresponds to a single observation, and this is unsuitable for POMDPs with large discrete or continuous observation space. The GPG generalizes the policy graph to POMDPs with continuous observation space by representing outgoing observation edges as a classifier that maps observations to subsequent policy graph nodes. Formally, a GPG G is a set of nodes. Each node $v = (a, \kappa)$ consists of an action $a \in A$ and a mapping $\kappa \colon O \to G$.

To execute a policy $\pi_{G,v}$ represented as a GPG G, we start at a node $v = (a, \kappa)$ in G and take the action a. Upon receiving an observation o, we move to the next node $v' = \kappa(o)$. The process then repeats at the new node $v' = (a', \kappa')$. We do not track beliefs explicitly using (1) and (2), but instead represent beliefs implicitly as histories of actions and observations.

Each node $v \in G$ induces an α-function $\alpha_v \colon S \to \mathbb{R}$, which defines the expected total reward of executing $\pi_{G,v}$ starting at an initial robot state $s \in S$. If the initial robot state is uncertain and described as a belief b, the expected total reward is then $\int_{s \in S} b(s)\alpha_v(s)\,\mathrm{d}s$. We define the *value* of a belief b with respect to a GPG G as the highest expected total reward, starting at any node in G:

$$V_G(b) = \max_{v \in G} \int_{s \in S} b(s)\alpha_v(s)\,\mathrm{d}s. \tag{3}$$

IV. Algorithm

A. Overview

Our algorithm computes a GPG as an approximation to an optimal policy. Following the highly successful point-based approach for discrete POMDPs [14], [17], [25], we sample a set B of points from the belief space and perform value iteration asynchronously over B. However, we iterate over a GPG instead of a value function. It is well known that value iteration converges under very general conditions.

We start with an initial GPG G_0. For each action $a \in A$, we create a node $v = (a, \kappa)$ in G_0, with $\kappa(o) = v$ for all $o \in O$. Basically, G_0 corresponds to a set of single-action policies.

We then iteratively apply the Bellman backup equation at a belief $b \in B$ to improve the current GPG G:

$$HV_G(b) = \max_{a \in A} \Big\{ R(b,a) + \gamma \int_{o \in O} p(o|b,a) V_G(b_{ao})\,\mathrm{d}o \Big\}, \tag{4}$$

Algorithm 1 Perform backup of a GPG G at a belief b.

MC-Backup(G, b, M, N, K)

1: **for** each $a \in A$ **do**
2: $\quad \kappa_a \leftarrow$ Build-Classifier(G, b, a, N, K).
3: $\quad v_a \leftarrow (a, \kappa_a)$.
4: $\quad V_a \leftarrow 0$.
5: $\quad$ **for** $i = 1, 2, \ldots, M$ **do**
6: $\quad\quad$ Sample a state s from the distribution $b(s)$.
7: $\quad\quad V_a \leftarrow V_a +$ Simulate$(G \cup \{v_a\}, v_a, s)$.
8: $a^* \leftarrow \arg\max_{a \in A} V_a$.
9: $G' \leftarrow G \cup \{v_{a^*}\}$.
10: **return** G'

Algorithm 2

Build-Classifier(G, b, a, N, K)

1: Sample a set S' of N states from the distribution $b_a(s)$.
2: $\Gamma \leftarrow \emptyset$.
3: **for** each $v \in G$ **do**
4: $\quad$ **for** each $s \in S'$ **do**
5: $\quad\quad \alpha_v(s) \leftarrow 0$
6: $\quad\quad$ **for** $i = 1, 2, \ldots, K$ **do**
7: $\quad\quad\quad \alpha_v(s) \leftarrow \alpha_v(s) +$ Simulate(G, v, s)
8: $\quad\quad \alpha_v(s) \leftarrow \alpha_v(s)/K$
9: $\quad \Gamma \leftarrow \Gamma \cup \{\alpha_v\}$.
10: **return** (S', G, Γ).

where H denotes the Bellman backup operator and $R(b,a) = \int_{s \in S} R(s,a)b(s)\,\mathrm{d}s$. The backup operation looks ahead one step and chooses the action that maximizes the sum of the expected immediate reward $R(b,a)$ and the expected value of the next belief with respect to G. The result is a new GPG G' with one new node $v = (a^*, \kappa_{a^*})$ added to G, where a^* is the maximizer in (4) and κ_{a^*} is the associated edge classifier that maps an observation o to a node in G. To execute the corresponding policy $\pi_{G',v}$, we start at v and take action a^*. After receiving an observation o, we move to the node $\kappa_{a^*}(o)$ in G and follow $\pi_{G,v}$ from then on.

It is important to note that our algorithm does not explicitly represent the value function V_G. It performs lazy evaluation of V_G through sampling, whenever needed.

We now elaborate on the backup and the belief space sampling procedures in the next two subsections.

B. Monte Carlo Backup and Classifier Construction

To perform backup of G at b using (4), it involves integrating over continuous state and observation spaces. Our algorithm performs the evaluation approximately through sampling (Algorithm 1 and 2).

First, we construct an edge classifier κ_a for each $a \in A$ (Algorithm 1, line 2). By substituting (3) into (4), it is clear that κ_a must map an observation o to the best node $v^* \in G$, implying that v^* maximizes

$$\int_{s \in S} b_{ao}(s)\alpha_v(s)\,\mathrm{d}s = \eta \int_{s \in S} b_a(s)p(o|s,a)\alpha_v(s)\,\mathrm{d}s. \tag{5}$$

To evaluate the above integral, we sample a set S' of states according to the distribution b_a. For each $v \in G$ and each

sample $s \in S'$, we estimate the value of $\alpha_v(s)$ with a set of Monte Carlo simulations. The procedure SIMULATE(G, v, s) starts at the initial state s and the node $v = (a, \kappa)$ in G. To simulate taking action a in state s, it samples a state s' from the distribution $T(s, a, s') = p(s'|s, a)$ and an observation o from the distribution $Z(s', a, o) = p(o|s', a)$. The process then repeats from the state s' and the node $\kappa(o) \in G$. The simulation length is chosen so that the estimation error is sufficiently small, as a result of the discount factor γ. We collect all the estimates together in

$$\Gamma_{S',G} = \{\alpha_v \mid \alpha_v \colon S' \to \mathbb{R} \text{ and } v \in G\}$$

which basically contains sampled values of a set of α-functions. The tuple $(S', G, \Gamma_{S',G})$ provides all the information necessary for constructing the classifier κ_a:

$$\kappa_a(o) = \arg\max_{v \in G} \sum_{s \in S'} p(o|s)\alpha_v(s), \tag{6}$$

in which the sum approximates the integral in (5). The values in Γ serve as the classifier coefficients.

Geometrically, κ_a maps an observation o to a n-dimensional feature vector $[p(o|s_1), p(o|s_2), \ldots, p(o|s_n)]$ for $s_i \in S'$ and then performs the classification in this feature space. Keep in mind, however, that the feature space is attached to a specific belief b_a, though we do not make the dependency explicit to simplify the notation.

Although a finite number of samples are used to construct κ_a, our analysis provides a uniform error bound on the classifier's performance for any o from a continuous observation space (Theorem 1).

After constructing κ_a for each $a \in A$, we perform another set of Monte Carlo simulations to find the best action a^* and the associated classifier κ_{a^*} (Algorithm 1, line 3–8), resulting in a new GPG node (a^*, κ_{a^*}).

Algorithm 1 summarizes the backup procedure, which takes $\mathcal{O}(|A||G|NK + |A|M)$ simulations. Of the three parameters M, N, and K that control the number of simulations and samples, N dominates the running time. It also controls the quality of the computed policy, as it determines the representational complexity and the accuracy of classifiers.

C. Belief Space Sampling

There are several approaches to sample the belief space [14], [17], [25]. For space limitation, we give a very brief description here, as it is not the main focus of this work. One approach is to spread samples evenly over the belief space $\mathcal{B}$ to cover it [17]. This is practical only if $\mathcal{B}$ is sufficiently small. Instead, we build a belief tree with an initial belief b_0 as the root. Each belief b at a tree node has upper and lower bounds on its value. The lower bound is the value of b with respect to the current policy. To compute the upper bound, we relax the model, for example, by assuming the states are fully observable and solving the resulting MDP. To sample new beliefs, we traverse a single path down the tree in the direction that tends to shrink the gap between the upper and lower bounds at the root b_0. We then expand the leaf node and add a new belief node to the tree. To perform backup, we retrace this path back to the root and invoke MC-BACKUP at each node along the way. See [14] or [25] for details.

TABLE I
THE SIZE AND EXECUTION SPEED OF COMPUTED POLICIES.

Task	N	$\lvert G\rvert$	Speed (KHz)
LQG	50	1024	25.5
intersection navigation	500	20	1.7
acrobot	100	825	14.8

N : number of samples for each α-function in the classifier
$|G|$: number of GPG nodes
Speed : policy execution speed in the number of GPG nodes processed per second

V. EXPERIMENTS

We evaluated our algorithm on three tasks. In *linear-quadratic-Gaussian (LQG) control*, we can solve for the optimal policy analytically and use it to calibrate the performance of the new algorithm. In *intersection navigation*, we investigate the benefit of sampling the observation space O during the policy computation, rather than discretizing O a priori. Finally, in *acrobot*, we use the algorithm for Bayesian reinforcement learning in order to handle model uncertainty.

A. LQG Control

An LQG system is basically a POMDP with linear system dynamics, Gaussian noise, and a quadratic reward function. Our simple LQG problem is given by

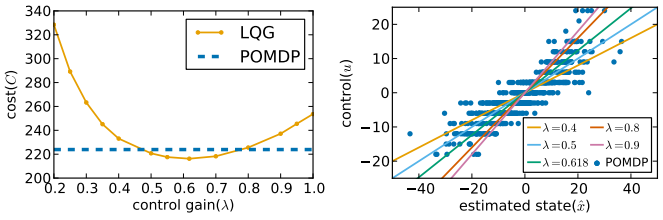
$$x_t = -x_{t-1} + u_{t-1} + w_t$$
$$y_t = x_t + v_t$$

where x_t, u_t, and y_t are the state, the action, and the observation at time t, and $w_t \sim \mathcal{N}(0, 10)$ and $v_t \sim \mathcal{N}(0, 10)$ represent zero-mean Gaussian system noise and observation noise. The goal is to minimize the infinite-horizon average cost $C = \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N} (x_t^2 + u_t^2)$. A linear feedback policy has the form $u_t = \lambda \hat{x}_t$, where $\hat{x}_t$ is the estimated mean state at time t and λ is the control gain. The optimal policy has $\lambda^* = 0.618$.

To recast the problem as a POMDP, we choose 17 equally spaced actions in the range $[-24, 24]$ and set the discount factor to 0.99 to approximate the infinite-horizon cost function. The state space and the observation remain unchanged. The computed policy contains $1,024$ policy graph nodes.

We evaluated the POMDP policy and several linear feedback policies with different λ by performing 10,000 simulations for each. Fig. 2 shows their costs and behaviors. The POMDP policy computation neither exploits the linearity of system dynamic nor possesses prior knowledge of the linear form of the optimal policy. Nevertheless, it discovers a policy that has a roughly linear form, up to action discretization and has a cost close to the minimum.

Table I shows the size of policies computed for this and the other two tasks, and their execution speed. The running times are obtained on a PC with a 2.83 GHz CPU and 4 GB memory. In case of multiple policies computed, Table I shows the results for the largest policy. The results confirm one main benefit



(*a*) (*b*)

Fig. 2. Comparing the LQG POMDP policy and the linear feedback policies. (*a*) Policy costs estimated from $10,000$ simulations. The dashed line indicates the cost of the POMDP policy. The curve plots the cost of linear feedback policies with different control gain λ. The standard errors of the estimated costs are all less than 1 and not visible on the plots. (*b*) Policy behaviors. For the POMDP policy, we plot the mean of the belief and the action associated with each policy graph node.

of the policy graph representation, very fast policy execution, which is important for some applications.

B. Intersection Navigation

Recall the example in Fig. 1. The autonomous vehicle R, in blue, stops at the intersection and waits for the other vehicle R' to clear before proceeding. R' cannot be localized accurately, as the measurements from R's proximity sensor are noisy. R wants to go through the intersection as fast as possible, while maintaining safety. So it must carefully balance exploration and exploitation by hedging against the noisy observations.

Our main objective here is to investigate the effect of observation modeling on the policy and not necessarily a high-fidelity model for vehicle navigation. We make a few simplifications to stay on the main issue. Assume that the vehicles move within a lane. The state space $S = [-10, 10]$ encodes the position of R', which is sufficient to decide the action of R. The initial belief on the position of R' is uniform over $[-10, 0]$. R has two actions. WAIT keeps R stopped. GO moves R forward through the intersection. There is no emergency stop in our simple model. If R goes through the intersection successfully, it gets a reward 1. If a collision occurs, it gets a large penalty $R_{\rm p}$. Hence the reward function

$$R(s,a) = \begin{cases} 0 & \text{if } a = \text{WAIT} \\ 1 & \text{if } a = \text{GO and } s \notin [-1,1] \\ -R_{\rm p} & \text{if } a = \text{GO and } s \in [-1,1] \end{cases} .$$

We tested two observation models. The first one follows the standard beam model for proximity sensing [28]. An observation $o = (h_1, h_2, \ldots, h_{30})$ consists of readings along 30 beams equally spaced over 160° field of view. We quantize each reading h_i into a binary value: $h_i = 1$ indicates that the ith beam hits R', and $h_i = 0$ indicates that the beam does not. There are false positives, due to unexpected obstacles, and false negatives, due to, *e.g.*, total reflection or glass. Let h_i^* denote true value for the ith beam. Our test uses a high-noise environment with $p(h_i = 1 | h_i^* = 1) = 0.7$ and $p(h_i = 0 | h_i^* = 0) = 0.9$. The beam model assumes that readings along the beams are independent: $p(o|s,a) = \prod_{i=1}^{30} p(h_i|s,a)$ [28].

The main difficulty with the beam model above is the high-dimensional observation space. With 2^{30} observations, no POMDP algorithm can cope. To avoid direct reasoning with the high dimensional observation space, our second model calculates the maximum-likelihood (ML) location x of R' from $o = (h_1, h_2, \ldots, h_{30})$, with x discretized into bins $X = \{-10, -9, \ldots, 9, 10\}$. Specifically, we have $x = \zeta(o) = \arg\max_{x \in X} p(x|o) = \arg\max_{x \in X} p(o|x)p(x)/p(o)$, where the prior $p(x)$ is uniform over X. We then use x as the observation for the POMDP model, resulting in only 21 observations in total. This drastic reduction in the number of observations, however, comes at a cost, as we see next.

TABLE II
PERFORMANCE COMPARISON OF POMDP POLICIES WITH TWO DIFFERENT OBSERVATION MODELS FOR INTERSECTION NAVIGATION.

$R_{\rm p}$	Observation	$\lvert G\rvert$	Time	Accident Rate
10	beam	14	2.61±0.0095	0.0029±0.00053
	ML	78	4.65±0.0015	0.0160±0.00014
100	beam	20	3.12±0.014	0.0009±0.00030
	ML	72	10.95±0.0037	0.0034±0.00005
1000	beam	18	5.03±0.030	0.0002±0.00014
	ML	41	12.92±0.00029	0.0004±0.000021

$|G|$: number of GPG nodes
Time : time to cross the intersection

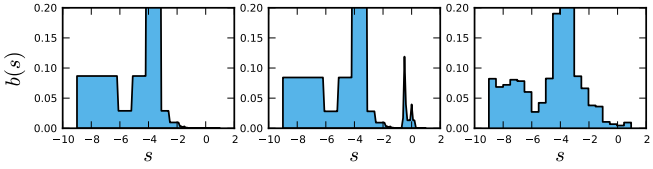


Fig. 3. Posterior beliefs b_1, b_2, and b, from left to right.

For the beam model, our new algorithm was the only option available to solve the resulting POMDP. For the ML model, we used the MCVI algorithm [4], which is specialized for continuous-state, discrete-observation POMDPs.

We solved several POMDP models with different values for the collision penalty $R_{\rm p}$ and evaluated each computed policy with $1,000,000$ simulations. The results are reported in Table II. Clearly the new algorithm with the beam model achieved consistently better results with lower accident rate and faster crossing time.

The performance gap results from *information loss* during the maximum-likelihood calculation. To understand this, consider a particular state $s = -4$ and choose two high-probability beam observations o_1 and o_2 from $p(o|s = -4)$ such that $\zeta(o_1) = \zeta(o_2) = x$. That is, we have the same ML location estimate for both o_1 and o_2 and cannot differentiate them in the ML observation model. Now consider the posterior beliefs b_1, b_2, and b (Fig. 3) for o_1, o_2, and x in their respective models, after R executes a single WAIT action and receives the observation. The posterior beliefs all have the same general shape. However, a careful comparison of b_1 and b_2 reveals a small secondary peak for b_2 in the region $[-1, 1]$, indicating the likely presence of R' in the intersection. A good policy must handle this low-probability, but critical event properly. Otherwise the vehicle will either get into an accident or unnecessarily wait. However, the ML model provides the same observation x whether it is actually o_1 or o_2, and the posterior belief b does not have a secondary peak. In general, there

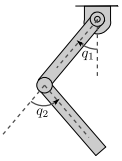


Fig. 4. The acrobot is a two-link articulated robot actuated only at the joint connecting the two links and thus unactuated. It resembles a gymnast swing on a high bar. In the standard acrobot, each link has mass $m = 1.0$ and length $\ell = 1.0$.

are 2^{30} beam observations, but only 21 ML observations. Many beam observations map into the same ML observation and cannot be differentiated in the ML model. This loss of information is a main contributor of the performance gap.

C. Acrobot with Model Uncertainty

Acrobot is a well-studied underactuated system (Fig. 4). In the swing-up task, the acrobot must get its tip above the height 1.95 and achieve the almost fully stand-up configuration. Our acrobot variant assumes that a key model parameter, the mass m of the acrobot's second link, is not known exactly, thus introducing model uncertainty.

This task is particularly challenging, because the acrobot dynamics is sensitive to m. An open-loop control policy that successfully swings up an acrobot with $m = 1.0$ fails completely on an acrobot with $m = 1.01$ [3]. To succeed, a control policy must simultaneously learn the acrobot's unknown parameter and plan the actions under an uncertain model. We apply the model-based Bayesian reinforcement approach [8] and formulate the task as a POMDP.

The POMDP state is $s = (q_1, q_2, \dot{q}_1, \dot{q}_2, m)$, where $q_1, q_2 \in [-\pi, \pi]$, $\dot{q}_1 \in [-4\pi, 4\pi]$, and $\dot{q}_2 \in [-9\pi, 9\pi]$ represent the joint angels and the angular velocities of the two links (Fig. 4). All the state variables, including m, are continuous. The acrobot can apply a torque $\tau \in \{-1, 0, +1\}$ at the elbow joint. We use the system dynamics equations in [26] and assume no action noise. An observation consists of the two joint-angle values under Gaussian noise with variance 0.1. The angular velocities $\dot{q}_1$ and $\dot{q}_2$ and the model parameter m cannot be observed directly. The reward is 10 if the acrobot reaches the specified height, and 0 for other states and actions. The discount factor is 0.95. The initial belief for m is uniform over $[0.95, 1.05]$.

Our new algorithm can solve this POMDP without a priori discretization of the state and observation spaces. State space discretization is difficult in general, because it introduces modeling errors that are difficult to quantify. It is exacerbated here by the acrobot's sensitive non-linear dynamics. Observation space discretization is also difficult, as it may lose information and degrade the quality of the computed policy (Section V-B). We will see further evidence of the difficulty here.

We solved the acrobot POMDP with different values for the sampling parameter N and evaluated each resulting policy with $10,000$ simulations. For comparison, we also evaluated an oracle policy, for which the model parameter m and the system state are fully observable. Table III shows that a relatively small N is sufficient to produce a good policy in this case. Increasing N consistently improves the results, as N controls the accuracy of edge classifiers and, in turn, policies.

Fig. 5 visualizes a particular edge classifier κ from the policy with $N = 100$. Each point in the plot represents an observation o collected from a simulation trace going through κ. The point is colored according to the output GPG node $\kappa(o)$. The observations fall into 6 classes, with very different sizes. The smallest one has width about 0.001. To obtain the same result with a regular discretization of the observation space, we have to use very fine resolution, roughly 0.001, in order to capture the small classes sandwiched between large ones. The resulting $1,000,000$ observations are beyond the reach of any discrete POMDP algorithm.

TABLE III
THE PERFORMANCE OF ACROBOT POMDP POLICIES WITH DIFFERENT VALUES OF SAMPLE PARAMETER N.

Policy	N	$\lvert G\rvert$	Average Height
oracle	-	-	1.97 ± 0.0000
POMDP	100	825	1.90 ± 0.0021
	50	871	1.87 ± 0.0027
	25	591	1.86 ± 0.0029
	10	815	1.84 ± 0.0031
	5	123	1.78 ± 0.0036
	3	244	1.66 ± 0.0144

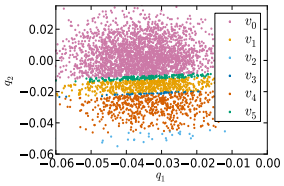


Fig. 5. Visualization of an edge classifier κ. Each point is a sampled observation o and colored according to the output GPG node $\kappa(o)$.

This example confirms again the difficulty of observation discretization. In practice, some observation discretization or aggregation is probably necessary. However, a priori discretization without a good understanding of its effect should be avoided. The new algorithm helps to reduce the need for aggressive discretization.

VI. Analysis

In this section, we analyze the approximation errors of our algorithm as a result of Monte Carlo sampling and provide a bound on its performance.

The analysis consists of four main steps showing that

1. given G, b and a, Algorithm 2 produces a classifier with uniformly bounded error for every observation $o \in O$ with high probability, if sample sizes N and K are sufficiently large;
2. for a given node $v \in G$, the same error integrated over all observations remains bounded, due to the uniform bound from the previous step (Theorem 1);
3. given G and b, the approximation error for a single backup (Algorithm 1) is bounded with high probability if M, N, and K are sufficiently large (Theorem 2);
4. finally, the accumulated approximation error after many backup steps is bounded with high probability, provided the sampled beliefs B approximate $\mathcal{B}$ well (Theorem 3).

We then conclude that the computed GPG converges to an optimal policy when M, N, and K are sufficiently large.

In the following analysis, we assume $R(s, a) \leq R_{\max}$ and $Z(s, a, o) = p(o|s, a) \leq P_{\max}$ for all $s \in S$, $a \in A$, and $o \in O$.

Define $V(b,a,o,v)$ to be the expression in (5):

$$V(b,a,o,v) = \int_{s\in S} b_a(s)p(o|s,a)\alpha_v(s)\,\mathrm{d}s. \tag{7}$$

Given belief b and $a \in A$, the optimal classifier $\kappa^*_{ba}(o)$ produces a node v that maximizes $V(b,a,o,v)$. By (6), Algorithm 2 computes a classifier $\kappa_{ba}(o)$ that produces a node maximizing a sampled approximation of (7)

$$\hat{V}(b,a,o,v) = \frac{1}{|S'|}\sum_{s\in S'} p(o|s,a)\alpha_v(s). \tag{8}$$

We define the error in step 1 as $|V(b,a,o,\kappa_{ba}(o)) - V(b,a,o,\kappa^*_{ba}(o))|$, for a fixed observation o. In step 2, we integrate over all $o \in O$. Define

$$V(b,a,\kappa) = R(b,a) + \gamma\int_{o\in O} V(b,a,o,\kappa(o))\,\mathrm{d}o. \tag{9}$$

The error for step 2 is then $|V(b,a,\kappa_{ba}) - V(b,a,\kappa^*_{ba})|$.

To analyze this error, we need to characterize the complexity of observation functions using a notion called the *covering number*. Let X denote a set of points in $\mathbb{R}^n$. Given $\epsilon > 0$, a finite subset $Y \subset \mathbb{R}^n$ *covers* X, if for every $x \in X$, there exists $y \in Y$ with $\|x-y\| < \epsilon$, where $\|x-y\| = \frac{1}{n}\sum_{i=1}^n |x_i - y_i|$. The covering number $\mathcal{C}(\epsilon, X)$ is the minimum number of points required to cover X.

Now consider a set of observation functions, $\mathbf{F}_a = \{f_{a,o} \mid f_{a,o}(s) = p(o|s,a), o \in O\}$ for some action $a \in A$. Let $\bar{s} = (s_1, s_2, \ldots, s_N)$ be a sequence of N states sampled uniformly at random from S, and $\mathbf{F}_{a|\bar{s}} = \{(f(s_1), f(s_2), \ldots, f(s_N)) \mid f \in \mathbf{F}_a\}$. In our analysis, we bound the complexity of observation functions by the maximum covering number

$$\mathcal{C}_Z(\epsilon, N) = \max_{a\in A}\sup_{\bar{s}\in S^N} \mathcal{C}(\epsilon, \mathbf{F}_{a|\bar{s}}). \tag{10}$$

Let

$$\rho_N(\epsilon,\tau) = \frac{2048(P_{\max}R_{\max})^2}{\epsilon^2(1-\gamma)^2}\left(\ln\left(4|G|\mathcal{C}_Z\left(\tfrac{\epsilon(1-\gamma)}{32R_{\max}}, N\right)\right) - \ln\tau\right).$$

The following theorem bounds the error between the optimal classifier and the approximate classifier computed.

Theorem 1. *Given a policy graph G, $b \in \mathcal{B}, a \in A$, a set S' of N states sampled independently from S according to $p(s|b,a)$, and a permissible class[1] of observation functions,*

$$p(|V(b,a,\kappa_{ba}) - V(b,a,\kappa^*_{ba})| > \epsilon) \le \tau \tag{11}$$

for any $\epsilon, \tau > 0$, if $N \ge \rho_N(\epsilon,\tau)$.

The theorem assumes that the observation space $O = [0,1]^n$, *i.e.*, an n-dimensional unit hypercube with the Euclidean metric. To simplify the presentation, we ignore the error of estimating $\alpha(s)$ with K Monte Carlo simulations (Algorithm 2). This error can be made arbitrarily small with sufficiently large K.

For the approximation error to converge to 0, Theorem 1 requires that for any $\epsilon > 0$, $\mathcal{C}_Z(\epsilon, N)\mathrm{e}^{-C\epsilon^2 N} \to 0$, as $N \to \infty$,

[1]Measurability conditions that usually hold in practice (see [10]).

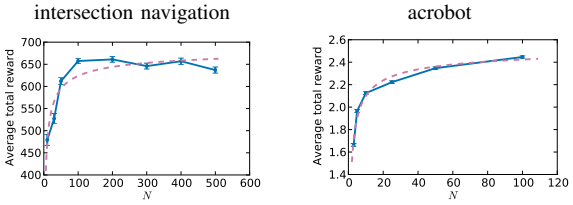


Fig. 6. Empirical convergence rates with respect to N. A solid line indicates the average total reward obtained from 100,000 simulations. A dashed line indicates the best-fit curve $a/\sqrt{N} + b$.

for a constant C. This condition is satisfied by many common classes of functions (see [10]), *e.g.*, the set of Gaussian functions $\mathbf{G} = \{g_o(s) \mid g_o(s) = \mathrm{e}^{-(s-o)^2}, s \in [0,1],\ o \in [0,1]\}$. Consider two functions from $\mathbf{G}$: $\mathrm{e}^{-(s-o)^2}$ and $\mathrm{e}^{-(s-(o+\epsilon))^2}$. By the Mean Value Theorem, for any $\epsilon > 0$,

$$|\mathrm{e}^{-(s-o)^2} - \mathrm{e}^{-(s-(o+\epsilon))^2}| = |\epsilon\cdot 2(s-c)\mathrm{e}^{-(s-c)^2}| \le \epsilon, \tag{12}$$

for some $c \in (o, o+\epsilon)$. Since (12) holds for any s, we have $\mathcal{C}(\epsilon, N) \le 1/\epsilon$. Finally, it is easy to show that $\mathcal{C}(\epsilon,N)e^{-C\epsilon^2 N} \le (1/\epsilon)e^{-C\epsilon^2 N} \to 0$ as $N \to \infty$.

In our setting, when the covering number $\mathcal{C}_Z(\epsilon, N)$ grows slowly with ϵ and N, the approximation error decreases at the rate of $\mathcal{O}(1/\sqrt{N})$ approximately. Figure 6 shows the empirical convergence rates for the intersection navigation and acrobot swing-up tasks. They correlate well with the theoretical analysis.

Theorem 1 bounds the error of a classifier computed for a given action. Algorithm 1 then performs backup and constructs the best policy graph node by selecting from the $|A|$ candidate action-classifier combinations by Monte Carlo simulation. Simulation introduces additional error in the backup.

In step 3, we bound the error of approximate backup: $|HV(b) - \hat{H}V(b)|$, where $HV(b)$ denotes the exact backup using (4) and $\hat{H}V(b)$ denotes the approximate backup produced by Algorithm 1. Let

$$\rho_M(\epsilon,\tau) = \frac{(P_{\max}R_{\max})^2}{2\epsilon^2(1-\gamma)^2}(\ln 2 - \ln\tau).$$

The following theorem bounds the error of a single step of point-based backup, as computed in Algorithm 1. The error has two parts, one part from errors in step 1 and 2 and the other part from Monte Carlo evaluation of the classifiers (Algorithm 1, line 5–7).

Theorem 2. *Given a policy graph G, a belief $b \in \mathcal{B}$, and a permissible class of observation functions,* MC-BACKUP(G, b, M, N, K) *produces an improved policy graph such that for any $\epsilon, \tau > 0$,*

$$p(|HV(b) - \hat{H}V(b)| > \epsilon) \le \tau$$

if $N \ge \rho_N(\epsilon/5, \tau/2|A|)$ and $M \ge \rho_M(\epsilon/5, \tau/2|A|)$.

Finally, in step 4, we combine all sources of error. We analyze the case where the algorithm runs synchronous MC-backup on a sampled belief set $B \subset \mathcal{B}$ for t iterations. Let $\delta_B = \sup_{b\in\mathcal{B}}\min_{b'\in B}\int_s |b(s) - b'(s)|ds$ denote the largest distance for any belief in $\mathcal{B}$ to its nearest point in the set

B where backup is performed. We bound the approximation error between the value function on the t-th iteration V_t and the optimal value function V^*. The result is similar to that for the MCVI algorithm [2].

By bounding the error propagation across the backup iterations, we obtain the following theorem:

Theorem 3. *Given a POMDP with a permissible class of observation functions, choose* $N \geq \rho_N(\epsilon/5, \tau/2|A||B|t)$, $M \geq \rho_M(\epsilon/5, \tau/2|A||B|t)$, *and perform* t *iterations of synchronous MC-backup over a sampled belief set* $B \subset \mathcal{B}$. *Then for every* $b \in \mathcal{B}$ *and every* $\epsilon, \tau > 0$,

$$|V^*(b) - V_t(b)| \leq \frac{\epsilon}{1-\gamma} + \frac{2R_{\max}\delta_B}{(1-\gamma)^2} + \frac{2\gamma^t R_{\max}}{1-\gamma},$$

with probability at least $1 - \tau$,

Theorem 3 shows that the approximation error comes from three main sources: MC-backup, the approximation of the belief space by a finite set of belief, and the finite number of back-up iterations. The error from MC-backup can be reduced by increasing the number of samples in the Monte Carlo sampling. The approximation of the belief space can be improved by using more belief points, and error decreases exponentially with the number of back-up iterations.

VII. Conclusion

This paper presents a new algorithm for solving POMDPs with continuous states and observations. These continuous models are natural for robotic tasks that require integrated perception and planning. We provide experimental results demonstrating the potential of this new algorithm for robot planning and learning under uncertainty. We also provide a theoretical analysis on the convergence of the algorithm.

Our algorithm uses sampling instead of fixed discretization to handle continuous state and observation spaces. Sampling opens up a range of new opportunities to scale up the algorithm for complex robot planning and learning tasks. Currently our algorithm performs a huge number of Monte Carlo simulations. It often takes hours to compute a policy for tasks at a scale similar to those in our experiments. Some of these simulations are redundant, and we are looking for ways to *reuse* simulations. We will apply standard techniques to prune the dominated α-functions and the GPG, thereby reducing the simulations needed. We will also explore parallelization to increase the practical performance of the algorithm.

Acknowledgments D. Hsu is supported in part by MoE AcRF grant 2010-T2-2-071 and National Research Foundation Singapore through the SMART IRG program. W.S. Lee is supported in part by the US Air Force Research Laboratory under agreement number FA2386-12-1-4031.

References

[1] J.A. Bagnell, S. Kakade, A. Ng, and J. Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, volume 16. The MIT Press, 2003.

[2] H.Y. Bai, D. Hsu, Mykel J. Kochenderfer, and W.S. Lee. Unmanned aircraft collision avoidance using continuous-state POMDPs. In *Proc. Robotics: Science and Systems*, 2011.

[3] H.Y. Bai, D. Hsu, and W.S. Lee. Planning how to learn. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2013.

[4] H.Y. Bai, D. Hsu, W.S. Lee, and V.A. Ngo. Monte Carlo value iteration for continuous-state POMDPs. In D. Hsu et al., editors, *Algorithmic Foundations of Robotics IX—Proc. Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)*. Springer, 2010.

[5] S. Brechtel, T. Gindele, and R. Dillmann. Solving continuous POMDPs: Value iteration with incremental learning of an efficient space representation. In *Proc. Int. Conf. on Machine Learning*, 2013.

[6] A. Brooks, A. Makarendo, S. Williams, and H. Durrant-Whyte. Parametric POMDPs for planning in continuous state spaces. *Robotics & Autonomous Systems*, 54(11):887–897, 2006.

[7] J.C. Davidson and S.A. Hutchinson. Hyper-particle filtering for stochastic systems. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2008.

[8] M.O.G. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts Amherst, 2002.

[9] E.A. Hansen. Solving POMDPs by searching in policy space. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 211–219, 1998.

[10] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[11] R. He, E. Brunskill, and N. Roy. Efficient planning under uncertainty with macro-actions. *J. Artificial Intelligence Research*, 40(1):523–570, 2011.

[12] J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, pages 1332–1338, 2005.

[13] K. Hsiao, L.P. Kaelbling, and T. Lozano-Pérez. Grasping POMDPs. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2007.

[14] 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.

[15] M. Lagoudakis and R. Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proc. Int. Conf. on Machine Learning*, 2003.

[16] N. Meuleau, L. Peshkin, K.E. Kim, and L.P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proc. Uncertainty in Artificial Intelligence*, pages 427–436, 1999.

[17] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, pages 477–484, 2003.

[18] R. Platt Jr, R. Tedrake, L.P. Kaelbling, and T. Lozano-Pérez. Belief space planning assuming maximum likelihood observations. In *Proc. Robotics: Science and Systems*, 2010.

[19] J.M. Porta, N. Vlassis, M.T.J. Spaan, and P. Poupart. Point-based value iteration for continuous POMDPs. *J. Machine Learning Research*, 7:2329–2367, 2006.

[20] P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems (NIPS)*, volume 16, pages 823–830. The MIT Press, 2003.

[21] W.B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, 2007.

[22] 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.

[23] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for POMDPs. *J. Artificial Intelligence Research*, 32(1):663–704, 2008.

[24] N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems (NIPS)*, volume 12, pages 1043–1049. The MIT Press, 1999.

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

[26] R. S. Sutton and A. G. Barto. *Reinforcement learning*. MIT Press, 1998.

[27] S. Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*. The MIT Press, 2000.

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

[29] J. van den Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. In *Proc. Robotics: Science and Systems*, 2010.

[30] Y. Wang, K.S. Won, D. Hsu, and W.S. Lee. Monte Carlo Bayesian reinforcement learning. In *Proc. Int. Conf. on Machine Learning*, 2012.