Partially Observable Markov Decision Processes (POMDPs) and Robotics

Hanna Kurniawati

School of Computing, Australian National University
Email: hanna.kurniawati@anu.edu.au

Xxxx. Xxx. Xxx. Xxx. YYYY. AA:1–25

https://doi.org/10.1146/((please add article doi))

Keywords

POMDP, planning under uncertainty, motion planning

Abstract

Planning under uncertainty is critical to robotics. The Partially Observable Markov Decision Process (POMDP) is a mathematical framework for such planning problems. It is powerful due to its careful quantification of the non-deterministic effects of actions and partial observability of the states. But precisely because of this, POMDP is notorious for its high computational complexity and deemed impractical for robotics. However, since early 2000, POMDPs solving capabilities have advanced tremendously, thanks to sampling-based approximate solvers. Although these solvers do not generate the optimal solution, they can compute good POMDP solutions that significantly improve the robustness of robotics systems within reasonable computational resources, thereby making POMDPs practical for many realistic robotics problems. This paper presents a review of POMDPs, emphasizing computational issues that have hindered its practicality in robotics and ideas in sampling-based solvers that have alleviated such difficulties, together with lessons learned from applying POMDPs to physical robots.

Contents

1. Introduction

The ability to compute reliable and robust decisions in the presence of uncertainty is essential in robotics. Specifically, an autonomous robot must decide how to act strategically to accomplish its tasks, despite being subject to various types of errors and disturbances affecting their actuators, sensors, and perception, and despite the lack of information and understanding about itself and its environment. The errors and limited information cause the effects of performing actions to be non-deterministic from the robot's point of view and cause the robot's state to only be partially observable, which means the robot never knows its exact state.

The Partially Observable Markov Decision Process (POMDP) (1, 2) is a mathematically principled framework to model decision-making problems in the non-deterministic and partially observable scenarios mentioned above. The POMDP quantifies the non-deterministic effects of actions and errors in sensors and perception stochastically. It estimates the robot's state as probability distribution functions over states, called beliefs, and computes the best actions to perform with respect to these beliefs, rather than single states. The computed action strategies will automatically balance information gathering and goal attainment. This concept is powerful: It is general and could enable robust operation even when the robot operates near environment boundary or near the limit of the robot's capability.

However, exactly because of its careful consideration of uncertainty, computing the exact optimal solution to a POMDP problem is computationally intractable (3). In fact, not long ago, most benchmark problems for POMDPs have less than 30 states and the best algorithms that could solve them took many hours (4, 5), which is grossly insufficient for realistic robotics problems. As a result, POMDPs were considered impractical for robotics and abandoned at the expense of robustness.

Nevertheless, in the past two decades, tremendous advances have been made in computing good action strategies for POMDP problems, thanks to the sampling-based approach. Although the computed strategies are not the optimal solution to the problems, they are often sufficient to substantially improve robustness. Hence, these progress enable the POMDP to become practical for a variety of realistic robotics problems.

In this paper, we describe an overview of these advances in POMDPs. We start by describing the POMDP problems and model in Section 2. Subsequently, we describe sampling-based methods that have advanced the practicality of POMDPs in robotics and the computational issues these methods alleviated. In Section 4, we present the implementation side of POMDPs in relation to robotics applications. Finally, we end with a brief discussion on the similarity of the progression of POMDPs and motion planning, as well as the relation between POMDPs and machine learning.

2. The Problem and POMDP Formulation

The POMDP is a natural representation of sequential decision-making problems where the results of actions are non-deterministic and the state is only partially observable. Sequential decision-making (aka. planning) is the problem of computing action strategies for a robot to achieve good long-term returns when actions may have long-term consequences. In such problems, the robot has some information about the effects of actions prior to execution, though these information may not be perfect nor complete. In other words, the robot's understanding of the actions' results are non-deterministic. The robot can use the perceived observations to help infer its state. However, in partially observable scenarios, due to errors in sensor measurements and in perception, the robot may perceive the same observations from multiple states, causing these states to be indistinguishable and the robot's exact state to remain unknown.

Many robotics problems fit the above planning in non-deterministic and partially observable scenarios. For example:

Underwater Navigation: How should an Autonomous Underwater Vehicle (AUV) navigates to a pre-specified goal, despite not knowing the exact underwater currents affecting its motion and despite substantial localization errors underwater?

Manipulation: How should a robot pick up an oil container from one location to another when it does not know exactly how full the container is? This lack of information means a relevant property of the problem is partially observable and the robot has uncertainty on the effect of its grasping. For example, if the container is almost empty, it will be easily moved and perhaps fall over when the robot tries to pick it up from the side.

Human Robot Collaboration: How to communicate effectively, so as to ensure effective collaboration with human, even though the robot does not know the exact characteristics nor intentions of the human? These variables are partially observed and due to a lack of information about the human characteristics and intention, the reaction of the human with respect to the robot's actions becomes non-deterministic.

The above examples are obviously far from being exhaustive in the robotics topics nor in the problems within each robotics topic, but hopefully they gave an indication of how diverse and common non-deterministic and partially observed planning problems are in robotics.

The above type of planning problems can naturally be formulated as a POMDP. Formally, the POMDP model is defined as a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$, where:

$\mathcal{S}$ denotes the state space —the set of all possible *states*, which can include the states of the robot and the environment.

$\mathcal{A}$ denotes the action space —the set of all *actions* the robot can perform.

$\mathcal{O}$ denotes the observation space —the set of all *observations* the robot can perceived.

$T(s, a, s')$ denotes the transition function, representing the non-deterministic effects of actions. It is a conditional probability function $P(s' \mid s, a)$ representing the probability that the robot will be in state $s' \in \mathcal{S}$ after performing action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$. In robotics, this function is sometimes

represented as a noisy dynamics function $s' = f(s, a, \eta)$, where $s, s' \in \mathcal{S} \subseteq \mathbb{R}^n$ and $\eta \sim \mathcal{N}$ is a noise vector sampled from noise distribution $\mathcal{N}$, while $f(\cdot)$ denotes the system's dynamics.

$Z(s', a, o)$ denotes the observation function, representing errors and noise in measurement and perception. It is a conditional probability function $P(o \mid s', a)$ that represents the observation the robot may perceive when it is in state $s' \in \mathcal{S}$ after performing action $a \in \mathcal{A}$.

R denotes the immediate reward function. This function can be parameterized by a state, a pair of state and action, or a tuple of state, action, and subsequent state.

A POMDP agent with model $\langle \mathcal{S}, \mathcal{A}, O, T, Z, R \rangle$ will operate as follows. At each time step, the agent is in some state $s \in \mathcal{S}$. However, due to partial observability, s is hidden to the agent, and instead the agent maintains a belief $b \in \mathcal{B}$ as an estimate of its state. The notation $\mathcal{B}$ denotes the belief space, which is the set of all beliefs. This set forms a simplex with $|\mathcal{S}| - 1$ dimensions due to the requirement that each belief must sum to 1. The agent infers the *best* action $a \in \mathcal{A}$ to execute from b (what *best* means is defined in the following paragraph). Once the action is performed, the hidden state may move to a new state $s' \in \mathcal{S}$. The state s' is hidden to the agent, but the agent perceives an observation $o \in O$ that may reveal some information about s'. The possible state s' the agent moves to and the observation o it may perceive follows the transition T and observation functions Z, respectively. Since the state s' is hidden to the agent, the agent updates its estimate of the state from b to belief b' via Bayesian inference based on the previous estimate b, the action a it just performed, and the observation o it just perceived. Finally, the agent receives a reward, based on the reward function R, from which the objective function, and hence best action is derived from. This sequence forms a single step of a POMDP agent, and the process repeats. Figure 1 illustrates this single step.

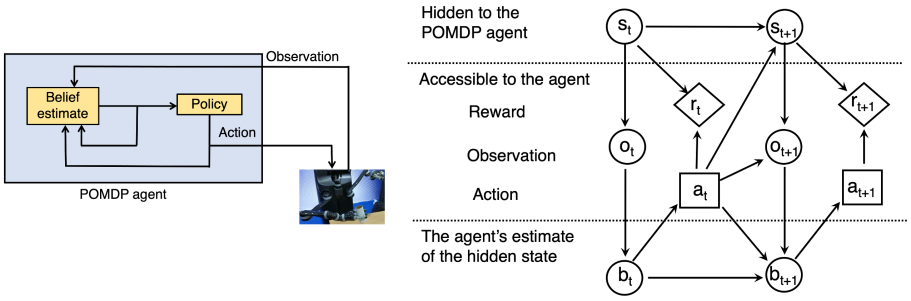

Figure 1

Illustration of a single time-step of a POMDP agent (a) and process (b).

Solving a POMDP problem modelled as $\langle \mathcal{S}, \mathcal{A}, O, T, Z, R \rangle$ means finding an optimal policy —that is, a mapping $\pi^* : \mathcal{B} \to \mathcal{A}$ from beliefs to actions that maximise the objective function. Many objective functions have been proposed. One that is often used in robotics is the following value function, which is based on the expected total discounted reward. This value function assumes the problem has an infinite horizon, meaning, at each time step, the POMDP agent can still move infinitely many steps.

$$V^*(b) = \max_{a \in \mathcal{A}} \underbrace{\left(R(b, a) + \gamma \underbrace{\sum_{o \in O} P(o \mid b, a) V^*(\tau(b, a, o))}_{J(b,a)} \right)}_{Q(b,a)} \tag{1.}$$

where $R(b,a) = \sum_{s\in\mathcal{S}} R(s,a)\cdot b(s)$ is the expected immediate reward, while $\tau(b,a,o)$ updates the belief estimate b after the agent performs action $a \in \mathcal{A}$ and perceives observation $o \in O$. Suppose $b' = \tau(b,a,o)$, then

$$b'(s') = P(s' \mid o,a,b) = \frac{P(o \mid s',a,b)\,P(s' \mid a,b)}{P(o \mid a,b)} = \frac{Z(s',a,o)\sum_{s'\in\mathcal{S}} T(s,a,s')\,b(s)}{\sum_{s''\in\mathcal{S}} Z(s'',a,o)\sum_{s\in\mathcal{S}} T(s,a,s'')\,b(s)} \quad 2.$$

The probability $P(o \mid a,b)$ can be computed as a normalizing factor, i.e., the denominator in eq. (2), to ensure the belief b' sums to one. The notation $\gamma \in (0,1)$ is a discount factor to ensure that the objective function for infinite horizon problems is well defined. Finding the best action from a belief b then involves solving an optimization of the Q-value $Q(b,a)$ for b and computing an estimation of the expected future total reward $J(b,a)$.

A related objective function is the finite horizon, where the POMDP agent has a finite number of steps to perform. In this case, the value function is non-stationary, and the expected total reward for a problem with horizon T is V_T^*, as defined below:

$$V_t^*(b) = \max_{a\in\mathcal{A}}\left(R(b,a) + \gamma \sum_{o\in O} P(o \mid b,a)V_{t-1}^*(\tau(b,a,o))\right) \text{ where } V_0^*(b) = \max_{a\in\mathcal{A}}(R(b,a)) \quad 3.$$

where $V_t^*(b)$ is the value of b when the POMDP agent can still move for t steps.

Another related and commonly used objective function in robotics is Goal-POMDP, or otherwise known as Shortest Path POMDP. Goal-POMDP assumes the state space contains a set of goal states. Let's denote this set of goals as $G \subset \mathcal{S}$. The reward function of a Goal-POMDP problem reflects the cost of actions, and the objective is then to reach a target belief with the lowest total cost. A target belief b is one where $b(s) = 0$ whenever $s \notin G$. A Goal-POMDP is equivalent to a POMDP with expected total discounted reward (6), in the sense that one can be transformed into another without changing the optimal policy and value function.

Throughout this paper, we will focus on POMDPs with expected total discounted reward (eq. (1)). The optimal value function of such POMDPs can be approximated arbitrarily closely by a Piecewise Linear Convex function (7). Furthermore, this infinite horizon objective function has a benefit that the optimal value function, and hence the optimal policy, is stationary.

Note that the state, action, and observation spaces of a POMDP model can be discrete or continuous. When the state and/or observation spaces are continuous, the summation in eq. (1) – eq. (3) are replaced with integrations over the respective spaces. In this paper, we focus on discrete and finite state, action, and observation spaces, unless otherwise stated.

2.1. Example

To make the above definition concrete, let's take a navigation problem as an example. Suppose a robot navigates in a GPS-denied environment (illustrated in Figure 2), discretized into uniform grid of size $d \times d$. Today's POMDP solvers can compute good policy for navigation in continuous state space, but discrete state space provides better clarity for our discussion. The robot needs to navigate to a goal cell, entering cells that are not robot-friendly (e.g., due to obstacles, challenging terrain, etc.). The goal-cell is equipped with a sensor, such that the robot knows exactly if and when it has entered it,

while two beacons are located in the environment to help the robot localise itself with small errors when it is within k cells away from the beacons.

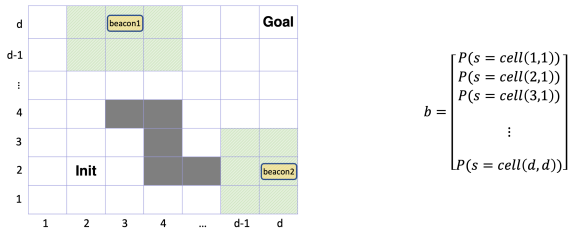

Figure 2

An example of navigation in GPS-denied environment, with the robot starting from cell-(2,2) marked as Init and the goal cell being in cell-(d, d) marked as Goal. The dark grey cells are non-robot friendly cells. Two beacons are set to help the robot localise, with beacon1 in cell-(3, d) and beacon2 in cell-(d, 2). The robot can localize with some error within k distance from a beacon (in this example, $k = 1$ and the cells are marked with green diagonals). Outside these cells and the goal cell, the robot receives no observation. The probability vector b is a belief representation for this problem.

In this example, the robot can be represented as a POMDP agent, defined as $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$ where:

$\mathcal{S}$**:** Robot's location, which is the set {cell-(1, 1), cell-(1, 2), cell-(1, d), ..., cell-(d, d)}.

$\mathcal{A}$**:** In this example, we take the simplest case of moving 1 cell in 4 direction, $\{North, South, East, West\}$.

$\mathcal{O}$**:** In this particular example, the observation space is a joint product between the distance to beacon1 and beacon2, {(n.a. from beacon1, n.a. from beacon2), (n.a. from beacon1, in cell containing beacon2), (n.a. from beacon1, within 1 cell from beacon2), ..., (within k cells from beacon1, within k cells from beacon2), in goal cell}. If the goal cell sensor is not perfect, then the observation space should be a joint product between the distance to both beacons as well as the goal-cell observation.

$T(s, a, s')$**:** A conditional probability function representing uncertainty in the effect of the robot's movement, e.g.,

- $P(s' = \text{cell-(2, 3)} \mid s = \text{cell-(2, 2)}, a = North) = 0.8$, $P(s' = \text{cell-(3, 3)} \mid s = \text{cell-(2, 2)}, a = North) = 0.1$, and $P(s' = \text{cell-(1, 3)} \mid s = \text{cell-(2, 2)}, a = North) = 0.1$
- $P(s' = \text{cell-(d, d)} \mid s = \text{cell-(d, d)}, a = *) = 1$ for any action taken (denoted as $*$) to indicate the goal state (cell-(d, d)) is an absorbing state

We can represent this transition function as four probability matrices of size $d^2 \times d^2$, where each matrix represent the state transition for each action.

$Z(s', a, o)$**:** A conditional probability function representing sensing errors, e.g., $P(o = \text{(n.a. from beacon1, within 1 cell from beacon2)} \mid s' = \text{cell-(d-1, 2)}, a = North) = 0.8$ and $P(o = \text{(n.a. from beacon1, within 0 cell from beacon2)} \mid s' = \text{cell-(d-1, 2)}, a = North) = 0.2$. Similar to the transition function, the observation function can be represented as four probability matrices, where each matrix represent the action the robot has just performed. However, the size of each matrix matrix is $|\mathcal{S}| \times |\mathcal{O}|$.

R**:** A real-valued function indicating the desirability of being at a particular state and the cost of movement, e.g., $R(s = \text{cell-(2, 2)}, a = North) = R(s = \text{cell-(2, 2)}, a = South) = R(s = \text{cell-(2, 2)}, a = East) = R(s = \text{cell-(2, 2)}, a = West) = \text{movementCost} = -1$, while the reward

for being in a non-robot friendly cell, such as $R(s = \text{cell-}(4, 2),\ a = *) = -10$, and the reward at the goal state $R(s = \text{cell-}(\text{d}, \text{d}),\ a = *) = 10$ for any action.

Note that many of today's POMDP solvers can work with generative models of transition, observation, and reward functions, which means the probabilities and reward functions do not need to be specified as explicitly as above.

Now, notice that due to the motion uncertainty and errors in sensing, the robot's position are generally not known exactly. To estimate the robot's position, POMDPs use beliefs, which in this example are probability distributions over the grid-cells and indicate the possible location of the robot. The belief space for this problem is a $d^2 - 1$ dimensional simplex, embedded in $\mathbb{R}^{d^2}$. For computation, each belief is generally represented as a probability vector of size d^2 with each element representing the probability the robot is at a particular cell. At each time step, the robot maintains a belief estimate and updates it based on the the action it performed and the observation it perceived.

Finding a policy that satisfies the infinite horizon value function as described in eq. (1) means that although the robot can take as many steps as it likes to navigate from the initial to the goal state, but reaching the goal state earlier will provides higher reward because the discount is less.

3. Sampling-Based Approximate POMDP Solvers

Finding the optimal solution to a POMDP problem is PSPACE-hard (3). Different sub-classes of POMDPs have slightly different hardness results, though most are still hard for classes above P (8, 9). Note, however, that planning under uncertainty in robotics is known to be a hard problem. For instance, motion planning for a 3D point robot with uncertainty in control and localization is PSPACE-hard (10), and if this robot is compliant, in the sense that when the robot is commanded to move through an obstacle, it complies with the obstacles' geometry instead of forcing itself to go through an obstacle (11), the problem is NEXP-hard (11). These results indicate that in general, the robotics problems of planning in non-deterministic and partially observable scenarios are computationally hard, even if they are not formulated as POMDPs.

Many methods to find the optimal policy to POMDP problems have been proposed. A survey of such methods are available in (5). However, the high computational complexity of these methods made them impractical for many realistic robotics problems.

A major breakthrough for POMDPs' applications in robotics comes when the sampling-based approximate POMDP solver (12) demonstrate that it can compute good policies for a problem with 870 states, in contrast to problems with under 30 states, which was the majority of the benchmark at the time. In this paper, we focus on the sampling-based approach for computing good POMDP policies and describe details of some of the methods under this approach in Section 3.1–Section 3.5. Now, let's first discuss an overview of the approach.

Sampling-based approximate POMDP solvers relax the optimality requirement to approximate optimality and restricts the problem only to scenarios where the POMDP agent starts from a given initial belief (let's denote this belief as b_0). Key to the approach is it samples a set of representative beliefs and computes the best action to perform only from the set of sampled beliefs, rather than the entire beliefs, thereby substantially reducing the complexity of finding good POMDP policies. Which set would be sufficiently representative and how difficult would it be to find such a set have been explored in (13), utilising the notion of set cover.

Many methods under the above mentioned approach have been proposed. However, they can in general be abstracted into the program skeleton in Algorithm 1. These methods iteratively sample a set of beliefs and estimate the values of these sampled beliefs until stopping criteria are satisfied.

Algorithm 1 A typical program skeleton for sampling-based POMDP solvers

1: Initialize policy π and a set of sampled beliefs B
   {Generally, B is initialised to contain only a single belief (e.g., the initial belief b_0)}
2: **repeat**
3: Sample a (set of) beliefs {Some methods sample histories (a history is a sequence of action–observation tuples) rather than beliefs. In POMDPs, beliefs provide sufficient statistics of the entire history (14), and therefore the two provide equivalent information}
4: Estimate the values of the sampled beliefs
   {Generally, via a combination of heuristics and update / backup operation}
5: Update π {In most methods, this step is a byproduct of the previous step}
6: **until** Stopping criteria is satisfied

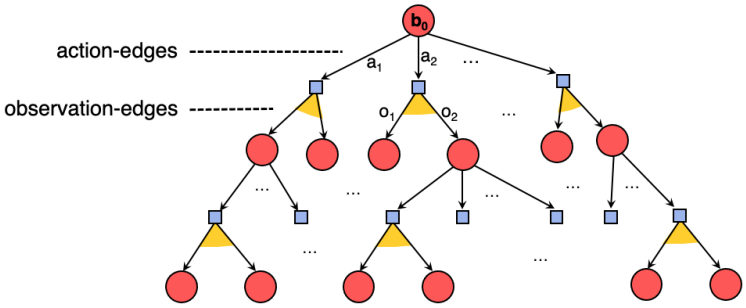

Figure 3

Illustration of belief tree $\mathcal{T}$. Circles represent beliefs.

A variety of sampling strategies have been proposed and are often critical to the performance of the method. Recall that sampling-based approach assumes the POMDP agent starts from a given initial belief b_0. This assumption implies that sampling beliefs from the set of beliefs reachable from b_0 (denoted as $\mathcal{R}(b_0)$) is sufficient. Therefore, the set of sampled beliefs can be represented as a belief tree, denoted as $\mathcal{T}$ and illustrated in Figure 3. This tree is akin to an AND-OR tree, where the nodes of $\mathcal{T}$ represent sampled beliefs, with the root representing b_0. The OR-edges are labelled by the actions they represent and referred to as action-edges, while the AND-edges are labelled by the observations they represent and referred to as observation-edges. For writing compactness, we use the same notation for the node and the belief it represents, and for the edges and the actions/observations the edges represent. A node b' is a child of b, connected by an action edge labelled $a \in \mathcal{A}$ and an observation edge labelled $o \in \mathcal{O}$ in $\mathcal{T}$, whenever $b' = \tau(b, a, o)$. Sampling a belief is then equivalent to selecting a node b of $\mathcal{T}$ to expand, together with an action $a \in \mathcal{A}$ and an observation $o \in \mathcal{O}$ to expand the node b. The newly sampled belief is computed as $b' = \tau(b, a, o)$, which is the belief reached after the agent with belief b performs action a and perceives observation o. Hence, the different strategies for sampling beliefs can simply be viewed as different strategies to construct and expand $\mathcal{T}$.

Similarly, multiple methods have been proposed to estimate the values of the sampled beliefs. Despite the variety, they all are based on the observation that for a node b of the belief tree $\mathcal{T}$, each path from b to a leaf node in $\mathcal{T}$ is a possible future the agent may encounter. Therefore, to estimate the

expected total future reward (aka., the value) of b, the value estimates of the descendants are back-propagated (often called backup) to b—that is, $\hat{V}(b) = \max_{a \in E_A(b)} \left(R(b,a) + \gamma \sum_{o \in E_O(b,a)} \hat{V}(b'_{b,a,o}) \right)$, where $E_A(b) \subseteq \mathcal{A}$ is the action-edges of b, $E_O(b,a) \subseteq \mathcal{O}$ is the observation-edges after following the edge $a \in E_A(b)$ from the node b, and $b'_{b,a,o}$ is the node child of b via action-edge $a \in E_A(b)$ and observation-edge $o \in E_O(b,a)$. This backup operation is applied iteratively from the leaves of $\mathcal{T}$ to the root node, so as to improve the estimated Q-values of performing different actions from the nodes of $\mathcal{T}$, which in turn improves identifying the best action to perform from the sampled beliefs, and hence the policy π. As indicated in Algorithm 1, this process of belief sampling and improvement of value estimate is repeated until stopping criteria are satisfied.

Most sampling-based approximate POMDP solvers are anytime, which means they can return a solution when stopped at any time, though of course there is a trade-off between the quality of the solution and the time the method has run. Therefore, the stopping criteria for this approach to solving POMDPs is often set to be the available planning time. Some methods (15, 16, 17) compute upper and lower bound estimates of the value functions, and hence the stopping criteria can be set to be when the difference between the upper and lower bounds for the initial belief b_0 is less than a pre-specified threshold. However, for practical purposes, for most realistic robotics problems, even methods that compute these upper and lower bounds often stop before the desired threshold gap is reached.

Many sampling-based approximate POMDP solvers can be broadly divided into offline and online. Offline solvers compute an approximately optimal POMDP policy π prior to execution. During execution, the agent only needs to estimate its current belief and execute the action $\pi(b)$. Online solver, on the other hand, interleaves policy computation and execution: At each step, prior to execution, the solver will compute the good action $a \in \mathcal{A}$ to perform from the current belief b. Once the action a is performed, the agent perceives an observation, updates its belief, and the process repeats.

Regardless of offline or online, these sampling-based methods aim to alleviate one or more of the following difficulties, which is crucial to enable POMDPs to become practical in robotics.

1. Large state space
2. Long planning horizon
3. Large observation space
4. Large action space
5. Complex transition dynamics

Among these issues, large state space and long planning horizon are two most discussed issues to date, often referred to as the curse of dimensionality and the curse of history, respectively. However, other issues become equally important when applying POMDPs to realistic robotics problems. In fact, the difficulty of solving a POMDP problem is influenced by a combination of problem characteristics related to the above issues, together with additional problem characteristics, such as the sparsity of the transition and observation functions. The work in (13) derives a criteria that captures these combined problem characteristics to identify how difficult different POMDP problems are for sampling-based methods, though this derived criteria is not always easy to compute.

In the next subsections, we describe the above issues in finding good POMDP policies, together with the sampling-based methods that have been proposed to explicitly alleviate them. Although the methods presented are not exhaustive, but we hope they provide some insights on the ideas that have significantly improve the practicality of POMDPs in robotics.

3.1. Large State Space

This issue is known as the curse of dimensionality: A POMDP solver must reason in the belief space, which is an $(n-1)$ dimensional continuous space, where n is the size of the state space. Key to sampling-based methods is that they restrict estimating the value function only on a set of representative beliefs, selected in an inexpensive manner via sampling. They relax the optimality requirement to substantially improve the scalability of POMDP solvers, with large state space being one of the first issues these solvers try to address. Below are some of the offline and online sampling-based methods that directly try to address the issue of large state space.

Offline Methods

Point-Based Value Iteration (PBVI) (12) was the first approximate POMDP solver that demonstrated good performance on problems with hundreds of states, i.e., an 870 states Tag (target finding) problem, albeit taking ~50 hours.

PBVI samples beliefs from the set $\mathcal{R}(b_0)$ of beliefs reachable from a given initial belief b_0 that are far from the already sampled beliefs. Specifically, given a set of sampled beliefs $B \subset \mathcal{B}$, PBVI expands the set by performing a single-step forward simulation for each pair of belief $b \in B$ and action $a \in \mathcal{A}$. Note that naively, this step will generate $|O|$ many beliefs for each belief and action pair. PBVI keeps only one of the resulting beliefs for each $b \in B$ as the newly sampled belief and add it to B. The belief kept is the one farthest away from any belief already in B based on L1 metric.

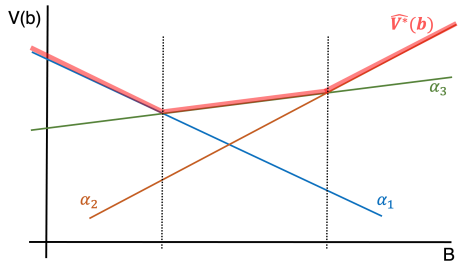

Figure 4

An illustration of $\Gamma = \{\alpha_1, \alpha_2, \alpha_3\}$ with three α-vectors. The X-axis represents beliefs, the Y-axis represents the values of the beliefs. Each linear function is represented by its gradient, which in this case is $\alpha_i \in \Gamma$ where $i \in [1, 3]$. The estimated optimal value function $\hat{V}^*(b)$ represented by Γ is the upper envelope of the three linear functions (marked as the thick red line).

Now, recall that the optimal value function eq. (1) can be approximated arbitrarily closely by a Piecewise Linear Convex (PWLC) function, as illustrated in Figure 4. PBVI utilises this characteristics by representing the estimated value function and policy using α-vectors. This representation maintains a finite set of α-vectors, denoted as Γ, where each α-vector represents the gradient of a linear function component of the PWLC value estimate. Therefore, given Γ, the estimated optimal value function can be computed as $\hat{V}^*(b) = \max_{\alpha \in \Gamma} \alpha \cdot b$, where $\alpha \cdot b$ represents the inner product between the two vectors, whose sizes are the same as the number of states in $\mathcal{S}$.

The question is then how does the set Γ relate to a policy. Intuitively, each $\alpha \in \Gamma$ corresponds to a policy tree T_{π_α}, where each node is associated with an action in $\mathcal{A}$ and each edge is associated with an observation in O. The value $\alpha(s)$ is then the expected total reward of starting from state $s \in \mathcal{S}$, executing the action associated with the root of T_{π_α}, traversing down the path in T_{π_α} based on the observation perceived, and executing the associated actions at each node of T_{π_α}. Each $\alpha \in \Gamma$ is then associated with the action at the root of the policy tree T_{π_α}. When a vector $\alpha \in \Gamma$ maximizes $V^*(b)$, the

policy maps the belief b to the action $a \in \mathcal{A}$ that is associated with α. More details about α-vectors representation of a POMDP policy are available in (4).

Point-based backup is used by PBVI to estimate the value function (eq. (1)). This backup operation computes the value function (eq. (1)) only at a finite set of sampled beliefs. It maintains a single α-vector for each sampled belief. Given an existing policy Γ and a newly sampled belief b, the new vector α that corresponds to b is constructed by assigning $\alpha_a(s) = R(s, a)$ and computing $\alpha = \arg\max_{a \in \mathcal{A}} \ \alpha_{b,a} \cdot b$, where $\alpha_{b,a} = \alpha_a + \sum_{o \in O} \arg\max_{\alpha \in \Gamma} \ \alpha \cdot \tau(b, a, o)$. Finally, α is added to Γ.

The idea of applying backup operation on only a finite set of beliefs have been proposed since the early work (18) and multiple subsequent works (4, 19). However, to ensure optimality, these methods select the set of beliefs systematically, which is expensive. The work in (20) introduces Point-based Dynamic Programming Update with backup operation that is very similar to the backup operation of PBVI. It selects beliefs based on some heuristics, which is much faster than the systematic selection proposed in (4, 18, 19). However, they interleave point-based backup with the much more expensive standard dynamic programming backup, to ensure optimality of the solution. By relaxing the optimality requirements, PBVI performs only point-based backup and replaces the expensive belief selection with inexpensive belief sampling, resulting in significant scaling up of POMDP solving capabilities.

Another sampling-based method, Perseus (21), separates belief sampling from backup operation, in the sense that backup is not performed to all sampled beliefs. Perseus uses α-vectors to represent the value function and policy and uses point-based backup too. However, by performing backup operation only on a subset of the sampled beliefs, it generates a smaller set of α-vectors, and hence reduces the memory requirements.

Later offline methods substantially improve the performance of PBVI and Persues further. For instance, Heuristic Search Value Iteration (HSVI) (15), and specifically HSVI2 (16), took 2 hours to generate a policy for Tag that has better quality than the policy generated by PBVI after 50 hours. Whilst, Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) (17) generates a better policy for Tag than the one generated by HSVI2 with only 6 seconds computation time. Since then, HSVI2 and SARSOP have been demonstrated to generate good policies for problems with over 15K states and 1K observations, while SARSOP has also been shown to generate good policies for RockSample(10,10) benchmark (15), which has over 100K states (22). Below, we present an overview of both HSVI2 and SARSOP, highlighting their strategies for sampling beliefs.

HSVI2 uses α-vectors policy representation and point-based backup, but differ from PBVI in its sampling strategy. HSVI2 maintains a lower and upper bound estimates of the value function, where the upper bound is used to guide sampling and is initialized with the value function of the fully observable (i.e., the Markov Decision Process (MDP)) simplification of the POMDP problem. This upper bound is represented as a set of points $U \subset \mathcal{B} \times \mathbb{R}$ and computed using sawtooth approximation (14). The lower bound is the current policy and is represented as a set Γ of α-vectors. Each sampled belief $b \in B$ is associated with a lower and upper bound, denoted as $\underline{V}(b)$ and $\overline{V}(b)$, where $\underline{V}(b)$ is associated with a vector $\alpha \in \Gamma$ and $\overline{V}(b)$ is associated with a point $u \in U$.

HSVI2 maintains the set of sampled beliefs in a belief tree, denoted as $\mathcal{T}$, where the nodes represent beliefs and an edge labelled with a pair of action–observation a–o from b to b' means there is an action $a \in \mathcal{A}$ and an observation $o \in O$, such that $b' = \tau(b, a, o)$. We will use the same notation for a node and the belief it represents. The root of $\mathcal{T}$ represents the initial belief b_0. HSVI2 interleaves belief sampling and backup until the gap between the upper and lower bound of b_0 is sufficiently small —that is, $|\overline{V}(b_0) - \underline{V}(b_0)| \leq \epsilon$ for a small threshold ϵ. To sample beliefs, HSVI2 performs multiple sequences

of forward simulations, starting from b_0 and following a path down the tree $\mathcal{T}$. Given a belief b, a single-step forward simulation selects the action with the best upper bound $a = \arg\max_{a \in \mathcal{A}} \overline{Q}(b, a)$ and observation with the highest excess uncertainty $o = \arg\max_{o \in \mathcal{O}} |\overline{V}(b_0) - \underline{V}(b_0)| - \gamma\epsilon^{-t}$, where γ is the discount factor and t is the depth of node b' in $\mathcal{T}$. The belief $b' = \tau(b, a, o)$ is then set as the child node of b in $\mathcal{T}$ via an edge labelled a–o. This single-step forward simulation process is repeated down the tree until the gap between the upper and lower bound of the newly added belief contribute less than the threshold ϵ to such a gap at b_0.

Now, SARSOP uses α-vectors policy representation, point-based backup, maintains upper and lower bounds estimates, and represents the set of sampled beliefs B as a belief tree $\mathcal{T}$ too. However, SARSOP explicitly aims to sample from the set of beliefs $\mathcal{R}^*(b_0)$ reachable from b_0 under the optimal policy. Although sampling from the set of beliefs $\mathcal{R}$ reachable from b_0 (as is PBVI and HSVI2) has significantly improved the scalability of POMDP solving, sampling useful beliefs —that is beliefs in or around $\mathcal{R}^*(b_0)$— becomes increasingly harder to sample for deeper levels of the belief tree because the size of $\mathcal{R}$ increases much faster than that of $\mathcal{R}^*$.

Of course, $\mathcal{R}^*$ is not known a priori, as otherwise we would have found the optimal policy. Therefore, SARSOP interleaves predicting the optimal value function with belief sampling. The prediction step uses a simple learning mechanism, where the belief space is discretized into bins based on features of the beliefs (in this case, the initial upper bound and entropy). The predicted value of a new belief b in $\mathcal{T}$ is then the average of the values of the sampled beliefs that lie in the same bin as b. If the bin is empty, the predicted value is set to be the upper bound. If the predicted value of b is higher than a target value, which indicates a better estimate of $V(b)$ may improve $V(b_0)$, SARSOP proceeds to expand b using single-step forward simulation similar to the one used in HSVI2. The target value at b is the lower bound of the root $\underline{V}(b_0)$ that has been propagated down from b_0 to b in the tree $\mathcal{T}$.

Furthermore, since value estimate and belief sampling are interleaved, beliefs in B that have been sampled early in the process may be based on a poor estimate of the value function. To keep B small and as close as possible to the set $\mathcal{R}^*(b_0)$, SARSOP prunes branches of $\mathcal{T}$ that are provably sub-optimal —that is, when $\overline{Q}(b, a) < \underline{Q}(b, a')$ for a node b in $\mathcal{T}$ and $a, a' \in \mathcal{A}$, all descendants of b via the edges labelled a–*, where * is any observation $o \in \mathcal{O}$, are pruned. Furthermore, SARSOP prunes a vector $\alpha \in \Gamma$ whenever it is δ-dominated by another vector in Γ at all points in B. The vector α is δ-dominated at belief $b \in B$ whenever $\alpha \cdot b' < \alpha' \cdot b'$ for all beliefs $b' \in B$ that are δ distance from b.

We hope the relatively detailed description of the three solvers above illustrates how substantial improvement in the scalability of POMDP solving can be achieved by altering only how beliefs are sampled, indicating the importance of this component.

The solvers described above relies on Value Iteration. Sampling-based approach has also been applied to Policy Iteration quite early on in Point-Based Policy Iteration (PBPI) (23). This methods represents policy explicitly as a Finite State Controller (FSC) together with the value function, as represented by the set of α-vectors. PBPI replaces the exact policy improvement step of the Policy Iteration method in (24) with the point-based backup used in PBVI together with PBVI's belief sampling strategy.

All of the above solvers assume that the state space, and also the action and observation spaces, are finite and discrete. Work have been proposed to extend them to continuous spaces. Many work in this extension focus on the policy representation. For instance, (25) proposes a point representation. Here, the value function is represented by a set of beliefs B along with their estimated Q-values. Given a new belief b, the Q-value for b and each action in $\mathcal{A}$ can be computed as an average of the Q-values for the particular action at b's k-nearest beliefs in B, where distance is computed using KL-divergence. The

method in (26) extends point-based methods to find good policies for POMDPs with continuous state space by replacing the value function representation α-vectors with α-functions, and using Gaussian mixture to represent the belief, transition, and observation functions. Monte Carlo Value Iteration (MCVI) (27) proposes a policy graph representation, where each node v in the policy graph represents an action and is associated with an α-function, where $\alpha_v(s)$ is the expected total reward of executing the policy graph when the agent starts at state s and execution starts by executing the action at node v. Another line of work, Guided Cluster Sampling (GCS) (28), represents the policy using either point representation (25) or policy graph (27), but focuses on the belief sampling strategy to alleviate the difficulty of sampling representative beliefs when the state space of the POMDP problem has many continuous state variables. We will see in the next subsection that online methods representation that no longer requires global value function representation, such as α-functions, makes it easier to scale-up solving capabilities to problems with continuous state spaces.

Online Methods

Online methods further improve the scalability of computing good POMDP policies by focusing to compute only the best action to perform from the current belief, rather than a policy for $\mathcal{R}(b_0)$ or $\mathcal{R}^*(b_0)$. The best action to perform is computed right before execution, and therefore time to compute them is in general very limited. However, by focusing on only the current belief, online methods have much lower memory requirements compared to offline methods, which is a major hindrance for further scalability of offline methods.

Moreover, compared to offline methods, most online methods take a different approach for belief sampling. They construct the belief tree $\mathcal{T}$ by sampling sequences of states–action–observation, and cluster the set of states reached via the same sequence of action–observation together to form the set of particles representing the same belief. The exact details of this sampling are different for different online methods, and some of them are discussed in this section.

RTDP-Bel (29) is one of the first online sampling-based methods for solving POMDPs approximately. It is designed for Goal-POMDP. However, the method presented in (6) can transform any discounted POMDP to Goal-POMDP. RTDP-Bel maintains a hashtable of discretized estimated values of the beliefs. Given the current belief $b \in \mathcal{B}$, RTDP-Bel performs a one-step forward simulation for each pair of b–a, where $a \in \mathcal{A}$, and uses a heuristics to estimate the expected total future reward. Specifically, for each b–a pair, it samples a state s from b, a subsequent state s' based on T (s, a, s'), and an observation o based on Z (s', a, o). It then computes $Q(b,a) = R(b,a) + \sum_{o \in \mathcal{O}} P(o \mid b,a)V(b')$ where $b' = \tau(b,a,o)$. The value $V(b')$ is computed using a heuristics or based on the values of the beliefs within the same bin as b' in the hashtable, if the bin is not empty. Finally, RTDP-Bel selects the action $a' = \arg\max_{a \in \mathcal{A}} Q(b,a)$ to execute and updates $V(b) = Q(b,a')$ and the hashtable of estimated value of sampled beliefs. The work in (6) demonstrated that RTDP-Bel is comparable to PBVI and HSVI2 for larger benchmark problems, such as Tag and Rock-Sample(7,8) (15). A recent work (30) have extended this method to use particle representation and multiple heuristics to guide sampling, and further demonstrate the capability of this line of work.

Another major line of work in online methods adopt the forward search idea of RTDP-Bel, but uses the Monte Carlo Tree Search (MCTS) (31) mechanism, which reduces reliance on heuristics. Furthermore, most online methods introduced below and subsequently rely on particle representation of beliefs and particle filter to update the beliefs, thereby making these solvers scalable for POMDPs with very large and even continuous state spaces.

The Partially Observable Monte Carlo (POMCP) (32) extends the Monte Carlo Tree Search (MCTS),

and specifically the Upper Confidence bounds for Trees (UCT) (33) to partially observable domain. UCT applies a multi-arm bandit method, called Upper Confidence Bound (UCB) (34), for action selection in MCTS. POMCP does not require an explicit transition, observation, and reward functions, rather it uses a generative model $\mathcal{G}(s, a)$, which maps a pair of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ to a tuple (s', o, r), where $s' \sim T(s, a, S')$, $o \sim Z(s', a, O)$, and r is the reward for performing a from s.

POMCP maintains a tree $\mathcal{T}$, where the root node corresponds to the current belief b_c. Each node of $\mathcal{T}$ represents a belief $b(h)$ associated with its history (denoted as h), which is the sequence of action–observation pairs $h = (a_0, o_0, a_1, o_1, \cdots, a_k, o_k)$ where $b = \tau(\cdots(\tau(\tau(b_c, a_0, o_0), a_1, o_1)\cdots), a_k, o_k)$. The history associated with the root node is an empty sequence. Furthermore, each node maintains statistical information to help guide future sampling. We will refer to nodes of $\mathcal{T}$ and the beliefs associated with them interchangeably. The belief $b(h)$ is represented as a set of particles. Note, however that the belief update is only performed during execution, and not during planning, as detailed below.

To find the best action to perform from the current belief b_c, POMCP constructs the tree $\mathcal{T}$ with b_c as the root node and performs many forward simulations from the root node. To perform a forward simulation form b_c, POMCP samples a state s_c from b_c and uses the sampled state to guide sampling. To sample subsequent beliefs, given a node $b(h)$, that is associated with history h, and a state $s \in support(b(h))$, POMCP performs a forward simulation from $b(h)$ by selecting an action a based on UCB1 (34), i.e., $a = \arg\max_{a \in \mathcal{A}} V(ha) + c\sqrt{\frac{N(h)}{N(h,a)}}$ where $N(h)$ is the number of times the node has been visited, $N(h, a)$ is the number of times the action a has been applied to node b, and c is a constant to balance exploitation and exploration. The value $V(h, a)$ is an estimate of $Q(b(h), a)$, which is computed as an average of the total discounted reward of multiple forward simulations. Now, let $(s', o, r) = \mathcal{G}(s, a)$, then $h' = append(h, (a, o))$ and s' is added to the particles set that represents the belief $b'(h')$ associated with h'. If h' has been visited before, the above forward simulation process is repeated from the node $b'(h')$ and sampled state s'. Otherwise, a new child of b is formed to correspond to the belief $b'(h')$ and history h'. The value $V(h')$ of this new leaf node is estimated by computing the total discounted reward following a pre-defined policy, often called as the rollout policy. The rollout policy can be replaced with a heuristic to estimate $V(h')$. A good estimate or rollout policy can help compensate for a lack of scalability in the planning horizon. Forward simulation is then restarted from the root node. Once the planning time for the step is over, the best action a from b_c is executed, an observation o is perceived, and the robot's belief is updated to $b'' = \tau(b_c, a, o)$ via particle filter. The tree $\mathcal{T}$ is reset, b'' is set as the root node of $\mathcal{T}$, and the process repeats.

Another method is Determinized Sparse Partially Observable Tree (DESPOT) (35). DESPOT constructs the belief tree and uses Monte Carlo sampling too, but uses a fixed number of scenarios (say K) to sample the beliefs. DESPOT expands every action, but uses the fixed number of scenarios to sample the observations during forward simulation. This strategy is akin to hindsight optimisation and generates a sparsely sampled belief tree, with $(|\mathcal{A}|^D K)$ many nodes for a constant K for any depth D of the tree. As a result, DESPOT can generally perform better than POMCP when the action space is relatively small but the observation space is large.

Another method, the Adaptive Belief Tree (ABT) (36), uses MCTS similar to POMCP too, but modifies POMCP in two areas. First, ABT performs backup operation along a path of $\mathcal{T}$ from a leaf node to the root, after each forward simulation down the tree is concluded (i.e., a new leaf node is added). This backup operation helps improve the estimated value function used for action selection in subsequent forward simulations. Second, ABT reuses previously built trees and estimated values of nearby beliefs to help improve estimating value function of newly added tree. These two modifications help ABT to

generate good strategies much faster than POMCP when good actions from nearby beliefs are similar, which is common in robotics problems (36, 37). Moreover, the approach of reusing planning results from previous steps allows ABT to generate good solutions to POMDP problems with dynamically changing models, especially when changes happen in a gradual manner.

3.2. Long Planning Horizon

This issue is often known as the curse of history. To compute good long-term return, a POMDP solver performs lookahead for k number of steps to consider future consequences of its action selection. However, in general, the size of the set of possible future consequences increases exponentially with the number of lookahead steps k.

Most methods discussed in Section 3.1 also claim to have alleviated the problem of long planning horizon. This is true because by estimating value function only for a small subset of the reachable space $\mathcal{R}(b_0)$ and even reachable space under an optimal policy $\mathcal{R}^*(b_0)$, computational resources can be reallocated to perform longer look-ahead, which in turn alleviate the long planning horizon issues.

However, the above strategies are often not sufficient for many robotics problems, where the required look-ahead can easily be 30 steps and more. Many methods have been proposed to directly alleviate these issues. They generally construct a more abstract action, and sample the belief space using this abstract action rather than the primitive single-step action. As a result, they reduce the effective planning horizon of the problem. Different methods to construct abstract actions have been proposed. Most (38, 39, 40) use macro-actions –that is, temporally extended sequences of actions where actions or sub-policy are run until some termination conditions are met. For example, the work in (40) constructs Partially Observable Semi-Markov Decision Processes (POSMDPs) to achieve sub-goals, and use these policies as macro-actions to offline solvers. Whilst, the work in (39) proposes macro-actions to reach subgoals in online solvers. These methods require sub-goals or termination conditions to be hand-designed to generate good problem decomposition.

Obviously, automatic generation of sub-problems are preferred. The work in (41) develops such an automatic generation mechanism, but for open loop policies for sampling beliefs, rather than macro actions. It sample milestones in the state space, biasing sampling towards states with high reward and high probability of generating useful observations. Sequences of actions to move from one milestone to another, assuming deterministic actions, become the actions used to guide sampling in the belief space. The optimality of the POMDP policy found depends on the density of the state and belief space sampling. Another method (42) restricts beliefs to be Gaussian and uses LQG (43) as macro-actions for an extension of the Probabilistic Roadmap (44) to belief space. Recently, (45) successfully constructs macro-actions for general POMDP solving automatically, based on the value of information. Specifically, it constructs macro-actions from sequences of open-loop policies with low value of information. It provides bounded regret on the quality of the policy generated by these macro actions.

3.3. Large Observation Space

Robotics problems often have rich and large (or even continuous) observation spaces, such as a combination of laser readings, joint torques reading, RGB images, etc.. Naive uniform discretization of such an observation space often results in too fine or too coarse a discretization. Overly fine discretization causes an unnecessarily large observation space, which slows down computation of good policies, while overly coarse discretization results in an effective observation space that cannot differentiate observations that induce different decisions.

Work have been proposed to better discretize continuous observation space (46) based on features derived from the value function and the associated best actions. Another work (47) proposes an extension of the policy graph representation (27) and combines it with a classification mechanism based on estimates of the value functions to identify which observations can be grouped together. Both of these methods are designed for offline methods.

For online methods, POMCPOW (48) uses the Double Progressive Widening to incrementally increase the set of observations to be considered. This strategy essentially discretizes the observation space incrementally based on the sampled observations. Although built on top of POMCP, POMCPOW diverges slightly, in the sense that it requires the use of weighted particles and an explicit observation function, rather than the generative model alone.

A more recent online method to alleviate the issue of continuous observation space is Lazy Belief Extraction for Continuous Observation POMDPs (LABECOP) (49), which avoids any form of discretization of the observation space. It maintains a set of sampled episodes, which is sequences of state–action–observation–reward quadruples, but postpone belief assignment until execution. During execution, LABECOP reweights the episodes to infer a belief based on the perceived observation, the action it just performed, and its current belief estimate, using a mechanism akin to particle filter. Finally, it estimates the Q-values of the actions based on the weighted average discounted total reward of appropriate components of the episodes.

3.4. Large Action Space

The solvers discussed above have significantly increased the scalability of POMDPs. However, most of them can only perform well for problems with a small discrete action space (i.e., $|\mathcal{A}| \leq 100$). To find the best action to perform, a POMDP solver must solve an optimization problem (eq. (1)) while estimating the Q-values of the actions, which in itself is expensive to compute. Sampling has been used to improve the speed of estimating Q-values, but most of the above solvers finds the best action naively by enumerating all actions. As a result, finding good POMDP policies becomes prohibitively expensive for problems with continuous or large discrete action space.

Perseus (21) is an offline sampling-based approximate POMDP solvers that have been extended to problems with continuous action space. It replaces maximization over all actions with sampled max operator, where maximization is computed over a random subset of the action space. GCS (28) is another offline solver for continuous action space. It performs maximization over only a subset of the action space too, but it uses geometric information from the robot operating environment to generate sequences of action space where optimization will be performed. The idea of sampling actions to alleviate problems with continuous action space have been proposed for tree-based solvers too in (50), albeit applied to the fully observable POMDP, i.e., MDP problems.

For POMDPs, an early work that extended tree search based solvers to problems with continuous action space is GPS-ABT (51). Due to the cost of estimating Q-values, and not to mention their gradient, GPS-ABT proposes to use the simplest non-gradient based optimization method, Generalized Pattern Search. The work also proposes an efficient data structure to efficiently keep track and reuse partially estimated Q-values of different pairs of belief–action. Despite using a simple optimization method, GPS-ABT is shown to converge to the optimal solution in probability, whenever the Q-value function is bounded and the gradient of the Q-value function is Lipschitz with respect to the action space. However, this method does not scale well for problems with more than 4-dimensional continuous action space. The work in (48) are also designed for handling continuous action space problems.

However, they have only been demonstrated for problems with 1-dimensional continuous action space. Another approach is to use Bayesian optimization (52, 53) for action selection, with Gaussian Process being used to represent beliefs and the estimated Q-value functions. However, they have only been demonstrated in problems with low (< 4) dimensional action space.

Another line of work (54) common to robotics, is to assume linear dynamics and that beliefs are Gaussian distributed. These assumptions allow one to apply LQG for solving, which has been demonstrated to show good performance on a 6-DOFs robot arm. Of course linearization does not always help. When and where linearization helps were explored in (55).

Another line of work (56) focuses on the problem of large discrete action space, rather than continuous action space. Problems with large discrete action space are sometimes harder than those with continuous action space because the first lack natural metric that can be used as heuristics to identify how close the performance of two actions will likely be. The work in (56) uses quantile statistics to construct a two-stage sampling mechanism for action selection, and has since been demonstrated to perform well on a logistic problem with up to 1M actions (57).

3.5. Complex Dynamics

To compute good approximate solutions, the above mentioned approximate POMDP solvers rely on a large number of forward simulations. They assume that each single-step forward simulation can be computed almost instantaneously. However, this assumption is false for robots with complex dynamics —that is, robots whose dynamics are non-linear and has no closed form solution—, where a single-step forward simulation may involve solving (Partial) Differential Equation(s), which is expensive to compute. Such complex dynamics are often required when a robot needs to perform fine motion, such as, opening screws, or when a robot operates near its maximum capability, such as, car racing.

The work in (58) proposes to represent complex dynamics as switching state-space dynamics model (hybrid dynamics model), and then proposes an offline sampling-based solver for POMDPs with such a hybrid dynamics model. Another method (59), which is typical for robotics applications, is to linearize the dynamics and uses LQR (43), a known method from control. The issue is linearization does not always work. A different approach is proposed in (37), which uses MCTS-based online solvers with the Multi Level Monte Carlo (MLMC) (60) to compute the single-step forward simulations. The MLMC is used to approximate dynamic computation with varying level of fidelity, with the goal of using the expensive original dynamics only occasionally, while the majority of the approximation uses less fidelity dynamics that are less expensive to compute.

4. Applying POMDPs to Physical Robots

Given the current scalability of POMDP solving, POMDPs have been applied to solve planning and control problems in various physical robot applications, including in a robot demonstration spanning over a 7 consecutive days and 7 hours per day at SIMPAR 2018 and ICRA 2018 (61). This section discusses some of the available software and lessons learned from applying POMDPs to physical robots.

4.1. Software

There has been a number of software tools for solving POMDPs being released as open-source software. For instance:

- Symbolic Perseus (62) implements Perseus (21) but with Algebraic Decision Diagrams (ADD) for factored representation (63). It accepts a text file with SPUDD format (64) as its inputs. Symbolic Perseus is written in Java and Matlab, and requires Matlab's Java Virtual Machine.
- ZMDP (65) implements HSVI (15) and HSVI2 (16). It is written in C++ and accepts text file with the Cassandra file format (66), which represents flat POMDP models, as inputs.
- Approximate POMDP Planning (APPL) Toolkit (67) implements SARSOP (17). It is written in C++. As its inputs, it accepts a text file in either the Cassandra (66) or PomdpX file formats (68). The latter represents factored representation and explicit separation of fully observed and partially observed state variables (69).
- APPL-online (70) implements DESPOT (35) and is written in C++.
- Toolkit for approximating and Adapting POMDP solutions In Realtime (TAPIR) (71) implements ABT (36) and is written in C++.
- On-line POMDP Planning Toolkit (OPPT) (72) is a software toolkit that provides a framework to ease interfacing with ROS. The POMDP model can be provided in two modes. First is via a text file where users can specify parameters for uncertainty. This mode of input is specifically designed for robot motion planning problems. For a more general problem, users can encode POMDP problems as plugins, with one plugin for each component (transition, observation, and reward functions). The default solver for this toolkit is ABT (36), though OPPT provides interface to incorporate other solvers too. OPPT is written in C++.
- pomdp_py (73) is a general purpose POMDP solving library, written in Python and Cython. It provides programming interface to implement POMDP models and solvers.

4.2. Implementation Tips

Parallelizing belief update, planning, and execution. Naive implementation of POMDP solvers, and specifically online solvers described in Section 3, are sequential —that is, belief update, then computing the best action to perform from the new belief, and finally executing the action. However, such an implementation often cause delays during execution, due to the often expensive computation to update beliefs and compute the best action from the new belief. These delays can be reduced by parallelizing the belief update and best action computation processes, and starting the computation as soon as an action started being executed (61).

For instance, suppose the robot is at belief b and has just started execution of the action $a^* \in \mathcal{A}$. Then, if the belief update performs the Sequential-Importance-Resampling (SIR) particle filter, the SIR process can start as soon as the robot decides to execute a^*. SIR particle filter consists of two steps. First is sampling from a proposal distribution, which in our case, $s' \sim T(s, a^*, S')$ where $s \in \mathcal{S}$ are sampled from b. Second is updating the importance weights of the samples s' based on the observation $o \in O$ perceived. The first step of drawing samples can start once the robot decides to execute a^*. By doing so, once the action is completely executed and an observation is perceived, SIR only needs to update the importance weights, which can be done fast.

In computing the best action, if ABT is used, one can sample additional episodes, starting from states sampled from the current belief b and performing a^*, as soon as the robot decides to execute a^*, so as to improve the policy within the entire descendent of b via a^* in the belief tree $\mathcal{T}$. This strategy increases the chances that after a^* is completely executed and the belief is updated based on the observation perceived, a good policy for the next belief is readily available in $\mathcal{T}$. Of course, there are cases where

even with the above strategy, the robot perceives observations that were not explored in $\mathcal{T}$. In such cases, one can reuse value estimates from nearby beliefs or restart planning from scratch.

Distance function. Some of the solvers use distance function between beliefs as part of their computation, often as a heuristic to assess how close two beliefs are. For this purpose, L1 metric and KL-divergence have often been used. However, for robotics problems, it is often desirable to account for the state space distance when computing distance between beliefs. For this purpose, Earth Mover Distance (EMD) is often more suitable than L1 or KL-divergence (74). Fast EMD computation has been developed in the computer vision community, including incorporated in OpenCV.

4.3. Some Notes on the POMDP Models

Below are three main concerns one often have about using POMDPs, together with a discussion that we hope would reduce such concerns.

How difficult is it to generate a suitable POMDP model? A POMDP model consist of six components. The state, action, and observation spaces are generally easy to define. However, the transition, observation, and reward functions are indeed harder to define. To model the transition and observation functions, one can use information about potential errors and develop a relatively conservative estimate, learn from data, or a combination of both. Setting the reward function can be quite involved if one wants to use the reward function as a heuristics to help guide the search. However, if we set the reward function to reflect desirability of being in a state, rather than as heuristics, then setting the reward functions are easier, though in this case, we do need solvers with good scalability.
Moreover, in most robotics problems, one can generate good POMDP policies without accurate POMDP models. For instance, the transition and observation functions used in a POMDP demonstration at ICRA'18 (61) are a very rough estimate, learned using a simple likelihood approach from a small amount of data. However, the POMDP strategies generated for this rough model resulted in a 100% success rate, while those generated without consideration of uncertainty resulted in only 35% success rate (61). Furthermore, results on end-to-end POMDP model learning and solving (75) indicate the models learned can often be different from the correct model but the policy generated are performing well.

What do we really gain by formulating and solving a problem as a POMDP? The short answer is robustness. A more nuanced answer is that by constructing a feedback policy that quantifies uncertainty, POMDPs can automatically balance trade-offs between information gathering actions and performing actions to achieve its task. In fact, it can even identify actions that could achieve both, as highlighted in the simulation result of (41). Such a capability is important when the solution space is small, such as when robots must operate in cluttered or confined environment.

Isn't first order Markov too restrictive? One concern with POMDPs is the first order Markov requirement. It might be useful to clarify that the POMDP policy actually accounts for the entire history because a POMDP policy maps beliefs to actions, and beliefs are sufficient statistics of the entire history (14). The first order Markov is indeed required for the transition dynamics and observation functions. However, the first order Markov requirements for those functions are also common in state space control (76), which is often used in robotics.

5. Discussion

The above two sections present an overview of some of the general sampling-based approximate POMDP solvers. This is by no means an exhaustive list of POMDP solving approaches. For instance, we did not cover policy search based approaches that have been introduced since (77). We also did not provide coverage on work that restricts beliefs to be Gaussian and those that extend deterministic sampling-based motion planning, such as the Probabilistic Roadmap (44), to belief space planning beyond the few mentioned above. Furthermore, the surveys (78) and (79) provide a more exhaustive list on offline sampling-based approximate POMDP solvers up to 2013 and approximate online POMDP solvers up to 2008, respectively. However, we focus to elaborate computational issues that have hindered the practicality of POMDPs in robotics and elucidate ideas that have alleviated them.

5.1. Comparison to Sampling-Based Motion Planning

Taking a step back, it is interesting to note that the key techniques and progressions that enable POMDPs to become practical in robotics is close to those of motion planning. Table 1 tries to capture this similarity.

Table 1 Sampling-based POMDP Solvers and Motion Planning

Progress in Capability	Towards Sampling-based methods	
	POMDPs	Motion Planning
Helpful concepts and theoretical results	Optimal value function is (or can be approximated arbitrarily close to) a piecewise linear convex function (2) for α-vectors representation, and heuristic based forward search (29) for online tree-based solvers.	The configuration space (80).
Fast primitive computation	Point-based dynamic programming (20) for offline solvers and Upper Confidence Tree (33) for online solvers.	Fast collision-check (81, 82).
Combined sampling-based with a more classical approach	Combined point-based and standard dynamic programming backup (20).	Potential field with randomization to exit local minima (83) and the Ariadne's Clew algorithm (84).
Full sampling-based to solve the problem start to become scalable	Off-line, where a good policy π is computed prior to execution, and during execution, the action $\pi(b)$ will be executed whenever the agent is at belief b, started with (12).	Multi-query, where the goal is to construct a compact representation of the free space component of the robot's configuration space, started with (44).
Sampling-based to solve a smaller problem (potentially, iteratively) to improve scalability	On-line, where the best action to perform is computed only for the belief at the current time-step, popularized by (32).	Single-query, where the goal is to answer a query to move the robot from a given initial to goal configurations, started with (85).

5.2. Relation to Learning

The POMDP is closely related to learning. It is a basic representation for model-based Bayesian Reinforcement Learning (86). Reinforcement Learning (RL) can be defined as a Markov Decision Process (MDP, the fully observed version of POMDP) with missing components. Since MDP models a fully observed system, MDP is defined as $\langle \mathcal{S}_{MDP}, \mathcal{A}_{MDP}, T_{MDP}, R_{MDP} \rangle$, where $\mathcal{S}_{MDP}$ is the state space, $\mathcal{A}_{MDP}$ is the action space, $T_{MDP}(s, a, s')$ is the transition function, representing the conditional proba-

bility function of moving to state $s' \in \mathcal{S}_{MDP}$ after performing action $a \in \mathcal{A}_{MDP}$ from state $s \in \mathcal{S}_{MDP}$, and R_{MDP} is the reward function. RL is then defined as MDP with initially unknown transition and/or reward functions.

POMDP representation of the RL problem is to model uncertainty over the T_{MDP} and R_{MDP} as probability distribution functions, generally as parametric distribution functions. The parameters of these functions are then set as partially observed state variables of the POMDP agent. As the POMDP agent perceives observations, its understanding about the true parameters improve. By solving this POMDP problem, the agent automatically balances the trade-off between information gathering actions to reduce uncertainty on the parameters and actions to achieve the task, and identifies actions that help improve both model understanding and task attainment. The difficulty of this approach of RL is that naive modelling often results in POMDP models that are much larger than the size of problems that state of the art POMDP solvers can handle.

On another note, as mentioned in Section 4.3, the transition and observation functions of POMDPs have often been learned from data. More recently, deep learning has been applied to solve POMDPs when its model is not fully known. Some of the early work (87, 88, 89) are model free, they directly learn the policy or value function without learning the POMDP model. However, better generalization has been achieved with methods (90, 75, 91, 92, 93) that embed the POMDP structure inside a neural network and training the network to learn a policy or value function, thereby combining model-based and model-free methods.

6. Conclusion

The Partially Observable Markov Decision Process (POMDP) is a mathematical framework for planning under uncertainty, and specifically for non-deterministic and partially observable scenarios. Finding the optimal solution to a POMDP problem is computationally intractable. However, sampling-based methods are now available to compute good optimal solutions —ones that significantly improve the robustness of robotics systems— within reasonable computational resources. Improving the scalability of POMDPs from a mere theoretical concept that can only work for small toy problems into a software tool that can be applied to a variety of realistic robotics problems requires multiple issues to be overcome. For robotics problems, five major issues are large state, observation, and actions spaces, long planning horizon, and complex dynamics. Various sampling-based techniques have been proposed to alleviate these issues. Software implementations of some of these methods and interfaces to typical robotics software are now available as Open Source Software to ease applying POMDPs to robotics problems. This paper presents an overview of the issues, methods, and practical tips on applying POMDPs to robotics.

Although some scalability issues in POMDPs remain, existing methods are efficient enough to improve the robustness of many robotics problems. We hope this paper could provide insights on the current state of POMDPs and help bring more awareness on the practicality of POMDPs in robotics.

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

This work is supported by the ANU Futures Scheme.

LITERATURE CITED

1. Drake AW. 1962. Observation of a markov process through a noisy channel. Ph.D. thesis, Massachusetts Institute of Technology
2. Sondik EJ. 1971. The optimal control of partially observable markov processes. Ph.D. thesis, Stanford University
3. Papadimitriou CH, Tsitsiklis JN. 1987. The complexity of markov decision processes. *Mathematics of operations research* 12:441–450
4. Kaelbling LP, Littman ML, Cassandra AR. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101:99–134
5. Monahan GE. 1982. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science* 28:1–16
6. Bonet B, Geffner H. 2009. *Solving POMDPs: RTDP-Bel vs. Point-based Algorithms.* In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1641–1646. Pasadena CA
7. Sondik EJ. 1978. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations research* 26:282–304
8. Mundhenk M, Goldsmith J, Lusena C, Allender E. 2000. Complexity of finite-horizon markov decision process problems. *Journal of the ACM (JACM)* 47:681–720
9. Vlassis N, Littman ML, Barber D. 2012. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Transactions on Computation Theory (TOCT)* 4:1–8
10. Natarajan BK. 1986. On moving and orienting objects. Tech. rep., Cornell University
11. Canny J, Reif J. 1987. *New lower bound techniques for robot motion planning problems*. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pp. 49–60. IEEE
12. Pineau J, Gordon G, Thrun S, et al. 2003. *Point-based value iteration: An anytime algorithm for POMDPs*. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, vol. 3, pp. 1025–1032. Citeseer
13. Hsu D, Lee W, Rong N. 2007. *What makes some POMDP problems easy to approximate?* In *Proc. Neural Information Processing Systems (NeurIPS)*
14. Hauskrecht M. 2000. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research* 13:33–94
15. Smith T, Simmons R. 2004. *Heuristic Search Value Iteration for POMDPs*. In *Uncertainty in Artificial Intelligence (UAI)*
16. Smith T, Simmons R. 2005. *Point-based POMDP Algorithms: Improved Analysis and Implementation*. In *Uncertainty in Artificial Intelligence (UAI)*
17. Kurniawati H, Hsu D, Lee W. 2008. *SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces*. In *Robotics: Science and Systems (RSS)*
18. Smallwood RD, Sondik EJ. 1973. The optimal control of partially observable markov processes over a finite horizon. *Operations research* 21:1071–1088
19. Cheng H. 1988. Algorithms for partially observable markov decision processes. Ph.D. thesis, The University of Britich Columbia
20. Zhang NL, Zhang W. 2001. Speeding up the convergence of value iteration in partially observable markov decision processes. *Journal of Artificial Intelligence Research* 14:29–51
21. Spaan MT, Vlassis N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of artificial intelligence research* 24:195–220
22. Ong SC, Png SW, Hsu D, Lee WS. 2010. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research* 29:1053–1068
23. Ji S, Parr R, Li H, Liao X, Carin L. 2007. *Point-Based Policy Iteration*. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 1243–1249

24. Hansen EA. 1998. *Solving POMDPs by Searching in Policy Space*. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 211–219
25. Thrun S. 1999. *Monte Carlo POMDPs.* In *NIPS*, vol. 12, pp. 1064–1070
26. Porta JM, Vlassis N, Spaan MT, Poupart P. 2006. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research* 7:2329–2367
27. Bai H, Hsu D, Lee WS, Ngo VA. 2010. Monte carlo value iteration for continuous-state POMDPs. In *Workshop on the Algorithmic Foundation of Robotics (WAFR)*. Springer
28. Kurniawati H, Bandyopadhyay T, Patrikalakis N. 2012. Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots: Special issue on RSS 2011* 30
29. Bonet B. 1998. *Solving large POMDPs using real time dynamic programming*. In *In Proc. AAAI Fall Symp. on POMDPs*
30. Kim SK, Salzman O, Likhachev M. 2019. *POMHDP: Search-based belief space planning using multiple heuristics*. In *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 29, pp. 734–744
31. Coulom R. 2006. *Efficient selectivity and backup operators in Monte-Carlo tree search*. In *International conference on computers and games*, pp. 72–83. Springer
32. Silver D, Veness J. 2010. *Monte-Carlo planning in large POMDPs*. In *Proc. Neural Information Processing Systems (NeurIPS)*
33. Kocsis L, Szepesvári C. 2006. *Bandit based monte-carlo planning*. In *European conference on machine learning*, pp. 282–293. Springer
34. Auer P, Cesa-Bianchi N, Fischer P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47:235–256
35. Somani A, Ye N, Hsu D, Lee WS. 2013. *DESPOT: Online POMDP Planning with Regularization*. In *Proc. Neural Information Processing Systems (NeurIPS)*, vol. 13, pp. 1772–1780
36. Kurniawati H, Yadav V. 2013. *An Online POMDP Solver for Uncertainty Planning in Dynamic Environment*. In *International Symposium on Robotics Research*
37. Hoerger M, Kurniawati H, Elfes A. 2019. *Multilevel Monte-Carlo for Solving POMDPs Online*. In *International Symposium on Robotics Research*
38. Theocharous G, Kaelbling L. 2003. Approximate planning in pomdps with macro-actions. *Advances in Neural Information Processing Systems* 16:775–782
39. He R, Brunskill E, Roy N. 2010. *PUMA: Planning under uncertainty with macro-actions*. In *Association for the Advancement of Artificial Intelligence (AAAI)*
40. Lim ZW, Hsu D, Lee WS. 2011. *Monte Carlo Value Iteration with Macro-Actions.* In *NIPS*, pp. 1287–1295
41. Kurniawati H, Du Y, Hsu D, Lee W. 2011. Motion planning under uncertainty for robotic tasks with long time horizons. *International Journal of Robotics Research* 30:308–323
42. Agha-Mohammadi AA, Chakravorty S, Amato NM. 2014. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *International Journal of Robotics Research* 33:268–304
43. Bertsekas DP. 2011. *Dynamic programming and optimal control 3rd edition, volume I*
44. Kavraki LE, Svestka P, Latombe JC, Overmars MH. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12:566–580
45. Flaspohler G, Roy NA, Fisher III JW. 2020. Belief-dependent macro-action discovery in POMDPs using the value of information. *Proc. Neural Information Processing Systems (NeurIPS)* 33
46. Hoey J, Poupart P. 2005. *Solving POMDPs with continuous or large discrete observation spaces*. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1332–1338
47. Bai H, Hsu D, Lee WS. 2014. Integrated perception and planning in the continuous space: A POMDP approach. *International Journal of Robotics Research* 33:1288–1302
48. Sunberg ZN, Kochenderfer MJ. 2018. *Online algorithms for POMDPs with continuous state, action, and observation spaces*. In *International Conference on Automated Planning and Scheduling (ICAPS)*
49. Hoerger M, Kurniawati H. 2021. *An On-Line POMDP Solver for Continuous Observation Spaces*. In *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*
50. Mansley C, Weinstein A, Littman M. 2011. *Sample-based planning for continuous action markov decision*

processes. In *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 21
51. Seiler K, Kurniawati H, Singh S. 2015. *An Online and Approximate Solver for POMDPs with Continuous Action Space*. In *IEEE International Conference on Robotics & Automation (ICRA)*
52. Morere P, Marchant R, Ramos F. 2016. *Bayesian optimisation for solving continuous state-action-observation POMDPs*. In *Proc. Neural Information Processing Systems (NeurIPS)*
53. Mern J, Yildiz A, Sunberg Z, Mukerji T, Kochenderfer MJ. 2021. *Bayesian Optimized Monte Carlo Planning*. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 11880–11887
54. Van Den Berg J, Abbeel P, Goldberg K. 2011. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research* 30:895–913
55. Hoerger M, Kurniawati H, Elfes A. 2020. Non-Linearity Measure for POMDP-based Motion Planning. *arXiv preprint arXiv:2005.14406* An earlier version has been published in WAFR 2016
56. Wang E, Kurniawati H, Kroese D. 2018. *An On-line Planner for POMDPs with Large Discrete Action Space: A Quantile-Based Approach*. In *International Conference on Automated Planning and Scheduling (ICAPS)*
57. Wang E, Kurniawati H, Kroese D. 2019. *Inventory Control with Partially Observable States*. In *Proc. Int. International Congress on Modelling and Simulation (MODSIM)*
58. Brunskill E, Kaelbling LP, Lozano-Perez T, Roy N. 2008. *Continuous-State POMDPs with Hybrid Dynamics*. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*
59. Van Den Berg J, Patil S, Alterovitz R. 2012. Motion planning under uncertainty using iterative local optimization in belief space. *International Journal of Robotics Research* 31:1263–1278
60. Giles MB. 2015. Multilevel monte carlo methods. *Acta Numerica* 24:259–328
61. Hoerger M, Song J, Kurniawati H, Elfes A. 2019. *POMDP-based Candy Server: Lessons Learned from a Seven Day Demo*. In *International Conference on Automated Planning and Scheduling (ICAPS)*
62. Poupart P. —. Symbolic-perseus. `https://cs.uwaterloo.ca/~ppoupart/software.html#symbolic-perseus`
63. Poupart P. 2005. Exploiting structure to efficiently solve large scale partially observable Markov decision processes. Ph.D. thesis, University of Toronto. Chapter 5
64. Hoey J, St-Aubin R, Hu A, Boutilier C. 1999. *SPUDD: Stochastic planning using decision diagrams*. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 279–288. Morgan Kaufmann Publishers Inc.
65. Smith T. —. ZMDP. `http://longhorizon.org/trey/zmdp/`
66. Cassandra AR. —. Pomdp format. `http://pomdp.org/code/pomdp-file-spec.html`
67. AdaComp-NUS. —. APPL. `https://github.com/AdaCompNUS/sarsop`
68. AdaComp-NUS. —. APPL. `https://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`
69. Ong S, Png S, Hsu D, Lee W. 2010. Planning under uncertainty for robotic tasks with mixed observability. *International Journal of Robotics Research* 29:1053–1068
70. AdaComp-NUS. —. APPL-Online. `https://github.com/AdaCompNUS/despot`
71. RDLLab. —. Toolkit for approximating and Adapting POMDP solutions In Realtime (TAPIR). `https://github.com/RDLLab/tapir`
72. Hoerger M, Kurniawati H, Elfes A. —. On-line POMDP Planning Toolkit (OPPT). `https://github.com/RDLLab/oppt`
73. H2RLab. —. pomdp_py. `https://h2r.github.io/pomdp-py/html/`
74. Littlefield Z, Klimenko D, Kurniawati H, Bekris KE. 2015. *The Importance of a Suitable Distance Function in Belief-Space Planning*. In *International Symposium on Robotics Research*
75. Karkus P, Hsu D, Lee WS. 2017. *QMDP-Net: Deep Learning for Planning under Partial Observability*. In *Proc. Neural Information Processing Systems (NeurIPS)*
76. Friedland B. 2012. *Control system design: an introduction to state-space methods*. Courier Corporation
77. Ng AY, Jordan M. 2000. *PEGASUS: A Policy Search Method for Large MDPs and POMDPs*. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 406–415
78. Shani G, Pineau J, Kaplow R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27:1–51
79. Ross S, Pineau J, Paquet S, Chaib-Draa B. 2008. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research* 32:663–704

80. Lozano-Pérez T, Wesley MA. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22:560–570
81. Larsen E, Gottschalk S, Lin MC, Manocha D. 1999. Fast proximity queries with swept sphere volumes. Tech. rep., Technical Report TR99-018, Department of Computer Science, University of . . .
82. Gottschalk S, Lin MC, Manocha D. 1996. *OBBTree: A hierarchical structure for rapid interference detection*. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180
83. Barraquand J, Latombe JC. 1990. *A Monte-Carlo algorithm for path planning with many degrees of freedom*. In *IEEE International Conference on Robotics & Automation (ICRA)*, pp. 1712–1717
84. Bessiere P, Ahuactzin JM, Talbi EG, Mazer E. 1993. *The "Ariadne's clew" algorithm: Global planning with local methods*. In *IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS)*, vol. 2, pp. 1373–1380
85. Hsu D, Latombe JC, Motwani R. 1997. *Path planning in expansive configuration spaces*. In *IEEE International Conference on Robotics & Automation (ICRA)*, vol. 3, pp. 2719–2726
86. Ghavamzadeh M, Mannor S, Pineau J, Tamar A. 2015. Bayesian Reinforcement Learning: A Survey. *Found. Trends Mach. Learn.* 8:359–483
87. Hausknecht M, Stone P. 2015. *Deep Recurrent Q-Learning for Partially Observable MDPs*. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*
88. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518:529 EP
89. Mirowski P, Pascanu R, Viola F, Soyer H, Ballard AJ, et al. 2016. *Learning to Navigate in Complex Environments*. In *International Conference on Learning Representations (ICLR)*
90. Shankar T, Dwivedy SK, Guha P. 2016. *Reinforcement Learning via Recurrent Convolutional Neural Networks*. In *International Conference on Pattern Recognition (ICPR)*, pp. 2592–2597
91. Oh J, Singh S, Lee H. 2017. *Value Prediction Network*. In *Proc. Neural Information Processing Systems (NeurIPS)*
92. Igl M, Zintgraf L, Le TA, Wood F, Whiteson S. 2018. *Deep Variational Reinforcement Learning for POMDPs*. In *International Conference on Machine Learning (ICML)*, pp. 2117–2126
93. Collins N, Kurniawati H. 2020. *Locally-Connected Interrelated Network: A Forward Propagation Primitive*. In *Workshop on the Algorithmic Foundation of Robotics (WAFR)*