High level robot motion planning using POMDP

by

Shyrailym Shaldambayeva

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

NAZARBAYEV UNIVERSITY

Apr 2021

Author .....
Department of Computer Science
Apr 28, 2021

Certified by .....
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by .....
Vassilios D. Tourassis
Dean, School of Engineering and Digital Sciences

High level robot motion planning using POMDP

by

Shyrailym Shaldambayeva

Submitted to the Department of Computer Science
on Apr 28, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Motion planning in uncertain environments is an essential feature of autonomous robots. Partially observable Markov decision processes (POMDP) provides a principled approach for such planning tasks and have been successfully employed for various robotic applications. Offline planning algorithms for POMDPs have proved to achieve optimal policies. However, these algorithms are computationally very expensive and are not often applicable to accomplish realistic robot motion planning scenarios. As an alternative to offline planning Monte Carlo algorithms for online planning were developed, e.g. DESPOT and POMCP are implemented for public use within a set of open-source POMDP libraries. The aim of this master thesis is to explore applicability of available open-source POMDP libraries to real-time robot motion planning tasks. We adopt these libraries for the POMDP models proposed in the literature and replicate a number of realistic benchmark experiments.

Thesis Supervisor: Almas Shintemirov
Title: Associate Professor

Acknowledgments

I would like to express my gratitude to my advisor, Almas Shintemirov, for his excellent guidance and active involvement in creating this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous robots have greatly advanced in a wide range of tasks that involve sensing, manipulation and navigation. Usually, they operate in stable and structured environments and perform precisely defined tasks. Such tasks involve only known objects and mostly have a repetitive character, so they can be solved by a sequence of small, predetermined subroutines. However, we would like them to be able to accomplish more complex tasks, not only in environments with perfect settings. To carry out the tasks, robots may have to deal with not only dynamic and uncertain environments, but also manipulate objects with unknown properties. Object's properties that can be unknown are its appearance, weight, location, number, partial occlusion, etc.

Despite the imperfect knowledge about their environment, autonomous robots are still expected to operate reliably, therefore motion planning under uncertainty is considered an essential capability though challenging. Without direct awareness of its own current state, a robot has to decide on the best action to take based on the set of all probable states consistent with observed information. Partially observable Markov decision processes (POMDP) is found suitable for such planning tasks as it provides a principled and general decision-making framework to capture uncertainties and estimate system states.

In a POMDP to meet the partial observability, a given problem is modelled using probabilities for all the uncertainties. The model obtains observations from its

environment that can be uncertain or incomplete. The interaction between the environment and the agent also includes a reward and penalty system. Observations, rewards/penalties and history of previously received information all take part in decision making and selecting actions to successfully complete the given task [23].

Offline planning algorithms such as value iteration and policy iteration are traditionally used to find the optimal policy in POMDPs. In worlds where the state space, the action space, the observation space and the planning horizon are all finite, offline planning algorithms for POMDPs have proved to achieve optimal policies. However, these algorithms are computationally very expensive and are not often applicable to accomplish realistic robot motion planning scenarios. Online planning algorithms developed as an alternative to offline planning are shown to handle different POMDP domains efficiently [29].

Two state-of-the-art online planning solvers, POMCP (Partially Observable Monte Carlo Planning) [26] and DESPOT (Determinized Sparse Partially Observable Trees) [28], use the Monte Carlo method which is based on repeated random sampling. POMCP performs Monte Carlo sampling both for sampling states from the current belief and sampling histories with a black-box simulator. DESPOT is a sparsely sampled belief tree determined by a number of scenarios sampled randomly, which is used for searching of a good enough policy. Both POMCP and DESPOT achieved a high level of performance in solving large planning tasks.

1.1 Project goals

Despite the success of different algorithms in solving POMDPs, it is still an open question which robotic motion planning tasks with some level of uncertainty benefit from the explicit modeling as a POMDP. One of the aims of this master thesis is to explore real world problems modeled and solved as POMDPs. We discover that POMDP has found use and improved performance in various robotic tasks such as navigation, localization, grasping and target tracking [12].

Defining a POMDP model for a particular task is a demanding process, as it

requires a thorough prior analysis to produce an appropriate model that integrates all the uncertainties. We learn how to build an appropriate POMDP model on the basis of two benchmark problems, namely Laser Tag and Rock Sample. These benchmark problems represent different types of robotic tasks, have their own distinct uncertainty and contrasting domains.

Additionally, this master thesis focuses on exploring applicability of available open-source POMDP libraries to real-time robot motion planning tasks. For this purpose, we test the performance of two state-of-the-art online solvers, POMCP and DESPOT, implemented by the APPL toolkit, on Laser Tag and Rock Sample. We connect the solvers to an external simulator to approximate their performance in the real world.

The above stated thesis goals aim, in general, to initiate, for the first time, theoretic and experimental research on POMDP framework applications to developing intelligent robotics systems at Nazarbayev University and Kazakhstan. The theoretical foundations developed as a result of this thesis work are planned to be continued as part of a funded research project on developing an autonomous multi-purpose mobile robot-manipulation system of the thesis supervisor and would ultimately result in developing a fully functioning intelligent mobile robot prototype.

1.2 Thesis outline

Chapter 2 starts with the literature review on robotic tasks with some degree of uncertainty modeled as POMDPs. It follows with a formal definition of a POMDP and its challenges. The chapter is concluded with a general approach to solving POMDPs and description of two existing solvers.

In Chapter 3 first two benchmark problems and their POMDP models are introduced. Next the APPL toolkit for POMDP planning is described. The experiments setup and the results conclude the chapter.

Chapter 4 summarizes the thesis work and provides an outlook on future work.

Chapter 2

Background

2.1 Modeling robotic tasks under uncertainty as POMPDs

Robotic tasks have been formulated as POMDPs in many different studies. One of the areas POMDP has found application in is robotic grasping. Hsiao et al. [9] proposed a method for partitioning the configuration space into a set of regions that further constitute the states in a partially observable Markov decision process (POMDP). They applied the method on the problem of performing pick-and-place operations by robotic arm and hand. In their setup, position of the robot, task dynamics and sensors were well known, and the uncertainty came from the robot's configuration and the state of the objects such as their shapes and relative pose.

The authors built their work on the ideas of one of the early approaches to planning in the presence of uncertainty [16]. This early approach was computationally complex and susceptible to failures since it tried to produce a plan that would work for all possible configurations. Hsiao et al. modeled the uncertainties using a probability distribution opposed to a set, which allowed to make trade-offs in order to deliver a feasible plan rather than an optimal one. This computational simplification was achieved by reducing the state and action space using abstraction. Such representation of the task allowed to focus on those parts of the space that had higher

probability to be encountered. In this study a planning algorithm named heuristic search value iteration (HSVI) presented in [27] was used. HSVI combines two well-known techniques: 1. attention-focusing search heuristics to sample belief states with a relatively high probability of being encountered; 2. piecewise linear convex representations of the value function to choose from those parts of the belief space.

More research works on planning for robotic grasping with uncertainty are presented in [5], [8], [13] and many others.

POMDPs have also been applied to robotic navigation problems. Ross et al. [24] adopted a Bayesian approach to a simple robot navigation task modeled as a POMDP. The robot moved in an open 2D area and tried to arrive at a specific goal location. In Bayesian reinforcement learning framework, given a prior distribution over model parameters, the posterior distribution is updated as the robot interacts with the environment. Optimal action is selected according to the posterior distribution over model parameters. Normal-Wishart distribution was chosen to represent the posterior distribution as it is more suitable for cases with continuous spaces. In their work authors used an extended POMDP model, BACPOMDP, which stands for the Bayes-Adaptive Continuous POMDP. In BACPOMDP the belief update operation includes the Normal-Wishart parameters as well as the state of the system, making the value of learning information about the environment an integral part of the planning. Also, during the belief update a particle filter based on Monte Carlo sampling methods is used to approximate the belief states.

Additional examples on navigation tasks modeled as POMDPs can be found in [4] and [13].

Object manipulation is another area where POMDPs are widely used. Pajarinen et al. [19] worked on the problem of manipulating unknown objects in a cluttered environment. In their study they considered different possible object compositions in decision making, rather than trying to find a single best object composition. In their experiments RGB-D data of the current scene is available to the robot, which is segmented and for each segment the probability of the segment belonging to an object is estimated. Based on these probabilities and past grasp attempts a probability

distribution over possible object compositions is created. For belief estimation they used an online POMDP method that computes probabilities of successfully grasping an object and observing its attributes based on the level of object's occlusion. Failed past grasps on the other hand are assumed to succeed only if the occlusion on the object changes. The last step is choosing the best long-term action for the current object distribution (with the highest grasp probability for an action) using POMDP and performing that action.

Li et al. [14] also studied the problem of object's search in clutter, specifically the task of finding a drink can in the jam-packed fridge. They prove that a multi-step POMDP planning has a clear advantage over simple greedy approaches. These findings agree with results of Pajarinen [19]. By multi-step planning they mean to find a sequence of actions to move objects in the cluttered environment to uncover the occluded target object. However, contrary to Pajarinen [19] who manipulated unknown objects, Li uses six known in advance different types of objects for their model with the maximum of eight objects on the shelf.

Li also states that moving objects inside the confined space rather than taking them out is preferable, but this poses a new problem of determining where exactly objects to be moved. The state space and the action space for this problem of placement are condemned to be large, since the model has to explicitly represent object poses in the coordinate system, which leads to over 1000 possible actions and around 10^{32} states. They used an approximate online POMDP algorithm, DESPOT (Determinized Sparse Partially Observable Tree) [28], which searches a sampled belief tree. DESPOT is able to handle large state spaces, but to make the search in DESPOT more efficient, authors reduced the number of actions by exploiting the spatial constraints.

POMDPs have been successfully employed for other robotic manipulation tasks such as sorting objects [17], [18], object detection [1], [3] and so on.

2.2 Partial Observable Markov Decision Process (POMDP)

A partially observable Markov decision process can be described as a tuple [11]

$$\langle S, A, T, R, \Omega, O \rangle$$

where

- S, A, T, and R describe a Markov decision process:
  - S is a set of states of the world. The number of states can be finite, countably infinite, or continuous.
  - A is a set of actions. The number of actions may be finite, countably infinite, or continuous.
  - $T : S \times A \to \Pi(S)$ is the state-transition function, giving for each world state and agent action, a probability distribution over world states. Due to uncertainty in action effect, the world has a certain probability of transitioning into any state in S. $T(s, a, s')$ denotes the probability of ending in state *s'*, given that the agent starts in state s and takes action a.
  - $R : S \times A \to R$ is the reward function, giving the expected immediate reward gained by the agent for taking each action in each state. $R(s, a)$ denotes the expected reward for taking action a in state s. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions.
- Ω is a finite set of observations the agent can experience of its world. In POMDP, the agent is not directly aware of its current state. Instead, the agent takes in an observation $o \in \Omega$ through its sensors after executing an action.
- O is the observation function, which gives a probability distribution over possible observations for each action and resulting state. $O(s', a, s)$ denotes the

probability of making observation o given that the agent took action a and landed in state s.

There are also two other important components of POMDP models, which are even included in the POMDP tuple in some literature [10][21]:

- γ is a discount factor. In some POMDP models, as in the "Light" environment presented in [15], the agent may receive an unbounded series of positive rewards by sitting in one state and executing a single action over and over again, never reaching the goal. To avoid this problem a concept of discounting is introduced. It is based on the idea that an infinite sequence of positive rewards is unrealistic, and rewards received in the near future must be valued more than those received in the distant future. The discount factor γ is commonly specified in the (0, 1) range, which assures that even in an infinite horizon the total reward is finite and the problem is well defined [10]. A reward r received at time t is contributed as $\gamma^t r$ to the total reward.

  In case when γ is equal to 1, no discounting occurs and the total reward is a simple sum of all reward values. If γ is close to 1, with each time step the contribution of future rewards to the total reward lessen exponentially. In case when γ is close to 0, distant rewards drop to near-zero values, so only immediate rewards are important. If γ is equal to 0, only reward at t_0 is counted. Another view on the discount factor is that it represents the probability that a trial will be allowed to continue after each step [15].

- b_0 is the initial belief.

  In POMDP, observations can be partial and/or noisy. They depend on the current state, as well as on the action taken, however they are not specific to a single state and the same observation may occur in different states. On top of that the observation function itself is probabilistic. Therefore, at any time an agent usually cannot determine its current state. One possible solution for the agent is to decide on its next action based on the complete history of actions

and observations up to the current time step. However, storing complete history is memory consuming. Instead, the agent stores a probability distribution over the state space, where each value represents the likelihood of the agent to be in each state. This probability distribution is called belief state [6]. Kaelbling et al. [11] prove that storing such distribution rather than complete history is a sufficient statistic and additional data about past actions or observations neither improve knowledge about the current state nor increase the expected total reward.

Thus, b_0 is the initial state probability distribution at $t = 0$. The distribution b_0 mainly depends on the domain. If the agent has no knowledge about its true state, the initial belief is usually uniform over the state space. Gaussian is another commonly used distribution for initial belief. It is found most suitable in robot localization problems, where the belief can be centered around the robot's actual location [29].

2.2.1 Tiger Problem

Let's consider the well-known Tiger problem to better understand each component of POMDP [11]:

Imagine an agent standing in front of two closed doors. Behind one of the doors is a tiger and behind the other is a large reward. If the agent opens the door with the tiger, then a large penalty is received (presumably in the form of some amount of bodily injury). Instead of opening one of the two doors, the agent can listen, in order to gain some information about the location of the tiger. Unfortunately, listening is not free; in addition, it is also not entirely accurate. There is a chance that the agent will hear a tiger behind the left-hand door when the tiger is really behind the right-hand door, and vice versa.

When we present this problem as a POMDP, it has the following components:

- **Set of states S**: There are two states of the world, s_l for when the tiger is behind the left door, s_r for when it is behind the right door.

- **Set of actions A**: There are three actions LEFT, RIGHT, and LISTEN.

- **Transaction function T**: If the agent takes the LISTEN action, the state of the world does not change. If the agent takes either LEFT or RIGHT actions, the state of the world changes to a respective s_l or s_r states with equal probability of 0.5.

- **Reward function R**: There are two possible outcomes, the agent receives +10 reward for opening the correct door or -100 for opening the door with the tiger behind it. The cost of listening is -1.

- **Set of observations** Ω: There are only two possible observations: to hear the tiger on the left (TL) or to hear the tiger on the right (TR).

- **Observation function O**: When the world is in state s_l, the LISTEN action results in observation TL with probability 0.85 and the observation TR with probability 0.15; conversely for world state s_r.

- **Initial belief** b_0: Uniform (i.e. [0.5, 0.5]), since tiger is on the left or on the right with equal probability.

2.3 POMDP Challenges

Despite being an efficient tool with strong mathematical foundation for planning under uncertainty, POMDP faces two computational difficulties known as the "curse of dimensionality" and the "curse of history" [10][12]. "Curse of dimensionality" has two sources. The first one is a possible large state space: with the number of state variables the number of states grows exponentially. The second source is due to the partially observable environment, planning uses the space of beliefs which are probability distributions over the states. Thus the size of the belief space is

approximated to grow exponentially with the number of states [28]. Hence, a task with 1,000 states will have a 1,000-dimensional belief space [12].

The second challenge that POMDP faces, the "curse of history", arises from the fact that in a planning task to reach its goal, an agent takes a long sequence of actions which results in a long planning horizon. The number of action-observation histories considered during planning grows exponentially with the planning horizon [28] [28].

We see that both curses result in exponential growth of computational complexity and many believed POMDPs to be practical only on the simplest problems. However, in recent years many POMDP solvers emerged that produce approximate, but computationally tractable solutions, rather than an optimal one. They are based on the principle that in many applications a near-optimal strategy is "good enough" to accomplish a given task [7]. These advances made it possible to solve complex robotic tasks modeled as large-scale POMDPs [28].

2.4 Solving POMDP

The solution to a POMDP is an optimal policy that maximizes the expected total reward. A policy is an action strategy that specifies the action an agent should take at every time step based on the available information [10]. For each POMDP, there is at least one optimal policy that achieves maximizing the total reward [29].

To understand how POMDPs are solved, they can be viewed as a special case of MDP over belief-state. As it is presented in Section 2.2, a belief state b is a probability distribution over state space S. Therefore, $b(s)$ denotes the probability assigned to state s by belief state b. To comply with the axioms of probability, these must always hold [11]:

1. $0 \leq b(s) \leq 1$
2. $\sum_{s \in S} b(s) = 1$

Then at time t given an old belief state b_{t-1}, an action a_t, and an observation o_t, a new belief state b_t is computed as a succeeding probability distribution using the

Bayes theorem:

$$b_t(s') = \eta O(s', a_t, o_t) \sum_{s \in S} T(s, a_t, s') b_{t-1}(s) \tag{2.1}$$

where η is a normalizing factor, that ensures that $b_t(s')$ is a well-defined probability distribution over states, i.e. the second requirement of axioms of probability is satisfied [28].

Accordingly, solving a POMDP can be thought of as constructing a policy function π that outputs an action $\pi(b) \in A$ for every possible belief b. $\pi(b)$ specifies the probability that the agent will execute any action in any given belief state [25]. The value of this policy π is the expected total discounted reward that the agent receives by executing π:

$$V_\pi(b_0) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t))] \tag{2.2}$$

And the goal is to find a policy that maximizes its value, i.e. the optimal policy:

$$\pi^* = argmax_\pi V_\pi(b_0) \tag{2.3}$$

The equation of an optimal policy (Equation 2.3) includes an expectation value of an infinite sum, which makes it computationally infeasible. Therefore, finite-horizon value functions are used instead [21]. Value iteration algorithm is such a function that can be used to compute an optimal value function for a POMDP with a specified finite horizon. This algorithm uses dynamic programming to recursively calculate values for each belief state b. It starts with evaluating the value of a belief state for a horizon length of 1 ($t = 1$). Next, at the second iteration of the algorithm, the value function for a horizon length of 2 ($t = 2$) is computed, which uses the results of the first iteration. Iterations continue until the value function for the desired horizon is found [25].

As t increases, the complexity of the optimal value function grows fast, due to the total number of linear functions needed to represent it. Calculating the optimal

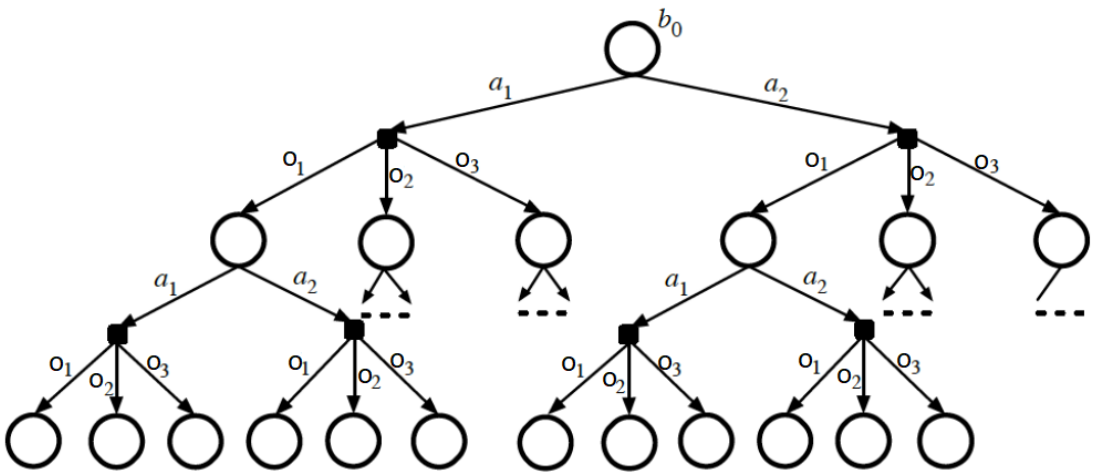

Figure 2-1: Belief tree

value function up to horizon H is estimated to require around $O(|S|^2|A|^H|\Omega|^H)$ work [15] [21].

2.4.1 Belief Tree

Belief tree is another way to search for an optimal action. A belief tree consists of nodes, where every node represents a belief. At the root of the belief tree is the initial belief b_0. Each node of the tree first branches into action edges, which represent valid actions to the system whose current belief is represented by the node. Each action edge in its turn branches into observation edges, which represent observations received after applying the action [28]. Therefore, the children of a node are precisely those beliefs derived using the Equation 2.1 [21]. Figure 2-1 illustrates a belief tree.

2.5 Online Algorithms

In general, POMDP solvers are categorized into two approaches: offline and online. Offline algorithms find the best action to execute for all possible situations before the execution. Even though these approximate algorithms result in an optimal policy, it takes them a significant amount of time to finish. Moreover, even the smallest changes in the environment cause recomputing the full policy. These limitations of

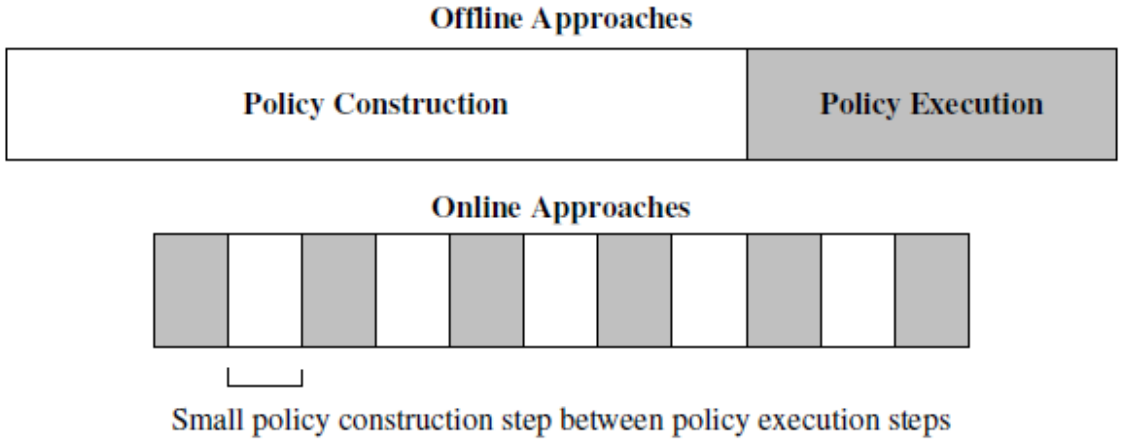

Figure 2-2: Comparison between offline and online approaches

offline algorithms make them applicable to problems with small to mid-size domains only [25].

Online algorithms are accepted as a better alternative to use in large POMDPs, as they consider only belief states reachable from the agent's current belief state and try to find an optimal local policy for it. Hence, the computation is focused on a small set of beliefs and is more feasible.

Figure 2-2 [25] illustrates the main difference between offline and online algorithms. We see from the figure that both offline and online algorithms consist of planning and execution phases. However, in offline algorithms the planning phase precedes the execution phase entirely, whereas in online algorithms they alternate at each time step.

Ross et al. [25] provide a general framework for online planning in their survey, where they describe what is involved in both phases of online algorithms. The planning phase includes passing the current belief state to the algorithm and computing the best action to execute in this belief. This is achieved by building an AND/OR tree which is another representation of a belief tree presented in Section 2.4.1. In an AND/OR tree, OR-nodes represent belief states reachable from the current belief state. At these nodes action must be selected. Then AND-nodes represent all possible observations from taking that action that bring to the following beliefs. Value

estimates from the fringe nodes are propagated all the way up to the root, so that the value of the current belief can be calculated using the Equation 2.3. Values of fringe nodes are usually estimated using an approximate value function computed offline. It is also quite common to keep not just a single value, but both a lower bound and upper bound on the value of the nodes. The result of the planning state is the best action identified for the current belief, which is then performed at the execution phase. Executing the action in its turn requires the current belief to update and a new belief tree obtained accordingly.

Online algorithms can be classified into three categories: Branch-and-Bound Pruning, Monte Carlo Sampling and Heuristic Search. They generally differ on either how they choose nodes of a belief tree to explore or how tree expansion is implemented. At the end they all aim to limit the number of explored nodes.

In the following sections two different online algorithms are presented. They are considered state-of-the-art online planning solvers and use Monte Carlo Sampling.

2.5.1 POMCP

As it has been already mentioned, both POMCP and DESPOT are examples of Monte Carlo algorithms. Monte Carlo algorithms try to overcome the problem of fully expanding the search tree over a large set of observations to find the best action. The idea behind these algorithms is to sample a subset of observations at each expansion and only consider beliefs reached by these sampled observations. This allows for deeper search, as it reduces the breadth of the search tree [25].

In Monte Carlo methods sampling happens from multiple random simulations, rather than a single one. They use a so-called black-box simulator as a generative model of the POMDP, which makes a fully defined model unnecessary. Such a simulator takes a state and an action as an input, and outputs a sample of a successor state, observation and reward. It is able to update the value function with no looking inside the black box that describes the dynamics of the model [26].

Partially Observable Monte-Carlo Planning (POMCP) was first introduced by Silver et al. [26] as a combination of a Monte Carlo update of the agent's belief state

with a Monte Carlo tree search from the current belief state.

Monte Carlo tree search (MCTS) uses Monte Carlo simulations to build a search tree. Each iteration of MCTS starts with sequentially selecting a node for expansion in best-first order according to the statistics stored in the nodes. In the next step children of the selected node are added to the tree, i.e. it gets expanded. If the node does not correspond to the terminal state, a random simulation is initiated starting from the state the node corresponds to. Simulation stops when it reaches a terminal state, and its outcome is backpropagated to the root. The iterations complete when a certain condition is met, and an action at the root gets selected [29].

In contrast to MCTS, POMCP constructs a search tree of histories instead of states by incorporating observations. POMCP tree contains two types of nodes: history nodes and sequence nodes. History nodes represent a possible history of actions and observations. Sequence nodes represent a sequence of the previous history and the action chosen at the previous history node. Edges from history nodes to sequence nodes represent the selected actions, and edges from sequence nodes to history nodes represent received observations [26].

The second component of POMCP is a Monte Carlo update of the agent's belief state instead of the usual Bayes theorem (Equation 2.1). POMCP uses an unweighted particle filter to approximate the belief state, and apply the Monte-Carlo method to update particles based on sample observations, rewards, and state transitions. Black box simulator provides an efficient implementation for unweighted particle filters, thus excellent scalability for larger problems [26].

2.5.2 DESPOT

Somani et al. [28] introduce Determinized Sparse Partially Observable Trees (DESPOT) as a sparse approximation of a standard belief tree. Instead of capturing the execution of all policies under all possible scenarios, a DESPOT captures the execution of all policies under a fixed set of K sampled scenarios. A DESPOT contains all the action branches, but not all the observation branches, only those observed under the sampled scenarios. Considering all the observation branches is found unnecessary to

produce an approximately optimal policy, since ultimately they get averaged in the Equation 2.3. As a result we get a tree of size $O(|A|^H K)$, which is significantly smaller than $O(|A|^H |\Omega|^H)$, and brings to dramatic improvement in computational efficiency for moderate K values.

A scenario is an abstract simulation trajectory with some start state s_0. It is formally defined as an infinite random sequence for a belief b, $\phi_b = (s_0, \phi_1, \phi_2, ...)$, where s_0 is a state sampled according to b and each ϕ_i is a real number sampled independently and uniformly from the range [0, 1]. To construct a DESPOT tree, a deterministic simulative model is applied to all possible action sequences under K sampled scenarios. For each action under a scenario we get a simulation trajectory, which traces out a path of actions and observations from the root of the standard belief tree. All the nodes and edges on this path are added to the DESPOT tree, each node of which stores a set of all scenarios that it encounters. The DESPOT tree is complete, when the above process is applied for every action sequence.

Next, the value function for a belief is determined as the average total discounted reward obtained by simulating the policy under each scenario. Somani [28] show in their work that as K increases, the value function converges to the optimal value function.

Chapter 3

Experiments

We applied two algorithms presented in Sections 2.5.1 and 2.5.2 to two benchmark problems: *Laser Tag* and *Rock Sample*. In the following sections we first present the problems and their POMDP models. Next the toolkit used for POMDP planning is presented, followed by the experiments setup and the results.

3.1 Laser Tag

Laser Tag is an expanded version of Tag, a standard benchmark problem presented by Pineau et al. [22]. Tag is based on the game of lasertag. There are two robots on a grid with 29 positions, where one robot searches and tries to tag another robot, who is trying to escape being tagged. Initially they start at random positions. The chasing robot is always aware of its own position, but not of its opponent's unless they happen to be in the same grid cell. Both robots can only move to four adjacent cells, and the chasing robot pays a cost of -1 for each move. The chasing robot aims to tag its target, and if it is in the same cell with the opponent during the attempt, it is rewarded +10. In case it is not successful, it is penalized -10.

In Laser Tag the goal remains the same as in Tag, but there are several significant differences. First, the grid in Laser Tag is a rectangular of the size 7×11 with eight randomly placed obstacles. Second, the chasing robot is unaware of its own position. Third and most importantly, it is equipped with a laser that returns the distances to

the nearest obstacles in eight directions as observations.

3.1.1 Laser Tag as a POMDP

A POMDP model for Laser Tag has the following components:

- **Set of states S**: A state is characterized by positions of the robots, which can take values {(0,0), (1,1), ..., (6, 10)} excluding the coordinates occupied by the obstacles.

- **Set of actions A**: There are 5 actions available to the robot: {$North$, $South$, $East$, $West$, Tag}. The first four as their names suggest are deterministic motion actions. The Tag action attempts to tag the opponent robot at the current position of the robot.

- **Transaction function T**: A successful Tag action terminates the simulation. The simulation continues in all other action-outcome cases. Since the robot does not know its own position on the grid, it keeps a vector of transition probabilities for every possible state and action pair. It also stores a distribution of laser readings for each state. Both transition probabilities and reading distributions get updated after any action, which does not lead to the terminal state.

- **Reward function R**: For every motion action the agent receives a penalty of -1. If the action Tag is successful, it is rewarded +10, and penalized -10 if unsuccessful.

- **Set of observations** Ω: An observation includes eight integers, each integer is a laser reading in one of the possible directions. The laser reading in each direction is generated from a normal distribution centered at the true distance of the robot to the nearest obstacle in that direction, with a standard deviation of 2.5.

- **Observation function O**: Probability for an observation is taken directly from the reading distributions which has been introduced in the transition function.

- **Initial belief** b_0: The initial belief contains all the states with equal probabilities that add up to 1.

3.2 Rock Sample

Rock Sample, first developed by Smith and Simmons [27], is a well-established benchmark that models rover's scientific exploration. $Rocksample(n, k)$ describes an instance of the Rock Sample problem, where the rover explores a grid map of size $n \times n$ containing k rocks. Some of the rocks have a scientific value, i.e. they are considered "good", and the rover receives reward by sampling such rocks. The rover is penalized for sampling "bad" rocks. There is an exit at the right side of the map, the rover is also rewarded upon reaching it. At the beginning, the robot is aware of its own position and the positions of the rocks, however unaware of their scientific value. Since sampling a rock is risky, the robot is equipped with a noisy long-range sensor which allows it to check whether a rock is good or not, so that it can decide on either approaching or avoiding it. The accuracy of the sensor depends on the distance between the robot and the rock being checked. Overall, the robot's goal is to sample good rocks and exit when it's finished.

3.2.1 Rock Sample as a POMDP

Components of a POMDP model for $Rocksample(n, k)$ are described as follows:

- **Set of states S**: A state is characterized by $k+1$ features: the rover's position, which can take values {(1,1), (1,2), ..., (n,n)}, and one binary feature for each of k rocks, which can take values $\{Good, Bad\}$. There is also a terminal state, which is reached when the rover moves to the exit.

- **Set of actions A**: There are $k + 1$ actions available to the robot: $\{North, South, East, West, Sample, Check_1, ..., Check_k\}$. The first four actions are deterministic single-step actions. The $Sample$ action samples the rock at the robot's current position. Each $Check_i$ action senses the rock i and returns either

Good or *Bad* observation, depending on the distance and the efficiency value. This will be discussed later in Set of observations.

- **Transaction function T**: Move actions deterministically affect only the rover's current position. $Check_i$ actions, on the other hand, leaves the agent's position unaffected, but changes the probability of the $rock_i$ being good.

- **Reward function R**: For sampling a good rock, the agent receives a reward of 10, and the rock's value is changed to bad. For sampling a bad rock, it receives a penalty of -10. The robot also gets a reward of 10, if it reaches the exit area. No other cost or reward can be obtained from taking other available moves.

- **Set of observations** Ω: There are three possible observations: *Good*, *Bad* and *None*. Observation *None* is produced for moving and sampling. Sensing a rock produces a noisy observation from {*Good*, *Bad*}. The noise in the sensor depends on the efficiency η, which decreases exponentially as a function of Euclidean distance between the rover and a rock. The probability of sensor's accuracy is calculated as $(1 + \eta)/2$, where η is equal to $2^{-d/d_0}$. Here d is the euclidean distance between the positions of the rover and the rock checked, and d_0 is a constant specifying the half efficiency distance.

- **Observation function O**: The probability of receiving observation *None*, given any of the motion actions or the action *Sample* is equal to 1, and of receiving either *Good* or *Bad* is 0. For the action $Check_i$ the probability of getting *None* is 0, and of getting a correct observation from {*Good*, *Bad*} is equal to the efficiency η, which was introduced above.

- **Initial belief** b_0: The initial belief state includes the initial position of the robot, which is known, and a set of k probabilities. These probabilities represent the probability of each rock being good, and are initially equally set to 0.5.

3.3 Approximate POMDP Planning Software

There are several available open-source POMDP libraries written in different languages and implementing a range of POMDP solvers. The *pomdp_py* framework [30] written in Python provides implementations for POMCP, POUCT, and basic value iteration. POMDPs.jl [2], a framework written in Julia, has implementations for POMCP, DESPOT and SARSOP.

To evaluate the performance of POMCP and DESPOT we have used Approximate POMDP Planning Software (APPL)[1], as it includes implementations to both algorithms of our interest and it provides an easy and clear integration with simulated environments. APPL is a C++ toolkit that implements a wide range of algorithms for POMDP planning, offline and online. Specifically, the *APPL Online* package includes implementation to both POMCP and DESPOT that we are interested in.

The *APPL Online* package can be used to solve a problem in the following three steps specified in documentation of the package [20]:

1. *Define a POMDP model of the problem:* POMDP can be represented either in POMDPX format or in C++ by inheriting the DSPOMDP class. Even though it requires more programming, we selected the latter as it can represent large problems and allows to integrate the solvers with external systems. DSPOMDP interface requires a representation of states, actions and observations; the deterministic simulative model; functions related to beliefs and starting states; bound-related functions, and memory management functions.

2. *Setup an interface for communication between the model and an external system:* The package provides the *World* abstract class to be inherited to create a custom world and to serve as such interface. If no communication with an external system is needed, the POMDP model can be used as a simulated world in a built-in implementation of *World*. This way the same model is shared by the solver for planning.

[1] https://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php

3. *Initialize the solver and execute the planning pipeline:* At this step the actual planning is performed through the *Planner* class, where the solver and problem-specific parameters are initialized.

3.4 Experiments Setup

For the experiments we considered instances of a Rocksample(11, 11) (i.e. a map of size $n \times n$ containing 11 rocks) and a Laser Tag scenarios in a simulated environment, designed on the Gazebo simulator[2]. Figure 3-1 illustrates the simulated Gazebo environment for Laser Tag, composed of a 7×11 rectangular grid surrounded by gray walls and containing eight obstacles and two robots. The target robot is colored red, and the chasing robot is green.

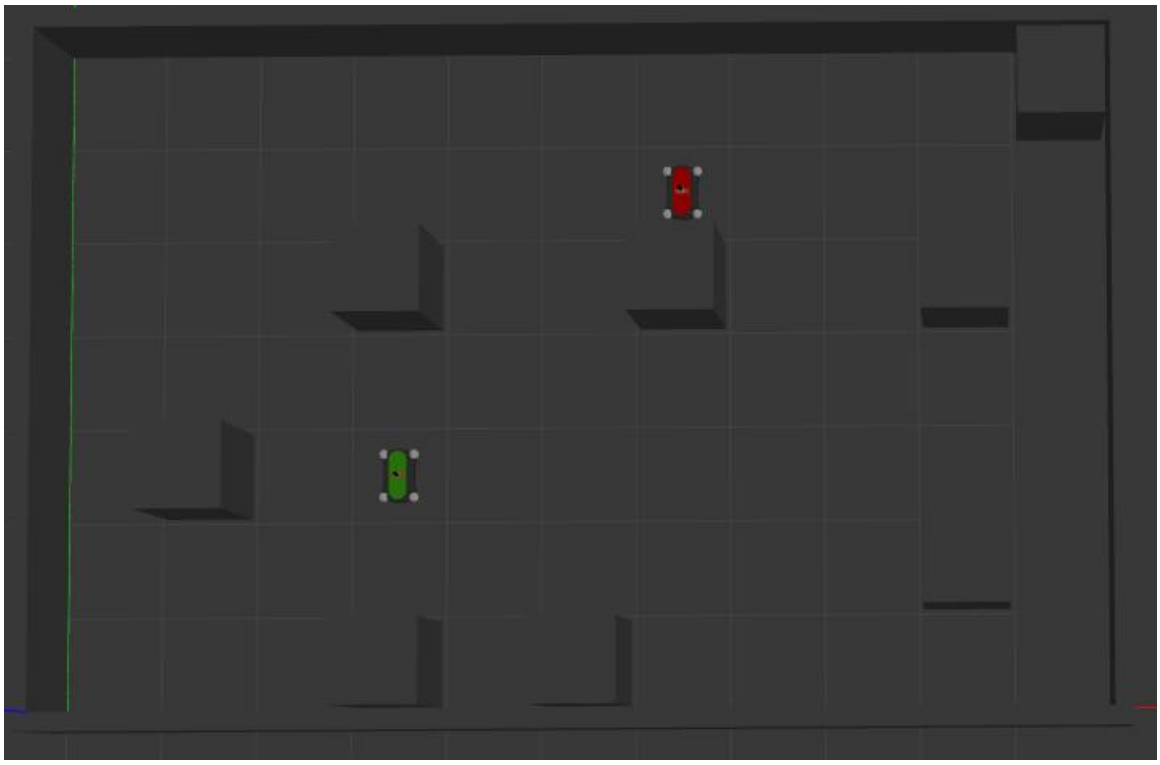

Figure 3-1: Laser Tag in Gazebo simulator

The simulated environment for Rocksample(11, 11) is illustrated on Figure 3-2. The map of Rock Sample is also surrounded by gray walls and of actual size of 11×12

[2]http://gazebosim.org/

with extra grids for the exit, which is located at the far right and marked in red. Good stones that have scientific value are colored blue and bad stones are red. The robot's and the rocks' positions stay unchanged on every simulation, but the scientific values of the rocks are randomized on each run, so that there are always five good rocks out of eleven. This implies that the maximum value for total undiscounted reward is 60.

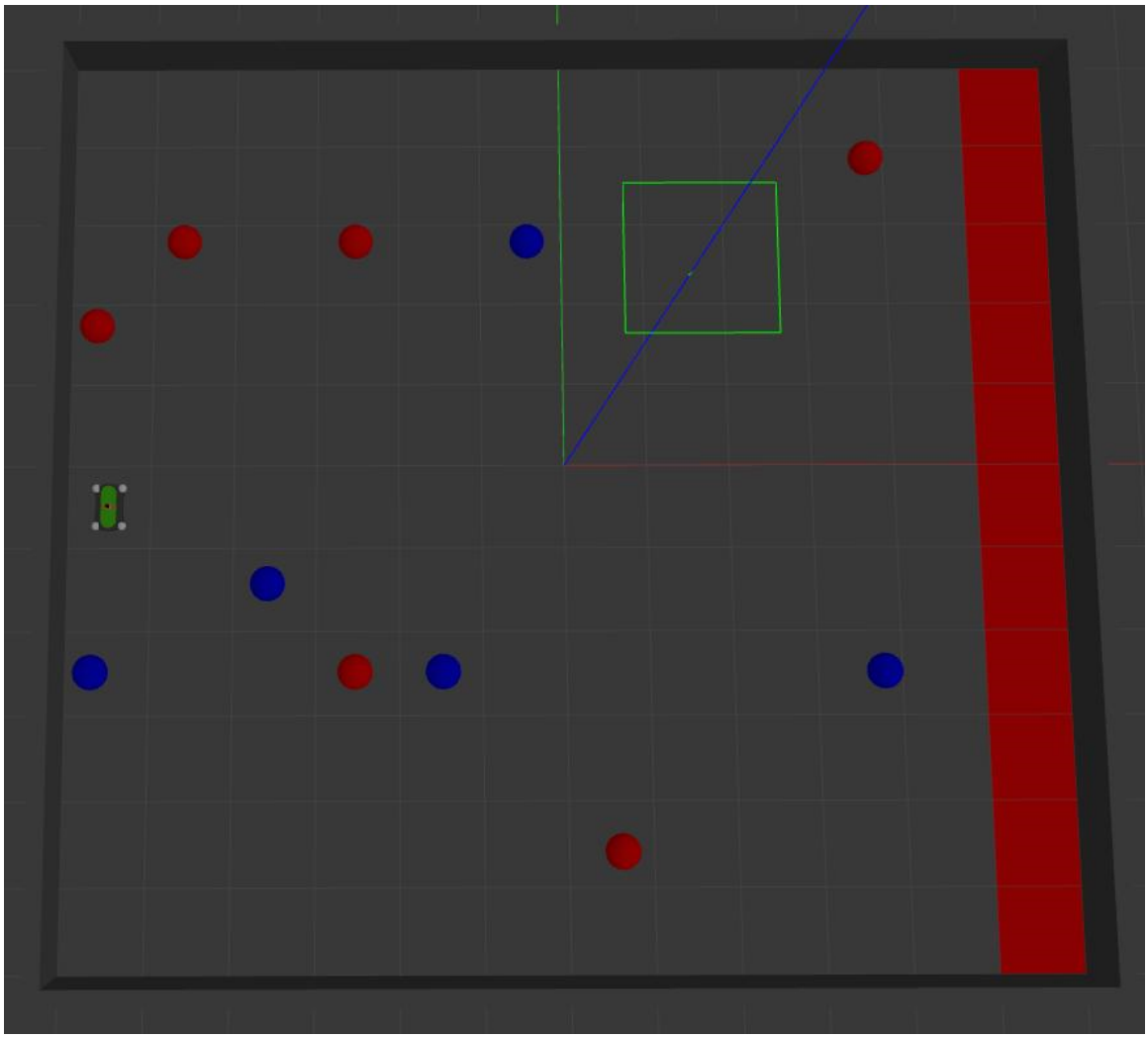

Figure 3-2: Rocksample(11,11) in Gazebo simulator

Our experimental setup includes several components: a POMDP planner, two ROS[3] (Robot Operating System) nodes and a Gazebo simulated environment. The POMDP planner provides actions to be executed by the robots in the simulated environment. The ROS nodes handle the communication among the modules. Actions

[3]https://www.ros.org/

generated by the planner are sent out by the node at the planner side in the custom world. They are received by another node run by the interface of the robot, which maps them to actual motor commands.

For each benchmark we ran each algorithm 10 times and recorded such performance measures as total discounted/undiscounted reward, number of steps taken to complete, and for Rock Sample also the number of rock checks performed.

3.5 Results

We compared the performance of a POMCP planner and a DESPOT planner on two benchmark problems, Laser Tag and Rock Sample. Discount factor in all the experiments was equal to 0.95, which is the typical value for the discount factor. All the experiments were conducted on the same experimental platform.

	Laser Tag	Rocksample(11,11)
|S|	4,830	247,808
|A|	5	16
|Ω|	$\sim 1.5 \times 10^6$	3
POMCP	-8.271	18.604
DESPOT	-5.088	21.652

Table 3.1: Performance comparison of POMCP and DESPOT for two benchmark problems. Average total discounted reward is reported for the algorithms.

The summarized results for both benchmarks are reported in Table 3.1. More detailed results for each problem are presented in Tables 3.2 and 3.3.

Performance measure	**DESPOT**	**POMCP**
Average total discounted reward	-5.088	-8.271
Average total undiscounted reward	-4	-10
Average # of steps	15	21
Average time (seconds)	72.032	111.865

Table 3.2: Performance comparison of POMCP and DESPOT for Laser Tag

From Table 3.1 we can also see how Laser Tag and Rock Sample compare in terms of their state space and observation space. Despite having the smaller state space (4,830 vs. 247,808), the observation space of Laser Tag is extremely large, whereas the

observation space of Rock Sample is small and has only three possibilities. The results indicate that both algorithms can scale up to large domains and solve efficiently the presented problems. However, in terms of average total discounted reward DESPOT outperforms POMCP in both problems.

Performance measure	**DESPOT**	**POMCP**
Average total discounted reward	21.652	18.604
Average total undiscounted reward	52	44
Average # of steps	39	33
Average time (seconds)	134.632	108.782
Average # of rock checks	12	10

Table 3.3: Performance comparison of POMCP and DESPOT for Rock Sample

Table 3.2 illustrates that POMCP not only has a smaller average total discounted reward in the Laser Tag problem, but also takes more steps, hence time to complete. Table 3.3 presents different results for the Rock Sample problem. There POMCP spent less time on average to complete, it took less steps and performed less checks on the rocks. However, this had a negative impact on the total reward and POMCP received substantially lower rewards than DESPOT. These results are comparable with the nature of the problems, specifically their reward systems. In Laser Tag each move that does not lead to the terminal state brings a penalty, therefore the more steps the robot makes, the less reward it receives. In Rock Sample a greater number of rock checks, hence a greater number of steps too, correlate with a larger reward, as the chances of finding good rocks increase and reaching a rock to sample requires additional steps. Therefore, the statistics on the average number of steps serve as additional evidence that DESPOT achieved better results than POMCP.

Chapter 4

Conclusion

4.1 Summary

In this thesis we have extensively introduced Partially observable Markov decision processes (POMDP). POMDP offers a rich mathematical model for framing planning problems with uncertainties. Uncertainties can be of various origins: motion, actuation, sensors and environment. As a result, the agent has to anticipate that its own state and/or the state of the environment it operates in may not be fully observable. In a POMDP these uncertainties are presented as conditional probability functions and the state of the system is approximated as probability distributions over the belief state. The interaction between the system and the agent is carried out through exchanges of actions from the agent to the system and observations in backward direction. Solving a POMDP is finding an optimal policy which maximizes the expected long-term reward by reasoning over belief states. Unfortunately, solving a POMDP exactly is impossible in the worst case and impractical in general due to two difficulties known as "curse of dimensionality" (exponential growth of beliefs) and ”curse of history” (exponential growth of action-observation histories).

However, there has been great progress in developing approximate methods, especially online algorithms have been successful in solving POMDPs efficiently and they even scale up to problems with extremely large domains. Two state-of-the-art online solvers, POMCP and DESPOT, use Monte Carlo method which relies on repeated

sampled simulations to find the best action for the current belief.

In order to illustrate the potential of a POMDP model we investigated two benchmark problems, Laser Tag and Rock Sample. These are standard problems widely used in POMDP research as test domains with large sizes. Laser Tag has a large observation space, and Rock Sample has a large state space. We see from these problems the importance of defining an accurate POMDP model to correctly capture uncertainties and transition dynamics, and produce a valid policy.

We employ the same benchmark problems to assess functionality of an open-source POMDP toolkit, named APPL, and study performance and behaviour of online solvers, POMCP and DESPOT. The APPL Toolkit provides a powerful and flexible interface for modeling even extremely large problems and easy integration with external systems using ROS. It has a clear documentation and the development of the toolkit is ongoing.

Both Laser Tag and Rock Sample have been extended to the Gazebo robotic simulator to approximate a real world scenario. The communication between the planners and the simulator is handled by ROS nodes. We compare POMCP and DESPOT under such metrics as total discounted reward, number of steps and time taken by the algorithm to complete. From our experimental evaluation we observe that POMCP and DESPOT both manage to scale up to both problems and produce a good solution to them. Even though both algorithms demonstrate comparable results, overall DESPOT has achieved stronger performance.

4.2 Future Work

The most immediate direction for future work is to design more complicated and realistic tasks, model them using a POMDP and solve with available solvers. The problems we considered can be easily modified to a range of real-world robot applications such as sensing and target tracking in hazardous environments that are naturally partially observable.

We have already mentioned that to obtain correct policies for a given task it is

essential to define a POMDP model appropriately. The model heavily depends on the assumptions we put in. Thus the real-world tasks can be too complex to the model, which implies they are longer to solve, perhaps even infeasible in real-time. Manipulating objects unknown to the model is another major problem, even insoluble if the model is not trained to predict for objects unseen before. However, there is already a ground work in tackling problems of these sorts. One of the approaches is to use deep neural networks to learn and generalize those policies generated by POMDP solvers[21]. Another data-driven approach uses visual camera images as inputs to a deep recurrent neural network model to enable a robot to push objects of unknown physical properties [10]. We think this a promising direction to turn the research to.

Bibliography

[1] Nikolay Atanasov, Bharath Sankaran, Jerome Le Ny, Thomas Koletschka, George J Pappas, and Kostas Daniilidis. Hypothesis testing framework for active object detection. In *2013 IEEE International Conference on Robotics and Automation*, pages 4216–4222. IEEE, 2013.

[2] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[3] Robert Eidenberger and Josef Scharinger. Active perception and scene modeling by planning with probabilistic 6d object poses. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1036–1043. IEEE, 2010.

[4] Amalia Foka and Panos Trahanias. Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*, 55(7):561–571, 2007.

[5] Neha P Garg, David Hsu, and Wee Sun Lee. Learning to grasp under uncertainty using pomdps. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2751–2757. IEEE, 2019.

[6] Janine Hoelscher, Bartolome de TirajanaSpanien, and J. Pajarinen. Interactive planning under uncertainty. 2017.

[7] Marcus Hoerger. Tractable pomdp-planning for robots with complex non-linear dynamics. 2020.

[8] M. Horowitz and J. Burdick. Interactive non-prehensile manipulation for grasping via pomdps. In *2013 IEEE International Conference on Robotics and Automation*, pages 3257–3264, 2013.

[9] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Grasping pomdps. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4685–4692. IEEE, 2007.

[10] Li Juekun. *Act to See and See to Act: A robotic system for object retrieval in clutter*. PhD thesis, 2018.

[11] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[12] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323, 2011.

[13] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Citeseer, 2008.

[14] Jue Kun Li, David Hsu, and Wee Sun Lee. Act to see and see to act: Pomdp planning for objects search in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5701–5707. IEEE, 2016.

[15] Michael L Littman. A tutorial on partially observable markov decision processes. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.

[16] Tomas Lozano-Perez, Matthew T Mason, and Russell H Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24, 1984.

[17] Ady-Daniel Mezei, Levente Tamás, and Lucian Buşoniu. Sorting objects from a conveyor belt using pomdps with multiple-object observations and information-gain rewards. *Sensors*, 20(9):2481, 2020.

[18] Pol Monsó, Guillem Alenyà, and Carme Torras. Pomdp approach to robotized clothes separation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1324–1329. IEEE, 2012.

[19] Joni Pajarinen and Ville Kyrki. Robotic manipulation in object composition space. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–6. IEEE, 2014.

[20] Cai Panpan, Luo Yuanfu, and Moritz Cremer. Tutorial on using despot. `https://github.com/AdaCompNUS/despot/blob/API_redesign/doc/cpp_model_doc/Tutorial%20on%20Using%20DESPOT%20with%20cpp%20model.md`.

[21] Ignacio Perez Bedoya. *Robotic grasping using POMDPs and machine learning*. PhD thesis, Massachusetts Institute of Technology, 2020.

[22] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032. Citeseer, 2003.

[23] J.R. Raol and A.K. Gopal. *Mobile Intelligent Autonomous Systems*. CRC Press, 2016.

[24] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayesian reinforcement learning in continuous pomdps with application to robot navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 2845–2851. IEEE, 2008.

[25] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[26] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[27] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. *arXiv preprint arXiv:1207.4166*, 2012.

[28] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *Advances in neural information processing systems*, pages 1772–1780, 2013.

[29] Andreas Ten Pas. *Simulation based planning for partially observable markov decision processes with continuous observation spaces*. PhD thesis, Citeseer, 2012.

[30] Kaiyu Zheng and Stefanie Tellex. pomdp_py: A framework to build and solve pomdp problems. *arXiv preprint arXiv:2004.10099*, 2020.