
THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

# Tractable POMDP-planning for robots with complex non-linear dynamics

Marcus Hoerger

Dipl.-Inf.

https://orcid.org/0000-0002-4698-5875

*A thesis submitted for the degree of Doctor of Philosophy at*

*The University of Queensland in 2020*

School of Information Technology and Electrical Engineering

# Abstract

Planning under partial observability is an essential capability of autonomous robots. While robots operate in the real world, they are inherently subject to various uncertainties such a control and sensing errors, and limited information regarding the operating environment. Conceptually these type of planning problems can be solved in a principled manner when framed as a Partially Observable Markov Decision Process (POMDP). POMDPs model the aforementioned uncertainties as conditional probability functions and estimate the state of the system as probability functions over the state space, called beliefs. Instead of computing the best strategy with respect to single states, POMDP solvers compute the best strategy with respect to beliefs. Solving a POMDP exactly is computationally intractable in general.

However, in the past two decades we have seen tremendous progress in the development of approximately optimal solvers that trade optimality for computational tractability. Despite this progress, approximately solving POMDPs for systems with complex non-linear dynamics remains challenging. Most state-of-the-art solvers rely on a large number of expensive forward simulations of the system to find an approximate-optimal strategy. For systems with complex non-linear dynamics that admit no closed-form solution, this strategy can become prohibitively expensive. Another difficulty in applying POMDPs to physical robots with complex transition dynamics is the fact that almost all implementations of state-of-the-art on-line POMDP solvers restrict the user to specific data structures for the POMDP model, and the model has to be hard-coded within the solver implementation. This, in turn, severely hinders the process of applying POMDPs to physical robots.

In this thesis we aim to make POMDPs more practical for realistic robotic motion planning tasks under partial observability. We show that systematic approximations of complex, non-linear transition dynamics can be used to design on-line POMDP solvers that are more efficient than current solvers. Furthermore, we propose a new software-framework that supports the user in modeling complex planning problems under uncertainty with minimal implementation effort.

# Declaration by author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my higher degree by research candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the Copyright Act 1968 unless a period of embargo has been approved by the Dean of the Graduate School.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis and have sought permission from co-authors for any jointly authored works included in the thesis.

# Publications included in this thesis

1. [1] **M. Hoerger**, H. Kurniawati and A. Elfes, Multilevel Monte-Carlo for Solving POMDPs Online, in: Proc. International Symposium on Robotics Research (ISRR) (To Appear), 2019

2. [2] **M.Hoerger**, J. Song, H.Kurniawati and A. Elfes, POMDP-based Candy Server: Lessons Learned From a Seven Day Demo, in: Proc. AAAI International Conference on Autonomous Planning and Scheduling (ICAPS), 2019, pp. 698-706

3. [3] **M. Hoerger**, H. Kurniawati, and A. Elfes, A Software Framework for Planning under Partial Observability. in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 7576-7582

4. [4] **M. Hoerger**, H. Kurniawati, T. Bandyopadhyay and A. Elfes, Linearization in Motion Planning under Uncertainty, in: Proc. 12th International Workshop on the Algorithmic Foundations of Robotics (WAFR), 2016.

# Submitted manuscripts included in this thesis

No manuscripts submitted for publication.

# Other publications during candidature

1. [5] **M. Hoerger**, H. Kurniawati, T. Bandyopadhyay, and A. Elfes, Effects of Obstacle Avoidance to LQG-Based Motion Planners, in: Australasian Conference on Robotics and Automation, ARAA, 2016

2. [6] A. Snoswell, V. Dewanto, **M. Hoerger**, H. Kurniawati and S. Singh, A Distributed, Any-Time Robot Architecture for Robust Manipulation, in: Australasian Conference on Robotics and Automation, ARAA, 2018

3. [7] A. Elfes, R. Steindl, F. Talbot, F. Kendoul, P. Sikka, T. Lowe, N. Kottege, M. Bjelonic, R. Dungavell, T. Bandyopadhyay and **M. Hoerger**, The Multilegged Autonomous eXplorer (MAX), in: Proceedings of the IEEE International Conference on Robots and Automation (ICRA), 2017

# Contributions by others to the thesis

- Dr. Hanna Kurniawati and Dr. Alberto Elfes guided and supported me through my candidature. This included the conception of the problem and substantial help in writing the publications that were generated during this candidature, particular for the earlier publications.

- Joshua Song designed and implemented the perception system for the Kinova MOVO mobile manipulator that was used for the experiments in Chapter 4.
- Aaron Snoswell and Vektor Devanto implemented the low-level system architecture for the Kinova MOVO mobile manipulator that was used for the experiments in Chapter 4

## Statement of parts of the thesis submitted to qualify for the award of another degree

No works submitted towards another degree have been included in this thesis.

## Research involving human or animal subjects

No animal or human subjects were involved in this research.

# Acknowledgments

First and foremost, I want to express my sincere gratitude to my principal advisor Dr. Hanna Kurniawati for her tremendous guidance throughout this journey. Her seemingly endless patience and ongoing support has helped me to grow as a researcher and professional on all levels. Without Hanna's leadership and advice this thesis would have not been possible.

Furthermore, I am grateful for Dr. Alberto Elfes to serve as my co-advisor. Alberto not only became an advisor, but a mentor and leader who keeps on inspiring me and countless other people. Alberto is the one who encouraged me to pursue a PhD program after working as a research intern within CSIRO's Robotics and Autonomous Systems Group. Thank you for granting me this amazing opportunity and making my PhD candidature possible in the first place.

Within UQ's School of ITEE, I want to thank my Confirmation and Review Committee for their time spent on assessing my candidature and providing valuable feedback. A special thank you goes to the Chair of the Committee Dr. Surya Singh. He always had an open ear for research related questions and granted valuable time for presentation rehearsals and interesting discussions. Thank you also to ITEE's HDR Liaison Officer Tracey Miller and all the people at the Graduate School for their support with administrative related tasks.

I thank all the great people I have met within CSIRO's Robotics and Autonomous Systems Group, particularly Dr. Navinda Kottege who supported me in securing a UQI International Student and CSIRO Postgraduate Student scholarship. Thank you to Dr. Tirthankar Bandyopadhyay for your advice and for being a great colleague.

Outside the professional domain I want to thank my mother Eva Hoerger and my sister Natascha Hoerger for being a major pillar in my life, for their tremendous personal support, and for providing a safe haven. Thank you to all my friends back home in Germany and the countless new friends I've met while staying in Australia. You provided a place for me to retreat, relax and enjoy life outside the lab.

Last but not least, I want to thank my dad Dieter Hoerger, who unfortunately passed away during this candidature. Thank you for always believing in me, supporting me in all my decisions and for being proud of me.

## Financial support

This research was supported by the University of Queensland International Scholarship (UQI) and the CSIRO Postgraduate Studentship.

## Keywords

partially observable markov decision process, pomdp, motion planning under uncertainty, modelling, control

## Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 010303, Optimisation, 40%
ANZSRC code: 080110, Simulation and Modelling, 40%
ANZSRC code: 080306, Open Software, 20%

## Fields of Research (FoR) Classification

FoR code: 0103, Numerical and Computational Mathematics, 40%
FoR code: 0801, Artificial Intelligence and Image Processing, 40%
FoR code: 0803, Computer Software, 20%

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Motion planning under partial observability is both challenging and essential for autonomous robots. To operate reliably, an autonomous robot must act strategically to accomplish its task, despite being subject to various types of uncertainties, such as motion, actuation and sensing uncertainty and uncertainty regarding the environment the robot operates in. Due to these uncertainties, the robot does not have full observability of its state and/or the state of its operating environment. Conceptually, this problem can be solved in a systematic and principled manner when framed as the Partially Observable Markov Decision Process (POMDP) [8]. A POMDP represents the aforementioned uncertainties as conditional probability functions, and estimates the state of the system (following a sequence of actions and observations that the robot has performed and perceived) as probability distributions over the state space, called *beliefs*. Solving a POMDP constitutes of systematically reasoning over the belief space $\mathcal{B}$, the set of all probability distributions over the state space, by taking into account the various uncertainties the robot is subject to. The robot then chooses the optimal strategy with respect to beliefs rather than single states in order to maximize its expected long-term outcome. Unfortunately, solving a POMDP exactly is computationally intractable in general [9], even for 3DOFs point robots [10, 11]. In particular the size of $\mathcal{B}$ grows exponentially with the size of the state space ("curse of dimensionality") and the number of possible strategies grows doubly exponential with the planning horizon ("curse of history"). Due to these computational challenges, POMDPs have often been considered impractical for all but the simplest problems.

However, in the past two decades we have seen tremendous progress in the development of approximate POMDP solvers that trade optimality for computational tractability. The underlying principle of these methods is the fact that in many applications a near-optimal strategy is "good enough" to accomplish a given task. Several general approximate POMDP solvers —that do not restrict the type of dynamics and sensing model of the system, nor the type of distributions used to represent uncertainty— can now compute good motion strategies on-line with a 1-10Hz update rate for a number of robotic problems [12–15]. Despite these advances, applying state-of-the-art approximate POMDP solvers to realistic robotic systems remains challenging. These systems often have a large number of degrees-of-freedom (DOFs) and complex, non-linear transition dynamics, causing the efficiency of

current POMDP solvers to degrade rapidly. To find a good strategy, most today's solvers simulate the effect of many sequences of actions from different beliefs. A simulation run generally requires expensive numerical integrations, and complex transition dynamics tend to increase the cost of each numerical integration, which in turn significantly increases the total planning cost. Of course, this cost increases rapidly for problems that require more or longer simulation runs, such as in problems with long planning horizons.

As a computationally attractive alternative to general POMDP-planning, various methods have been developed that extend algorithms from sampling-based deterministic motion planning to the stochastic domain. These methods typically simplify planning in the belief space by imposing restrictions on the type of systems and class of beliefs, which in turn speeds up the planning process. However, it is not clear how these problem simplifications affect the quality of the resulting motion strategies.

Another challenge in applying POMDPs to physical robots with complex transition dynamics is the fact most (if not all) implementations for state-of-the-art solvers restrict the user to specific data structures for the POMDP model, or the model has to be hard-coded within the solver implementation. This, in turn, severely hinders the process of applying POMDPs to physical robots.

In this thesis we aim to make POMDPs more practical for realistic robotic motion planning tasks under partial observability. We show that systematic approximations of complex, non-linear transition dynamics can be used to design on-line POMDP solvers that are more efficient than current state-of-the-art solvers. Furthermore, we propose a new software-framework which supports the user in modeling complex planning problems under partial observability with minimal implementation effort.

In this chapter we present a brief introduction to motion planning under uncertainty algorithms and discuss their shortcomings in Section 1.1. We then present a formal introduction to POMDPs Section 1.2, followed by a brief overview of POMDP solvers, with a focus on on-line solvers in Section 1.3. In Section 1.4 we discuss the limitations of on-line POMDP solvers for systems with complex, non-linear transition dynamics before providing an overview of our main contributions in Section 1.5. Finally, the remainder of this thesis is outlined in Section 1.6.

## 1.1 Motion Planning under Uncertainty

To reduce the complexity of planning in the belief space, several motion planning under uncertainty algorithms have been proposed that reap from the success of probabilistic sampling-based algorithms for deterministic problems, such as Probabilistic Roadmap (PRM) [16], Rapidly-Exploring-Random-Tree (RRT) [17] variants thereof [18, 19]. These algorithms enabled solving motion planning problems involving robots with many degrees-of-freedom that were deemed too difficult to solve with traditional methods, such as grid-search-based methods [20, 21].

To lift the planning problem from the state space to the belief space, Stochastic Roadmap [22] combines PRM with the Markov Decision Process (MDP) [23], a special class of POMDP that assumes the state of the robot is fully observable. Due to the use of MDP, this method cannot handle partial observability. The Belief Roadmap [24] can handle partial observability, but restricts the robot's

transition and observation dynamics to be linear(ized) and the beliefs to be Gaussian distributions. More recently methods have been proposed that formulate the belief space planning problem as a stochastic control problem: LQG-MP [25] combines RRT with a Linear-Quadratic-Gaussian (LQG) feedback-controller [26]. It samples a finite set of trajectories using RRT and uses an LQG-controller to follow the trajectory that minimizes an expected quadratic cost function. FIRM [27] extends PRM to the belief space and constructs independent LQG-controllers for each node. HFR [28] is an on-line variant of LQG-MP. Similar to LQG-MP, HFR uses RRT to sample a set trajectories that are evaluated in belief space, but does so at each planning step. In contrast to methods that extend deterministic sampling-based motion planning algorithms to belief space planning, [29] starts from a nominal trajectory that is iteratively modified using sequential quadratic programming [30] to minimize a cost function, assuming maximum likelihood observations.

Note that although these methods don't address the general POMDP problem, they can be seen as a special class of POMDP solvers for motion planning under uncertainty problems. We will therefore refer to them as *linearization-based POMDP solvers*.

A common limitation of this class of solvers (except for Stochastic Roadmap, which cannot handle partial observability), is the fact that they rely on linear(ized) system and observation dynamics and assume that beliefs can be represented as Gaussian distributions. However, it is known that linearization only works well for systems with "mild" [31] non-linearities. Additionally, for robots operating in cluttered environments or in the vicinity of dynamic constraints, Gaussian distributions can be inadequate to accurately model beliefs. Even though computationally more complex, POMDP solvers don't suffer from these limitations.

## 1.2 Partially Observable Markov Decision Process (POMDP)

The Partially Observable Markov Decision Process is a principled mathematical framework to model and solve motion planning problems under partial observability. Formally a POMDP is a tuple $< S, A, O, T, Z, R, \gamma >$, where $S$, $A$ and $O$ are the state, action and observation spaces of the robot. $T$ and $Z$ model the uncertainty in the effect of taking actions and receiving observations as conditional probability functions $T(s,a,s') = p(s'|s,a)$ and $Z(s',a,o) = p(o|s',a)$, where $s, s' \in S$, $a \in A$ and $o \in O$. $R(s,a)$ models the reward the robot receives when performing action $a$ from $s$ and $0 < \gamma < 1$ is a discount factor. Due to uncertainties in the effect of actions and receiving observations, the true state of the robot is only partially observable. Hence, instead of planning with respect to states, the robot has to plan with respect to probability distributions $b \in \mathcal{B}$ over the state space, called beliefs, where $\mathcal{B}$, called the belief space, is the set of all probability distributions over $S$. The solution of a POMDP is an optimal policy $\pi^*$, a mapping from beliefs to actions $\pi^* : b \mapsto a \in A$ such that the robot maximizes the expected discounted future reward when following $\pi^*$, *i.e.*

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1.1}$$

where $r_t$ is the reward received at time $t$. Once $\pi^*$ has been computed, it can be used as a feedback-controller: Given the current belief $b$, the robot performs $\pi^*(b)$, receives an observation $o \in O$ and updates its belief according to $b' = \tau(b,a,o)$, such that

$$b'(s') = \tau(b,a,o)(s') = \eta Z(s',a,o) \int_{s\in S} T(s,a,s')b(s)\mathrm{d}s \tag{1.2}$$

where $\eta = 1/\left(\int_{s'\in S} Z(s',a,o) \int_{s\in S} T(s,a,s')\mathrm{d}s\mathrm{d}s'\right)$ is a normalization constant. The value achieved by a policy $\pi$ at a particular belief $b$ can be compactly expressed as

$$V_\pi(b) = R(b,\pi(b)) + \gamma \int_{o\in O} Z(b,\pi(b),o)V_\pi(\tau(b,\pi(b),o))\mathrm{d}o \tag{1.3}$$

where $R(b,a) = \int_{s\in S} R(s,a)b(s)\mathrm{d}s$ and $Z(b,a,o) = \int_{s'\in S} Z(s',a,o) \int_{s\in S} T(s,a,s')b(s)\mathrm{d}s\mathrm{d}s'$. The value achieved by the optimal policy $\pi^*$ is then the policy that satisfies $\pi^*(b) = \arg\max_\pi V_\pi(b)$.

## 1.3 POMDP Solvers

POMDP solvers have seen tremendous advances in the past two decades. The key of their success is the use of sampling to trade optimality for computational tractability. Instead of computing a policy for the entire belief space, sampling based planners represent $\mathcal{B}$ by a set of sampled beliefs and compute the policy for this sampled set only. In general there are two classes of POMDP solvers: Off-line an on-line solvers. Off-line solvers compute a policy a-priori which is then executed during run-time. These solvers (*e.g.* [32–36]) typically use a point based approach to compute the value function at a representative set of sampled beliefs.

In contrast to off-line solvers, on-line solvers interleave policy computation and policy execution. They search for a single best action for the current belief only, execute the action and update the belief. This process then repeats from the new belief. By computing a policy with respect to the current belief only, on-line POMDP solvers usually scale much better to larger problems [37]. Hence, in this thesis, we are primarily focusing on on-line solvers.

To compute a policy for the current belief, on-line solvers typically construct and evaluate a belief tree using lookahead search. A belief tree is a set of nodes, representing beliefs, that are connected via action-observation edges, such that a belief $b'$ is a child of $b$ via edge $(a,o)$ if $b' = \tau(b,a,o)$. The root of the belief tree represents the initial belief. An illustration of a belief tree is shown in Figure 1.1.

Various search methods have been proposed: Heuristic search methods such as [38], [39] and [40] maintain upper- and lower bounds of the value of belief nodes and use heuristics to decide which leaf nodes to expand. RBTSS [41] uses branch-and-bound pruning to expand the belief tree in depth-first order up to a pre-defined depth, while pruning sub-optimal actions to focus the search on more promising parts of the belief tree. [42] expands the belief tree in depth-first order as well, but limits the number of observations expanded for each action to reduce the branching factor of the tree.

More recently Monte-Carlo-Tree-Search (MCTS) [43] has been successfully applied to POMDP planning. POMCP [13] represents beliefs as sets of particles so as to handle large or continuous state spaces. To construct and evaluate the belief tree, POMCP uses Monte-Carlo sampling: Starting

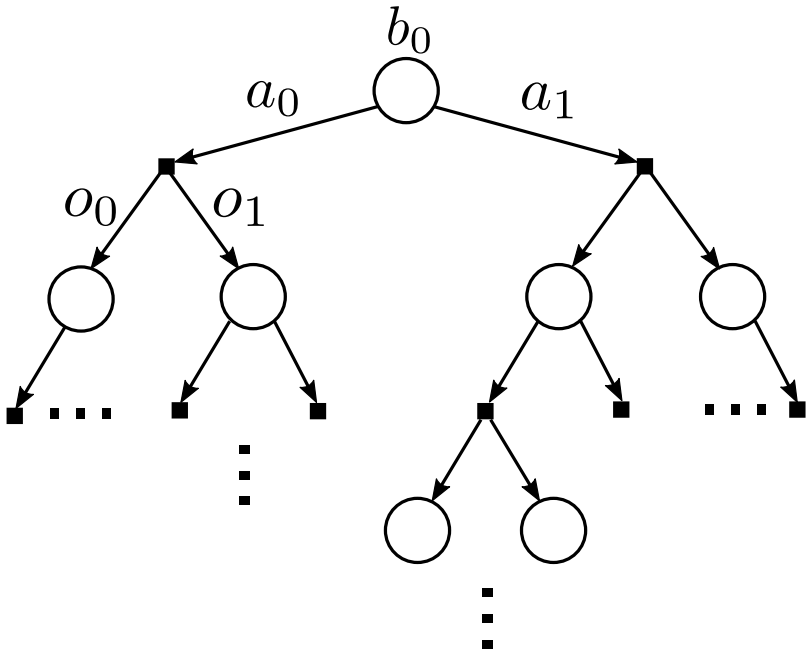



Figure 1.1: Illustration of a belief tree. The nodes represent beliefs. Every node has $|A|$ outgoing edges that represent actions. These actions branch into $|Z|$ observations. The root of the tree represents the initial belief $b_0$.

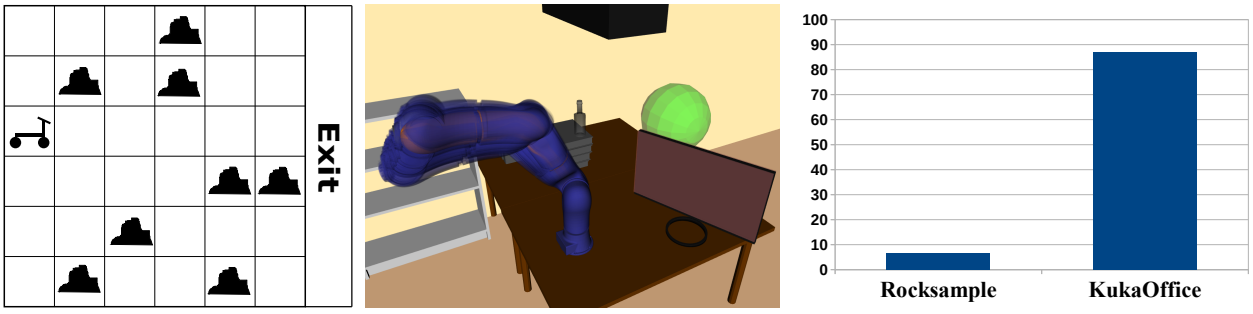
from the current belief, POMCP samples state-action-observation-reward trajectories of the system using forward-simulation. A crucial component of POMCP is the action-selection strategy during the simulations. POMCP frames this problem as a Multi-Armed-Bandid (MAB) [44] problem and uses Upper Confidence Bound1 (UCB1) [45] to select actions. A similar strategy is used by ABT [12]. ABT has been designed to quickly adapt it's policy to changes in the POMDP model during run-time. Instead of computing a policy from scratch, ABT identifies and updates only the parts of the belief tree that are affected by these changes. Although the primary contribution of ABT is to handle changes of the POMDP model during run-time, it has been successfully applied to problems where the model is static. A different approach is used by DESPOT [14]. DESPOT uses a combination of Monte-Carlo sampling, heuristic search and branch-and-bound pruning. It constructs a sparse representation of the belief tree by sampling a fixed number scenarios a-priori, and restricting the forward-search to the sampled scenarios.

While POMCP, ABT and DESPOT can handle continuous state spaces, they are limited to discrete action spaces and discrete or discretized observation spaces. Very recently, on-line solvers have been proposed that can handle continuous action [15, 46] and observation spaces [46, 47]. Nevertheless, efficiently solving POMDP problems with continuous action spaces remains an open problem.

## 1.4 Limitations of On-line POMDP Solvers

While state-of-the-art on-line POMDP solvers such as POMCP, ABT and DESPOT have greatly increased the capability of on-line POMDP planning, they all rely on a large number of sampled forward-simulations to estimate the expected outcome of sequences of actions. Although this strategy

works well for small to medium-sized problems with simple transition dynamics, it quickly becomes problematic for systems where even a single forward-simulation is expensive. As a motivating example that illustrates the problem, we applied the state-of-the-art MCTS-based solver ABT to two problem scenarios:




(a) (b) (c)

Figure 1.2: (a) An instance of the Rocksample problem, a common benchmark problem for on-line POMDP solvers. (b) The manipulator problem, where a 7DOFs torque-controlled manipulator has to reach a goal area (green sphere) with its end-effector, while being subject to control and observation errors. Blue particles represent samples from the belief. (c) Percentage of the total planning time (CPU time) used by ABT to perform the forward-simulations during policy computation in both problems.

The first problem scenario is *Rocksample* [48], a common scalable benchmark problem with simple, closed-form transition dynamics. Rocksample models a Mars rover seeking to collect samples from valuable rocks in the environment, before moving to exit area. In the second problem, *KukaOffice*, a torque-controlled 7DOFs manipulator operating inside a 3D-office environment has to reach a goal area inside the environment while being subject to control and observation errors. Here the transition dynamics are simulated using a physics engine. The two problem scenarios are shown in Figure 1.2(a) and (b) respectively. Figure 1.2(c) shows, for both problems, the percentage of the total planning time per step (1s for *Rocksample* and 8s for *KukaOffice*) that is spent on carrying out the forward-simulations. It shows that for the Rocksample problem, only around 6% of the total planning time is used to for these simulations, while for the *KukaOffice* problem almost 90% of the total planning time is attributed to the forward-simulations of the transition dynamics used by ABT to evaluate sequences of actions. This severely hinders ABT's ability to compute a good policy fast (hence the long planning time of 8s per step). Of course, ABT is not the only on-line solver whose efficiency degenerates for systems with complex, non-linear transition dynamics. On the contrary, many on-line solvers, including the state-of-the-art solver DESPOT, rely on a large number of forward-simulations for each action to expand a single belief. When simulations are expensive, this requires substantially more planning time. In a preliminary test we have found that in the *KukaOffice* problem DESPOT requires, on average, 12s to compute the forward-simulations for a single belief expansion using 50 samples (50 is a tenth of what is commonly used [14]). This is already significantly more than the allowed planning time per step.

It is therefore clear that in order to address these limitations, we have to find ways to reduce the cost of simulation during planning, without sacrificing quality of the resulting policies. This is precisely

what we aim to achieve in this thesis.

## 1.5 Contributions

This thesis focuses on making on-line POMDPs more practical for realistic robotic motion planning problems under partial observability. In particular we study two approaches of systematic approximations of complex, non-linear transition dynamics and how they can be used to design on-line POMDP solvers that are more efficient that current state-of-the-art solvers.

In the first approach we investigate if and how linearized system dynamics and Gaussian belief assumptions can be used to simplify non-linear motion planning problems under partial observability and, as a result, speed-up the planning process. We do this by proposing a new measure of non-linearity for stochastic systems, called Statistical-distance-based Non-linearity Measure (SNM) and show that SNM can approximately decide when linearization is effective and when it should be avoided. We also show both theoretically as well as experimentally that SNM is more suitable in identifying non-linearities induced by the operating environment compared to an existing state-of-the-art measure of non-linearity. This allows us to design a new on-line POMDP solver, called SNM-Planner. SNM-Planner consists of two component solvers, a general solver (ABT [12]) and a linearization-based solver (MHFR, adapted from [28]), and decides during run-time which solver to use to compute the policy, based on a local approximation of SNM around the current belief. Experimental results on a car-like robot with second-order dynamics and a 4DOFs and 7DOFs manipulator with torque-control indicate that this simple approach is more effective than the two component solvers alone.

In the second approach we use a *sequence* of approximations of the transition dynamics to design a new general sampling-based on-line POMDP solver called Multilevel POMDP Planner (MLPP). MLPP combines a recently proposed Monte-Carlo technique for variance-reduction, Multilevel Monte-Carlo (MLMC) [49, 50], with Monte-Carlo-Tree-Search to reduce the cost of sampling. We show experimentally that MLPP significantly outperforms two of the fastest today's on-line POMDP solvers, ABT and POMCP, on four different problem scenarios, involving POMDP-based torque control, navigation and grasping.

To further reap recent advances in POMDP planning and make POMDPs more applicable to complex motion planning problems under partial observability, we propose a new software-framework, called On-line POMDP Planning Toolkit (OPPT). Instead of hard-coding the problem within a given solver implementation, OPPT provides a rich and easy-to-use architecture that enables the user to model robotic motion planning problems under uncertainty with little-to-no additional implementation effort. Furthermore, OPPT is not tied to a specific solver implementation. Instead, OPPT provides a general solver API that allows to user to easily implement new POMDP solvers. We have tested OPPT on a physical manipulation task, where a 6DOFs manipulator must reliably pick up a cup from a table, use the object to scoop candies from a nearby box, and put the cup back on the table, while being subject to significant perception errors and unexpected events in the environment.

## 1.6 Outline

The following chapters are organized as follows. We start by presenting our proposed measure of non-linearity SNM in Chapter 2. We formally introduce SNM and show both theoretically as well as via an extensive empirical study that SNM is more suitable than a recently proposed measure of non-linearity to identify when a linearization-based POMDP solver likely yields a good policy and when a general solver should be used. Furthermore we introduce our derived on-line solver SNM-Planner that combines a general and a linearization-based solver and show experimentally that SNM-Planner yields better policies than using the component planners alone. In Chapter 3 we then propose our new general sampling-based on-line POMDP solver MLPP that uses a sequence of approximations to the robot's transition dynamics and combines them via a Multilevel Monte-Carlo estimator and Monte-Carlo-Tree-Search to reduce the cost of sampling. We then discuss under which assumptions MLPP is asymptotically optimal and show experimentally that MLPP computes near-optimal policies significantly faster than two of the fastest today's on-line POMDP solvers. In Chapter 4 we introduce our proposed POMDP software framework OPPT. In this chapter we describe the general architecture of OPPT and discuss how complex problems can be modeled within the OPPT framework. We then show the versatility of OPPT via a real-world manipulation task. Chapter 5 concludes this thesis and outlines possible future work.

The following publication has been incorporated as Chapter 2.

- [4] **M. Hoerger**, H. Kurniawati, T. Bandyopadhyay and A. Elfes, Linearization in Motion Planning under Uncertainty, in: Proceedings of the 12th International Workshop on the Algorithmic Foundations of Robotics (WAFR), 2016.

| Contributor | Statement of contribution | % |
|---|---|---|
| **Marcus Hoerger** | writing of text | 50 |
| | proof-reading | 50 |
| | theoretical derivations | 80 |
| | numerical calculations | 100 |
| | preparation of figures | 100 |
| | initial concept | 20 |
| Hanna Kurniawati | writing of text | 40 |
| | proof-reading | 50 |
| | supervision, guidance | 60 |
| | theoretical derivations | 20 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 80 |
| Alberto Elfes | writing of text | 0 |
| | proof-reading | 0 |
| | supervision, guidance | 40 |
| | theoretical derivations | 0 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 0 |
| Tirthankar Bandyopadhyay | writing of text | 10 |
| | proof-reading | 0 |
| | supervision, guidance | 0 |
| | theoretical derivations | 0 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 0 |

# Chapter 2

# Linearization in Motion Planning under Uncertainty

## 2.1 Introduction

Linearization is a common practice in in solving non-linear control and estimation problems. In the context of motion planning under uncertainty, linearization-based methods [24, 25, 27, 28, 51] commonly use the LQG-framework [52] to speed-up the evaluation of many sequences of actions from different beliefs. This is acheived by restricting the beliefs to be (multivariate) Gaussian distributions. Together with a linearized system, this allows for closed-form computations of subsequent beliefs that result from performing an action and perceiving an observation, which is not possible for non-parametric beliefs. In contrast, general POMDP solvers (*e.g.* [12–14]) typically represent beliefs as sets of particles. To accurately evaluate different sequences of actions, these general solvers often require a large number of expensive forward simulations from the current belief. As a result, the linearization-based planners require less time to simulate the effect of performing a sequence of actions from a belief, and therefore can *potentially* find a good strategy faster than the general method. However, it is known that linearization in control and estimation performs well only when the system's non-linearity is "weak" [53]. The question is, what constitute "weak" non-linearity in motion planning under uncertainty? Where will it be useful and where will it be damaging to use linearization (and Gaussian) simplifications?

This Chapter presents our work towards answering these questions. We propose a measure of non-linearity for stochastic systems, called *Statistical-distance-based Non-linearity Measure (SNM)*, to help identify the suitability of linearization in motion planning under uncertainty problems. SNM is based on the total variation distance between the original dynamics and sensing models, and their corresponding linearized models. It is general enough to be applied to any type of motion and sensing errors and any linearization technique, regardless of the type of approximation of the true beliefs (*e.g.*, with and without Gaussian simplification). We show that the difference between the value of the optimal strategy generated if we plan using the original model and if we plan using the linearized

model, can be upper-bounded by a function linear in SNM. Furthermore, our experimental results indicate that compared to recent state-of-the-art methods of non-linearity measures for stochastic systems, SNM is more sensitive to the effect that obstacles have on the effectiveness of linearization, which is critical for motion planning.

To further test the applicability of SNM in motion planning, we develop a simple on-line planner that uses a local estimate of SNM to automatically switch between a general planner [12] that uses the original POMDP model and a linearization-based planner (adapted from [28]) that uses the linearized model. Experimental results on a car-like robot with acceleration control, and a 4DOFs and 7DOFs manipulators with torque control indicate that this simple planner can appropriately decide if and when linearization should be used and therefore computes better strategies faster than each of the component planner.

## 2.2 Background

### 2.2.1 Measures of Non-Linearity

While linearization is often being used in solving non-linear control and estimation problems, it is known that linearization performs well only when the system's non-linearity are relatively mild. To identify the effectiveness of linearization in solving non-linear problems, many non-linearity measures have been proposed in the control and information fusion community.

Most non-linearity measures (*e.g.*, [54–56]) have been designed for deterministic systems. For instance, [54] proposed a measure derived from the curvature of the non-linear function. The work in [55, 56] computes the measure based on the distance between the non-linear function and its nearest linearization. A brief survey of non-linearity measures for deterministic systems is available in [53].

It is only recently that non-linearity measures for stochastic systems started to flourish. For instance, [53] extends the measures in [55, 56] to be based on the average distance between the non-linear function that models the motion and sensing of the system, and the set of all possible linearizations of the function.

[57] proposes a different class of measures, which is based on the distance between distribution over states and its Gaussian approximation, called Measure of Non-Gaussianity (MoNG), rather than based on the non-linear function itself. They assume a passive stochastic systems, and compute the negentropy of the non-linear function of the transformed belief — that is, the non-linearity measure of a belief is computed as the negentropy between the subsequent beliefs and their Gaussian approximations. Their results indicate that MoNG is more suitable to measure the non-linearity of stochastic systems, as it takes into account the effect that non-linear transformations have on the shape of the transformed beliefs. This advancement is encouraging and we will use this measure as a comparator for our proposed measure. However, for this purpose, this measure must be modified because our system is not passive and in fact, eventually we would like to have a measure that can be used to decide what strategy to use (*i.e.*, to use a linearization-based or a general solver). The exact modifications we made

can are discussed in Section 2.4.2

Despite the various non-linearity measures that have been proposed, most are not designed to take into account the effect of obstacles to the non-linearity of the robotic system. Except for MoNG, all of the aforementioned non-linearity measures will have difficulties in taking into account the effect of obstacles, even when these effects are embedded in the motion and sensing models. For instance, curvature-based measures requires the non-linear function to be twice continuously differentiable, but the presence of obstacles is very likely to break the differentiability of the motion model. Furthermore, the effect of obstacles is likely to violate the additive Gaussian error, required for instance by [53]. Although MoNG can potentially take into account the effect of obstacles, it is not designed to. The measure is based on the Gaussian approximation to the subsequent belief. In the presence of obstacles this subsequent belief would have support only in the valid region of the state space, and therefore computing the difference between this subsequent belief and its Gaussian approximation is likely to underestimate the effect of obstacles to the effectiveness of linearization. This is exactly the problem we try to alleviate in our proposed non-linearity measure SNM.

Instead of adopting existing approaches for non-linearity measures, SNM adopts the approach commonly used for sensitivity analysis [58, 59] of Markov Decision Processes (MDP) —a special class of POMDP where uncertainty is only in the effect of performing actions. It is based on the statistical distance measure between the true transition dynamics and its perturbed versions. Linearized dynamics can be viewed as a special case of perturbed dynamics, and hence this statistical distance measure can be applied as a non-linearity measure too. We do need to extend these analysis, as they are generally defined for discrete state spaces and are defined with respect to only the transition models (MDP assumes the state of the system is fully observable). Nevertheless, such extensions are feasible and the generality of this measure could help decide which linearization method to use.

## 2.3 Statistical Distance-based Non-linearity Measure (SNM)

Intuitively, our proposed measure SNM is based on the total variation distance between the effect of performing an action and perceiving an observation under the true dynamics and sensing model, and the effect under the linearized dynamic and sensing model. The total variation distance distance $D_{TV}$ between two probability measures $\mu$ and $\nu$ over a measurable space $\Omega$ is defined as $D_{TV}(\mu,\nu) = \sup_{E\in\Omega} |\mu(E) - \nu(E)|$. An alternative expression of $D_{TV}$ is the functional form $D_{TV}(\mu,\nu) = \frac{1}{2}\sup_{|f|\leq 1} |\int f \mathrm{d}\mu - \int f \mathrm{d}\nu|$. Formally, SNM is defined as:

**Definition 1.** *Let $P = \langle S, A, O, T, Z, R, b_0, \gamma\rangle$ be the POMDP model of the system and $\widehat{P} = \langle S, A, O, \widehat{T}, \widehat{Z}, R, b_0, \gamma\rangle$ be a linearization of P, where $\widehat{T}$ is a linearization of the transition function $T$ and $\widehat{Z}$ is a linearization of the observation function Z of P, while all other components of P and $\widehat{P}$ are the same. Then, the SNM (denoted as $\Psi$) between P and $\widehat{P}$ is $\Psi(P,\widehat{P}) = \Psi_T(P,\widehat{P}) + \Psi_Z(P,\widehat{P})$,*

*where*

$$\Psi_T(P,\widehat{P}) = \sup_{s\in S, a\in A} D_{TV}(T(s,a,s'),\widehat{T}(s,a,s')) \tag{2.1}$$

$$\Psi_Z(P,\widehat{P}) = \sup_{s\in S, a\in A} D_{TV}(Z(s,a,o),\widehat{Z}(s,a,o)) \tag{2.2}$$

Note that SNM can be applied as both a global and a local measure. As a local measure, the supremum over the state $s$ can be restricted to a subset of $S$, rather than the entire state space. Also, SNM is general enough for any approximation to the true dynamics and sensing model, which means that it can be applied to any type of linearization and belief approximation techniques, including those that do and do not assume Gaussian belief simplifications.

We want to use SNM to bound the difference between the expected total reward received if the system were to run the optimal policy of the true model $P$ and if it were to run the optimal policy of the linearized model $\widehat{P}$. Note that since our interest is in the actual reward received, the values of these policies are evaluated with respect to the original model $P$ (we assume $P$ is a faithful model of the system). More precisely, we want to show that:

**Theorem 1.** *Given a POMDP $P = \langle S,A,O,T,Z,R,b_0,\gamma\rangle$ and its linearized version $\widehat{P} = \langle S,A,O,\widehat{T},\widehat{Z},R,b_0,\gamma\rangle$, where $\widehat{T}$ is a linearization of $T$, $\widehat{Z}$ a linearization of $Z$. Let $\pi^*$ denote the optimal policy for $P$ and $\widehat{\pi}$ denote the optimal policy for $\widehat{P}$, then,*

$$V_{\pi^*}(b) - V_{\widehat{\pi}}(b) \leq 4\gamma\frac{R_{max}}{(1-\gamma)^2}\Psi(P,\widehat{P}) \tag{2.3}$$

*where*

*$V_\pi(b) = R(b,\pi(b)) + \gamma\int_{o\in O} Z(b,a,o)V_\pi(\tau(b,a,o))\mathrm{d}o$ and $\tau(b,a,o)$ is the belief transition function as defined in eq.(1.2)*

*Proof:* To proof Theorem 1, we first assume, without loss of generality, that a policy is represented by a set of $\alpha$-functions $\Gamma$. Each $\alpha$-function corresponds to a policy tree $T_\pi$. Let us denote $\alpha_\pi$ the $\alpha$-function that corresponds to policy $\pi$. The value $\alpha_\pi(s)$ is the expected total reward the robot receives when executing $T_\pi$ from$s$. The nodes of a policy tree represent actions whereas the edges represent observations. Assume the robot is at state $s$. Executing $T_\pi$ from $s$ means that the action that corresponds to the root of the tree is executed. After the robot receives an observation the tree is traversed along the edge that represents the received observation. Then the action corresponding to the node that is connected to the root via the received observation is executed. This process is repeated until a leaf node is reached. The value $\alpha_\pi(s)$ can be written as

$$\alpha_\pi(s) = R(s,a_0) + \gamma\int_{s'\in S}\int_{o\in O} T(s,a_0,s')Z(s',a_0,o)\alpha_\pi^o(s')\mathrm{d}o\mathrm{d}s' \tag{2.4}$$

where $a_0$ is the immediate action of $\pi$ from $s'$ and $\alpha_\pi^o$ is the alpha function that corresponds to the subtree of $T_\pi$ whose root is the child of $a_0$ via $o$. Given an $\alpha_\pi$-function that corresponds to policy $\pi$,

the policy-value function of belief $b$ (that is, the expected value of executing $\pi$ starting from $b$) can be written as

$$V_\pi(b) = \int_{s' \in S} b(s)\alpha_\pi(s)\mathrm{d}s \tag{2.5}$$

The optimal policy-value function (which we know is equal to the optimal value function) starting from belief $b$ is then

$$V_{\pi^*}(b) = \max_{\alpha_\pi \in \Gamma} \int_{s' \in S} b(s)\alpha_\pi(s)\mathrm{d}s \tag{2.6}$$

Looking at 2.4, we see that $\alpha_\pi$ is defined with respect to the original transition and observation functions. Using the linearized transition and observation functions, we can define the linearized $\alpha$-function with corresponding policy $\pi$, denoted as $\widehat{\alpha}_\pi$:

$$\widehat{\alpha}_\pi(s) = R(s,a_0) + \gamma \int_{s' \in S} \int_{o \in O} \widehat{T}(s,a_0,s')\widehat{Z}(s',a_0,o)\widehat{\alpha}^o_\pi(s')\mathrm{d}o\mathrm{d}s' \tag{2.7}$$

Consider now a POMDP solver that computes a policy $\widehat{\pi}$ that is optimal for POMDP $\widehat{P}$. For any $s \in S$ we have that $\alpha_{\widehat{\pi}}(s) \geq \widehat{\alpha}_{\widehat{\pi}}(s) - |\alpha_{\widehat{\pi}}(s) - \widehat{\alpha}_{\widehat{\pi}}(s)|$ and $\widehat{\alpha}_{\pi^*}(s) \geq \alpha_{\pi^*}(s) - |\alpha_{\pi^*}(s) - \widehat{\alpha}_{\pi^*}(s)|$. Therefore

$$\int_{s' \in S} b(s)\alpha_{\widehat{\pi}}(s)ds \geq \int_{s' \in S} b(s)\widehat{\alpha}_{\widehat{\pi}}(s)\mathrm{d}s - |\alpha_{\widehat{\pi}}(s) - \widehat{\alpha}_{\widehat{\pi}}(s)| \tag{2.8}$$

and

$$\int_{s' \in S} b(s)\widehat{\alpha}_{\pi^*}(s)\mathrm{d}s \geq \int_{s' \in S} b(s)\alpha_{\pi^*}(s)\mathrm{d}s - |\alpha_{\pi^*}(s) - \widehat{\alpha}_{\pi^*}(s)| \tag{2.9}$$

Since $\widehat{\pi}$ is the optimal policy for $\widehat{P}$, we also know that

$$\int_{s' \in S} b(s)\widehat{\alpha}_{\widehat{\pi}}(s)\mathrm{d}s \geq \int_{s' \in S} b(s)\widehat{\alpha}_{\pi^*}(s)\mathrm{d}s \tag{2.10}$$

From 2.8, 2.9 and 2.10 it immediately follows that

$$\begin{aligned} \int_{s' \in S} b(s)\alpha_{\widehat{\pi}}(s)\mathrm{d}s \geq \int_{s' \in S} b(s)\alpha_{\pi^*}(s)\mathrm{d}s' - 2\sup_\pi |\alpha_\pi(s) - \widehat{\alpha}_\pi(s)| \\ V_{\widehat{\pi}}(b) \geq V_{\pi^*}(b) - 2\sup_\pi |\alpha_\pi(s) - \widehat{\alpha}_\pi(s)| \end{aligned} \tag{2.11}$$

Before we continue, we first have to show the following Lemma:

**Lemma 1.** *Let $R_m = \max\{|R_{min}|, R_{max}\}$. For any policy $\pi$ and any $s \in S$, the absolute difference between the original and linearized $\alpha$-functions is upper bounded by*

$$|\alpha_\pi(s) - \widehat{\alpha}_\pi(s)| \leq 2\gamma \frac{R_m}{(1-\gamma)^2} \Psi(P, \widehat{P}) \tag{2.12}$$

The proof of Lemma 1 is presented in Section 2.6.1.

Using the result of Lemma 1, we can now conclude the proof for Theorem 1. Substituting the upper bound derived in Lemma 1 into eq.(2.11) and re-arranging the terms gives us

$$V_{\pi^*}(b) - V_{\widehat{\pi}}(b) \leq 4\gamma \frac{R_m}{(1-\gamma)^2} \Psi(P, \widehat{P}) \tag{2.13}$$

which is what we are looking for. □

## 2.4 SNM-Planner: An Application of SNM for Planning

SNM-Planner is an on-line planner that uses SNM as a heuristic to decide at each planning step whether a general or a linearization-based solver should be used for the policy computation. The general solver we use is Adaptive Belief Tree (ABT) [12], while the linearization-based solver is Modified High Frequency Replanning (MHFR), which is an adaptation of HFR [28]. HFR is designed for chance-constraint POMDPs, *i.e.*, it explicitly minimizes the collision probability, while MHFR is a POMDP solver where the objective is to maximize the expected total reward. At each planning step SNM-Planner approximates the local value of SNM around the current belief $b$. This value and a given threshold will then be used to decide whether to use ABT or MHFR to compute the policy from $b$. An overview of SNM-Planner is presented in Algorithm 1. In the following two subsections we provide a brief overview of the two component planners ABT and MHFR.

### 2.4.1 Adaptive Belief Tree (ABT)

ABT is an Monte-Carlo-Tree-Search based on-line and anytime general POMDP solver that updates (rather than recomputes) its policy at each planning step. Given the current belief $b_t$, ABT constructs and maintains a belief-tree by sampling sequences of state-–action-–observation-–reward quadruples, called episodes, starting from $b_t$ using a generative model. Each node in the belief tree represents a belief, while an edge from belief $b$ to $b'$ means that there is an action $a \in A$ and an observation $o \in O$, such that $b' = \tau(b,a,o)$. ABT approximates beliefs by sets of particles. After performing an action and receiving an observation, the current belief is updated using a particle filter. Details of the method can be found in [12].

### 2.4.2 Modified High-Frequency Replanning (MHFR)

The main difference between HFR and MHFR is that HFR is designed for chance-constrained POMDPs, i.e., it explicitly minimizes the collision probability, while MHFR is a POMDP solver, whose objective is to maximize the expected total reward. Similar to HFR, MHFR approximates the current belief $b_t$ by a multivariate Gaussian distribution $b_t = N(\bar{s}_t, \Sigma_t)$ where $\bar{s}$ is the mean and $\Sigma_t$ the covariance matrix of the distribution. To find a policy, MHFR computes, at each planning step, a set of trajectories from $\bar{s}_t$ to a goal state using multiple instances of RRTs [60]. It then computes the expected total discounted reward of each trajectory by tracking the beliefs around the trajectory using a Kalman Filter [61], assuming maximum-likelihood observations. The policy then becomes the first action of the trajectory with the largest expected total discounted reward. After executing this action and perceiving and observation, the belief is updated using an Extended Kalman Filter [62]. The process then repeats from the updated belief. To increase efficiency, MHFR additionally adjusts the previous trajectory to start from the mean of the updated belief using LQG and adds the adjusted trajectory to the set of trajectories. More details and a precise derivation of the method are available in [28].

**Algorithm 1** SNM-Planner (POMDP $P$, initial belief $b_0$, SNM threshold $\mu$, planning time per step $t_{plan}$, max num. planning steps $t_{max}$)

```
1: InitializeABT(P)
2: InitializeMHFR(P)
3: t = 0
4: b = b_0
5: while terminal == False and t < t_max do
6:     Ψ = approximatePsi(b)
7:     t_p = t_plan − t_m                ▷ t_m is the time that was required to approximate SNM
8:     if Ψ < μ then
9:         a = MHFR(P̂, b, t_p)
10:    else
11:        a = ABT(P, b, t_p)
12:    end if
13:    terminal = Execute a
14:    o = get observation
15:    b = τ(b, a, o)                    ▷ We use Sequential Importance Resampling [63]
16:    t = t + 1
17: end while
```

### 2.4.3 Approximating SNM

From the definition of SNM, we can see that approximating SNM is equivalent to solving an optimization problem over a set of states, subsequent states, actions and observations. For systems with simple transition and observation dynamics and small numbers of degrees-of-freedom, this can in principle be done on-line in a tractable manner. However, for more complex systems, computing SNM on-line quickly becomes infeasible. We therefore resort to approximating SNM off-line and utilize the results during run-time. Here we will only discuss how to approximate the transition component $\Psi_T$ of SNM, however, the same method applies to the observation component $\Psi_Z$.

Let us first rewrite the transition component of $\Psi_T$ as

$$\Psi_T = \sup_{s \in S} \Psi_T(s) = \sup_{s \in S} \sup_{a \in A} D_{TV}(T(s,a,s'), \widehat{T}(s,a,s')) \tag{2.14}$$

where $\Psi_T(s)$ is simply the transition component of SNM, given a particular state. Recall from Section 2.3 that we can use SNM as a local measure by restricting the supremum in eq.(2.14) to a subset of the state space. This allows us to replace $S$ in eq.(2.14) by a sampled representation of $S$, which we denote as $\tilde{S}$, evaluate the term $\Psi_T(s)$ off-line for each state in $\tilde{S}$, and save the results in a lookup-table. This lookup-table can then be used during run-time to get a local approximation of $\Psi_T$ around the current belief.

The first question is, *how do we efficiently sample the state space*? A naive approach would be to use uniform sampling. However, for large state spaces this would be wasteful, since a large amount of samples would be required to sufficiently cover the entire state space. Furthermore, for motion planning problems, large portions of the state space are often irrelevant since they are either disconnected, or unlikely to be traversed by the robot during run-time. A better strategy is to consider only the subset of the state space that is reachable from the support set of the initial belief under any policy, denoted as

$S_{b_0}$. To sample from $S_{b_0}$, we use a simple but effective method: Assuming deterministic dynamics, we solve the motion planning problem off-line using kinodynamic RRTs and use the nodes in the trees as a sampled representation of $S_{b_0}$. In principle any deterministic sampling-based motion planner can be used to generate samples from $S_{b_0}$, however, in our case RRT is a particularly suitable choice due to its space-filling property [64]. On the other hand, it is known that the exploration property of RRT is dependent on the underlying state space metric [65]. For problems where an appropriate metric is not available, various extensions to RRT have been proposed [65–67] that improve the exploration property of RRT when a suitable metric is unavailable. Additionally, sampling-based methods with optimality guarantees (*e.g.* [68, 69]) could be applied to generate samples that are likely within the subset of the state space reachable under a near-optimal policy. Furthermore, note that any of the above methods generates states according to a deterministic transition function only. If required, one could also generate additional samples according to the actual stochastic transition dynamics of the robot. However in our experiments we found that the simplicity of RRT outweighed the benefits of these possible extensions and the generated state samples constituted a sufficient approximation of $S_{b_0}$.

After we have generated a pre-defined number of samples from $S_{b_0}$, another difficulty in approximating $\Psi_T(s)$ is the computation of the supremum over the action space. Similarly to restricting the approximation to a discrete set of states, we can impose a discretization on the action space - the same discretization used by ABT- which leaves us with a maximization over discrete actions, denoted as $\tilde{A}$. Using the set $\tilde{A}$, we can then approximate eq.(2.14) for each state in $S_{b_0}$. This is done as follows: Given a particular state $s \in S_{b_0}$ and action $a \in \tilde{A}$, we draw $n$ samples of subsequent states according to the original and linearized transition function, and construct a multidimensional histogram from both sample sets. In other words, we discretize the distributions that follow from the original and linearized transition function, given a particular state and action. Suppose the histogram consists of $k$ bins. The value $\Psi_T(s,a)$ is then approximated as

$$\Psi_T(s,a) \approx \frac{1}{2} \sum_{i=1}^{k} |n_i - \widehat{n}_i| \tag{2.15}$$

where $n_i$ is the normalized number of states inside bin $i$ sampled from the original transition function, and $\widehat{n}_i$ the normalized number of states inside bin $i$ sampled from the linearized transition function. The right-hand side of eq.(2.15) is simply the definition of the total variation distance between two discrete distributions.

By repeating this process for each action in $\tilde{A}$ and taking the maximum, we end up with an approximation of $\Psi_T(s)$. This procedure is repeated for every state in the set $S_{b_0}$. As a result we get a lookup-table, assigning each state in $S_{b_0}$ an approximated value of $\Psi_T(s)$ that is used during run-time to approximate the transition component of SNM around the support set of the current belief. Recall that beliefs in SNM-Planner are represented by sets of particles, that are updated after the robot performs and action and receives and observation. Hence for the set of particles that represent the current belief we can extract an approximation of the transition component SNM using the lookup-table that we calculated off-line. We do this by iterating over the belief particles, finding the nearest neighbor $s_{near} \in S_{b_0}$ for each particle, and assigning the pre-computed $\Psi_T(s_{near})$ to the particle. Taking the

maximum of the $\Psi_T(s_{near})$ values gives us an approximation of the transition component of SNM with respect to the support set of the belief.

Clearly this approximation method assumes that states that are close together (close according to some distance metric in $S$) should yield similar values for SNM. At first glance this is a very strong assumption. In the vicinity of obstacles or constraints, states that are close together could potentially yield very different SNM values. However, we will now show that under mild assumptions, pairs of states that are elements within certain subsets of the state space indeed yield similar SNM values.

Consider a partitioning of the state space into a finite number of local-Lipschitz subsets $S_i$ that are defined as follows:

**Definition 2.** *Let $S$ be a metric space with distance metric $D_S$. $S_i$ is called a local-Lipschitz subset of $S$ if for any $s_1, s_2 \in S_i$, any $s' \in S$ and any $a \in A$ : $|T(s_1,a,s') - T(s_2,a,s')| \leq C_{T_i} D_S(s_1,s_2)$ and $\left|\widehat{T}(s_1,a,s') - \widehat{T}(s_2,a,s')\right| \leq C_{\widehat{T}_i} D_S(s_1,s_2)$, where $C_{T_i} \geq 0$ and $C_{\widehat{T}_i} \geq 0$ are finite constants*

In other words, $S_i$ are subsets of $S$ in which the original and linearized transition functions are Lipschitz continuous with Lipschitz constants $C_{T_i}$ and $C_{\widehat{T}_i}$. This definition essentially says that applying an action to states in $S_i$ that are close together yields similar subsequent states under both the original and the linearized transition function. With this definition at hand, we can now show the following lemma:

**Lemma 2.** *Let $S$ be a $n-$dimensional metric space with distance metric $D_S$ and assume $S$ is normalized to $[0,1]^n$. Furthermore let $S_i$ be a local-Lipschitz subset of $S$, then*

$$|\Psi_T(s_1) - \Psi_T(s_2)| \leq \frac{1}{2} D_S(s_1,s_2) \left[C_{T_i} + C_{\widehat{T}_i}\right] \tag{2.16}$$

*for any $s_1, s_2 \in S_i$*

The proof for this Lemma is presented in Section 2.6.2. This Lemma says that the difference between the SNM values for two states from the same local-Lipschitz subset $S_i$ depends only on the distance $D_S$ between them, since $C_{T_i}$ and $C_{\widehat{T}_i}$ are constant for each subset $S_i$, and $n$ is constant for a particular POMDP problem. Thus, as the distance between two states converges towards zero, the SNM value difference converges towards zero as well. This implies that if we approximate SNM for a sufficient number of samples from $S_{b_0}$, we can re-use these approximated values on-line with a small error, without requiring an explicit representation of the $S_i$ subsets, or even having complete knowledge of the underlying state space metric.

Note that we assume that the Lipschitz constants $C_{T_i}$ and $C_{\widehat{T}_i}$ for each subset $S_i$ are finite. This is not necessarily always the case. Consider the special case of a robot with deterministic dynamics for which the transition function is the delta function. In this case the two Lipschitz constants are infinite for any subset of $S$. However, we argue that for most robotic systems that are subject to uncertainty the assumption of finite Lipschitz constants holds.

## 2.5 Experiments and Results

### 2.5.1 Experimental setup

Our experiment is two-fold: To test the applicability SNM and to test the planner as proposed in Section 2.4. For our first objective, we compare SNM with a modified version of the Measure of Non-Gaussianity (MoNG) [57]. Details on this measure are in Section 2.5.2. We use ABT as the general POMDP solver and MHFR as the linearization-based POMDP solver.

All algorithms are implemented in C++, while all experiments are conducted on Intel Xeon E5-2650 CPUs with 16GB RAM. For the parallel construction of the RRTs in MHFR, we utilize 8 CPU cores throughout the experiments. All parameters are set based on preliminary runs over the possible parameter space, the parameters that generate the best results are then chosen to generate the experimental results. For the comparison between SNM and MoNG, we use a car-like robot with $2^{nd}$ order dynamics and a 4DOFs-manipulator with torque control operating inside various environments with increasing uncertainty and clutterness. To test SNM-Planner, we additionally use a scenario where a torque-controlled 7DOFs Kuka iiwa manipulator operates inside an office environment.

### 2.5.2 Measure of Non-Gaussianity

The Measure of Non-Gaussianity (MoNG) proposed in [57] is based on the negentropy between the PDF of a random variable and its Gaussian approximation. Consider a $n$-dimensional random variable $X$ distributed according to PDF $p(x)$. Furthermore, let $\widehat{X}$ be a Gaussian approximation of $X$ with PDF $\widehat{p}$ (x), such that $\widehat{X} \sim N(\mu, \Sigma_x)$, where $\mu$ and $\Sigma_x$ are the first two moments of $p(x)$. The negentropy between $p$ and $\widehat{p}$ (denoted as $J(p, \widehat{p})$) is then defined as

$$J(p, \widehat{p}) = H(\widehat{p}) - H(p) \tag{2.17}$$

where

$$\begin{aligned} H(\widehat{p}) &= \frac{1}{2} ln\left[(2\pi e)^n \left|det(\Sigma_x)\right|\right] \\ H(p) &= -\int p(x) \ln p(x) dx \end{aligned} \tag{2.18}$$

are the differential entropies of $p$ and $\widehat{p}$ respectively. A (multivariate) normal distribution has the largest differential entropy amongst all distributions with equal first two moments, therefore $J(p, \widehat{p})$ is always non-negative. In practice, since the PDF $p(x)$ is not known exactly in all but the simplest cases, $H(p)$ has to be approximated.

In [57] this measure has originally been used to assess the non-linearity of passive systems. Therefore, in order to achieve comparability with SNM, we need to extend the Non-Gaussian measure to general active stochastic systems of the form $s_{t+1} = f(s_t, a_t, v_t)$. We do this by evaluating the non-Gaussianity of distribution that follows from the transition function $T(s, a, s')$ given state $s$ and

action $a$. In particular for a given $s$ and $a$, we can find a Gaussian approximation of $T(s,a,s')$ (denoted by $T_G(s,a,s')$) by calculating the first two moments of the distribution that follows from $T(s,a,s')$.

Using this Gaussian approximation, we define the Measure of Non-Gaussianity as

$$MoNG(T,T_G) = \sup_{s\in S, a\in A} \left[H(T_G(s,a,s')) - H(T(s,a,s'))\right] \tag{2.19}$$

Similarly we can compute the Measure of Non-Gaussianity for the observation function:

$$MoNG(Z,Z_G) = \sup_{s\in S, a\in A} \left[H(Z_G(s,a,o)) - H(Z(s,a,o))\right] \tag{2.20}$$

where $Z_G$ is a Gaussian approximation of $Z$.

In order to approximate the entropies $H(T(s,a,s')))$ and $H(Z(s,a,o))$, we are using a similar histogram-based approach as discussed in Section 2.4.3. The entropy terms for the Gaussian approximations can be computed in closed form, according to the first equation in eq.(2.18) [70].

### 2.5.3 Robot Models

#### 4DOFs-Manipulator

The 4DOFs-manipulator consists of 4 links connected by 4 torque-controlled revolute joints. The first joint is connected to a static base. In all problem scenarios the manipulator must move from a known initial state to a state where the end-effector lies inside a goal region located in the workspace of the robot, while avoiding collisions with obstacles the environment is populated with.

The state of the manipulator is defined as $s = (\theta, \dot{\theta}) \in \mathbb{R}^8$, where $\theta$ is the vector of joint angles, and $\dot{\theta}$ the vector of joint velocities. Both joint angles and joint velocities are subject to linear constraints: The joint angles are constrained by $(-3.14, 3.14)rad$, whereas the joint velocities are constrained by $(6,2,2,2)rad/s$ in each direction. Each link of the robot has a mass of $1kg$.

The control inputs of the manipulator are the joint torques, where the maximum joint torques are $(20,20,10,5)Nm/s$ in each direction. Since ABT assumes a discrete action space, we discretize the joint torques for each joint using the maximum torque in each direction, which leads to 16 actions.

The dynamics of the manipulator is defined using the well-known Newton-Euler formalism [71]. For both manipulators we assume that the input torque for each joint is affected by zero-mean additive Gaussian noise. Note however, even though the error is Gaussian, due to the non-linearities of the motion dynamics the beliefs will not be Gaussian in general. Since the transition dynamics for this robot are quite complex, we assume that the joint torques are applied for 0.1s and we use the ODE physics engine [72] for the numerical integration of the dynamics, where the discretization (*i.e.*$\delta t$) of the integrator is set to $\delta t = 0.004s$.

The robot is equipped with two sensors: The first sensor measures the position of the end-effector inside the robot's workspace, whereas the second sensor measures the joint velocities. Consider a function $g : \mathbb{R}^8 \mapsto \mathbb{R}^3$ that maps the state of the robot to an end-effector position inside the workspace, then the observation model is defined as

$$o = [g(s), \dot{\theta}]^T + w \tag{2.21}$$

where $w_t$ is an error term drawn from a zero-mean multivariate Gaussian distribution with covariance matrix $\Sigma_w$.

The initial state of the robot is a state where the joint angles and velocities are zero.

When the robot performs an action where it collides with an obstacle it enters a terminal state and receives a penalty of -500. When it reaches the goal area it also enters a terminal state, but receives a reward of 1,000. To encourage the robot to reach the goal area quickly, it receives a small penalty of -1 for every other action.

### 7DOFs Kuka iiwa manipulator

The 7DOFs Kuka iiwa manipulator is very similar to the 4DOFs-manipulator. However, the robot consists of 7 links connected via 7 revolute joints. We set the POMDP model to be similar to that of the 4DOFs-manipulator, but expand it to handle 7DOFs. For this robot, the joint velocities are constrained by $(3.92, 2.91, 2.53, 2.23, 2.23, 2.23, 1.0) rad/s$ in each direction and the link masses are $(4, 4, 3, 2.7, 1.7, 1.8, 0.3) kg$. Additionally, the torque limits of the joints are $(25, 20, 10, 10, 5, 5, 0.5) Nm/s$ in each direction. For ABT we use the same discretization of the joint torques as in the 4DOFs-manipulator case, *i.e.*we use the maximum torque per joint in each direction, resulting in 128 actions. Similarly to the 4DOFs-manipulator, we assume that the input torques are applied for 0.1s and we use the ODE physics engine with an integration step size of 0.004s to simulate the transition dynamics. The observation and reward models are the same as for the 4DOFs-manipulator. The initial joint velocities are all zero and almost all joint angles are zero too, except for the second joint, for which the initial joint angle is $-1.5 rad$. Figure 2.1(c) shows the Kuka manipulator operating inside an office scenario.

### Car-like robot

A nonholonomic car-like robot of size $(0.12 \times 0.07 \times 0.01)$ drives on a flat xy-plane inside a 3D environment populated by obstacles The robot must drive from a known start state to a position inside a goal region without colliding with any of the obstacles. The state of the robot at time t is defined as a 4D vector $s_t = (x_t, y_t, \theta_t, \upsilon_t) \in \mathbb{R}^4$, where $x_t, y_t \in [-1, 1]$ is the position of the center of the robot on the $xy$-plane, $\theta_t \in [-3.14, 3.14] rad$ the orientation and $\upsilon_t \in [-0.2, 0.2]$ is the linear velocity of the robot. The initial state of the robot is $(-0.7, -0.7, 1.57 rad, 0)$ while the goal region is centered at $(0.7, 0.7)$ with radius 0.1. The control input at time $t$, $a_t = (\alpha_t, \phi_t)$ is a 2D real vector consisting of the acceleration $\alpha \in [-1, 1]$ and the steering wheel angle $\phi_t \in [-1 rad, 1 rad]$. The robot's dynamics is subject to control noise $v_t = (\tilde{\alpha}_t, \tilde{\phi}_t) \sim N(0, \Sigma_v)$. The robot's transition model is

$$s_{t+1} = f(s_t, a_t, v_t) = \begin{bmatrix} x_t + \Delta t \, \upsilon_t \cos\theta_t \\ y_t + \Delta t \, \upsilon_t \sin\theta_t \\ \theta_t + \Delta t \tan(\phi_t + \tilde{\phi}_t)/0.11 \\ \upsilon_t + \Delta t (\alpha_t + \tilde{\alpha}_t) \end{bmatrix} \tag{2.22}$$

where $\Delta t = 0.3 s$ is the duration of a timestep and the value 0.11 is the distance between the front and rear axles of the wheels.

This robot is equipped with two types of sensors, a localization sensor that receives a signal from two beacons that are located at $(\hat{x}_1, \hat{y}_1)$ and $(\hat{x}_2, \hat{y}_2)$. The second sensor is a velocity sensor mounted on the car. With these two sensors the observation model is defined as

$$o_t = \begin{bmatrix} \frac{1}{((x_t-\hat{x}_1)^2+(y_t-\hat{y}_1)^2+1)} \\ \frac{1}{((x_t-\hat{x}_2)^2+(y_t-\hat{y}_2)^2+1)} \\ v_t \end{bmatrix} + w_t \tag{2.23}$$

where $w_t$ is an error vector drawn from a zero-mean multivariate Gaussian distribution with covariance matrix $\Sigma_w$.

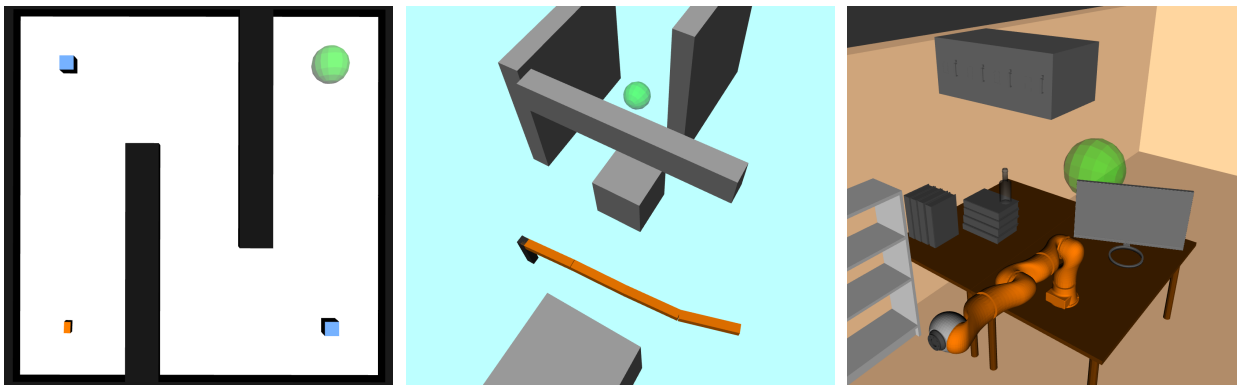
Similar to the manipulators described above, the robot receives a penalty of -500 when it collides with an obstacle, a reward of 1,000 when reaching the goal area and a small penalty of -1 for any other action.

### 2.5.4 Testing SNM

In this set of experiments we want to understand the performance of SNM compared to existing non-linearity measures for stochastic systems in various scenarios. In particular, we are interested in the effect of increasing uncertainties and the effect that obstacles have on the effectiveness of SNM, and if these results are consistent with the performance of a general solver relative to a linearization-based solver. Additionally, we want to see how different observation models – one with additive Gaussian noise and non-additive Gaussian noise – affect our measure. For the experiments with increasing motion and sensing errors, recall from Section 2.5.1 that the control errors are drawn from zero-mean multivariate Gaussian distributions with covariance matrices $\Sigma_v$. We then define control error (denoted as $e_T$) to be the standard deviation of the Gaussian distributions, such that $\Sigma_v = e_T^2 \times \mathbb{1}$. Similarly for the covariance matrices of the zero-mean multivariate Gaussian sensing errors, we define the observation error as $e_Z$, such that $\Sigma_w = e_Z^2 \times \mathbb{1}$. Note that during all the experiments, we use normalized spaces, which means that the error vectors affect the normalized action and observation vectors. For SNM and MoNG we first generated 100,000 state samples for each scenario, and computed a lookup table for each error value off-line, as discussed in Section 2.4.3. Then, during run-time we calculated the average approximated SNM and MonG values.

#### Effects of increasing uncertainties in cluttered environments

To investigate the effect of increasing control and observation errors to SNM, MoNG and the two solvers ABT and MHFR in cluttered environments, we ran a set of experiments where the 4DOFs-manipulator and the car-like robot operate in empty environments and environments with obstacles, with increasing values of $e_T$ and $e_Z$, ranging between 0.001 and 0.075. The environments with obstacles are the Factory and Maze environments shown in Figure 2.1(a) and (b). For each scenario and each control-sensing error value (we set $e_T = e_Z$), we ran 100 simulation runs using ABT and MHFR respectively with a planning time of 2s per step.



(a) Maze (b) Factory (c) KukaOffice

Figure 2.1: Test scenarios for the different robots. The objects colored black and gray are obstacles, while the green sphere is the goal region. (a) The Maze scenario for the car-like robot. The blue squares represents the beacons, while the orange square at the bottom left represents the initial state. (b) The 4DOFs-manipulator scenario. (c) The KukaOffice scenario

| **(a) Empty environment 4DOFs-manipulator** | | | |
|---|---|---|---|
| $e_T = e_Z$ | **SNM** | **MoNG** | $\frac{\mathbf{V_{ABT}(b_0) - V_{MHFR}(b_0)}}{\mathbf{V_{ABT}(b_0)}}$ |
| 0.001 | 0.207 | 0.548 | 0.0110 |
| 0.0195 | 0.213 | 0.557 | 0.0346 |
| 0.038 | 0.243 | 0.603 | 0.0385 |
| 0.057 | 0.254 | 0.617 | 0.0437 |
| 0.075 | 0.313 | 0.686 | 0.0470 |

| **(b) Empty environment Car-like robot** | | | |
|---|---|---|---|
| $e_T = e_Z$ | **SNM** | **MoNG** | $\frac{\mathbf{V_{ABT}(b_0) - V_{MHFR}(b_0)}}{\mathbf{V_{ABT}(b_0)}}$ |
| 0.001 | 0.169 | 0.473 | 0.1426 |
| 0.0195 | 0.213 | 0.479 | 0.1793 |
| 0.038 | 0.295 | 0.458 | 0.1747 |
| 0.057 | 0.350 | 0.476 | 0.1839 |
| 0.075 | 0.395 | 0.446 | 0.2641 |

Table 2.1: Average values of SNM, MonG and the relative value difference between ABT and MHFR for the 4DOFs-manipulator (a) and the car-like robot (b) operating inside empty environments.

The average values for SNM and MoNG and the relative value differences between ABT and MHFR in the empty environments are presented in Table 2.1. The results show that for both scenarios SNM and MoNG are sensitive to increasing transition and observation errors. This resonates well with the relative value difference between ABT ant MHFR. The more interesting question is now, how sensitive are both measures to obstacles in the environment? Table 2.2(a) and (b) shows the results for the Factory and the Maze scenario respectively. It is evident that SNM increases significantly compared to the empty environments, whereas MoNG is almost unaffected. Overall obstacles increase the relative value difference between ABT and MHFR, except for large uncertainties in the Maze scenario. This indicates that MHFR suffers more from the additional non-linearities that obstacles introduce. SNM is able to capture these effects well.

An interesting remark regarding the results for the Maze scenario in Table 2.2(b) is that the relative value difference actually decreases for large uncertainties. The reason for this can be seen in Figure 2.2. As the uncertainties increase, the problem becomes so difficult, such that both solvers fail to compute a reasonable policy within the given planning time. However, clearly MHFR suffers earlier from these large uncertainties compared to ABT.

| (a) Factory environment | | | |
|---|---|---|---|
| $e_T = e_Z$ | SNM | MoNG | $\frac{V_{ABT}(b_0)-V_{MHFR}(b_0)}{V_{ABT}(b_0)}$ |
| 0.001 | 0.293 | 0.539 | 0.0892 |
| 0.0195 | 0.351 | 0.567 | 0.1801 |
| 0.038 | 0.470 | 0.621 | 0.5818 |
| 0.057 | 0.502 | 0.637 | 0.7161 |
| 0.075 | 0.602 | 0.641 | 1.4286 |

| (b) Maze environment | | | |
|---|---|---|---|
| $e_T = e_Z$ | SNM | MoNG | $\frac{V_{ABT}(b_0)-V_{MHFR}(b_0)}{V_{ABT}(b_0)}$ |
| 0.001 | 0.215 | 0.482 | 0.2293 |
| 0.0195 | 0.343 | 0.483 | 1.4473 |
| 0.038 | 0.470 | 0.491 | 1.1686 |
| 0.057 | 0.481 | 0.497 | 0.0985 |
| 0.075 | 0.555 | 0.502 | 0.0040 |

Table 2.2: Average values of SNM, MonG and the relative value difference between ABT and MHFR for the 4DOFs-manipulator operating inside the Factory environment (a) and the car-like robot operating inside the Maze environment (b).

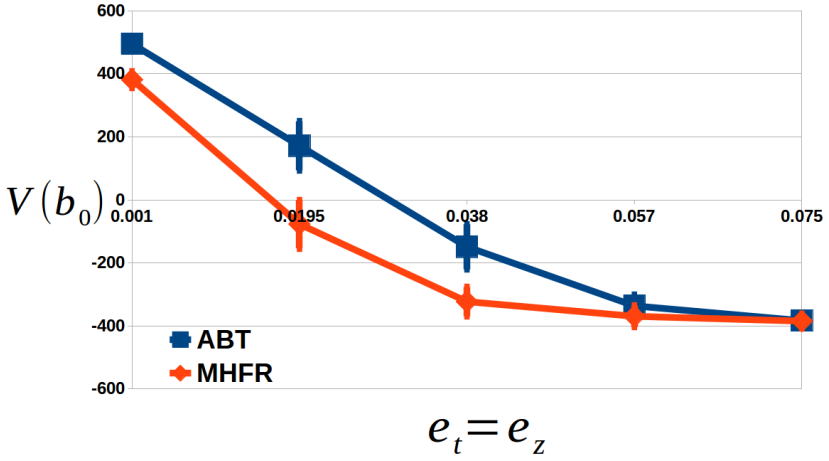



Figure 2.2: The average total discounted rewards achieved by ABT and MHFR in the Maze scenario, as the uncertainties increase. Vertical bars are the 95% confidence intervals.

### Effects of collision dynamics

In a second set of experiments we investigated the effects of collision dynamics to SNM and MoNG. For this, the robots are allowed to collide with the obstacles. In other words, colliding states are not terminal and the dynamic effects of collisions are reflected in the transition model. For the 4DOFs-manipulator these collisions are modeled as additional constraints (contact points) that are resolved by applying "correcting velocities" to the colliding bodies in the opposite direction of the contact normals.

For the Car-like robot, we modify the transition model eq.(2.22) to consider collision dynamics such that

$$s_{t+1} = \begin{cases} f_{col}(s_t, a_t, v_t) & \text{if } f(s_t, a_t, v_t) \text{ collides} \\ f(s_t, a_t, v_t) & \text{else} \end{cases} \tag{2.24}$$

where

$$f_{coll}(s_t, a_t, v_t) = [x_t, y_t, \theta_t, -3v_t]^T \tag{2.25}$$

This transition function causes the robot to slightly "bounce" off obstacles upon collision. There are two interesting remarks regarding this transition function: The first one is that eq.(2.25) is a deterministic. In other words, a collision causes an immediate reduction of the uncertainty regarding

| (a) Maze environment with collision dynamics | | | |
|---|---|---|---|
| $e_T = e_Z$ | **SNM** | **MoNG** | $\frac{\mathbf{V_{ABT}(b_0)-V_{MHFR}(b_0)}}{\mathbf{V_{ABT}(b_0)}}$ |
| 0.001 | 0.425 | 0.490 | 0.3807 |
| 0.0195 | 0.576 | 0.505 | 7.0765 |
| 0.038 | 0.636 | 0.542 | 8.6847 |
| 0.057 | 0.740 | 0.569 | 2.0194 |
| 0.075 | 0.776 | 0.611 | 1.7971 |

| (b) Factory environment with collision dynamics | | | |
|---|---|---|---|
| $e_T = e_Z$ | **SNM** | **MoNG** | $\frac{\mathbf{V_{ABT}(b_0)-V_{MHFR}(b_0)}}{\mathbf{V_{ABT}(b_0)}}$ |
| 0.001 | 0.492 | 0.639 | 0.07141 |
| 0.0195 | 0.621 | 0.621 | 0.4007 |
| 0.038 | 0.725 | 0.738 | 0.6699 |
| 0.057 | 0.829 | 0.742 | 1.0990 |
| 0.075 | 0.889 | 0.798 | 1.7100 |

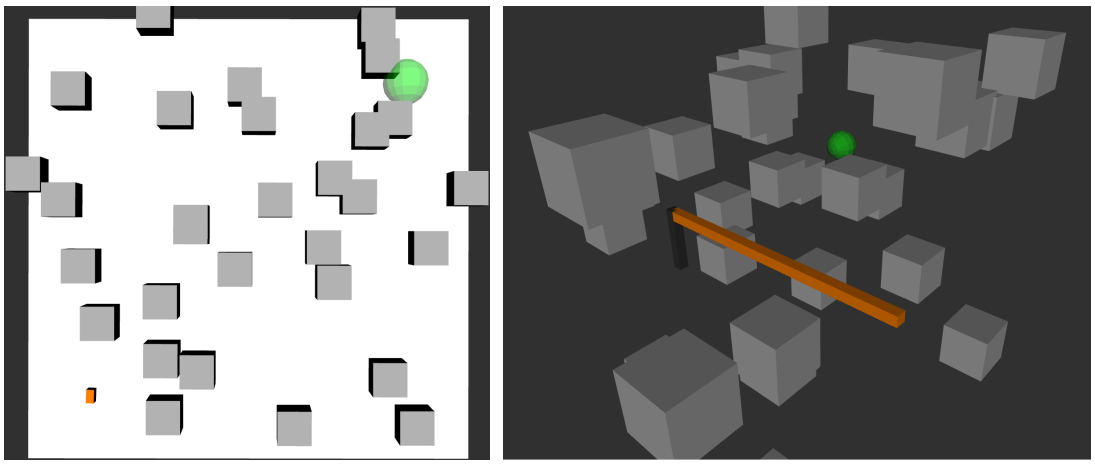
Table 2.3: Average values of SNM, MonG and the relative value difference between ABT and MHFR for the 4DOFs-manipulator operating inside the Factory environment (a) and the car-like robot operating inside the Maze environment (b) while being subject to collision dynamics.

the state of the robot. Second, while the collision effects eq.(2.25) are linear, eq.(2.24) is not smooth since the collision dynamics induce discontinuities when the robot operates in the vicinity of obstacles.

Intuitively, collision dynamics are highly non-linear effects. Here we investigate if SNM is capable of capturing these effects. Table 2.3 shows the comparison between SNM and MoNG and the relative value difference between ABT and MHFR for the 4DOFs-manipulator operating inside the Factory environment (a) and the car-like robot operating inside the Maze environment (b) while being subject collision dynamics. It can be seen that the additional non-linear effects are captured well by SNM. Interestingly, compared to the results in Table 2.2(a), were the 4DOFs-manipulator operates in the same environment without collision dynamics, MoNG captures the effects of collision dynamics as well, which indicates that collision dynamics have a large effect on the Gaussian assumption made by MHFR. Looking at the relative value difference between ABT and MHFR confirms this. MHFR suffers more from the increased non-linearity of the problems caused by collision dynamics compared to ABT. This effect aggravates as the uncertainty increases, which is a clear indication that the problem becomes increasingly non-linear with larger uncertainties. Looking at the results for the car-like robot operating in the Maze scenario presents a similar picture. Comparing the results in Table 2.3(b) where collision dynamics are taken into account to Table 2.2(b), shows that collision dynamics have a significant effect both to SNM as well as Measure of Non-Gaussianity.

**Effects of increasingly cluttered environments**

To investigate the effects increasingly cluttered environments have on both measures, we ran a set of experiments in which the Car-like robot and the 4DOFs-manipulator operate inside environments with an increasing number of randomly distributed obstacles. For this we generated test scenarios with 5, 10, 15, 20, 25 and 30 obstacles that are uniformly distributed across the environment. For each of these test scenarios, we randomly generated 100 environments. Figure 2.3(a)-(b) shows two example environments with 30 obstacles for the Car-like robot and the 4DOFs-manipulator. For this set of experiments we don't take collision dynamics into account and the control and observation errors were fixed at $e_t = e_z = 0.038$. Table 2.4 presents the results for SNM, MoNG and the relative value difference between ABT ant MHFR for the 4DOFs-manipulator (a) and the car-like robot (b). From these results it is clear that, as the environments become increasingly cluttered, the advantage



(a) Maze (b) Factory

Figure 2.3: Two example scenarios for the Car-like robot (a) and the 4DOFs-manipulator (b) with 30 randomly distributed obstacles.

**(a) 4DOFs-manipulator with increasing number of obstacles**

| Num obstacles | SNM | MonG | $\frac{\lvert V_{ABT}(b_0) - V_{MHFR}(b_0) \rvert}{V_{ABT}(b_0)}$ |
|---|---|---|---|
| 5 | 0.359 | 0.650 | 0.0276 |
| 10 | 0.449 | 0.643 | 0.0683 |
| 15 | 0.514 | 0.673 | 0.2163 |
| 20 | 0.527 | 0.683 | 0.2272 |
| 25 | 0.651 | 0.690 | 0.2675 |
| 30 | 0.698 | 0.672 | 0.3108 |

**(b) Car-like robot with increasing number of obstacles**

| Num obstacles | SNM | MonG | $\frac{\lvert V_{ABT}(b_0) - V_{MHFR}(b_0) \rvert}{V_{ABT}(b_0)}$ |
|---|---|---|---|
| 5 | 0.327 | 0.459 | 0.0826 |
| 10 | 0.387 | 0.473 | 0.1602 |
| 15 | 0.446 | 0.482 | 0.1846 |
| 20 | 0.468 | 0.494 | 0.4813 |
| 25 | 0.529 | 0.489 | 0.5788 |
| 30 | 0.685 | 0.508 | 0.7884 |

Table 2.4: Average values of SNM, MonG and relative value difference between ABT and MHFR for the 4DOFs-manipulator (a) and the car-like robot (b) operating inside environments with increasing numbers of obstacles. here $e_T = e_Z = 0.038$.

of ABT over MHFR increases, indicating that the obstacles have a significant effect on the Gaussian belief assumption of MHFR. Additionally SNM is clearly more sensitive to those effects compared to MoNG, whose values remain virtually unaffected by the clutterness of the environments.

### Effects of non-linear observation functions with non-additive errors

In the previous experiments we assumed that the observation functions are non-linear functions with additive Gaussian noise, a special class of non-linear observation functions. This class of observation functions has some interesting implications: First of all, the resulting observation distribution remains Gaussian. This in turn means that MoNG for the observation function evaluates to zero. Second, linearizing the observation function results in a Gaussian distribution with the same mean but different covariance. We therefore expect that the observation component SNM remains small, even for large uncertainties. To investigate how SNM reacts to non-linear observation functions with non-additive noise, we ran a set of experiments for the 4DOFs-manipulator operating inside the Factory environment and the car-like robot operating inside the Maze environment where we replaced the both observation functions with non-linear functions with non-additive noise. For the 4DOFs-manipulator we replaced

| **(a) Factory environment with additive observation errors** | | | | | |
|---|---|---|---|---|---|
| $\mathbf{e_Z}$ | 0.001 | 0.0195 | 0.038 | 0.057 | 0.075 |
| **SNM** | 0.001 | 0.004 | 0.013 | 0.036 | 0.047 |
| **MonG** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **(b) Factory environment with non-additive observation errors** | | | | | |
| $\mathbf{e_Z}$ | 0.001 | 0.0195 | 0.038 | 0.057 | 0.075 |
| **SNM** | 0.012 | 0.087 | 0.173 | 0.234 | 0.317 |
| **MonG** | 0.0 | 0.047 | 0.094 | 0.136 | 0.173 |

Table 2.5: Comparison between the observation component of SNM and MoNG for the 4DOF-manipulator operating inside the Factory environment with observation function eq.(2.21) (a) and eq.(2.26) (b)

| **(a) Maze environment with additive observation errors** | | | | | |
|---|---|---|---|---|---|
| $\mathbf{e_Z}$ | 0.001 | 0.0195 | 0.038 | 0.057 | 0.075 |
| **SNM** | 0.002 | 0.012 | 0.037 | 0.048 | 0.060 |
| **MonG** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **(b) Maze environment with non-additive observation errors** | | | | | |
| $\mathbf{e_Z}$ | 0.001 | 0.0195 | 0.038 | 0.057 | 0.075 |
| **SNM** | 0.083 | 0.086 | 0.101 | 0.198 | 0.207 |
| **MonG** | 0.0 | 0.012 | 0.032 | 0.053 | 0.075 |

Table 2.6: Comparison between the observation component of SNM and MoNG for the car-like robot operating inside the Maze environment with observation function eq.(2.23)(a) and observation function eq.(2.27)(b)

the observation function defined in eq.(2.21) with

$$o_t = g(s_t + w_t) \tag{2.26}$$

where $w_t \sim N(0, \Sigma_w)$. In other words, the manipulator has only access to a sensor that measure the position of the end-effector in the workspace.

For the car-like robot we use the following observation function:

$$o_t = \begin{bmatrix} \frac{1}{((x_t + w_t^1 - \hat{x}_1)^2 + (y_t + w_t^2 - \hat{y}_1)^2 + 1)} \\ \frac{1}{((x_t + w_t^1 - \hat{x}_2)^2 + (y_t + w_t^2 - \hat{y}_2)^2 + 1)} \\ v_t + w_t^3 \end{bmatrix} \tag{2.27}$$

where $\left(w_t^1, w_t^2, w_t^3\right)^T \sim N(0, \Sigma_w)$.

Table 2.5 shows the values for the observation components of SNM and MoNG for the 4DOFs-manipulator operating inside the Factory environment as the observation errors increase. As expected, for additive Gaussian errors, MoNG is zero, whereas SNM is small but measurable. This shows that SNM is able to capture the difference of the variance between the original and linearized observation functions. For non-additive errors, the observation function is non-Gaussian, therefore we can see that both measures increase as the observation errors increase. Interestingly for both measures the observation components yield significantly smaller values compared to the transition components.

| Factory environment with non-additive observation errors | | | |
|---|---|---|---|
| $e_T = e_Z$ | **SNM** | **MoNG** | $\frac{\mathbf{V_{ABT}(b_0) - V_{MHFR}(b_0)}}{\mathbf{V_{ABT}(b_0)}}$ |
| 0.001 | 0.012 | 0.0 | 0.06992 |
| 0.0195 | 0.0878 | 0.0476 | 0.43861 |
| 0.038 | 0.1732 | 0.0941 | 0.89720 |
| 0.057 | 0.2347 | 0.1363 | 1.46063 |
| 0.075 | 0.3178 | 0.1740 | 8.34832 |

| Maze environment with non-additive observation errors | | | |
|---|---|---|---|
| $e_T = e_Z$ | **SNM** | **MoNG** | $\frac{\mathbf{V_{ABT}(b_0) - V_{MHFR}(b_0)}}{\mathbf{V_{ABT}(b_0)}}$ |
| 0.001 | 0.0837 | 0.0 | -0.12451 |
| 0.0195 | 0.0868 | 0.0121 | 0.33872 |
| 0.038 | 0.1017 | 0.0321 | 1.41429 |
| 0.057 | 0.1983 | 0.0531 | 8.70111 |
| 0.075 | 0.2072 | 0.0758 | 0.95132 |

Table 2.7: Average values of SNM, MonG and the relative value difference between ABT and MHFR for the 4DOFs-manipulator operating inside the Factory environment (a) and the car-like robot operating inside the Maze environment (b) with non-additive observation errors

This indicates that the non-linearity of the problem stems mostly from the transition function. For the car-like robot operating inside the Maze environment we see a similar picture. For the observation function with additive Gaussian errors, Table 2.6(a) shows that MoNG remains zero for all values of $e_Z$, whereas SNM yields a small but measurable value. Again, both measures increase significantly in the non-additive error case in Table 2.6(b).

The question is now, how do ABT and MHFR perform in both scenarios when observation functions with non-additive Gaussian errors are used? Table 2.7(a) shows this relative value difference for the 4DOFs-manipulator operating inside the Factory environment. It can be seen that as the errors increase, the relative value difference between ABT and MHFR increase significantly, compared to the relative value difference show in Table 2.2(a), where an observation function with additive errors is used. Similarly, for the car-like robot operating inside the Maze scenario using the observation function with non-additive errors, the relative value difference shown in table Table 2.7(b) between the two solvers is much larger compared to Table 2.2(b).

This is in line with our intuition that non-Gaussian observation functions are more challenging for linearization-based solvers.

### 2.5.5 Testing SNM-Planner

In this set of experiments we want to test the performance of SNM-Planner in comparison with the two component planners ABT and MHFR. To this end we tested SNM-Planner on three problem scenarios: The Maze scenario for the car like robot shown in Figure 2.1(a) and the Factory scenario for the 4DOFs-manipulator. Additionally we tested SNM-Planner on a scenario in which the Kuka iiwa robot operates inside an office environment, as shown in Figure 2.1(b). Similarly to the Factory scenario, the robot has to reach a goal area while avoiding collisions with the obstacles. The planning time per step is 8s in this scenario. For the SNM-threshold we chose 0.51.

The results in Table 2.8 indicate that SNM-Planner is able to approximately identify when it is beneficial to use a linearization-based solver and when a general solver should be used. In all three scenarios SNM-Planner outperforms the two component planners. In the Maze scenario, the difference between SNM-Planner and the component planners is significant. The reason is, MHFR is well suited to compute a long-term strategy, as it constructs nominal trajectories from the current state estimate all the way to the goal, whereas the planning horizon of ABT is limited by the depth of the search tree. In

| Planner | Car-like robot | 4DOFs-manipulator | Kuka iiwa |
|---|---|---|---|
| ABT | -148.87 ± 71.75 | 799.175 ± 51.13 | 500.727 ± 67.16 |
| MHFR | -322.84 ± 46.57 | 334.23 ± 125.35 | -179.195 ± 75.98 |
| SNM-Planner | **15.43 ± 68.54** | **837.21 ± 12.64** | **625.35 ± 49.71** |

Table 2.8: Average total discounted reward and ± 95% confidence interval over 100 simulation runs. Here $e_T = pez = 0.038$ for all scenarios. The proportion of ABT being used in the Maze, Factory and Office scenarios is 37.85%, 56.43% and 42.33% respectively.

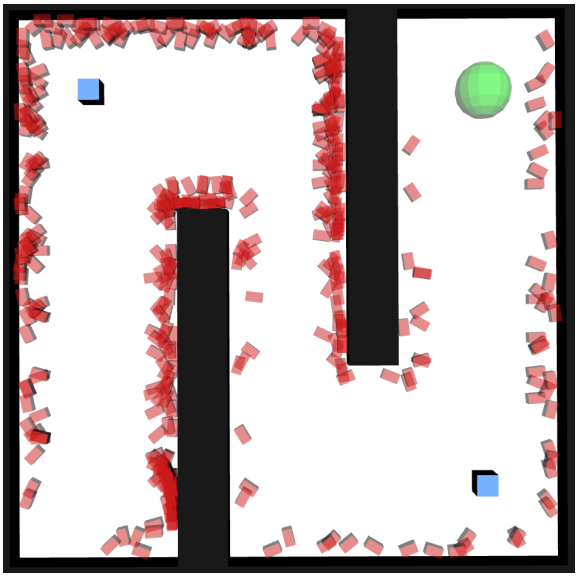


Figure 2.4: State samples (shown in red) in the Maze scenario for which the approximated SNM value exceeds the chosen threshold of 0.51

our experiments ABT is usually unable to plan more than 4 steps ahead. However, in the proximity of obstacles the Gaussian belief assumption of MHFR are no long valid, and careful planning is required to avoid collisions with the obstacles. In general ABT handles these situations better than MHFR. SNM-Planner combines the benefits of both planners and alleviates their shortcoming. Figure 2.4 shows state samples for which the SNM-values exceed the given threshold of 0.51. It is obvious that many of these samples are clustered around obstacles. In other words, when the support set of the current belief (i.e. the subset of the state space that is covered by the belief particles) lies in open areas, MHFR is used to drive the robot towards the goal, whereas in the proximity of obstacles, ABT is used to compute a strategy that avoids collisions with the obstacles.

A similar behavior was observed in the KukaOffice environment. During the early planning steps, when the robot operates in the open area, MHFR is well suited to drive the end-effector towards the goal area, but near the narrow passage at the back of the table, ABT in general computes better motion strategies. Again, SNM-Planner combines both strategies to compute better motion strategies than each of the component planners alone.

## 2.6 Proofs

For writing compactness we use the following shorthand notations for the transition and observation functions throughout the next two sections: $T = T(s,a,s')$, $\widehat{T} = \widehat{T}(s,a,s')$ and $Z = Z(s,a,o)$, $\widehat{Z} = \widehat{Z}(s,a,o)$. Additionally in Section 2.6.2 we use the notations $T_k = T(s_k,a,s')$ and $\widehat{T}_k = \widehat{T}(s_k,a,s')$.

### 2.6.1 Proof of Lemma 1

Consider $a \in A$ being the immediate action of policy $\pi$. Then for any $s \in S$:

$$\begin{aligned}
|\alpha_\pi(s) - \hat{\alpha}_\pi(s)| &= \left| R(s,a) + \gamma \int_{s'\in S}\int_{o\in O} TZ\alpha^o_\pi(s')\mathrm{d}o\mathrm{d}s' - R(s,a) - \gamma \int_{s'\in S}\int_{o\in O} \widehat{T}\widehat{Z}\widehat{\alpha}^o_\pi(s')\mathrm{d}o\mathrm{d}s' \right| \\
&= \gamma \left| \int_{s'\in S}\int_{o\in O} TZ\alpha^o_\pi(s') - \widehat{T}\widehat{Z}\widehat{\alpha}^o_\pi(s') do ds' \right| \\
&\leq \gamma \left( \left| \int_{s'\in S}\int_{o\in O} TZ\left[\alpha^o_\pi(s') - \widehat{\alpha}^o_\pi(s')\right]\mathrm{d}o\mathrm{d}s' \right| + \left| \int_{s'\in S}\int_{o\in O} \widehat{\alpha}^o_\pi(s')\left[TZ - \widehat{T}\widehat{Z}\right]\mathrm{d}o\mathrm{d}s' \right| \right)
\end{aligned} \tag{2.28}$$

Let's have a look at the second term on the right-hand side of eq.(2.28), that is

$$term2(s,a) = \left| \int_{s'\in S}\int_{o\in O} \widehat{\alpha}^o_\pi(s')\left[TZ - \widehat{T}\widehat{Z}\right]\mathrm{d}o\mathrm{d}s' \right| \tag{2.29}$$

We can expand and upper-bound this term as follows:

$$\begin{aligned}
term2(s,a) &= \left| \int_{s'\in S}\int_{o\in O} \widehat{\alpha}^o_\pi(s')\left[TZ - \widehat{T}Z + \widehat{T}Z - \widehat{T}\widehat{Z}\right]\mathrm{d}o\mathrm{d}s' \right| \\
&\leq \left| \int_{s'\in S}\left[T - \widehat{T}\right]\int_{o\in O} \widehat{\alpha}^o_\pi(s')Z\mathrm{d}o\mathrm{d}s' \right| + \left| \int_{s'\in S}\widehat{T}\int_{o\in O} \widehat{\alpha}^o_\pi(s')\left[Z - \widehat{Z}\right] \right| \\
&\leq \int_{s'\in S}\left|T - \widehat{T}\right|\int_{o\in O}\left|\widehat{\alpha}^o_\pi(s')\right| Z\mathrm{d}o\mathrm{d}s' + \int_{s'\in S}\widehat{T}\int_{o\in O}\left|\widehat{\alpha}^o_\pi(s')\right|\left|Z - \widehat{Z}\right|\mathrm{d}o\mathrm{d}s'
\end{aligned} \tag{2.30}$$

The term $|\widehat{\alpha}^o_\pi|$ can be upper-bounded from above using $|\widehat{\alpha}^o_\pi(s)| \leq \frac{R_m}{1-\gamma}$ for any $s \in S$, which yields

$$term2(s,a) \leq \frac{R_m}{1-\gamma}\left[\int_{s'\in S}\left|T - \widehat{T}\right|\mathrm{d}s' + \int_{s'\in S}\widehat{T}\int_{o\in O}\left|Z - \widehat{Z}\right|\mathrm{d}o\mathrm{d}s'\right] \tag{2.31}$$

From the definition of the total variation distance, it follows that $\int_{s'\in S}\left|T - \widehat{T}\right|\mathrm{d}s' = 2D^{s,a}_{TV}(T,\widehat{T})$ for any given $s \in S$ and $a \in A$ and $\int_{o\in O}\left|Z - \widehat{Z}\right|\mathrm{d}o = 2D^{s',a}_{TV}(Z,\widehat{Z})$ for any given $s' \in S$. Substituting these equalities into eq.(2.31) and taking the supremum over the conditionals $s, s'$ and $a$ allows us to upper-bound eq.(2.31) by

$$term2(s,a) \leq 2\frac{R_m}{1-\gamma}\Psi(P,\widehat{P}) \tag{2.32}$$

Substituting this upper bound into 2.28 yields

$$\begin{aligned}
|\alpha_\pi(s) - \hat{\alpha}_\pi(s)| &\leq \gamma \left| 2\frac{R_m}{1-\gamma}\Psi(P,\widehat{P}) + \int_{s'\in S}\int_{o\in O} TZ\left[\alpha^o_\pi(s') - \widehat{\alpha}^o_\pi(s')\right] do ds' \right| \\
&\leq \gamma \left( 2\frac{R_m}{1-\gamma}\Psi(P,\widehat{P}) + \int_{s'\in S}\int_{o\in O} TZ\left|\alpha^o_\pi(s') - \widehat{\alpha}^o_\pi(s')\right| do ds' \right)
\end{aligned} \tag{2.33}$$

The last term on the right hand side of eq.(2.33) is essentially a recursion. Unfolding this recursion yiels

$$|\alpha_\pi(s) - \hat{\alpha}_\pi(s)| \leq 2\gamma \frac{R_m}{(1-\gamma)^2} \Psi(P, \widehat{P}) \tag{2.34}$$

which is Lemma 1. □

### 2.6.2 Proof of Lemma 2

We can write the absolute difference between the SNM-values conditioned on two states $s_1, s_2 \in S_i$ as

$$\begin{aligned} |\Psi_T(s_1) - \Psi_T(s_2)| &= \left| \sup_{a \in A} D_{TV}(T_1, \widehat{T}_1) - \sup_{a \in A} D_{TV}(T_2, \widehat{T}_2) \right| \\ &= \left| \frac{1}{2} \sup_{a \in A} \sup_{|f| \leq 1} \left| \int_{s' \in S} f(s') \left[ T_1 - \widehat{T}_1 \right] \mathrm{d}s' \right| - \frac{1}{2} \sup_{a \in A} \sup_{|f| \leq 1} \left| \int_{s' \in S} f(s') \left[ T_2 - \widehat{T}_2 \right] \mathrm{d}s' \right| \right| \end{aligned} \tag{2.35}$$

Manipulating the algebra allows us to write

$$\begin{aligned} |\Psi_T(s_1) - \Psi_T(s_2)| &\leq \frac{1}{2} \sup_{a \in A} \left| \sup_{|f| \leq 1} \left( \int_{s' \in S} f(s') \left[ T_1 - T_2 \right] \mathrm{d}s' + \int_{s' \in S} f(s') \left[ \widehat{T}_1 - \widehat{T}_2 \right] \mathrm{d}s' \right) \right| \\ &\leq \frac{1}{2} \sup_{a \in A} \left( \sup_{|f| \leq 1} \int_{s' \in S} f(s') \left| T_1 - T_2 \right| \mathrm{d}s' + \sup_{|f| \leq 1} \int_{s' \in S} f(s') \left| \widehat{T}_1 - \widehat{T}_2 \right| \mathrm{d}s' \right) \\ &\leq \frac{1}{2} D_S(s_1, s_2) \left[ C_{T_i} + C_{\widehat{T}_i} \right] \end{aligned} \tag{2.36}$$

For the last inequality we bound the terms $|T_1 - T_2|$ and $\left| \widehat{T}_1 - \widehat{T}_2 \right|$ using Definition 2. Furthermore we use the fact that $\sup_{|f| \leq 1} \int_{s' \in S} f(s') ds' = 1$, assuming that the state space $S$ is normalized. This concludes the proof of Lemma 2. □

## 2.7 Summary

This chapter presents our work in identifying the suitability of linearization for planning under partial observability. To this end, we present a measure of non-linearity for stochastic systems, called Statistical-distance-based Non-linearity Measure (SNM), which is based on the total-variation distance between the distributions that represent the system's motion–sensing model and its linearized version. Comparison studies with one of state-of-the-art methods for non-linearity measure indicate that SNM is more suitable in measuring the effects cluttered environments have on the efficiency of linearization-based solvers. We also propose a simple threshold-based on-line solver SNM-Planner that uses a local estimate of SNM around the current belief to select a general POMDP solver or a linearization-based solver to compute the policy. Experimental results indicate that our simple solver can appropriately decide where linearization should be used and when it should be avoided. This results in policies that are better than the ones computed by each of the component solver alone.

The following publication has been incorporated as Chapter 3.

- [1] **M. Hoerger**, H. Kurniawati and A. Elfes, Multilevel Monte-Carlo for Solving POMDPs Online, in: Proc. International Symposium on Robotics Research (ISRR) (To Appear), 2019

| Contributor | Statement of contribution | % |
|---|---|---|
| **Marcus Hoerger** | writing of text | 60 |
| | proof-reading | 50 |
| | theoretical derivations | 90 |
| | numerical calculations | 100 |
| | preparation of figures | 100 |
| | initial concept | 80 |
| Hanna Kurniawati | writing of text | 40 |
| | proof-reading | 50 |
| | supervision, guidance | 90 |
| | theoretical derivations | 10 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 20 |
| Alberto Elfes | writing of text | 0 |
| | proof-reading | 0 |
| | supervision, guidance | 10 |
| | theoretical derivations | 0 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 0 |

# Chapter 3

# Multilevel Monte-Carlo for Solving POMDPs On-Line

## 3.1 Introduction

In the previous Chapter we have seen that simplifications such as linearized system dynamics and Gaussian belief assumptions can help in speeding-up the planning process for POMDP solvers. Recall that the reason why simplified system dynamics are necessary in the first place is because state-of-the-art general on-line POMDP solvers rely on a large number of expensive forward simulations of the system and standard Monte-Carlo to estimate the expected values of different sequences of actions.

The problem of computing expectations from expensive simulations using Monte-Carlo techniques arises in many disciplines. Various methods have been developed to increase the efficiency of plain Monte-Carlo, such as Quasi-Monte-Carlo [73], Importance sampling [74] and Control Variates [75]. More recently the concept of Multilevel Monte-Carlo (MLMC) has been proposed [49, 50]. The idea is to use cheap and coarse approximations of the system to carry out the majority of the simulations and combine them with a small number of accurate but expensive simulations to maintain correctness. While the underlying approach is conceptually simple, MLMC has been used to significantly reduce the computational effort on an number of applications that involve computing expectations from expensive simulations [76–78].

Motivated by this idea, we propose a sampling-based general on-line POMDP solver, called Multilevel POMDP Planner (MLPP) that uses multiple levels of approximation of the system's dynamics, to reduce the number and complexity of forward simulation needed to compute a near-optimal policy. In particular, MLPP combines the commonly used Monte-Carlo-Tree-Search [43] with MLMC. By constructing a set of correlated samples from a sequence of approximations of the original system's dynamics, in conjunction with applying Multilevel Monte-Carlo estimation to compute the expected value of sequences of actions, MLPP is able to compute near-optimal policies substantially faster than two of the fastest today's on-line POMDP solvers on four challenging robotic motion planning tasks under uncertainty, involving POMDP-based torque control, navigation and grasping.

We also show that under certain conditions, MLPP converges asymptotically to the optimal solution.

## 3.2 Background

### 3.2.1 Multilevel Monte-Carlo

In this section, we provide a brief overview of the underlying concept of MLMC. A more extensive overview of MLMC and its applications is available at [50].

Suppose we have a random variable $X$ and we wish to compute its expectation $\mathbb{E}[X]$. A simple Monte-Carlo (MC) estimator for $\mathbb{E}[X]$ is

$$\mathbb{E}[X] \approx \frac{1}{N}\sum_{i=1}^{N} X^{(i)} \tag{3.1}$$

where $X^{(i)}$ are iid. samples drawn from $X$. In many applications sampling from $X$ directly is expensive, causing the MC-estimator eq.(3.1) to converge slowly.

The idea of MLMC is to use the linearity of expectation property to reduce the cost of sampling. Suppose, $X_0, X_1, X_2, \ldots, X_L$ is a sequence of approximations to $X$, where $\lim_{L\to\infty} X_L = X$ and the approximation increases in accuracy and sampling cost as the index increases. Using the linearity of expectation, we have the simple identity:

$$\mathbb{E}[X] = \mathbb{E}[X_0] + \sum_{l=1}^{L} \mathbb{E}[X_l - X_{l-1}] \tag{3.2}$$

and can design the unbiased estimator:

$$\mathbb{E}[X] \approx \frac{1}{N_0}\sum_{i=1}^{N_0} X_0^{(i)} + \sum_{l=1}^{L} \frac{1}{N_l}\sum_{i=1}^{N_l} (X_l^{(i)} - X_{l-1}^{(i)}) \tag{3.3}$$

with independent samples at each level. The key here is that even though the samples at each level are independent, the individual samples $X_l^{(i)}$ and $X_{l-1}^{(i)}$ at level $l$ are correlated, such that their differences have a small variance. Note that in Monte-Carlo techniques such as Quasi-Monte-Carlo [79] or Markov-Chain-Monte-Carlo [80], correlation between samples can, in general, lead to a reduced convergence rate. However, in the context of MLMC, correlated samples for the expected difference on multiple levels has the opposite effect, *i.e.*an increased convergence rate. This becomes obvious when looking at the identity

$$\mathbb{V}[X_l - X_{l-1}] = \mathbb{V}[X_l] + \mathbb{V}[X_{l-1}] - 2COV(X_l, X_{l-1}) \tag{3.4}$$

A strong correlation between $X_l$ and $X_{l-1}$ results in $2COV(X_l, X_{l-1})$ being large, thereby reducing the variance of $X_l - X_{l-1}$ which in turn increases the convergence rate of the Monte-Carlo estimator of $\mathbb{E}[X_l - X_{l-1}]$.

Of course, the aim is to be able to sample only from the first few approximations while still computing a relatively good approximation of $\mathbb{E}[X]$. It turns out, if we define the sequence of approximations appropriately [50], the variance $\mathbb{V}[X_l - X_{l-1}]$ becomes smaller for increasing level

$l$, and therefore we require fewer and fewer samples to accurately estimate the expected differences. This means we can take the majority of the samples at the coarser levels, where sampling is cheap, and only a few samples are required on the finer levels, thereby leading to a substantial reduction of the cost to estimate $\mathbb{E}[X]$ accurately.

## 3.3 Multilevel POMDP Planner (MLPP)

MLPP is an anytime on-line POMDP solver. Starting from the current history $h_t$, MLPP computes an approximation to the optimal policy by iteratively constructing and evaluating a search tree $\mathcal{T}$, a tree whose nodes are histories and edges represent a pair of action-observation. From hereafter, we use the term *nodes* and the *histories* they represent interchangeably. A history $h'$ is a child node of $h$ via edge $(a,o)$ if $h' = hao$. The root of $\mathcal{T}$ corresponds to an empty history $h_0$. The policy of MLPP is embedded in $\mathcal{T}$ via $\pi(h) = \arg\max_{a\in A} \widehat{Q}(h,a)$, where $\widehat{Q}(h,a)$ is an approximation of $Q(h,a) = R(h,a) + \gamma\mathbb{E}_{o\in O}[V_{\pi^*}(hao)]$, *i.e.*the expected value of executing $a$ from $h$ and continuing optimally afterwards. Note that we can equip each node with a representation of its underlying belief (*e.g.*a set of particles). Hence calculating a policy using a tree of histories is equivalent to using a belief tree.

To compute $\widehat{Q}$, MLPP constructs $\mathcal{T}$ using a framework similar to POMCP [13] and ABT [12]: Given the current history $h_t$, MLPP repeatedly samples *episodes* starting from $h_t$. An episode $e$ is a sequence of $(s,a,o,r)$-quadruples, where the state $s \in S$ of the first quadruple is distributed according to the current belief $b_t$ – we approximate beliefs by sets of particles – and the states of all subsequent quadruples are sampled from the transition function $T$, given the state and action of the previous quadruple. The observations $o \in O$ are sampled from the observation function $Z$, while the reward $r = R(s,a)$ is generated by the simulation process. Each episode corresponds to a path in $\mathcal{T}$. Details on how the episodes are sampled are given in Section 3.3.1.

Key to MLPP is the adoption of the MLMC concept: Episodes are sampled using multiple levels of approximations of the transition function. Suppose $T$ is the transition function of the POMDP problem. MLPP first defines a sequence of increasingly accurate approximations of the transition function $T_0, T_1, ..., T_L$ with $T_L = T$, and uses the less accurate but cheaper transition functions for the majority of the episode samples, to approximate the $Q$-value function fast. Note that to ensure asymptotic convergence of MLPP, we slightly modify MLMC such that $L$ is finite and $T_L$ is the most refined level MLPP samples from.

Let $V_k(e)$ be the total discounted reward of an episode starting from the $k$-th quadruple. For a node $h$ of depth $k$, MLPP approximates $Q(h,a)$ according to:

$$\begin{aligned}\widehat{Q}(h,a) &= \widehat{Q}_0(h,a) + \sum_{l=1}^{L} (\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a)) \\ &= \frac{1}{N_0(h,a)} \sum_{i=1}^{N_0(h,a)} V_k(e_0^{(i)}) + \sum_{l=1}^{L} \frac{1}{N_l(h,a)} \sum_{i=1}^{N_l(h,a)} (V_k(e_l^{(i)}) - V_k(e_{l-1}^{(i)}))\end{aligned} \tag{3.5}$$

where an episode $e_l$ on level $l$ is sampled using $T_l$, and $N_l(h,a)$ is the number of all episodes on level $l$ that start from $h_0$, pass through $h$ and execute $a$ from $h$. Similar to eq.(3.3), the key here is that even though we draw independent samples on each level, the episode samples for the value differences $V_k(e_l^{(i)}) - V_k(e_{l-1}^{(i)})$ are *correlated*. The question is, *how do we correlate the sampled episodes?*

We adopt the concepts of *determinization* [14] and common random numbers [81], a popular variance reduction technique: To sample states and observations for an episodes on level $l$, we use a deterministic simulative model, i.e. a function $f_l : S \times A \times [0,1] \mapsto S \times O$ such that, given a random variable $\psi$ uniformly distributed in $[0,1]$, $(s',o) = f_l(s,a,\psi)$ is distributed according to $T_l(s,a,s')O(s',a,o)$. For an initial state $s_0 \sim b_t$ and a sequence of actions, the states and observations of an episode on level $l$ are then *deterministically* generated from $f_l$ using a sequence $\Psi = (\psi_0, \psi_1, ...)$ of iid. random numbers. Now, to sample a correlated episode on level $l-1$, we use the same initial state $s_0$, the same sequence of actions and the same random sample $\Psi$ used for the episode on level $l$, but generate next states and observations from the model $f_{l-1}$ corresponding to $T_{l-1}$, such that for a given $s$ and $a$, $(s',o) = f_{l-1}(s,a,\psi)$ is distributed according to $T_{l-1}(s,a,s')O(s',a,o)$. Using the same initial state, action sequence and random sample $\Psi$ results in two closely correlated episodes, reducing the variance of $V_k(e_l^{(i)}) - V_k(e_{l-1}^{(i)})$.

To incorporate the above sampling strategy to the construction of $\mathcal{T}$, MLPP computes the estimator eq.(3.5) in two subsequent stages: In the first stage, MLPP samples episodes using the coarsest approximation $T_0$ of the transition function to compute the first term in eq.(3.5). In the second stage, MLPP samples correlated pairs of episodes to compute the value difference terms in eq.(3.5). These two stages are detailed in the next two subsections. An overview of MLPP is shown in Algorithm 2, procedure RUNMLPP. We start by initializing $\mathcal{T}$, containing the empty history $h_0$ as the root, and setting the current belief to be the initial belief (line 1). Then, in each planning loop iteration (line 3-7) we first sample an episode using $T_0$ (line 4), followed by sampling two correlated episodes (line 6). Once the planning time for the current step is over, MLPP executes the action that satisfies $\arg\max_{a \in A} \widehat{Q}(h_0,a)$. Based on the executed action $a$ and perceived observation $o$, we update the belief using a SIR particle filter [63] (line 11) and continue planning from the updated history $h_0ao$. This process repeats until a maximum number of steps is reached, or the system enters a terminal state (we assume that we know when the system enters a terminal state).

### 3.3.1 Sampling the episodes using $T_0$

To sample an episode using $T_0$, starting from the current history $h$, we first sample a state from the current belief which will then correspond to the state of the first quadruple of the episode (line 1 in Algorithm 2, procedure SAMPLEEPISODE). To sample a next state and observation, we first need to select an action from $h$ (line 4). The action-selection strategy is similar to the strategy used in POMCP and ABT. Consider the set of actions $A'(h) \subseteq A$ that have already been selected from $h$. If $A'(h) = A$, *i.e.*all actions have been selected from $h$ at least once, we formulate the problem of which action to select as a Multi-Arm-Bandit problem (MAB) [44]. MABs are a class of reinforcement learning

**Algorithm 2** MLPP

RUNMLPP

1: $\mathcal{T}$ = initializeTree(); $b = b_0$; $h$ = Root of $\mathcal{T}$; terminal = False; $t = 1$
2: **while** terminal is False and $t < t_{max}$ **do**
3: **while** planning time not over **do**
4: $(e, \Psi)$ = SampleEpisode($\mathcal{T}$, $b$, $h$, 0)
5: backupEpisode($\mathcal{T}$, $e$)
6: SampleCorrelatedEpisodes($\mathcal{T}$, $b$, $h$)
7: **end while**
8: $a$ = get best action in $\mathcal{T}$ from $h$
9: terminal = Execute $a$
10: $o$ = get observation
11: $b = \tau(b,a,o)$; $h = hao$
12: $t = t + 1$
13: **end while**

SAMPLEEPISODE(Search tree $\mathcal{T}$, Belief $b$, History node $h$, level $l$)

1: $s$ = sample a state from $b$
2: $e$ = init episode; $\Psi$ = init random number sequence; unvisitedAction = False
3: **while** unvisitedAction is False and $s$ not terminal **do**
4: $(a$, unvisitedAction$) = \text{UCB1}(h,l)$ ▷ For $l = 0$ we select actions from $A$, for $l > 0$ from $A'(h)$
5: **if** $a$ is $\varnothing$ **then** break **end if**
6: $\psi \sim [0,1]$
7: $(s',o) = f_l(s,a,\psi)$ ▷ Generate $(s',o)$ such that $(s',o) \sim T_l(s,a,s')Z(s',a,o)$
8: $r = R(s,a)$; insert $(s,a,o,r)$ to $e$ and $\psi$ to $\Psi$
9: $s = s'$; $h$ = child node of $h$ via edge $(a,o)$. If no such child exists, create one
10: **end while**
11: $r = 0$
12: **if** unvisitedAction is True **then** $r$ = calculateHeuristic($s$, $h$) **end if**
13: insert $(s,-,-,r)$ to $e$
14: **return** $(e,\Psi)$

SAMPLECORRELATEDEPISODES(Search tree $\mathcal{T}$, Belief $b$, History node $h$)

1: $l \sim 2^{-l}$ ▷ Sample a level $l$ proportional to $2^{-l}$
2: $(e_l,\Psi) = \text{sampleEpisode}(\mathcal{T},b,h,l)$
3: $e_{l-1}$ = init episode
4: $s = e_l[1].s$ ▷ State of the first quadruple of $e_l$
5: **for** $i = 1$ to $|h_l|$ **do**
6: $a = e_l[i].a$ ▷ Action of the $i$-th quadruple of $e_l$
7: $(s',o) = f_{l-1}(s,a,\Psi[i])$ ▷ $s'$ is generated according to $T_{l-1}$
8: $r = R(s,a)$; insert $(s,a,o,r)$ to $e_{l-1}$
9: $s = s'$; $h$ = child node of $h$ via edge $(a,o)$. If no such child exists, create one
10: **if** $s'$ is terminal **then** break **end if**
11: **end for**
12: $r = 0$
13: **if** $i$ is $|e_l|$ **then**
14: $r$ =calculateHeuristic($s$, $h$)
15: **end if**
16: insert $(s,-,-,r)$ to $e_{l-1}$ and backupRewardDifference($\mathcal{T}$, $e_l$, $e_{l-1}$)

problems where an agent has to select a sequence of actions to maximise the total reward, but the rewards of selecting actions is not known in advance. Futhermore, MABs are stateless, in a sense that the reward obtained at a particular time-step doesn't depend on the previously obtained reward. One of the most successful algorithms to solve MAB problems is Upper Confidence Bounds1 (UCB1) [45]. UCB1 selects an action according to

$$a = \arg\max_{a \in A} \left( \widehat{Q}(h,a) + c_0 \sqrt{\frac{\log(N_0(h))}{N_0(h,a)}} \right) \tag{3.6}$$

where $N_0(h)$ is the number of episodes that were sampled using $T_0$ that pass through $h$, $N_0(h,a)$ is the number of episodes that were sampled using $T_0$, pass through $h$ and select action $a$ from $h$ and $c_0$ is an exploration constant. In case there are actions that haven't been selected from $h$, we use a rollout strategy that selects one of these actions uniformly at random. The use of MABs to formulate the action-selection problem during the search has previously been successfully used in [12, 13, 82].

We then sample a random number $\psi \sim [0,1]$ (line 6) and, based on $\psi$ and the selected action, generate a next state and observation (line 7) from the model $f_0$ using $T_0$, an immediate reward (line 8) and add the quadruple to the episode. Additionally we set $h$ to the child node that is connected to $h$ via the selected action and sampled observation. If this child node doesn't exist yet, we add it to $\mathcal{T}$ (line 9). Note that selecting a previously unselected action always results in a new node.

To get a good estimate of $\widehat{Q}_0(h,a)$ for a newly selected action, MLPP computes a problem dependent heuristic estimate (line 12) in its rollout strategy using the last state of the episode. Computing a heuristic estimate of $\widehat{Q}_0(h,a)$ helps MLPP to quickly focus its search on more promising parts of $\mathcal{T}$.

Once we have sampled the episodes, we backup the expected discounted reward of the episode all the way back to the current history (line 5 in procedure RUNMLPP) to update the $\widehat{Q}_0$-values along the selected action sequence.

### 3.3.2 Sampling the correlated episodes

Once MLPP has sampled an episode using the coarsest approximation of $T$, it samples two correlated episodes, via procedure SAMPLECORRELATEDEPISODES in Algorithm 2. For this we first sample a level $l$ proportional to $2^{-l}$ (line 1), with $l \geq 1$. This is motivated by the idea that as we increase the level, fewer and fewer samples are needed to get a good estimate of the expected value difference. The idea of randomizing the level is motivated by [83]. Based on the sampled level $l$, we first sample an episode using the finer transition function $T_l$ (line 2). Sampling this episode is similar to the coarsest level, with some notable differences in the action-selection strategy: At each node $h$, we only consider actions from the set $A'(h) \subseteq A$ that have been selected at least once during sampling of the coarsest episodes. This is because actions that haven't been selected on the coarsest level yet, don't have an estimate for the first component $\widehat{Q}_0(h,a)$ of eq.(3.5), therefore we wouldn't be able to update the $Q$-value estimates in a meaningful way. Additionally, for each level, we maintain separate visitation counts $N_l(h)$ and $N_l(h,a)$, which allows us to use UCB1 as the action selection strategy, i.e.

$a = \arg\max_{a \in A'(h)} \left( \widehat{Q}(h,a) + c_l \sqrt{\frac{\log(N_l(h))}{N_l(h,a)}} \right)$. In case we end up in a node where $A'(h)$ is empty, we stop the sampling of the episode.

To sample a correlated episode on the coarser level $l-1$, we use the model $f_{l-1}$ corresponding to $T_{l-1}$, but the same initial state (line 4), the same action sequence (line 6) and the same random number sequence (line 7) that was used for the episode on level $l$. After we have obtained two correlated episodes on level $l$ and $l-1$, we backpropagate the discounted reward difference between the two episodes along the action sequence all the way to the current history (line 16), to update the expected $Q$-value difference between level $l$ and $l-1$, i.e. $\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a)$ for each action in the sequence. Note that even though we use the same action sequence for both episodes, the sequence of visited nodes in $\mathcal{T}$ might be different due to different observations, or because the coarse episode terminates earlier than fine episode. If this is the case, we backup both episodes individually until we arrive at an action edge that is the same for both episodes (there is always at least one common action edge, which is the outgoing action of the current history). The actual $Q$-value estimates $\widehat{Q}(h,a)$ along the common action sequence are then updated according to

$$\widehat{Q}(h,a) = \widehat{Q}_0(h,a) + \sum_{l=1}^{K} w_l(h,a) \left( \widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a) \right) \tag{3.7}$$

During the early stages of planning, when only a few discounted reward differences have been sampled, the estimator $\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a)$ might have a large variance, causing it to "overcorrect" the policy. To alleviate this issue, we use a weighting function $w_l$ defined as $w_l(h,a) = \left(1 + \frac{\widehat{\mathbb{V}}[Q_l(h,a) - Q_{l-1}(h,a)]}{N_l}\right)^{-1}$, where $\widehat{\mathbb{V}}[\cdot]$ is an estimate of the variance of the $Q$-value difference $Q_l(h,a) - Q_{l-1}(h,a)$, obtained from the history samples, and $N_l$ is the number of samples used to estimate $Q_l(h,a) - Q_{l-1}(h,a)$. As the number of samples on level $l$ and $l-1$ increases, $w_l(h,a)$ converges towards 1, hence the limit of eq.(3.7) is the actual MLMC-estimator of $\widehat{Q}(h,a)$ defined in eq.(3.5).

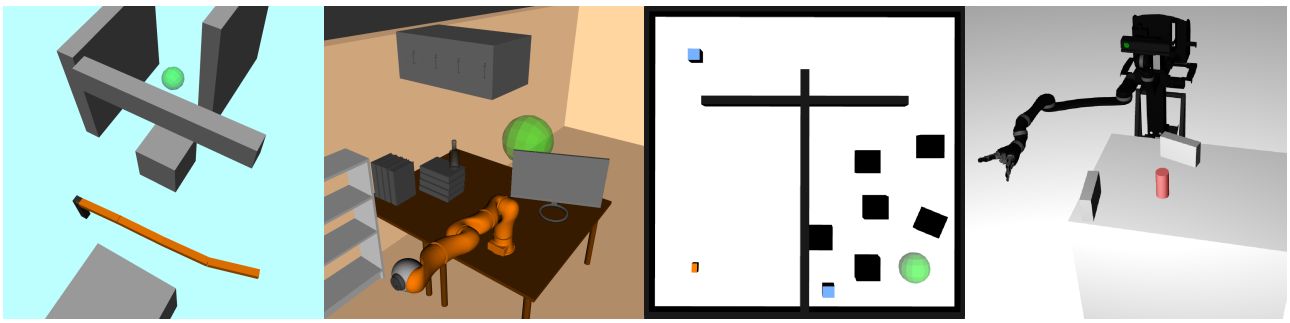
## 3.4 Discussion

We now discuss under which conditions MLPP converges to the optimal policy.

Suppose we have an action sequence $(a_1, a_2, a_3, ..., a_K)$ and an initial state $s_0 \sim b_t$. Applying the action sequence to $s_0$ results in a *trajectory* $(s_0, a_1, s_1, o_1, a_2, s_2, o_2, ...)$ which is distributed according to $\prod_{i=1}^{K} T(s_{i-1}, a_i, s_i) Z(s_i, a_i, o_i)$. Now suppose we have a sequence of approximations of the transition function $T_0, T_1, ...T_L$ with $T_L = T$.

**Assumption 1.** *Given a POMDP P, with transition function $T$ and a sequence of approximations of the transition function $T_0, T_1, ...T_L$ with $T_L = T$, then for any action sequence $(a_1, a_2, a_3, ..., a_K)$, $\prod_{i=1}^{K} T(s_{i-1}, a_i, s_i) Z(s_i, a_i, o_i) > 0$ implies $\prod_{i=1}^{K} T_l(s_{i-1}, a_i, s_i) Z(s_i, a_i, o_i) > 0$ for $0 \leq l \leq L$.*

Intuitively, under this assumption, any node in $\mathcal{T}$ than can be reached by episodes that are sampled using the original transition function $T$ can also be reached by episodes that are sampled using $T_l$. Given this assumption, and the fact that we select actions according to UCB1 on each level independently,



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

Figure 3.1: Test scenarios used to evaluate MLPP. (a) `Factory` (b) `KukaOffice` (c) `CarNavigation` (d) `MovoGrasping`

the estimator $\widehat{Q}(h,a)$ in 3.7 converges to $Q(h,a)$ in probability as the number of episodes that pass through $h$ and execute $a$ from $h$ increases on each level. This is based on the analysis in [13, 43]. Therefore MLPP's policy converges to the optimal policy in probability, too. Assumption 1 is quite strong and might be too restrictive for some problems. Relaxing this assumption is subject to future work. Nevertheless, problems whose transition and observation functions for all stat–action pairs are represented as distributions with infinite support (e.g., Gaussian) satisfy the assumption above.

## 3.5 Experiments and Results

MLPP is tested on two motion-planning problems under uncertainty with expensive non-linear transition dynamics and two problems with long-planning horizon. The scenarios are shown in Figure 3.1 and described below.

### 3.5.1 Problem scenarios with expensive transition dynamics

#### Factory

In this problem scenario we look at the 4DOFs-manipulator defined in Section 2.5.3, operating inside the Factory environment presented in Figure 3.1(a)). However, here the transition function of the robot is slightly different. Recall from Section 2.5.3 that we use the ODE physics engine to simulate the transition dynamics, where for the discretization of the numerical integrator of ODE, we used $\delta t = 0.004s$. Here we use $\delta t = 0.0001s$ instead. In other words, simulating the transition dynamics becomes is a lot more expensive, due to the finer discretization of the integrator. Apart from the modification of the transition dynamics, the remaining components of the POMDP model (observation function, reward function and initial belief) are the same as the ones defined in Section 2.5.3, however, here we use a discount factor of 0.98.

#### KukaOffice

The robot used in this scenarion is the 7DOFs Kuka iiwa manipulator presented in Section 2.5.3. Here the robot operates in the KukaOffice scenario, shown in Figure 2.1(c). Similarly to the `Factory` problem, we use the ODE physics engine to simulate the transition dynamics, but again, change the discretization step-size of the integrator used by ODE from $\delta t = 0.004s$ to $\delta t = 0.0001s$, which results in significantly more expensive transition dynamics.

### 3.5.2 Problem scenarios with long planning-horizons

#### CarNavigation

The car-like robot used for this problem scenario is the one used in Section 2.5.3, with a slight modification of the transition function: In Section 2.5.3 the transition function was parametrized by the duration of a time step $\Delta t$, which was set to $\Delta t = 0.3s$. Here we set $\Delta t = 0.05s$. Additionally we use the environment, which is shown in Figure 2.1(c). Here the robot receives a penalty of -500 when it collides with an obstacle, a reward of 10,000 when reaching the goal area (in both cases it enters a terminal state) and a small penalty of -1 for every step. The discount factor is 0.99 and we allow a maximum of 500 planning steps.

#### MovoGrasp

A 6-DOF Movo manipulator equipped with a gripper must grasp a cylindrical object placed on a table in front of the robot while avoiding collisions with the table and the static obstacles on the table. The environment is shown in Figure 3.1(d). The state space of the manipulator is defined as $S = \Theta \times GripperStates \times GraspStates \times \Phi_{obj}$, where $\Theta = (-3.14rad, 3.14rad)^6$ are the joint angles of the arm, $GripperStates = \{gripperOpen, gripperClosed\}$ indicates whether the gripper is open or closed, $GraspStates = \{grasp, noGrasp\}$ indicates whether the robot is grasping the object or not, and $\Phi_{obj} \subseteq \mathbb{R}^6$ is the set of poses of the object in the robot's workspace. The action space is defined as $A = A_\theta \times \{openGripper, closeGripper\}$ where $A_\theta \subseteq \mathbb{R}^6$ is the set of fixed joint angle increments/decrements for each joint, and $openGripper, closeGripper$ are actions to open/close the gripper, resulting in 66 actions. When executing a joint angle increment/decrement action $\hat{\theta}$, the joint angles evolve linearly according to $\theta_{t+1} = \theta_t + \Delta t\hat{\theta} + v_t$, where $\Delta t = 0.25$ and $v_t$ is a multivariate zero-mean Gaussian control error. We assume that the $openGripper$ and $closeGripper$ are deterministic.

Here the robot has access to two sensors: A joint-encoder that measures the joint angles of the robot and a grasp detector that indicates whether the robot grasps the object or not. For the joint-encoder, we assume that the encoder readings are disturbed by a small additive error drawn from a uniform distribution $[-0.05, 0.05]$. For the grasp detector we assume that we get a correct reading 90% of the time. The robot starts from an initial belief where the gripper is open, the joint angles of the robot are $(0.8, -0.2, 0.8, -0.03, 0.0, 0.7)rad$ and the object is placed on the table such that the $x$ and $y$ positions of the object are uniformly distributed according to $[0.86m \pm 0.01m, 0.2 \pm 0.01m]$. When the robot

collides with the environment or the object, it enters a terminal state and receives a penalty of -250. In case the robot closes the gripper but doesn't grasp the object, it receives a penalty of -100. Additionally, when the gripper is closed and a grasp is not established, the robot receives a penalty of -700 if it doesn't execute the *openGripper* action. Each motion also incurs a small penalty of -3. When the robot successfully grasps the object, it receives a reward of 1,750 and enters a terminal state. The discount factor is 0.99 and we allow a maximum of 200 planning steps.

Similarly to the CarNavigation problem, the difficulty for this problem is the large number of steps that are required for the robot to complete its task (around 100). Additionally, the robot must act strategically when approaching the object to ensure a successful grasp.

### 3.5.3 Experimental setup

All four test scenarios and the solvers are implemented in C++ within the OPPT framework [3], ensuring that all solvers use the same problem implementation. For ABT we used the implementation provided by the authors [84]. For POMCP we used the implementation provided by `https://github.com/AdaCompNUS/despot`. Note that all three solvers rely on heuristic estimates of the action values in their rollout strategy. For a fair comparison, we use the same heuristic function for all three solvers, where we use methods from motion-planning, assuming the problem is deterministic.

All simulations were run single-threaded on an Intel Xeon Silver 4110 CPU with 2.1GHz and 128GB of memory. For the `Factory` and `KukaOffice` problem, we use the ODE physics engine [72] to simulate the transition dynamics. The levels $l$ used by MLPP in these scenarios are associated with the "discretization" (i.e., $\delta t$) used by the numerical integration of ODE. In particular, $\delta t = C_1 \cdot 2^{-C_2 l}$. For the scenarios `CarNavigation` and `MovoGrasp`, since the dynamics of these problems are simple, MLPP associates the levels $l$ to the time-step, i.e., $\Delta t = C_1 \cdot 2^{-C_2 l}$. The exact parameters (i.e., $C_1$, $C_2$, and the number of levels $L$) were determined via systematic preliminary trials. As a result of these trials, we set the parameters used by MLPP for `Factory` and `KukaOffice` to be $C_1$=0.0128, $C_2$=1, $L$=7, for `CarNavigation` to be $C_1$=0.4, $C_2$=1, $L$=3, and for `MovoGrasping` to be $C_1$=1, $C_2$=0.5, $L$=4.

The purpose of our experiments are three folds. First is to test whether our particular choice for the multiple levels of approximation of the transition functions results in a reduction of the variance of the $Q$-value difference terms in eq.(3.7). This ensures that, as we increase the level, fewer and fewer episode samples are required to accurately estimate the difference terms. To do this, we ran MLPP on each problem scenario for 10 runs with a planning time of 20s per step. Then, at each step, after planning time is over, we use the computed policy $\pi$ and sample 50,000 additional episodes from the current history $h$ on each level $l$ to compute the variance $\mathbb{V}[Q_l(h,a)]$ and 50,000 correlated episodes on each level $l$ to compute $\mathbb{V}[Q_l(h,a) - Q_{l-1}(h,a)]$, where $a$ is the action performed from $h$ according to $\pi(h)$. Taking the average of these variances over all steps and all simulation runs then gives us an indication how the variance of the $Q$-value difference terms in eq.(3.7) behaves as we increase the level of approximation of the transition function.

Second is to compare MLPP with two state-of-the-art POMDP solvers ABT [12] and POMCP [13].

For this purpose, we used a fixed planning-time per step for each solver, where we used 1s for the `Factory`, `CarNavigation` and `MovoGrasp` problem, and 5s for the `KukaOffice` problem. For each problem scenario we tested ABT and POMCP using different levels of approximations of $T$ for planning, to see whether using a single approximation of $T$ helps to speed-up computing a good policy, compared to MLPP that uses all levels of approximations of $T$ for planning.

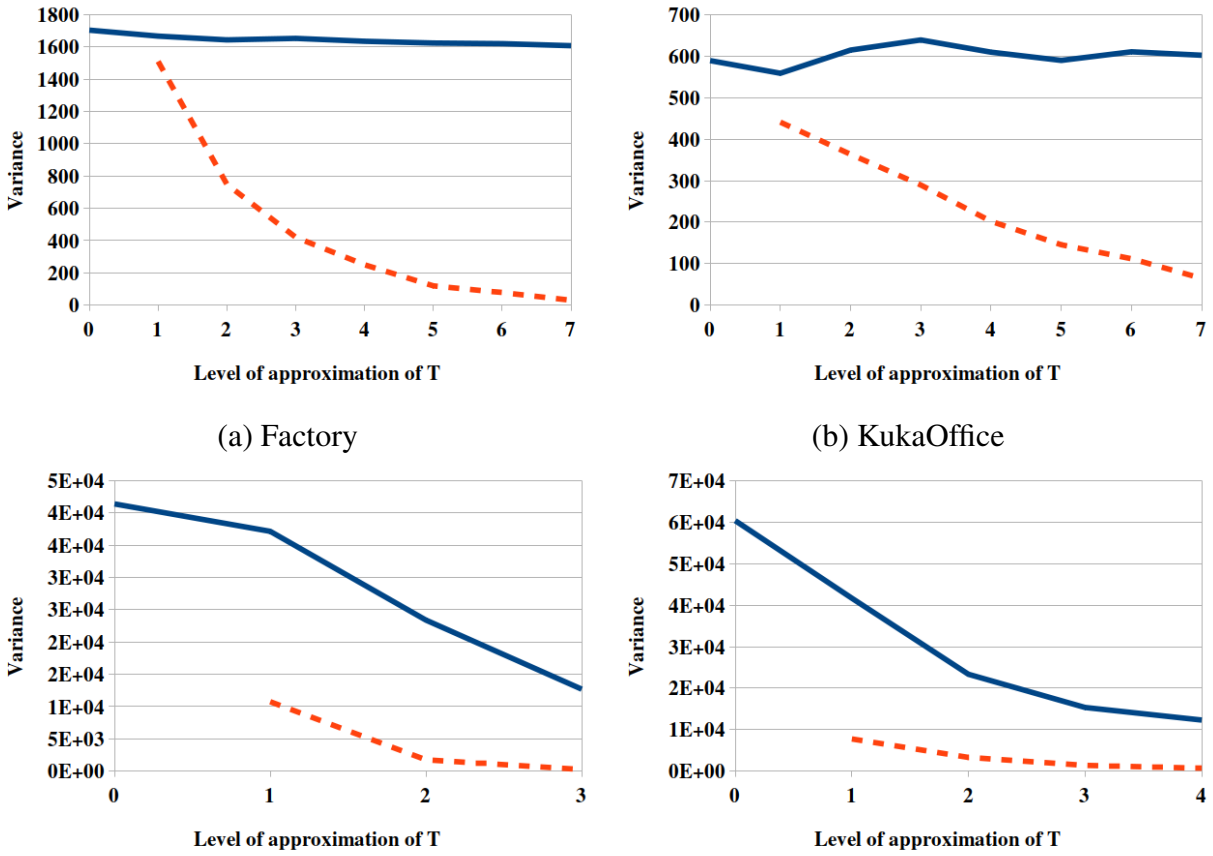
DESPOT [14] is not used as a comparator because for the type of problems we try to address, DESPOT's strategy of expanding each belief with every action branch (via forward simulation) is uncompetitive. For example, for `Factory`, expanding a single belief takes, on average, $\sim$14.4s using K=50 scenarios (50 is a tenth of what it commonly used [14]), which is already much more than the time for a single planning step in our experiments (1s). Similarly, for the long planning-horizon problem `MovoGrasp`, DESPOT must expand all 66 actions using K scenarios from every belief it encounters, which quickly becomes infeasible for a planning horizon of more than 5 steps.

Last, we investigated if and how fast MLPP converges to a near-optimal policy compared to ABT and POMCP, when the latter two solvers use the original transition function for planning. To do this, we used multiple increasing planning times per step for the `Factory` problem, starting from 1s to 20s per step. The results of all three experiments are discussed in the next section.

### 3.5.4 Results

#### Variances of $Q_l - Q_{l-1}$

Figure 3.2 shows the average variances of $Q_l$ and $Q_l - Q_{l-1}$ for all four problem scenarios. It is clear that in all scenarios the variance of the $Q$-value differences decreases significantly as we increase the level $l$, indicating that we indeed require fewer and fewer episode samples for increasing $l$. Note that the rate of decrease depends on the particular choice of the sequence of approximate transition functions. Multiple sequences can be possible for a particular problem, but preference should be given to the sequence for which the variance of the $Q$-value difference decreases fastest.




(a) Factory

(b) KukaOffice

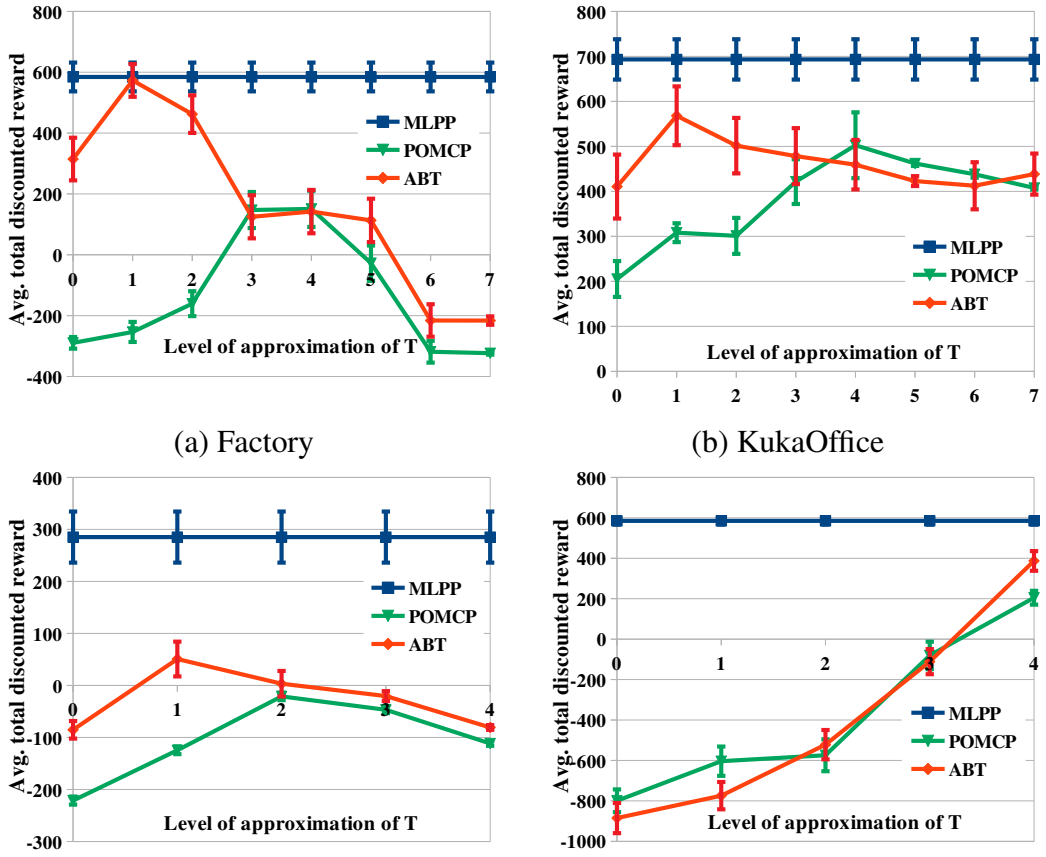
(c) CarNavigation

(d) MovoGrasping

Figure 3.2: Average variance of $Q_l$ (solid blue line) and $Q_l - Q_{l-1}$ (dashed red line) for the problem scenarios (a) Factory, (b) KukaOffice, (c) CarNavigation and (d) MovoGrasping. The $x$-axis represents the level $l$, whereas the $y$-axis represents the variance.

#### Average total discounted rewards

Figure 3.3(a)-(d) shows the average total discounted rewards achieved by ABT, POMCP and MLPP in all four test scenarios. The results indicate that, for ABT and POMCP, using a single coarse approximation of $T$ for planning can help compute a better policy, compared to using the original transition function. However, different regions of the belief space are likely to require different level of approximations. For instance in `Factory` and `KukaOffice`, when the states in the support of the belief




(a) Factory

(b) KukaOffice

(c) CarNavigation

(d) MovoGrasping

Figure 3.3: Average total discounted reward of MLPP, ABT and POMCP on the `Factory` (a), `KukaOffice` (b), `CarNavigation` (c) and `MovoGrasp` (d) scenarios. The $x$-axis represents the level of approximation of the transition function used for planning. Note that MLPP uses all levels for planning (hence the horizontal lines), whereas ABT and POMCP use only a single level as indicated by the $x$-axis. For each scenario, the largest level of approximation is equal to the original transition function. Vertical bars are the 95% confidence intervals.

place the robot in the relatively open area, coarse levels of approximation suffice but, when they are in the cluttered area, higher accuracy is required. Unlike ABT and POMCP, MLPP covers multiple levels of approximations and is able to quickly reduce errors in the estimates of the action values caused by coarse approximations. Subsequently MLPP consistently outperforms ABT and POMCP in these scenarios. The lack of coverage causes difficulties for ABT and POMCP in `MovoGrasping` as well, where a high accuracy is neccessary for grasping.

### Increasing planning times

Figure 3.4 shows the average total discounted rewards achieved by each solver for the `Factory` scenario as the planning time per step increases. The results indicate MLPP converges to a good policy much faster than ABT and POMCP: ABT requires 10 seconds per step to generate a policy whose quality is similar to the policy generated by MLPP in only 1 second per step, while POMCP is unable to reach similar level of quality, even with a planning time of 20 seconds per step (in our experiments it takes roughly 5 minutes of planning time per step for POMCP to converge to a near-optimal policy).

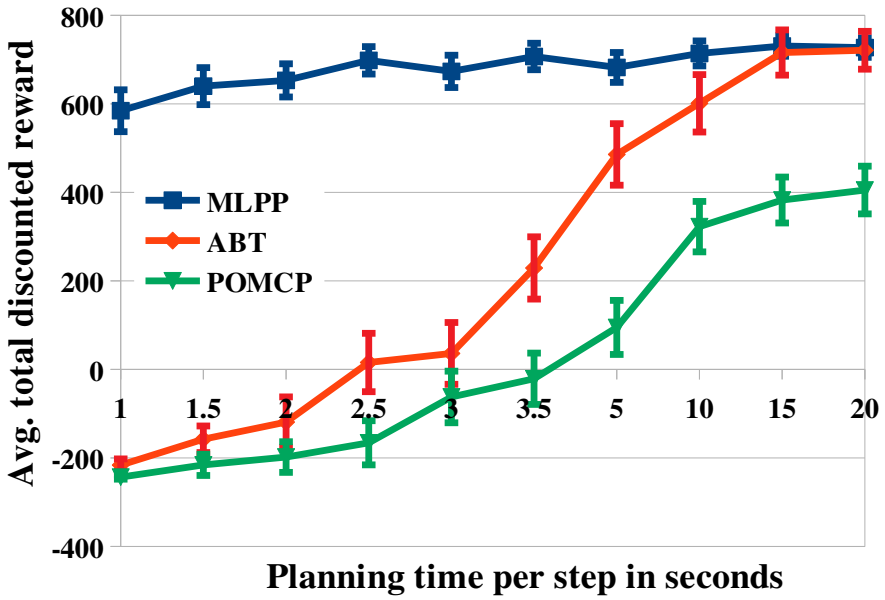



Figure 3.4: Average total discounted rewards for ABT, POMCP and MLPP for `Factory` using increasing planning times per step. The average is taken over 500 simulation runs for each planning time and algorithm. Vertical bars are the 95% confidence intervals.

## 3.6 Summary

Despite the rapid progress in on-line POMDP planning, computing robust policies for systems with complex dynamics and long planning-horizons remains challenging. Today's fastest on-line solvers rely on a large number of forward simulations and standard Monte-Carlo methods to estimate the expected outcome of action sequences. While this strategy works well for small to medium-sized problems, their performance quickly deteriorates for problems with transition dynamics that are expensive to evaluate and problems with long planning-horizons.

To alleviate these shortcomings, we propose MLPP, an on-line POMDP solver that extends Multilevel Monte-Carlo to POMDP planning. MLPP samples histories using multiple levels of approximation of the transition function and computes an approximation of the $Q$-values using a Multilevel Monte-Carlo estimator. This enables MLPP to significantly speed-up the planning process while retaining correctness of the $Q$-value estimates. We have successfully tested MLPP on four robotic tasks that involve expensive transition dynamics and long planning-horizons. In all four tasks, MLPP outperforms ABT and POMCP, two of the fastest on-line solvers, which shows the effectiveness of the proposed method.

The following publications have been incorporated as Chapter 4.

- [3] **M. Hoerger**, H. Kurniawati, and A. Elfes, A Software Framework for Planning under Partial Observability, in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 7576–7582

- [2] **M.Hoerger**, J. Song, H.Kurniawati and A.Elfes. POMDP-based Candy Server: Lessons Learned From a Seven Day Demo, in: Proc. AAAI International Conference on Autonomous Planning and Scheduling (ICAPS), AAAI, 2019, pp. 698-706

| Contributor | Statement of contribution | % |
|---|---|---|
| **Marcus Hoerger** | writing of text | 50 |
| | proof-reading | 40 |
| | theoretical derivations | 50 |
| | numerical calculations | 70 |
| | preparation of figures | 100 |
| | initial concept | 40 |
| Hanna Kurniawati | writing of text | 40 |
| | proof-reading | 40 |
| | supervision, guidance | 80 |
| | theoretical derivations | 40 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 40 |
| Alberto Elfes | writing of text | 0 |
| | proof-reading | 0 |
| | supervision, guidance | 20 |
| | theoretical derivations | 0 |
| | numerical calculations | 0 |
| | preparation of figures | 0 |
| | initial concept | 0 |
| Joshua Song | writing of text | 10 |
| | proof-reading | 20 |
| | supervision, guidance | 0 |
| | theoretical derivations | 10 |
| | numerical calculations | 30 |
| | preparation of figures | 0 |
| | initial concept | 20 |

Joshua Song was responsible for implementing the robot-perception system used for [2]. He made no contributions to [3].

# Chapter 4

# On-line POMDP Planning Toolkit

## 4.1 Introduction

In the previous two chapters we have seen that systematic approximations of complex, non-linear transition functions allows us to design on-line POMDP solvers that are more efficient than current state-of-the-art solvers. However, for real-world scenarios, modeling the problem as a POMDP is often a challenging and tedious process itself. This process involves modeling the kinematic and dynamic properties of the robot and the environment, as well as defining and implementing the transition, observation and reward functions of the problem. It is therefore desirable to have access to easy-to-use software tools that support the user in the modeling process.

Several software tools for solving POMDPs do exists, e.g., Symbolic Perseus [85], ZMDP [86], APPL [87], and TAPIR [84]. To use these solvers, a user needs to first encode the POMDP model of the problem. This encoding is easy for discrete POMDP problems: Users only need to list down the values of the components that define a POMDP problem in simple file formats, such as the Cassandra file format [88], PomdpX file format [89], and SPUDD format [90]. But, all software that can solve continuous POMDP problems or POMDP problems on-line require users to hard-code the problem in the software.

To alleviate the above difficulties we have developed On-line POMDP Planning Toolkit (OPPT), a software-toolkit for approximating POMDP solutions, on-line. OPPT uses a plugin-based framework to provide flexibility and ease for users to implement new POMDP models, without being tied to a specific uncertainty model. For general POMDP problems, the user can implement these plug-ins. However, for standard motion planning under uncertainty problems —that is, moving from one configuration to another with errors in the effect of actions and sensing—, OPPT provides a default POMDP model, such that users only need to specify 3D models of the robot and environment, and a configuration file that specifies parameters for the probability density functions that represent uncertainties in the effect of actions, observations, and starting state, and the reward function.

OPPT allows a user to separate the POMDP model (including the robot's environment) for planning and for simulated execution. It is known that developing a faithful POMDP model is often difficult.

However, it is also known that strategies computed with imperfect POMDP models can still generate relatively good robot behaviours. The ability to separate planning and execution environments will better facilitate sensitivity analysis studies of on-line POMDP solvers and allow users to better predict the performance these solvers in the physical world.

OPPT allows users to implement new POMDP solvers, too. For this purpose, OPPT provides an abstract and general POMDP solver class that is not restricted to specific data structures. Furthermore, users also have access to a rich framework that provides functionalities common for many robotic planning problems, such as kinematic computations, physical simulation (via ODE [72] in Gazebo [91]) of the robot and the environment it operates in and collision detection (via FCL [92]).

We tested OPPT on a complex manipulation task that involves a 6DOFs Kinova Jaco arm with KG3 gripper attached to the Kinova MOVO mobile manipulator platform. The task of the robot is to pick-up a cup from a table in front of the robot and use the cup to scoop candy from a candy box located next to the table. The rest of this Chapter is organized as follows: Section 4.2 provides an overview of the OPPT architecture and discusses some of its core components. In Section 4.3 we introduce the manipulation tasks we tested OPPT on. Here we also discuss a strategy that was developed for this problem to alleviate long-planning horizons. Additionally we provide implementation strategies that allow for a smooth operation of the robot with minimal delays between the planning steps.

## 4.2 Architecture

OPPT separates implementations of POMDP models from solvers. To ease the implementation of POMDP models, OPPT allows users to specify the state, action, and observation spaces via a configuration file, and uses a plug-in architecture to implement transition, observation, reward functions, and the initial belief. These plug-ins may have optional variables. The values of such variables are specified in the configuration file (the same file that specifies the three spaces of a POMDP problem). The details of the plug-in architecture are given in Section 4.2.2.

OPPT implements a POMDP model of a standard motion planning under partial observability problem (defined in Section 4.2.3), which means that for such problems, a user only needs to specify a configuration file. For more general problems, a user needs to implement the appropriate plug-in(s) as necessary and write a configuration file that specifies the state, action, and observation spaces, and plug-in options.

A new solver can be implemented via program API, as described in Section 4.2.3. The default solver in OPPT is Adaptive Belief Tree (ABT) [12].

Before discussing the details of how the POMDP models and solvers are implemented in OPPT, let's first discuss its overall architecture.

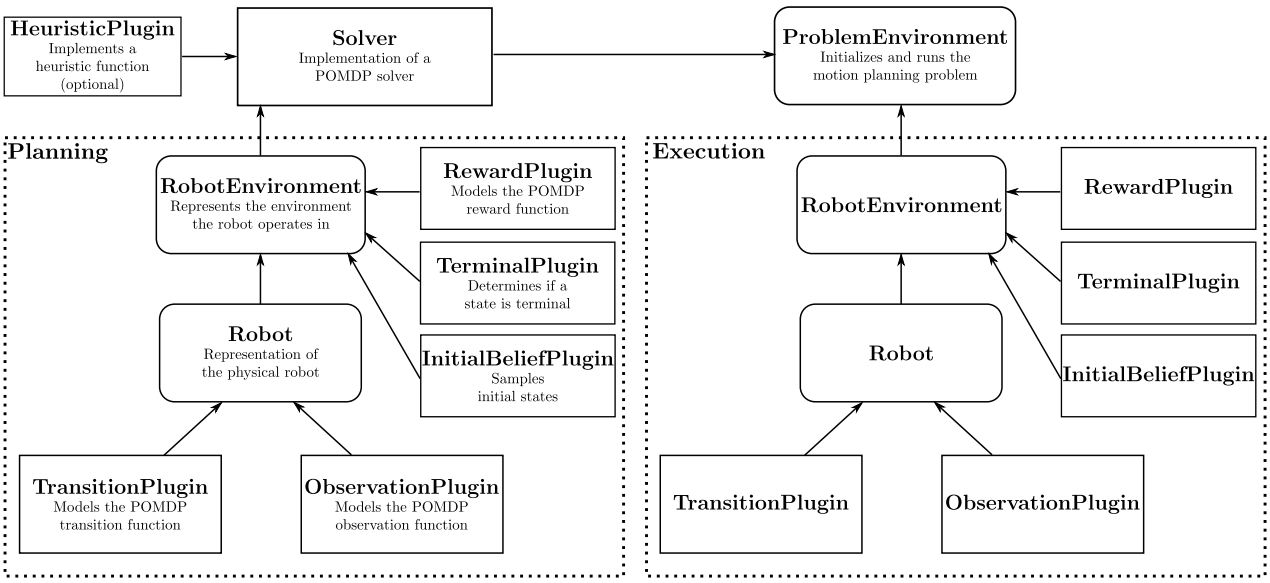



Figure 4.1: OPPT architecture overview. Rounded boxes represent core components of the OPPT framework. Square-shaped boxes represent components for which alternative implementations can be provided. A directed arrow from A to B depicts ”A is owned by B”

Table 4.1: Main classes of OPPT and their respective core methods

| Class | Key methods |
|---|---|
| *ProblemEnvironment* | `setup`, `setupSolver`, `runProblem` |
| *RobotEnvironment* | `getGazeboInterface`, `getScene`, `getReward`, `isTerminal`, `isValid` |
| *Robot* | `transitionState`, `getObservation` |
| *Solver* | `improvePolicy`, `getNextAction`, `updateBelief` |

### 4.2.1 Architecture Overview

The overall architecture of OPPT is shown in Figure 4.1. OPPT separates implementations of POMDP models and solver. Furthermore, for model implementation, OPPT allows separate models for planning and execution, although of course the same model can be used both for planning and execution. At the core of the model are the `ProblemEnvironment`, `RobotEnvironment` and `Robot` classes. These classes and their key-methods are described below, while Table 4.1 summarizes them.

#### ProblemEnvironment

The `ProblemEnvironment` is the main component of OPPT. It is responsible for initializing, setting-up, and running a POMDP problem (via the `setup`, `runProblem` methods), as well as loading and initializing an instance of the POMDP solver (via the `setupSolver`) that will be used to solve the specified POMDP problem. `ProblemEnvironment` follows the general work-flow of online POMDP planning, which includes the following high-level steps:

1. Use the `Solver` to improve the policy with respect to the current belief (via `Solver::improvePolicy`).

2. Get the best action to apply to the robot (via `Solver::getNextAction`).

3. Apply the action to the robot and sample an observation and a reward.

4. Inform the solver about the action taken and observation received to update its belief (via `Solver::updateBelief`).

5. Repeat steps 1–4 until a terminal state is reached, or a maximum number of steps is exceeded

After a simulation run is finished, the `ProblemEnvironment` generates an output file containing statistics about the simulation run.

### RobotEnvironment

The `RobotEnvironment` class represents an interface for the `Solver` to communicate with a `Robot`. It also contains a geometric representation of the environment and the robot (the latter via the `Robot` class). At start-up, the `ProblemEnvironment` instantiates two `RobotEnvironments`: One will be used for planning, while the other for simulated execution. This class also sets up the Gazebo interface. Many robot motion planning problems involve robots with complex non-linear dynamics and observation function that cannot be modeled in closed-form. For such problems, OPPT uses the ODE physics engine (via Gazebo) to compute the robot's dynamics. The interface thereby serves two purposes: It maintains a kinematic and dynamic model of the environment and the robot, using the underlying model structure of Gazebo, and provides simple methods that allow the transition and observation plug-ins to communicate with the physics engine (by obtaining a reference via the `RobotEnvironment::getGazeboInterface` method) to simulate environment, robot and sensor dynamics, or do collision checking with the maintained representation of the environment (by obtaining a refernce to the environment via the `RobotEnvironment::getScene` method). The `RobotEnvironment` class automatically sets-up Gazebo, based on a particular environment and robot model, and a state, action and observation description within the problem configuration file, to free the user from any additional set-up operations. Furthermore, this class provides methods to determine if a state is valid (via the `isValid` method) or a terminal state (via the `isTerminal` method), which use the `TerminalPlugin` and a method to sample rewards (`RobotEnvironment::getReward`) utilized by the `Solver` and the `ProblemEnvironment`.

### Robot

The `Robot` class is a general representation of the physical robot. It maintains the `TransitionPlugin` and the `ObservationPlugin` and provides methods (`transitionState` and `getObservation`) that allows the `Solver` to communicate with these plugins. The `Robot` class is also responsible to construct the underlying state, action and observation spaces of the robot. This is done automatically given a specific problem description.

### 4.2.2 Plug-In Architecture

OPPT uses a plug-in architecture similar to the one used in Gazebo [91], to provide flexibility in extending both POMDP model and solver. These plug-ins are implemented as shared libraries that are loaded dynamically during runtime.

In terms of the model implementation, plug-ins are used to implement the transition, observation, and reward functions, and the initial belief (and terminal states, if required) components of a POMDP model. OPPT provides one plug-in type for each POMDP component. For each plug-in, OPPT provides an implementation for the standard motion planning problem (see Section 4.2.3). Standard plug-ins that specify the transition and observation functions and initial belief defines only the types of the distributions. The parameters of the distributions are provided as inputs and are specified in the configuration file. This means that to implement a POMDP model for a standard motion planning under partial observability, a user only needs to write a configuration file that specifies the state, action, and observation spaces, and the parameters of the distributions.

Furthermore, since plug-ins can be exchanged during run-time, OPPT provides better support for changing environments, commonly encountered in long-term autonomy tasks. For instance, when the quality of a sensor deteriorates, one can replace the observation function, while OPPT is running.

For solvers that are derived from Monte-Carlo-Tree-Search (such as the default solver ABT), OPPT provides a heuristic plug-in that estimates the values of the leaf nodes of the tree.

### 4.2.3 Working with OPPT

#### Standard Motion Planning Problems

Table 4.2: State and observation configuration variables for motion planning problems

| | |
|---|---|
| *jointPositions,*<br>*jointVelocities* | Joint angles in rad, joint velocities in rad/s |
| *linkPoses,*<br>*linkPositionsX,*<br>*linkPositionsY,*<br>*linkPositionsZ,*<br>*linkOrientationsX,*<br>*linkOrientationsY,*<br>*linkOrientationsZ* | Poses, positions and orientations of the local link frames w.r.t. world frame |
| *linkVelocitiesLinear,*<br>*linkVelocitiesAngular,*<br>*linkVelocitiesLinearX,*<br>*linkVelocitiesLinearY,*<br>*linkVelocitiesLinearZ,*<br>*linkVelocitiesAngularX,*<br>*linkVelocitiesAngularY,*<br>*linkVelocitiesAngularZ* | Linear and angular velocities of the local link frames w.r.t. the world frame |
| *additionalDimensions* | Additional state dimensions that are not considered by the physics engine |

Table 4.3: Action configuration variables for motion planning problems

| | |
|---|---|
| *jointTorques* | Input for torque controlled joints |
| *jointPositions* | Input for position controlled joints |
| *jointVelocities* | Input for velocity controlled joints |
| *additionalDimensions* | Additional action dimensions that are not considered by the physics engine |

As discussed earlier, OPPT implements the model plug-ins for a standard motion planning under partial observability problem. In this problem, one or more robots must move from an initial configuration to a configuration in a goal region. The initial configuration is not known exactly, and is represented as a uniform distribution with bounded support. The exact bounds are given as input parameters. Actions and observations are disturbed by additive Gaussian noise, whose parameters are given as input parameters. The robot and sensor dynamics are simulated by the physics engine within Gazebo. These dynamics can be linear or non-linear. Therefore, despite additive Gaussian noise in the transition and observation functions, the beliefs may not be Gaussian (even if the initial belief was Gaussian). The robot's environment is fully observable, though it may change. The reward function is designed such that the robot receives a penalty when colliding with an obstacle, a small penalty for every step it performs and a large reward when reaching the goal area. The exact penalty and reward are given as input parameters. States are terminal when the robot collides with an obstacle or reaches the goal area.

The configuration file enables the user to specify an instance of this class of standard motion planning problems. Here, the user provides state, action and observation descriptions of the POMDP model. A list of state, action and observation descriptions the user can define are shown in Table 4.2 and Table 4.3. Additionally the problem configuration file specifies the set of POMDP plugins that will be loaded during runtime, and provides a reference to the robot and environment model files. Furthermore it is possible to define additional parameters that are being used by a specific `Solver` implementation.

For motion planning problem that fits the above description, a user can use the default plug-ins, specifies the state, action, and observations spaces and the input parameters in the configuration file. The kinematic and dynamic model of the robot and the environment the robot operates in are given as inputs, and defined using the SDF format [93], a XML-like descriptive format that contains a precise kinematic and dynamic description of the environment and the robot. These SDF-models are used by the `GazeboInterface` to initialize the underlying physics engine. Within the SDF-model files, the user can attach sensors to the robot that are used by the standard observation plug-in.

### General Planning Problem

For problems that do not fall into the class of standard motion planning problems, such as grasping, target-tracking and environmental exploration problems, the user can provide custom implementations of the plug-ins that define a particular POMDP problem. The plug-ins are designed such that only a small number of virtual methods have to be implemented. Each plug-in must implement the

`load` method, which is called after a plugin has been instantiated. Here, the user can perform set-up operations for any custom data structures that are maintained within a plugin. Additionally, a pointer to the `RobotEnvironment` is passed to the `load` method that can be used by a specific plugin implementation (e.g. for collision checking).

We emphasize that OPPT does not enforce a particular uncertainty model. Instead, the user has to define how states, actions and observations are affected by the uncertainties within the `transitionState` (for the transition plugin) and `getObservation` (for the observation plugin) methods. The user can also define their own distribution representation within the plug-ins.

### User Interaction

A benefit of the default solver, ABT, is that its policy can be adjusted when the environment changes. To reap this benefit, OPPT allows users to interact with the robot's operating environment using the Gazebo client GUI, during run-time. They can add and remove obstacles or change the pose of obstacles. If changes in the environment are known *a priori*, users can define them inside the configuration file. Here, the user specifies at which time step a specific change to the environment occurs. When the environment changes, the `Solver` will be informed about these changes via the `Solver::handleEnvironmentChanges` method. Implementing this method is optional, but if the user wishes to adapt the policy to the environment changes, the user must implement this method. This method has been implemented for our default solver.

For problems with fully observable environments, additional care is needed. Changes in the environment during run-time are always applied to the `RobotEnvironment` that is used by the `ProblemEnvironment` to execute a policy. However, it is possible reflect these changes in the planning environments as well.

For visualizing the motion planning progress, OPPT provides a lightweight standalone GUI that visualizes the 3D-environment, the current state of the robot, and the current belief during run-time. It is based on RViz [94], a visualization toolkit within the ROS framework [95]. This GUI can also be used to replay the output files that are generated by the `ProblemEnvironment` after each simulation run.

### Implementing New Solvers

In addition to reducing the difficulty of using on-line POMDP solvers, OPPT aims to ease implementation of new POMDP solvers. To this end, OPPT provides a general `Solver` interface, which is an abstract class that provides three key methods that must be implemented for new solvers.

The first method is `improvePolicy`, which is called by the `ProblemEnvironment` at each planning step. Within this method the `Solver` calculates the best policy from the current belief. Note that OPPT does not enforce a specific belief data structure. Depending on the solver, a belief can have very different representations, such as a set of particles or a multivariate-normal distribution. A `Solver` implementation therefore has to provide its own internal belief data structure.

The second key method is the `Solver::getNextAction`, which is called by the `ProblemEnvironment` after the `Solver::improvePolicy` method is finished. This method should return an action according to the calculated policy, such that this action maximizes the expected discounted future reward the robot receives when executing this action.

The third key method is the `Solver::updateBelief`, which takes the action the robot has performed and the observation that has been received as input arguments. In this method, the `Solver` performs a belief update according to the action and the observation. As mentioned above, a `Solver` is not restricted to a specific type of belief, therefore the user has to implement his/her own belief update functionality. OPPT provides an implementation of the Sequential-Importance-Resampling particle filter [63], which can be used when the belief is represented by a set of particles.

Apart from these three core methods, the `Solver` interface provides a set of optional methods for serialization and visualization.

## 4.3 CandyScooper

Figure 4.2: The problem scenario for the CandyScooper problem. A MOVO mobile manipulator equiped with a 6DOFs Jaco arm with KG3 gripper must pick-up at cup located on a table in front of the robot, use the cup to scoop candy from a candy box located next to the table, and put the cup back on the table.

As part of the Kinova demo program at the IEEE SIMPAR and ICRA 2018, we apply a general on-line POMDP solver, implemented in OPPT, as the sole motion planner of a 6DOFs manipulator performing a task that requires a relatively large number of steps to accomplish: The robot must move to pick-up a cup placed on a table, scoop candies using the cup, and put the cup back on the table (Figure 4.2 illustrates the task). In this scenario, the robot must trigger when scanning of the

environment is required and cope with ∼3cm perception errors, user's behaviour of moving the cup when the robot is trying to pick it up, and changes to the surface of the candies after each scoop.

We divide the above task into four motion planning problems: Picking up the cup, Approaching the candy box, Scooping, and Placing the cup back on the table. Each problem was modeled as a POMDP and solved using an on-line solver. We found that with almost naive modeling, existing solvers are capable in handling the size of the state and observation spaces posed by each problem.

However, for the picking problem, a major difficulty comes from the curse of history. Due to the various "playful" user's behaviour, a planner needs to perform a large number of lookahead steps to compute a robust strategy. Methods have been proposed to alleviate the curse of history issue [96, 97]. However generally, on-line POMDP solvers perform well for lookahead step of up to 10–15 steps (within reasonable computational resources), which is often insufficient for the above problem.

Moreover, the curse of history is amplified by the large action space imposed by a 6DOFs manipulator. Methods have been proposed to alleviate this issue [15, 46]. However, existing solvers can only perform well for problems with 3–4 continuous action spaces [15], while a method that can perform well for problems with 100,000 discrete actions was only recently proposed [98]. To alleviate this issue, we resort to use a small discrete number of fixed increments and decrements of the joint angles as the action space. This strategy helps to keep the size of the action space manageable, but at the cost of an increased planning horizon.

Here we propose to alleviate the curse of history issue by using macro-actions, represented as finite state machines. We also present the POMDP models for each of the four problems mentioned above and how they are combined together, so as to allow a smooth transition between problems. Furthermore, we present the perception system and how it is integrated with the POMDP-based planner, and some implementation tips and ricks to ensure smooth operation of the POMDP planner on a physical 6DOFs manipulator.

The above robotics scenario and system was demonstrated as part of a robotics demo that ran for a total of seven days. During this time, 150 runs were attempted, and our solution reached 98% success rate. After the demo, we slightly improved the system and conducted more systematic experiments, in particular in scenarios where a user displaces the cup that the robot tries to pick-up and in cluttered environments. Experiments in such scenarios indicate that our solution reaches no less than 90% success rate.

### 4.3.1 The System

The manipulator we use is a Kinova 6DOFs Jaco arm with KG3 gripper, attached to the Kinova MOVO mobile manipulator platform. In this work, we use only one of the arms of the MOVO and viewed the MOVO as a static base for the Jaco arm. The sensor used is the one attached to the MOVO, i.e., Kinect v2 mounted on the MOVO's head and encoders at the joints of the Jaco arm. By default, MOVO comes with a non-calibrated Kinect v2 sensor. We perform internal calibration of the sensor, but not the external calibration. We found that the error due to no external calibration causes model and perception

error that can easily be handled by our solver.

MOVO runs the Robot Operating System (ROS) and comes with two Next Unit of Computing (NUC) computers. Each NUC is an Intel Core i7-5557U@3.1GHz machine with 16 GB RAM. One of these computers is used for sensor processing, while the other is used to process control input and sensor scheduling. We use an off-board computer to run OPPT and the solver. The off-board computer is an Intel Xeon E5-1620@3.6GHz with 16 GB RAM. This use of off-board computing is due to different ROS versions used by the solver and the interfaces to the robot.

### 4.3.2 Problem Scenario and Formulation

We consider a manipulation task in which the above Jaco arm interacts with its environment while being subject to significant uncertainties in motion, sensing and its understanding of the environment. Specifically, the manipulator must grasp and pick-up a cup (with known geometry) that is placed at an arbitrary position on a table in front of the robot, use the cup to scoop candies from a box of candies, and then place the cup containing candies back on the table. The location of the table, the location of the cup on the table, and the location of the box of candies are initially unknown. The average perception error in the object localization is 3cm. Furthermore, the cup location can change at run-time, e.g. a user might pick-up the cup and place it at a different location on the table while the robot is moving to pick it up, whereas the surface of the candies in the box changes after every scoop.

We divide the above task into 4 motion planning problems, i.e.:

**Picking:** Moving the manipulator from its initial configuration to pick-up a cup from a table, despite not knowing the exact initial position of the table and the cup, possible changes of the cup's position during run-time, and perception errors.

**Pre-Scoop:** Moving the cup to a pose above the candy box that is beneficial for scooping

**Scooping:** Scooping candies from a box using the cup, despite not knowing the exact surface of the candies and perception errors

**Placement:** Placing the cup back to the table, while ensuring that no candies fall out of the cup.

To enable robust operation despite the various types of uncertainties, we formulate each problem as a Partially Observable Markov Decision Process (POMDP). The different POMDP problems are designed to have overlapping state variables, so that the end of one problem automatically transition into the beginning of the next problem with ease.

#### POMDP Formulation of the Picking Problem

In this problem, the state space is a joint product between the robot's and the cup configurations. Specifically, the state space is defined as $S = \Theta \times GripperStates \times GraspStates \times \Phi_{obj}$, where $\Theta = (-3.14rad, 3.14rad)^6$ are the joints angles of the arm, $GripperStates = \{gripperOpen, gripperClosed\}$ indicate whether the gripper is open or closed, and $GraspStates = \{grasp, noGrasp\}$

indicates whether the robot has successfully grasped the cup or not. A successful grasp means that the gripper grasps the cup from the side, so as to help accomplish the scooping problem. This condition can be checked easily because the geometry of the cup to be picked is known a priori. The last component, $\Phi_{obj} \subseteq \mathbb{R}^6$ is the set of all poses of the cup in the workspace of the robot. The orientation of the cup is represented as Euler angles, for compactness. Note that in this problem, we do not worry about gimbal lock issues because, the cup does not move until it is in contact with the robot and once in contact, rather than simulating the cup's motion, we derive the pose of the cup based on the robot's configuration when contact happens.

In this problem we do consider obstacles. However, to keep the size of the state space manageable, we do not include the obstacles as part of the POMDP state space. Uncertainties in the estimate of the poses and shapes of the obstacles are dealt with conservatively, by constructing bounding boxes around the obstacles, and enlarging them by the average localization error.

To keep the action space small, the set of primitive actions consists of the set of fixed angle increments/decrements for each joint (denoted as $A_\theta \subseteq \mathbb{R}^6$), plus `scan`, `openGripper` and `closeGripper` actions, resulting in a discrete action space of size $2^n + 3$, where $n$ is the robot's DOFs. The `scan action` takes $\sim$1.5s, while all other primitive actions take $\sim$0.7s to be executed. In subsection **Alleviating the Curse of History**, we discuss how this set of primitive actions is augmented with macro-actions, so as to reduce the effective planning horizon, thereby alleviating the curse of history issue.

The joint angles of the robot evolve linearly according to

$$\theta' = \theta + a_\theta + e_\theta \tag{4.1}$$

where $\theta, \theta' \in \Theta$ are the current and next joint angles, $a_\theta \in A_\theta$ is a vector of joint angle increments/decrements and $e_\theta \sim N(0, \Sigma)$ is an additive control error drawn from a multivariate Gaussian distribution. The parameters are set through experiments with the physical system. Note that this component of the transition function is used throughout the subsequent POMDP problems described in the next subsections. The actions `openGripper` and `closeGripper` are assumed to be perfect. To determine if the `closeGripper` action results in a successful grasp, we use a simple threshold based-method: Executing the `closeGripper` action corresponds to moving the finger-joint encoders from an opening angle of $0.0rad$ to a closing angle of $1.0rad$. If the resulting closing angles are less than $0.95rad$, we assume that the gripper was not fully closed, i.e. a grasp is established. Otherwise we assume that a grasp was unsuccessful.

The observation space is a joint product of four components. The first component is an estimate of the pose of the cup, which is computed by our perception system (discussed in section **Perception**). This observation component is perceived only when the `scan` action is performed. This component is continuous but, for the purpose of solving, we uniformly discretize the space, in the sense that two observations will be considered the same if their L2 distance is less than a pre-defined threshold. The second component comes from the joint-encoders, which measure the joint-angles of the arm. Similar to observation w.r.t. the cup's pose, we uniformly discretize observation regarding the joint-angles

of the robot. The third component is a gripper-encoder that indicates whether the gripper is open or closed, and the last component is a grasp detector which observes whether the robot grasps the object or not.

For the observation function, we assume that the gripper-encoder sensor provides perfect information. For the grasping sensor we assume to get a correct reading 90% of the time. For the joint-angle sensor we assume that the encoder readings are disturbed by a small additive error drawn from a uniform distribution with support $[-0.05rad, 0.05rad]$. To accommodate for possible errors in the observations of the object pose, we assume that these pose estimates are drawn from a uniform distribution around the actual pose of the object. Note that the robot only receives an observation on the object pose when a *scan* action is performed.

The POMDP agent receives a reward of 1,750 for a successful grasp and a reward of 1,000 for successfully bringing the the cup to the goal region, which is located slightly above the table. We set a penalty of -250 for every self-collision or collision with the environment, i.e., the table, the cup, and obstacles. Collision with the cup happens whenever the robot touches the cup with any part other than the inside parts of its gripper. Each motion also incurs a small penalty of -3, to encourage the robot to accomplish the task sooner than later. Furthermore, a penalty of -100 is given to failed grasps, to ensure that the cup is firmly grasped, so that the subsequent problem (i.e., scooping) can be accomplished.

In addition to the above rewards, in this particular problem, we added the following three reward components to act as a heuristic to help guiding the search. In case the gripper is closed and a grasp has not been established, the robot receives a penalty of -700 if it does not execute the `openGripper` action. This forces the robot to immediately re-open the gripper after an unsuccessful grasp attempt. On the other hand, in case a grasp has been established, the robot receives a penalty of -250 if the gripper is opened. Additionally, the robot receives a penalty of -200 for states in which the object is ”behind” the gripper.

#### POMDP Formulation of the Pre-Scoop Problem

Here, the state, primitive action, and observation spaces, as well as the transition and observation functions are the same as for the picking problem. Since the curse of history of this problem is not as severe as the picking problem, the action space of this problems consists of only the primitive actions. During the initial scan, the robot obtains an estimate of the location of the box that contains the candy, as well as an estimate of the height of the candy surface. Based on the estimated location of the box, we construct a small goal-area above the box that must be reached by the robot, such that the cup is inside the goal-area and it faces the candy surface.

Furthermore, the reward function for this problem is simpler than the one used for the picking problem: The robot receives a penalty of -250 when it collides with itself or the environment, and a small penalty -3 for every step it takes. Additionally, the robot receives a large penalty of -250 when the `openGripper` action is executed, as this would result in loosing the grasp. If the robot successfully moves the cup to the goal-area such that the cup faces the candy surface, it receives a reward of 1,000.

#### POMDP Formulation of the Scooping Problem

In this problem, the robot has to move the cup beneath the candy surface, scoop the candy, and leave the candy box without spillage. The state space consists of the same state variables as the previous two problems, plus two additional variables. The first variable represents the height of the candy surface, so as to account for uncertainty in the candy surface estimate. In the initial belief, this variable is distributed according to a uniform distribution $[-3cm, 3cm]$ around the (point) estimate of the height of the candy surface provided by the perception system. Estimating the height of the candy surface is performed once, prior to execution. The second variable, `enteredCandyBox`, is a boolean that indicates whether the robot has moved the cup beneath the candy surface. The action and observation spaces, as well as the transition and observation functions are the same as the pre-scoop problem.

To achieve scooping we separate the reward function into two parts: The part where the cup hasn't entered the candy box yet and the part where the cup is beneath the candy surface. For the first part, the robot receives a reward of $1,000$ if the cup enters the candy box such that the opening of the cup faces the candy surface. For the second part, the robot receives a reward of $1,000$ once the cup emerges in a near upright orientation. Additionally we penalize backward-motions by $-50$ as this would prevent the robot from filling the cup with candy. With this simple reward function, the robot is encouraged to perform a scooping-like motion. The robot receives another reward of $1,000$ once it reaches a goal area located above the candy box and holds the cup at a near upright orientation. To avoid spillage, the robot receives a penalty of $-500$ if the orientation of the cup is more than a pre-defined threshold from an upright orientation.

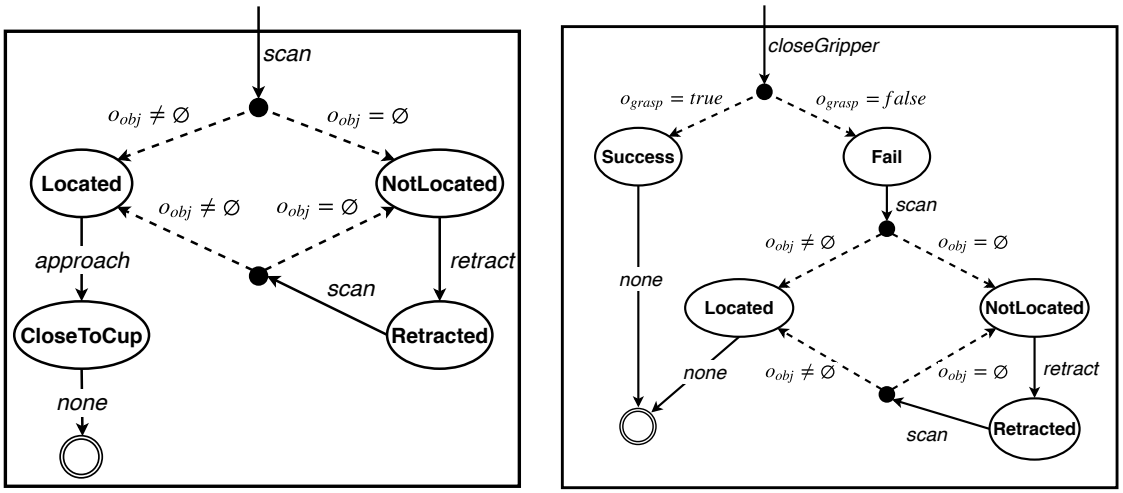
#### POMDP Formulation of the Placement Problem

In this problem, we use the same state, action and observation spaces and the same transition and observation functions as in the pre-scoop problem. In terms of the reward function, similar to the scooping problem, we need to make sure that the cup faces upwards to avoid spillage. Hence the robot recieves the same penalty of $-500$ if the orientation of the cup about the $X$ and $Y$ axis exceeds 10 degrees. Then, once the robot has reached a state such that the cup is inside the goal area above the table, it receives a reward of 1000 for the `openGripper` action. At the same time, after performing the `openGripper` action, the robot enters a terminal state which concludes the task.

### 4.3.3 Planning

To solve the aforementioned POMDP problems, we use the standard solver of OPPT, Adaptive Belief Tree (ABT), detailed in Section 2.4.1. Note that we chose ABT as the solver due to its stable code-base at the time of modeling the problem. However, in principle any other on-line solver could be used instead, such as MLPP proposed in Chapter 3.

The following three subsections describe some implementation details that enable ABT to generate a good strategy to solve the manipulation problems set in this work.




(a) `scan-and-approach` (b) `grasp-and-rescan`

Figure 4.3: (a) and (b): Finite state machines of the macro-actions. Ovals are machine-states, labeled solid arrows are actions, dashed arrows are observations, and double circles are exit states. $o_{obj}$ are observations with respect to the pose of the cup, whereas $o_{grasp}$ are observation from the grasping sensor.

### Alleviating the Curse of History

The curse of history issue in this work is particularly severe in the picking problem. In this problem, the robot must cope with possible changes of the cup's position during run-time, which means it has to consider multiple additional sensing actions to re-localize the cup. However, such a re-localization becomes relevant only when the lookahead step performed reaches beliefs with substantial probability mass of being in states where the gripper is close enough to the cup, that attempting to grasp the cup is not futile, which in general, requires sufficiently long lookahead steps from when a scanning action should take place.

To alleviate this curse of history issue, we propose macro actions, represented as finite state machines, that augment the action space of the POMDP problem. In particular, we propose two macro actions ( the finite state machines representations are illustrated in Figure 4.3):

1. The `scan-and-approach` macro action, which consists of two sub-actions: A `scan` action to localize the cup, and an `approach` action, which is a sequence of primitive actions that attempt to bring the gripper close to the estimated location of the cup. For the `scan` action the robot queries the perception system to receive an estimated object pose, whereas the `approach` action utilizes a deterministic motion planner.

2. The `grasp-and-rescan` macro action. This macro action is designed to hedge failed grasps. A failed grasp could occur due to two reasons: There is too much uncertainty with respect to the object pose, or the object pose has changed during run-time.In both cases the robot must re-localize the object before attempting another grasp. Hence, if a grasping attempt during execution of the `grasp-and-rescan` fails, the robot immediately performs another `scan` action. However, if grasping attempt is successful, the robot continues with the next action.

In both macro actions, during a `scan` action, if the cup is occluded by the arm, such that the robot does not "see" the object, the robot performs a `retract` action, utilizing a deterministic motion planner to compute a sequence of primitive-actions that brings the arm to a configuration where the object is more likely to be occlusion-free. The transition function for the macro actions are derived from the transition functions of the primitive actions.

Note that to speed up planning, our solver does not run the deterministic planner when `approach` or `retract` action are selected at planning time. Rather, it computes a sampled set of various different joint angle values ($\subseteq \Theta$) and corresponding gripper pose, off-line. During on-line planning, it samples a joint angle value associated with a gripper pose nearest to the desired gripper pose, and assumes that this sampled joint angle is the result of executing the `approach` or `retract` action. During execution, a deterministic motion planner is used to find a sequence of primitive actions to move the robot towards the assumed resulting joint angles. This strategy performs well because we only require the `approach` or `retract` action brings the gripper reasonably close to the object pose estimate.

### Rollout Strategy

ABT and most on-line solvers today use a rollout strategy as a heuristic to estimate the value of $Q(b,a)$, when the node $b$ is first expanded via action $a \in A$. In practice, this heuristic is critical for the performance of ABT, particularly when the belief tree is still shallow. A good heuristic helps ABT to focus on the most promising parts of the tree and therefore converge to a good policy faster.

A commonly used rollout strategy is a greedy approach, i.e., at each rollout step, the planner selects actions with the highest immediate reward. However, two issues arise with this strategy. First, it is too myopic for our problem. Second, it requires simulations to be run at each rollout step, which can be quite expensive. Therefore, we propose to use a heuristic that does not require repeated simulation runs to estimate the aforementioned $Q$-value. Suppose we need to estimate $Q(b,a)$ for the first time. We first sample a next state $s' \in S$ according to $T(s,a,s')$. Let $\theta \in \Theta$ be the joint-angle component of $s'$. The heuristic estimate $\widehat{Q}(b,a)$ is:

$$\widehat{Q}(b,a) = R_{max} * exp(-\lambda * d_{pos}(g, h(\theta))) \tag{4.2}$$

where $R_{max}$ is the maximum possible immediate reward, $\lambda$ is a scaling factor, $h$ is a function that maps the joint angles $\theta$ to a gripper pose in the robot's workspace and $d_{pos}$ is the Euclidean distance function. The notation $g \in \mathbb{R}^3$ is a goal position that is different for each of the four POMDP problems. For the picking problem $g$ refers to the position of the cup in the robot's workspace, whereas for the remaining problems $g$ refers to the position of the goal areas. With this heuristic, we favor expanding nodes where the gripper is closer to the goal position of the respective problem, and therefore help the solver to quickly focus on parts of the belief tree where the gripper moves towards the goal.

### Reducing Delay between Steps

Now, the naive implementation of ABT, or any other on-line POMDP solver, follows a strictly sequential order of execution, i.e., policy computation – policy execution – belief update. Such an

implementation is likely to result in significant delays during execution, as the robot would have to wait for the solver to update the belief and compute a policy for the new belief. To substantially reduce such delays, we parallelize these tasks where possible by running two processes at the same time.

The first process is the belief-update process. Recall that in ABT, beliefs are represented by sets of particles that are updated using a particle filter. In our implementation we use the Sequential-Importance-Resampling (SIR) particle filter [63]. SIR particle filtering consists of two steps. First, drawing samples from a proposal distribution, which in our case, $s'_k \sim T(s_k, a_k, S)$ where $s_k$ are particles that are sampled from the current belief and $a_k \in A$ is the action currently performed. Second, updating the importance weights of the samples $s'_k$ up to a normalization constant based on the observation $o_k \in O$ perceived, which in our case, $w'_k = w_k Z(s'_k, a_k, o_k)$. Generating samples from the proposal distribution is often quite expensive. However, we can start drawing these samples once the robot starts executing $a_k$. Then, once the robot receives an observation, all that remains for the belief update is to update the importance weights which can be done fast.

The second process is the policy-update process. Suppose the current belief maintained by the solver is $b$ and $a_k \in A$ is the action that the solver has estimated to be the best to perform from $b$. In our implementation, once the robot starts executing $a_k$, our implementation of ABT will start planning for the next step. The planning time is set to be 0.7s (the smallest execution time for a primitive action) or until the current action finishes execution, whichever is higher. During planning, ABT will sample additional episodes, starting from states sampled from $b$ and performing action $a_k$ as its first action, thereby improving the policy within the entire descendent of $b$ via $a_k$ in the belief tree. This strategy increases the chances that after the robot has executed $a_k$ and the belief is updated based on the observation perceived, a good policy for the next belief is readily available.

Of course, there are cases where even with this strategy, the robot perceives observations that were not explored in the belief tree. In such cases, we restart planning from scratch. However, we found that the above parallel implementation of ABT is sufficient to reduce the delay between perceiving an observation and executing the subsequent action to be under 0.1 sec in all our experiments.

### 4.3.4 Perception

MOVO is equipped with a Kinect v2 sensor mounted on a pan and tilt actuator. This mechanism is only used to scan the entire scene (i.e., the table and candy box) at the beginning of the execution. The action `scan` uses a pre-defined position and orientation of the sensor, set to ensure that each `scan` action covers the entire workspace of the particular POMDP problem.

The Kinect sensor provides RGB and depth images. We use both the RGB image and a point cloud obtained from the depth image to detect and localize the table where the cup and obstacles are located, the cup and obstacles themselves, the candy box location, and the height of the remaining candy in the candy box. The two key components of the perception system are a Convolutional Neural Network (CNN) that processes RGB images, and a point cloud algorithm.

To detect the cup, we first trained a CNN, for which we used the SSDLite-MobileNetV2 [99]

architecture, via Tensorflow [100]. The CNN was pre-trained on the COCO [101] dataset and then fine-tuned for 200k steps on 200 photos of the cup. However, this CNN only provides a rough estimate of the cup's pose. To get a better estimate, we first crop the Kinect point cloud around the CNN pose estimate, which results in a much smaller point cloud. We then convert the cup's CAD model to an object point cloud, randomly sample object poses close to the CNN estimate and transform the object point cloud to each sample. The final pose estimate is then the pose sample whose transformed object point cloud is closest (in terms of the sum of pairwise distances) to the cropped Kinect point cloud.

To detect obstacles on the table, we remove points corresponding to large planes (i.e. the table) and use a clustering algorithm on the remaining points. We then fit 3D bounding boxes around each cluster to estimate the poses and dimensions of the obstacles. For the localization of the candy box, we use a fiducial marker and information about the size of the box.

In an initial experiment we found that the average error with respect to the pose of the cup is approximately 3cm. Furthermore, the pose estimate can be increasingly unpredictable if the cup is too close to the Kinect. Even factors such as sunlight, varying sensor temperature, sharp object curvature, and highly reflective surfaces decrease the quality of the pose estimate. In principle we could reduce the error by performing external calibration. However, since this is often a tedious process, we instead chose to let the POMDP solver handle these errors.

### 4.3.5 Results

To evaluate our system, we implement our models and strategies using OPPT and ran three sets of experiments on the Kinovoa Movo mobile manipulator platform, equipped with a 6-DOF arm and gripper. The first set evaluates the effectiveness of our system in picking up the cup when the position of the cup is changed during run-time. The second set evaluates our approach in increasingly cluttered environments. In the last set of experiments, we evaluate our system for the entire scenario in a live demo setting.

At the start of each run, the user places the cup at an arbitrary position on the table that is unknown to the robot a-priori. For the first two sets of experiments, a run is considered successful if the robot picks-up the cup and unsuccessful if the robot pushes the cup off the table or pushes the cup to a position where it lies outside the observation range. Runs in which one or more pick-up attempts fail, but the robot is able to recover and eventually pick-up the cup are considered successful. For the third set of experiments, a run is considered successful if the robot is able to pick-up the cup, scoop candies from the candy box and deliver the cup back to the table without spillage or collisions with the environment.

### Changing Cup Positions

In this set of experiments, we actively change the position of the cup during run-time by placing it at random positions on the table. For this we divided the experiments into 20 runs with 0, 2, 4 and 6 cup position changes. The occurrence of these cup position changes are random as well. Figure 4.4 shows snapshots of a typical run where we change the cup location twice during runtime.

Figure 4.4: These figures show snapshots of a successful run with two obstacle position changes. Here the robot attempts to pick-up the cup (first picture) but the cup position is changed to the left corner of the table (second picture). It performs a `scan` action, and the robot attempts to pick up the cup again. But, as the robot attempts to pick-up the cup at the new position, the cup is again moved to a location close to the robot (third picture). A `scan` action is performed again. In the last picture the robot successfully picks-up the cup at its the new location.

Table 4.4 summarizes the results. In general, the robot showed robust behaviour in picking-up the cup. It successfully handled multiple initially unknown changes in the cup position, occlusions of the cup caused by the arm and imperfect information regarding the environment. Out of the 20 runs, only 2 were unsuccessful. In one of the failure cases, querying the perception system using the `scan` action resulted in a wildly incorrect cup pose estimation. Subsequently the robot attempted to pick-up the cup at a very different location, and in doing so it pushed the cup off the table. In the second unsuccessful run, the cup slipped, toppled, and rolled off the table during a pick-up attempt.

To obtain a better understanding about the utility of the macro-actions for robustly picking up the cup, we ran an additional experiment where the robot has no access to the macro-actions.

We tested our system using 5 execution runs where for each run, we changed the position of the cup once during run-time. In none of the runs the robot was able to pick-up the cup. This is because due to the limited lookahead, the robot couldn't "see" the benefit of performing the `scan` action to re-localize the cup after an unsuccessful pick-up attempt, hence the `scan` action was never used. This indicates the importance of reducing the effective planning-horizon in this problem.

### Cluttered Environments

Similar to the previous set of experiments, the task is to pick-up the cup, but now the robot has to additionally avoid collisions with obstacles. This makes the problem significantly harder, since the robot has to plan how to negotiate the obstacles without compromising a successful grasp. For this experiment we do not manually change the cup position. We tested our system using two scenarios, a scenario with one obstacle and a scenario with three obstacles. Figure 4.5 shows the scenarios used for this set of experiments.


(a) One obstacle (b) Three obstacles

Figure 4.5: The problem scenario used for the set of experiments with one (a) and three (b) obstacles. The task for the robot is pick-up the cylindrical cup while avoiding collisions with the two boxes and the black cup.

We tested the system in both scenarios using 20 execution runs. For the scenario with one obstacle, 19 out of 20 execution runs (i.e. 95%) were successful, i.e. the robot was able to pick-up the cup while simultaneously avoiding collisions with the environment. For the scenario with 3 obstacles, the robot managed to pick-up the cup in 19 out of 20 execution runs as well (95%). During the two failure cases the robot collided with the white carton (see Figure 4.5) causing it to tip over. This was caused by the use of a conservative approach in handling the uncertainties with respect to the obstacle poses and dimensions. The high success rate indicates that the robot exhibited robust behaviour in scenarios with increasingly cluttered environments.

### Live Demo

To test the entire planning process, from picking up the cup, approaching the candy box, scooping the candy to delivering the cup back to the table, we ran the entire system for 7 consecutive days in a live-demo setting. For this we let bystanders place the cup at random positions on the table ahead of every execution run. In approximately 150 runs we achieved a success rate of over 98%, i.e. the robot was able to pick up the cup, scoop candy and deliver the filled cup back to the table. This shows the effectiveness of our approach in solving the entire planning problem in a non-sterile setting.

| Num cup position changes | % of successful runs |
|---|---|
| 0 cup position changes | 100 |
| 2 cup position changes | 95 |
| 4 cup position changes | 100 |
| 6 cup position changes | 90 |

Table 4.4: Percentage of successful runs for scenarios with 0, 2, 4, and 6 cup position changes. For each scenario the robot performed 20 execution runs

### 4.3.6 Comparison with Deterministic Motion Planner

For comparison, we apply a commonly used motion planner for high DOFs, RRT-Connect [18], on the picking problem in an environment without obstacles where the cup's pose is stationary. Given the initial state of the robot, RRT-Connect computes a trajectory in the robot's state space that moves the gripper towards the most-likely estimate (from the perception system) of the cup's pose, before attempting to pick-up the cup. Note that for this experiment, we are only interested in how effective RRT-Connect is in achieving a successful grasp. Therefore we do not optimize or smoothen the computed trajectories. The RRT-Connect-based picking is executed $20\times$. Among them, only 7 runs resulted in a successful pick-up (compared to a 100% success rate of our POMDP-based planner, see Table 4.5). While RRT-Connect is reasonably effective in moving the gripper towards the estimated cup-pose, due to the perception errors, the resulting gripper pose where a grasp is attempted is often unsuitable to pick up the cup. In contrast, the policies computed by our POMDP-solver result in the robot to carefully approach the cup as to reduce uncertainty with respect to the pose of the cup, e.g. by slightly pushing the cup before attempting to pick it up.

Furthermore, a POMDP-based planner is able to automatically identify actions that can reduce uncertainty while accomplishing its goals. Such motion does not always have to be explicit scanning, especially when scanning is expensive (as in our problem). For instance, the accompanying video at 00:13 and 00:43 shows that instead of moving straight to the most-likely location of the cup and attempt to grasp it, the robot carefully approaches the cup and "pushes" it slightly a few times before eventually grasping it. This strategy reduces uncertainty on the relative position of the cup w.r.t. the hand, enabling more robust grasps compared to RRT-Connect-based approaches.

## 4.4 Summary

In this section we present OPPT, an open-source software framework for on-line POMDP planning. Current software tools for on-line POMDP planning are either limited to a specific solver, or require problems to be hard-coded within the provided implementation. OPPT alleviates both limitations by providing a rich framework for the standard class of motion planning under partial observability problems and a plug-in based architecture. It provides a general API for developing on-line POMDP solvers further, and implements ABT [12] —an on-line POMDP solver that can adapt its solutions to changes in the POMDP model— as its default solver. OPPT allows users to specify a POMDP model via plug-ins and a simple configuration files. Furthermore, for the standard class of motion planning problems, users can specify a POMDP model with no coding effort, via a configuration file and 3D models of the robot and the environment.

We have successfully applied OPPT and its default solver ABT as the sole motion planner of a relatively long manipulation task. A 6DOFs Jaco arm must pick-up a cup from a table, use the cup to scoop candies from the candy box, and place the cup back on the table without spillage. Here the robot is subject to significant perception errors regarding the location of of the cup, the obstacles and

the surface of the candy. Additionally uncertainties stem from "playful" users that keep moving the cup while the robot tries to pick it up. Our POMDP-based system, implemented in OPPT, was tested in a 70 hours live demo and achieved a success rate of 98%. Further experiments to test robustness demonstrate promising results too.

We hope OPPT reduces the difficulty of applying recent advances in POMDP-based planning to complex and realistic robotic tasks under partial observability. We also hope this software will encourage and support the community to further extend the capabilities of POMDP solvers, and decision making under uncertainty in general, so as to address a major bottleneck for reliable and robust autonomous robot operations.

# Chapter 5

# Conclusion

## 5.1 Summary

Planning under partial observability is an essential capability of autonomous robots. Framing such planning problems as POMDPs allows for mathematically principled solutions. Even though solving POMDPs exactly is computationally intractable in general, over the past few decades we have seen substantial progress in the development of approximate-optimal on-line POMDP solvers that trade optimality for computational tractability. Despite this progress, solving POMDPs for systems with complex, non-linear transition dynamics remains challenging. State-of-the-art general on-line solvers rely on a large number of forward simulations of the system to evaluate sequences of actions the robot can perform from the current belief, which can become prohibitively expensive when the robot's transition dynamics admit no closed-form solution. In fact, in many realistic problem scenarios, the majority of the planning cost is attributed to simulating the robot's transition dynamics. Moreover, attempting to apply current on-line solvers to real-world planning tasks reveals additional issues from an implementation point of view: In almost all implementations of current POMDP solvers – particularly the ones that consider continuous state, action and observation spaces – the user is tied to specific data structures of the POMDP model, or the model has to be hard-coded. In addition, developing and testing new on-line POMDP solvers using existing POMDP software is not easy, as many implementations are tied to a specific solver, or class of solvers.

In this thesis we have shown how to tackle the above problems and make POMDPs more practical for motion planning problems under partial observability. In particular we have studied two approaches of systematic approximations of complex, non-linear dynamics that help in designing new POMDP solvers that are more efficient than current state-of-the-art POMDP solvers: In the first approach we have investigated if, and how *linearization*, a common method from estimation and control, can help to simplify the problem and subsequently speed-up the planning process. Our experimental results indicate that naively restricting belief to be Gaussian and assuming linearized transition and observation dynamics and can be harmful to the quality of the resulting policies, particular when the robot operates in cluttered environments. To measure the effect of cluttered environments to

linearization-based solvers, we have developed a new measure of non-linearity for stochastic systems, called Statistical-distance-based Non-linearity Measure (SNM). Our results indicate that SNM is, compared to a state-of-the-art measure, more effective in measuring the effect cluttered environments have on the quality of policies computed by linearization-based solvers. Based on this insight, we have developed a simple on-line solver, called SNM-Planner, that switches between a general solver (ABT) and a linearization-based solver (MHFR), depending on a local approximation of SNM around the current belief. Our experimental results indicate that by combining these two classes of solvers, we are able to compute higher-quality policies than the two component solvers alone. This shows that the proposed measure SNM is indeed capable of deciding when a linearization-based solver likely yields a good policy and when a general solver is the better choice.

In the second approach we have developed a new general on-line solver, called Multilevel POMDP Planner (MLPP). Instead of replacing the robot's complex, non-linear transition function with a single approximation (such as a linearized transition function), MLPP uses *multiple* levels of approximation of the transition function to simplify the planning problem. By combining Monte-Carlo-Tree-Search, a search method that has shown state-of-the-art performance in on-line POMDP planning, with a relative new Monte-Carlo technique called Multilevel Monte-Carlo (MLMC) to estimate the expected value of sequences of actions, MLPP is able to significantly reduce the cost of sampling. This, in turn, enables MLPP to compute near-optimal policies significantly faster than ABT and POMCP, two of the fastest on-line solvers today, on four problem scenarios that involve POMDP based torque-control, navigation and grasping. We have also shown that under certain conditions, MLPP retains asymptotic optimality guarantees. A further benefit of our new solver is that, compared to linearization-based solvers, MLPP imposes no restrictions on the type of beliefs. In contrast, linearization-based solvers require the beliefs to be approximated by (multivariate) Gaussian distributions.

To alleviate the restrictions of current POMDP software, we have developed On-line POMDP Planning Toolkit (OPPT) a new software framework that is designed to support the user in modeling POMDP problems with minimal additional coding effort. Instead of hard-coding the problem, the user can define new problems using simple model- and configurations files only. In addition to that, OPPT provides a general solver API that allows the user to implement new solvers (independent of the POMDP models) without being tied to a specific class of solvers. Note that each test problem considered in this thesis has been modeled using OPPT. Furthermore, most solvers, with the exception for POMCP in Chapter 3 have been implemented using the solver API provided by OPPT. Additionally we have successfully applied OPPT on a real-world manipulation task that involves a 6DOFs Jaco arm. In this task the robot has to pick-up a cup from a table, use the cup to scoop candy from a box near the table, and return the cup to the table without spillage. The robot is hereby subject to perception errors regarding the exact location of the cup the location of the obstacles and the height of the candy surface. Additionally, the location of the cup might be altered by bystanders during run-time. We have divided the entire task into 4 POMDP problems (each of them modeled in OPPT) that are being solved sequentially. Additionally we have provided some implementation details that enable the robot to handle long-planning horizons and ensure the delay between individual planning steps is negligible.

We have successfully demonstrated the entire system over a total of seven days at the 2018 SIMPAR and ICRA conferences where we achieved a 98% success rate over 150 runs.

## 5.2 Future Work

Future work abounds. For instance, the question for a better measure of non-linearity remains. SNM relies on computing a maximization over the state- and action spaces of the robot, which is often difficult to estimate on-line. We therefore resort to an off-line method which, however, is not suitable for dynamic environments. Furthermore, the derived upper-bound in Theorem 1 is relatively loose and can only be applied as a sufficient condition to identify whether linearization likely performs well. It would be useful to find a tighter bound that remains general enough for the various linearization and distribution approximation methods in robotics. The presented comparison studies in Chapter 2 also reveal several insights regarding the suitability of linearization-based solvers for specific types of problems. For instance, linearization tends to perform poorly for systems operating in cluttered environment or for systems that are subject to large errors, but tends to perform well in finding a coarse strategy for problems that require long planning horizon. Such insights can potentially help to design better solvers.

Regarding the proposed on-line solver MLPP, a worthwhile direction for future research involves relaxing the assumptions that are required for MLPP to asymptotically converge to the optimal policy. This might require a different approach to combine MLMC with Monte-Carlo-Tree-Search. Additionally MLPP requires the user to parametrize the transition function and define the levels of approximation of the transition function a-priori. This is a limitation that is inherent to all current Multilevel Monte-Carlo methods [50]. For many problems, as the ones we studied in Chapter 3, parameterizing the transition function and defining the levels is rather trivial. However, for some transition functions, it might not be immediately obvious how the transition function should be parametrized. An interesting venue could be to develop methods that learn a parametrization that admit multiple levels of approximation of the transition function. Moreover, instead of using a fixed number of levels, the number of levels could be adapted during run-time. This might possibly yield quite substantial performance gains, particular when finer levels have a negligible contribution to the $Q$-value estimates.

While we believe that the OPPT-framework presented in Chapter 4 provides a significant step forward in simplifying the modelling of complex POMDP-problems, there are several areas in which the framework could be extended. Modern robotics software is often implemented within a larger operating framework such as ROS [95]. To support the integration of OPPT into these frameworks, a standard ROS-wrapper for OPPT would provide opportunities to use OPPT as an in-place module for modelling and planning under partial observability within already existing robot systems. Additionally, the support of different physics engines outside the Gazebo domain, such as Mujoco [102] would increase the flexibility of OPPT in modelling different robotic planning problems even further.

In general the method proposed in this thesis require a good understanding of the underlying

transition model of the problem. For example SNM requires a transition model that is at least easy to sample from, in order to numerically compute a linearized model. Similarly, MLPP requires a transition model that admits the definition of multiple levels of approximation. Such models, however, are often not available, motivating the increasingly popular field of model-free reinforcement learning methods [44]. However, model-free methods typically suffer from poor sample efficiency [103]. To increase sampling efficiency, model-based reinforcement learning methods have been proposed [104], in which an explicit representation of a transition model is learned that is subsequently used to compute a policy. An interesting research venue could be to adapt the methods proposed in this thesis by learning models on different levels of accuracy, thereby speeding-up the learning process and further increasing sampling efficiency.

# Bibliography


[1] M. Hoerger, H. Kurniawati, A. Elfes, Multilevel monte-carlo for solving pomdps online, in: Proc. International Symposium on Robotics Research (ISRR) (To Appear), 2019.

[2] M. Hoerger, J. Song, H. Kurniawati, A. Elfes, Pomdp-based candy server: Lessons learned from a seven day demo, in: Proc. AAAI International Conference on Autonomous Planning and Scheduling (ICAPS), AAAI, 2019, pp. 698–706.

[3] M. Hoerger, H. Kurniawati, A. Elfes, A software framework for planning under partial observability, in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018, pp. 7576–7582.

[4] M. Hoerger, H. Kurniawati, T. Bandyopadhyay, A. Elfes, Linearization in motion planning under uncertainty, in: Proceedings of the 12th International Workshop on the Algorithmic Foundations of Robotics (WAFR), 2016.

[5] M. Hoerger, H. Kurniawati, T. Bandyopadhyay, A. Elfes, Effects of obstacle avoidance to lqg-based motion planners, in: Australasian Conference on Robotics and Automation, ARAA, 2016.

[6] A. Snoswell, V. Dewanto, M. Hoerger, J. Song, H. Kurniawati, S. Singh, A distributed, anytime robot architecture for robust manipulation, in: Australasian Conference on Robotics and Automation, ARAA, 2018.

[7] A. Elfes, R. Steindl, F. Talbot, F. Kendoul, P. Sikka, T. Lowe, N. Kottege, M. Bjelonic, R. Dungavell, T. Bandyopadhyay, M. Hoerger, The multilegged autonomous explorer (max), in: Proc. of the IEEE International Conference on Robots and Automation (ICRA), IEEE, 2017, pp. 1050–1057.

[8] E. J. Sondik, The optimal control of partially observable markov decision processes, Ph.D. thesis, Stanford, California (1971).

[9] C. H. Papadimitriou, J. N. Tsitsiklis, The complexity of markov decision processes, Mathematics of operations research 12 (3) (1987) 441–450.

[10] J. Canny, J. Reif, New lower bound techniques for robot motion planning problems, in: Foundations of Computer Science, 1987., 28th Annual Symposium on, IEEE, 1987, pp. 49–60.

[11] B. Natarajan, The complexity of fine motion planning, The International journal of robotics research 7 (2) (1988) 36–42.

[12] H. Kurniawati, V. Yadav, An online pomdp solver for uncertainty planning in dynamic environment, in: Proc. Int. Symp. on Robotics Research, 2013.

[13] D. Silver, J. Veness, Monte-carlo planning in large POMDPs, in: Advances in neural information processing systems, 2010, pp. 2164–2172.

[14] A. Somani, N. Ye, D. Hsu, W. S. Lee, Despot: Online pomdp planning with regularization, in: Advances in neural information processing systems, 2013, pp. 1772–1780.

[15] K. M. Seiler, H. Kurniawati, S. P. Singh, An online and approximate solver for pomdps with continuous action space, in: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 2290–2297.

[16] L. E. Kavraki, P. Svestka, J. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Transactions on Robotics and Automation 12 (4) (1996) 566–580. doi:10.1109/70.508439.

[17] S. M. Lavalle, Rapidly-exploring random trees: A new tool for path planning, Tech. rep., Computer Science Department, Iowa State University (1998).

[18] J. J. Kuffner, S. M. LaValle, Rrt-connect: An efficient approach to single-query path planning, in: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation, Vol. 2, 2000, pp. 995–1001. doi:10.1109/ROBOT.2000.844730.

[19] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, The international journal of robotics research 30 (7) (2011) 846–894.

[20] J. Lengyel, M. Reichert, B. Randall Donald, D. P. Greenberg, Real-time robot motion planning using rasterizing computer graphics hardware, in: ACM Siggraph Computer Graphics, Vol. 24, 1990, pp. 327–335. doi:10.1145/97879.97915.

[21] J. Barraquand, J. Latombe, Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles, in: Proceedings. 1991 IEEE International Conference on Robotics and Automation, Vol. 3, 1991, pp. 2328–2335. doi:10.1109/ROBOT.1991.131750.

[22] R. Alterovitz, T. Siméon, K. Goldberg, The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty, in: Proceedings Robotics: Science and systems, 2007.

[23] R. Bellman, A markovian decision process, Journal of Mathematics and Mechanics (1957) 679–684.

[24] S. Prentice, N. Roy, The belief roadmap: Efficient planning in linear pomdps by factoring the covariance, in: Robotics Research, Springer, 2010, pp. 293–305.

[25] J. Berg, P. Abbeel, K. Goldberg, LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information, 2010.

[26] A. Lindquist, On feedback control of linear stochastic systems, SIAM Journal on Control 11 (2) (1973) 323–343.

[27] A.-A. Agha-Mohammadi, S. Chakravorty, N. M. Amato, Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements, The International Journal of Robotics Research 33 (2) (2014) 268–304.

[28] W. Sun, S. Patil, R. Alterovitz, High-frequency replanning under uncertainty using parallel sampling-based motion planning, IEEE Transactions on Robotics 31 (1) (2015) 104–116.

[29] R. Platt, R. Tedrake, L. Kaelbling, T. Lozano-Perez, Belief space planning assuming maximum likelihood observations, in: Robotics Science and Systems Conference (RSS), 2010.
URL `http://groups.csail.mit.edu/robotics-center/public_papers/Platt10.pdf`

[30] J. Nocedal, S. Wright, Numerical optimization, Springer Science & Business Media, 2006.

[31] M. Nikolaou, V. Hanagandi, Nonlinearity quantification and its application to nonlinear system identification, Chemical Engineering Communications 166 (1) (1998) 1–33.

[32] J. Pineau, G. Gordon, S. Thrun, Point-based value iteration: An anytime algorithm for pomdps, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI'03, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003, pp. 1025–1030.

[33] T. Smith, R. G. Simmons, Point-based pomdp algorithms: Improved analysis and implementation., in: UAI, AUAI Press, 2005, pp. 542–547.

[34] E. Brunskill, L. Kaelbling, T. Lozano-Perez, N. Roy, Continuous-state POMDPs with hybrid dynamics, in: International Symposium on Artificial Intelligence and Mathematics, 2008.
URL `http://lis.csail.mit.edu/pubs/tlp/brunskill_continuous.pdf`

[35] J. M. Porta, N. Vlassis, M. T. Spaan, P. Poupart, Point-based value iteration for continuous pomdps, J. Mach. Learn. Res. 7 (2006) 2329–2367.

[36] H. Kurniawati, D. Hsu, W. S. Lee, Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces, in: In Proc. Robotics: Science and Systems, 2008.

[37] S. Ross, J. Pineau, S. Paquet, B. Chaib-Draa, Online planning algorithms for pomdps, Journal of Artificial Intelligence Research 32 (2008) 663–704.

[38] J. Satia, R. Lave, Markovian decision processes with probabilistic observation of states, Management Science 20 (1) (1973) 1–13.

[39] R. Washington, Bi-pomdp: Bounded, incremental partially-observable markov-model planning, in: European Conference on Planning, Springer, 1997, pp. 440–451.

[40] S. Ross, B. Chaib-Draa, Aems: An anytime online search algorithm for approximate policy refinement in large pomdps, in: Proceedings of the 20th International Joint Conference on Artifical Intelligence, IJCAI'07, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007, pp. 2592–2598.
URL http://dl.acm.org/citation.cfm?id=1625275.1625693

[41] S. Paquet, L. Tobin, B. Chaib-draa, Real-time decision making for large pomdps, in: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, 2005, pp. 450–455.

[42] D. A. McAllester, S. Singh, Approximate planning for factored pomdps using belief state simplification, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 409–416.
URL http://dl.acm.org/citation.cfm?id=2073796.2073843

[43] L. Kocsis, C. Szepesvári, Bandit based monte-carlo planning, in: European conference on machine learning, Springer, 2006, pp. 282–293.

[44] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, 2012.

[45] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning 47 (2-3) (2002) 235–256.

[46] Z. N. Sunberg, M. J. Kochenderfer, Online algorithms for pomdps with continuous state, action, and observation spaces, in: Twenty-Eighth International Conference on Automated Planning and Scheduling, 2018.

[47] H. Bai, D. Hsu, W. S. Lee, Integrated perception and planning in the continuous space: A pomdp approach, The International Journal of Robotics Research 33 (9) (2014) 1288–1302.

[48] T. Smith, R. Simmons, Heuristic search value iteration for pomdps, in: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, UAI '04, AUAI Press, Arlington, Virginia, United States, 2004, pp. 520–527.
URL http://dl.acm.org/citation.cfm?id=1036843.1036906

[49] S. Heinrich, Multilevel monte carlo methods, in: S. Margenov, J. Waśniewski, P. Yalamov (Eds.), Large-Scale Scientific Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 58–67.

[50] M. B. Giles, Multilevel monte carlo methods, Acta Numerica 24 (2015) 259–328.

[51] J. Berg, D. Wilkie, S. Guy, M. Niethammer, D. Manocha, LQG-Obstacles: Feedback Control with Collision Avoidance for Mobile Robots with Motion and Sensing Uncertainty, 2012.

[52] D. P. Bertsekas, Dynamic Programming and Optimal Control, 2nd Edition, Athena Scientific, 2000.

[53] X. R. Li, Measure of nonlinearity for stochastic systems, in: Information Fusion (FUSION), 2012 15th International Conference on, IEEE, 2012, pp. 1073–1080.

[54] D. M. Bates, D. G. Watts, Relative curvature measures of nonlinearity, Journal of the Royal Statistical Society. Series B (Methodological) (1980) 1–25.

[55] E. Beale, Confidence regions in non-linear estimation, Journal of the Royal Statistical Society. Series B (Methodological) (1960) 41–88.

[56] K. Emancipator, M. H. Kroll, A quantitative measure of nonlinearity., Clinical chemistry 39 (5) (1993) 766–772.

[57] J. Duník, O. Straka, M. Šimandl, Nonlinearity and non-gaussianity measures for stochastic dynamic systems, in: Information Fusion (FUSION), IEEE, 2013, pp. 204–211.

[58] A. Mastin, P. Jaillet, Loss bounds for uncertain transition probabilities in markov decision processes, IEEE, 2012, pp. 6708–6715.

[59] A. Müller, How does the value function of a markov decision process depend on the transition probabilities?, Mathematics of Operations Research 22 (4) (1997) 872–885.

[60] S. M. LaValle, Planning Algorithms, Cambridge University Press, New York, NY, USA, 2006.

[61] R. E. Kalman, A new approach to linear filtering and prediction problems, Journal of basic Engineering 82 (1) (1960) 35–45.

[62] H. Sorenson, Kalman Filtering: Theory and Application, IEEE Press selected reprint series, IEEE Press, 1985.
URL `https://books.google.com.au/books?id=2pgeAQAAIAAJ`

[63] M. S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking, IEEE Transactions on signal processing 50 (2) (2002) 174–188.

[64] J. J. Kuffner, S. M. LaValle, Space-filling trees: A new perspective on incremental search for motion planning, in: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE, 2011, pp. 2199–2206.

[65] P. Cheng, S. M. LaValle, Reducing metric sensitivity in randomized trajectory design, in: Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180), Vol. 1, IEEE, 2001, pp. 43–48.

[66] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, T. Lozano-Perez, Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 2537–2542.

[67] J. Denny, R. Sandström, A. Bregger, N. M. Amato, Dynamic region-biased rapidly-exploring random trees, in: Twelfth International Workshop on the Algorithmic Foundations of Robotics (WAFR), 2016.

[68] Y. Li, Z. Littlefield, K. E. Bekris, Sparse methods for efficient asymptotically optimal kinodynamic planning, in: Algorithmic foundations of robotics XI, Springer, 2015, pp. 263–282.

[69] K. Hauser, Y. Zhou, Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space, IEEE Transactions on Robotics 32 (6) (2016) 1431–1443.

[70] N. A. Ahmed, D. V. Gokhale, Entropy expressions and their estimators for multivariate distributions, IEEE Transactions on Information Theory 35 (3) (1989) 688–692. doi:10.1109/18.30996.

[71] M. W. Spong, S. Hutchinson, M. Vidyasagar, Robot Modeling and Control, Vol. 3, Wiley New York, 2006.

[72] R. Smith, Open dynamics engine, http://www.ode.org/.

[73] H. Niederreiter, Random number generation and quasi-Monte Carlo methods, Vol. 63, Siam, 1992.

[74] P. W. Glynn, D. L. Iglehart, Importance sampling for stochastic simulations, Management Science 35 (11) (1989) 1367–1392.

[75] R. Y. Rubinstein, R. Marcus, Efficiency of multivariate control variates in monte carlo simulation, Operations Research 33 (3) (1985) 661–677.

[76] M. B. Giles, Multilevel monte carlo path simulation, Operations Research 56 (3) (2008) 607–617.

[77] C. Bierig, A. Chernov, Approximation of probability density functions by the multilevel monte carlo maximum entropy method, Journal of Computational Physics 314 (2016) 661–681.

[78] D. F. Anderson, D. J. Higham, Multilevel monte carlo for continuous time markov chains, with applications in biochemical kinetics, Multiscale Modeling & Simulation 10 (1) (2012) 146–179.

[79] S. Asmussen, P. W. Glynn, Stochastic simulation: algorithms and analysis, Vol. 57, Springer Science & Business Media, 2007.

[80] D. Van Ravenzwaaij, P. Cassey, S. D. Brown, A simple introduction to markov chain monte–carlo sampling, Psychonomic bulletin & review 25 (1) (2018) 143–154.

[81] A. B. Owen, Monte carlo theory, methods and examples (2013).

[82] T. Keller, M. Helmert, Trial-based heuristic tree search for finite horizon mdps, in: Twenty-Third International Conference on Automated Planning and Scheduling, 2013.

[83] C.-h. Rhee, P. W. Glynn, A new approach to unbiased estimation for sde's, in: Proceedings of the Winter Simulation Conference, Winter Simulation Conference, 2012, p. 17.

[84] D. Klimenko, J. Song, H. Kurniawati, Tapir: a software toolkit for approximating and adapting pomdp solutions online, in: Proceedings of the Australasian Conference on Robotics and Automation, 2014.

[85] P. Poupart, Symbolic-perseus, `https://cs.uwaterloo.ca/~ppoupart/software.html#symbolic-perseus`.

[86] T. Smith, ZMDP, `http://longhorizon.org/trey/zmdp/`.

[87] N. U. of Singapore, APPL, `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`.

[88] A. R. Cassandra, Pomdp format, `http://pomdp.org/code/pomdp-file-spec.html`.

[89] S. Ong, S. Png, D. Hsu, W. Lee, Planning under uncertainty for robotic tasks with mixed observability 29 (8) (2010) 1053–1068.

[90] J. Hoey, R. St-Aubin, A. Hu, C. Boutilier, Spudd: Stochastic planning using decision diagrams, in: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 279–288.

[91] N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, Vol. 3, IEEE, 2004, pp. 2149–2154.

[92] J. Pan, S. Chitta, D. Manocha, Fcl: A general purpose library for collision and proximity queries, in: Robotics and Automation (ICRA), 2012 IEEE International Conference on, IEEE, 2012, pp. 3859–3866.

[93] O. S. R. Foundation, SDFormat, `http://sdformat.org`.

[94] O. S. R. Foundation, RViz, `http://wiki.ros.org/rviz`.

[95] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, Ros: an open-source robot operating system, in: ICRA Workshop on Open Source Software, 2009.

[96] R. He, E. Brunskill, N. Roy, Puma: Planning under uncertainty with macro-actions., in: Proceedings of the National Conference on Artificial Intelligence, Vol. 2, 2010.

[97] H. Kurniawati, Y. Du, D. Hsu, W. S. Lee, Motion planning under uncertainty for robotic tasks with long time horizons, The International Journal of Robotics Research 30 (3) (2011) 308–323.

[98] E. Wang, H. Kurniawati, D. P. Kroese, An on-line planner for pomdps with large discrete action space: A quantile-based approach, in: ICAPS, AAAI Press, 2018, pp. 273–277.

[99] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[100] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning., in: OSDI, Vol. 16, 2016, pp. 265–283.

[101] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft coco: Common objects in context, in: European conference on computer vision, Springer, 2014, pp. 740–755.

[102] E. Todorov, T. Erez, Y. Tassa, Mujoco: A physics engine for model-based control, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2012, pp. 5026–5033.

[103] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, arXiv preprint arXiv:1801.01290.

[104] A. S. Polydoros, L. Nalpantidis, Survey of model-based reinforcement learning: Applications on robotics, Journal of Intelligent & Robotic Systems 86 (2) (2017) 153–173.