Project Thesis
PRO-XXX

Partially Observable Markov Decision Processes for Planning in Uncertain Environments

by
Lasse Peters

Supervisors at TUHH
Prof. Dr.-Ing. Robert Seifried
Daniel Duecker, M.Sc.

Supervisors at UC Berkeley
Prof. Claire J. Tomlin
Zachary Sunberg, PhD.

Hamburg University of Technology
Institute of Mechanics and Ocean Engineering
Prof. Dr.-Ing. R. Seifried

Hamburg, July 2019

Contents

List of Abbreviations

DESPOT Determinized Sparse Partially Observable Tree. 9

DPW Double Progressive Widening. 13

HRI Human Robot Interaction. 34

MCTS Monte-Carlo Tree Search. 13

MDP Markov Decision Process. 5

MLMPC Most Likely Model Predictive Control. 25

MLRA Most Likely Reflex Agent. 25

MOMDP Mixed Observability Markov Decision Process. 38

MPC Model Predictive Control. 25

PDF Probability Density Function. 30, 32

POMCP Partially Observable Monte-Carlo Planning. 13

POMCPOW Partially Observable Monte-Carlo Planning with Observation Widening. 9

POMDP Partially Observable Markov Decision Process. 2

PSRP Probabilistically Safe Robot Planning. 39

SEM Standard Error of the Mean. 28

UCT Upper Confidence Bound for Trees. 15

Chapter 1

Introduction

Many decision making problems are subject to inherent uncertainty. Examples of such problems range from aircraft collision avoidance and robotic navigation tasks to applications in health care and medical treatment [KochenderferHollandChryssanthacopoulos12, BandyopadhyayEtAl13, PineauEtAl03, SchaeferEtAl05]. While humans have developed good intuition for decision making problems present in their day to day lives, many of the same tasks – like planning in autonomous driving – pose difficult problems for robotic agents [LevinsonEtAl11].

Actively considering uncertainty in planning promises to improve robustness, safety and performance of the system. In conventional control theory a typical approach is to model uncertainty in terms of a bounded disturbance, robustifying the controller through reasoning over worst case disturbance sequences [PetersenUgrinovskiiSavkin12]. However, modelling the disturbance as an adversarial player is often impractical, since long tail distributions may cause the controller to come up with overly conservative, thus poorly performing plans.

Therefore, a tremendous amount of research has focused on incorporating uncertainty in decision making through probabilistic models, rather than adversarial game type approaches [RoyEtAl99, AmatoEtAl15, FisacEtAl18, ChoudhuryKochenderfer19].

One of the most general frameworks for modeling uncertainty in sequential decision making in a probabilistic fashion is provided by the Partially Observable Markov Decision Process (POMDP). Formulating and solving a planning problem as a POMDP allows the planner to reason over both state and outcome uncertainty. Also, by taking into account future observations, solving a POMDP gives rise to behavior that actively performs information gathering.

While POMDPs provide a comprehensive way of taking into account uncertainty in the planning procedure, finding the optimal solution to these problems is in practice often intractable since in worst case it cannot be done in polynomial time [PapadimitriouTsitsiklis87]. However, there exists a wide range of approximate strategies to compute useful solutions to these problems. In this work, we distinguish between two fundamentally different approaches to approximation: Solving an *exact formulation* of the POMDP approximately; or finding an *approximate formulation* of the POMDP to design a problem specific solver for the simplified domain. For the sake of conciseness, we henceforth refer to the former as *full POMDP solutions* while the latter are denoted as *problem specific simplifications*.

1.1 Motivation

In robotics applications, characterized by limited compute and real time constraints, full POMDP solution methods are traditionally avoided. Instead, domain specific simplifications are used that neglect the partial observability or make other assumptions about the problem structure [SadighEtAl16, FisacEtAl18]. These simplifications are typically justified by the assumption that a full POMDP solution is intractable. Furthermore, the implementation complexity of state-of-the-art POMDP solvers inhibits the adaption of these algorithms.

However, recent progress in POMDP research has yielded new solution methods that have been shown to provide good performance even on medium to large scale POMDPs [SomaniEtAl13, SunbergKochenderfer18]. Additionally, the introduction of *POMDPs.jl* [EgorovEtAl17] – a modern framework for modelling and solving POMDPs in the Julia programming language – has lowered the barrier to modelling problems as POMDPs by abstracting away large parts of the implementation complexity.

Motivated by these advances this work aims to provide insight into the use of POMDPs for modelling and solving planning problems in robotics.

1.2 Contributions

At the example of two application domains we examine the use of behaviors generated by state-of-the-art POMDP solvers for robotic planning problems. Specifically, this work contains the following contributions:

First, we systematically quantify the performance advantage of full POMDP solutions with both analytic and empirical heuristic guidance compared to problem

specific simplifications in a continuous-space problem domains. For this purpose we focus on *simultaneous localization and planning*, a problem characterized by inherent uncertainty in the physical state of the robot.

The second contribution is a detailed comparison of the problem specific planning approach by [FisacEtAl18] to a state-of-the-art POMDP solver. This comparison aims to show how well a problem may be solved without considering the full complexity of POMDPs and to which extent such an approximation can not keep up with the full POMDP solution. For this purpose we study an instance of motion planning in a shared space with a human actor. In this application domain, latent human intentions force the agent to consider uncertainty in order to reach its goal safely.

Finally, as part of our work on motion planning with latent human intentions we contribute an implementation of a software model to simulate interacting humans and robots with near real time POMDP planning capabilities. This implementation is designed to accommodate convenient interchangeability of different human models and builds upon *POMDPs.jl* to allow for a direct comparison of different planning approaches.

1.3 Outline

write, when structure has converged

Theory Theoretical properties and background of POMDPs. Solution methods: POMCPOW, DESPOT.

Experiments Applications of POMDPs to two problem domains: Simultaneous localization an planning, Motion planning with latent human intentions.

Summary The summary of this work.

Chapter 2

Fundamentals

This work showcases the use of POMDPs to model and solve decision making problems in robotics with inherent uncertainty. In order to provide a baseline for further discussions of specific application domains (Chapters 3 and 4), we first introduce some of the theory used throughout this work. It should be noted that in the interest of conciseness we focus this theoretical introduction on only the main tools used. Fundamental concepts of statistical inference and tools like *Monte Carlo Integration* are assumed to be known. For a thorough discussion of these underlying concepts the reader may refer to [Kochenderfer15, Bertsekas05, ThrunBurgardFox05].

In the following, we first introduce the theoretical framework of POMDPs used for sequential decision making under uncertainty (Section 2.1). This section discusses modelling assumptions, structure of solutions as well as theoretical properties of POMDPs. Thereafter, Section 2.2 describes two state-of-the-art online solution methods for problems of this domain.

2.1 Sequential Decision Making Problems Under Uncertainty

The Partially Observable Markov Decision Process (POMDP) is a principled mathematical formalism capable of representing a broad range of sequential decision making problems under uncertainty. As the name suggests, this framework is a generalization of the more popular Markov Decision Process (MDP) to the partially observable case. Thus, before proceeding with the full complexity of a POMDP let us first examine its fully observable version.

2.1.1 Markov Decision Process (MDP)

Markov Decision Processes (MDPs) are sequential decision making problems in which an agent takes *actions* a that affect the *state* s of the environment and receives *rewards* r based the state-transition and the action taken [Kochenderfer15, Bertsekas05]. The state evolves according to a stochastic transition model $\mathcal{T}$ and obeys the Markov property. That is, future states are independent of past states given the current state and action. By this means, MDPs allow to model outcome uncertainty. As is common in literature, we denote quantities at time t with an according subscript. When examining only a single step in a context where time does not explicitly matter, we may also refer to states before and after the transition as s and s' rather than s_t and s_{t+1}.

Formally, an MDP is fully characterized by the following quantities:

State Space $\mathcal{S}$. The set of all possible states.

Action Space $\mathcal{A}$. The set of all possible actions.

Transition Model $\mathcal{T}$. A model to represent the likelihood of each transition. This model provides $\mathcal{T}(s' \mid s, a)$, the probability of state s' given that previously the environment was in state s and the agent took action a.

Reward Function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. A deterministic mapping that assigns a real-valued reward r to each transition (s, a, s') with finite transition probability, $\mathcal{T}(s' \mid s, a) > 0$.

Discount Factor γ. A scalar that governs how rewards are discounted in the future.

The objective of the agent in the MDP is to maximize the expected cumulative rewards. Formally, this translates to finding a *policy* $\pi : \mathcal{S} \to \mathcal{A}$, that maps each encountered s to an action $a = \pi(s)$, such that following this policy maximizes the objective,

$$J(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, \pi(s_t), s_{t+1})\right]. \tag{2.1}$$

An optimal solution to an MDP can always be formulated as a deterministic Markov policy, even though the reward that a policy achieves may be randomized through $\mathcal{T}$. Consequently, there always exists a maximizer π^* of Eq. (2.1), where π^* assigns a single action to every state. Also, since the state obeys the Markov property, no additional information beyond the current state at every time is necessary to maximize J [Altman99].

The objective of an MDP is often discussed in terms of the *optimal value function* $V^* : \mathcal{S} \to \mathbb{R}$. The optimal value function evaluated at state s represents the expected cumulative reward if the agents acts optimally starting from s. Additionally, we introduce the *optimal state-action value function* $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, concisely referred to as *Q-function* or *Q-value* of a state-action tuple. The Q-value, $Q(s, a)$, is defined as the expected cumulative reward given that the agent starts at state s, immediately takes action a, and then follows the optimal policy [Bertsekas05].

2.1.2 Partially Observable MDP (POMDP)

A Partially Observable Markov Decision Process (POMDP) is an MDP where the agent cannot directly observe the state, but instead at every time step t receives an *observation* o_t emission of the latent state s_t and action a_t. By this means, a POMDP is able to encode state uncertainty in addition to the outcome uncertainty present in the underlying MDP.

Having introduced the properties of an MDP in the previous section, a generalization of this formalism to the partially observable case is obtained by augmenting the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ describing the underlying MDP with the following quantities:

Observation Space $\mathcal{O}$. The set of all possible observations.

Observation Model $\mathcal{Z}$. A model to represent the likelihood of each observation o given the current state of the environment and the action taken at that time. Formally, this model provides the conditional probability $\mathcal{Z}(o \mid s, a)$ for each $(o, s, a) \in (\mathcal{O} \times \mathcal{S} \times \mathcal{A})$.

The dynamic decision network representing the information structure for a finite time horizon POMDP is depicted in Fig. 2.1. As evident from this figure, the decision of the agent at time t is informed by the sequence of all previous actions and observations as well as some prior knowledge, b_0. In contrast to the fully observable case where the policy is only a function of quantities at the current time, the policy in a POMDP maps each possible *history*, $h_t = (b_0, a_0, o_1, a_1, \ldots, a_{t-1}, o_t)$ to an action, a_t.

In cases where this history can be compressed to a probability for every state at time t, we will refer to this distribution as the *belief*, $b_t(s)$. This belief is a sufficient statistic for optimal decision making. Consequently, there exists a policy $\pi^*(b_t)$ only dependent on the belief at the current time step, such that choosing $a_t = \pi^*(b_t)$ maximizes Eq. (2.1) subject to the constraints on the information pattern imposed by the POMDP, [KaelblingLittmanCassandra98, Kochenderfer15].

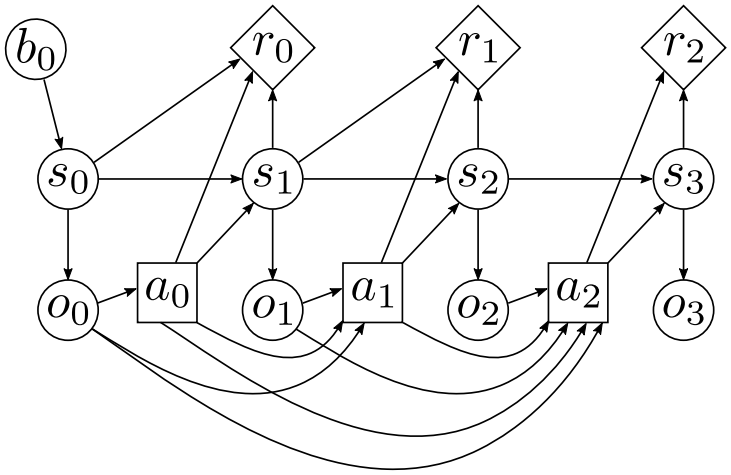


Figure 2.1: The POMDP as a dynamic decision network.

Loosely speaking, b_t preserves all relevant information contained in the action-observation-history, h_t necessary to compute the optimal action. The belief is maintained by recursively performing a Bayesian update,

$$b'(s') = \frac{\int_{s\in\mathcal{S}} \mathcal{Z}(o \mid s, a, s')\mathcal{T}(s' \mid s, a)b(s)\, ds}{\int_{s'\in\mathcal{S}} \int_{s\in\mathcal{S}} \mathcal{Z}(o \mid s, a, s')\mathcal{T}(s' \mid s, a)b(s)\, ds\, ds'}. \tag{2.2}$$

with incoming observations, o. In the case of a discrete state space, $\mathcal{S}$ integrals a to be replaced with sums. In cases where this update rule is too complex to be evaluated analytically, a particle filter may be used for approximate inference [Kochenderfer15, ThrunBurgardFox05].

Solutions to a POMDP are Markov policies on belief states. That is, the optimal policy $\pi^*(b(s)) : \mathcal{B} \to \mathcal{A}$ selects action $a = \pi(b(s))$ for each belief in the *belief space*, $\mathcal{B}$ under the objective of maximizing Eq. (2.1). Alternatively, when thinking of this procedure in terms of action-observation-histories, the policy may be envisioned as a conditional plan reasoning over the optimal action to take for every possible sequence of actions and observations starting from the root belief, b_0. This perspective reveals that the search space of possible policies rapidly grows with increasing size of $\mathcal{A}$ and $\mathcal{O}$. In fact, it has been shown that even in the case of finite-horizon problems POMDPs remain PSPACE-complete. Hence, it is reasonable to assume that no efficient general algorithm for solving large problems of this class can be found [PapadimitriouTsitsiklis87]. However, recent research has made significant progress on approximate solution methods, some of which we will use in this work [SilverVeness10, SomaniEtAl13, SunbergKochenderfer18].

2.2 Online POMDP Solvers

Once a problem has been formalized as a POMDP, a wide range of solution methods can be applied to solve it. On an abstract level, there exist two classes of solution techniques:

Offline methods perform most of the computation prior to execution. That is, a policy for the entire belief space is computed a priori. At execution time, actions are selected according to the assignment computed offline.

Online methods compute the optimal policy at execution time by planning from the current belief state. Solution techniques of this kind potentially leads to redundant computation and more compute per decision step at runtime. However, in practice they allow to solve much larger POMDPs since the reachable belief states are typically a small subset of the full belief space. By exploiting this local structure, online methods allow to compute solutions to large problems that may be intractable for offline approaches [Kochenderfer15, SilverVeness10, SomaniEtAl13, KurniawatiYadav16].

In practice, due to the complexity of POMDPs, even for seemingly simple problems finding the exact solution is computationally intensive. However, recent research has put forth online solvers that provide good approximations to a wide range of problems. This work makes use of two of the most advanced state of the art online POMDP solvers – Determinized Sparse Partially Observable Tree (DESPOT) and Partially Observable Monte-Carlo Planning with Observation Widening (POMCPOW). We choose these methods since they show superior performance in a wide range of benchmark problems and outperform other approximate solutions especially for large POMDPs [SomaniEtAl13, SunbergKochenderfer18].

In the following subsections we introduce DESPOT and POMCPOW in greater detail and discuss their theoretical properties.

2.2.1 DESPOT

Determinized Sparse Partially Observable Tree (DESPOT) is an approximate online solver for POMDPs that performs policy search on a sparse approximation of the standard belief tree [SomaniEtAl13]. In this paper we use the *anytime heuristic search* version of DESPOT as implemented by [Sunberg18a]. In this section we outline the general idea of the algorithm. While we aim to provide insight to enable a high level understanding of the algorithm, a detailed discussion of implementation details and convergence properties is beyond the scope of this work. For further insight into the theoretical foundations of this algo-

rithm the reader may refer to [SomaniEtAl13]. A detailed benchmark comparing the performance of this algorithm to other state-of-the-art solvers is provided in [SomaniEtAl13, SunbergKochenderfer18]. Beyond that, [Sunberg18a] provides a well structured implementation that closely matches the pseudo code of the original paper.

First, it must be noted that the term *DESPOT* is used interchangeably for both, the representation of the belief tree, $\mathcal{D}$, as well as the POMDP solver using this representation to compute a solution. We proceed by first introducing the high level idea of the belief tree representation in Section 2.2.1. Based on that, in Section 2.2.1 we discuss an anytime heuristic search method that incrementally constructs a DESPOT $\mathcal{D}$ to simultaneously find the optimal policy on this sparse belief tree.

Determinized Sparse Partially Observable Trees

A DESPOT is a sparse approximation of a belief tree that alleviates the exponentially complex problem of reasoning over the belief evolution by sampling states from the root belief and considering only a limited number of random *scenarios* when simulating future observations.

A *scenario* is an abstract concept which may be envisioned as an object that determinizes the outcome of a random process. Under a fixed scenario $\boldsymbol{\phi}$ an action sequence $(a_1, a_2, ...)$ applied starting from belief b always traverses the same state-action-observation sequence $(s_0, a_1, o_1, a_2, o_2, \dots)$. In DESPOT, scenarios are used to obtain a sparse presentation of the future. Figure 2.2 depicts a partially constructed DESPOT $\mathcal{D}$ rooted at the current belief, b_0, obtained under two scenarios (marked in blue and orange). Each node implicitly represents a particle belief, $\boldsymbol{\Phi}$, constructed from the distribution of outcomes encountered under all scenarios, $\boldsymbol{\phi} \in \boldsymbol{\Phi}$, passing through that node.

The tree is constructed by sampling an initial state under each scenario. Starting from this initial state, for each scenario *all actions* $a \in \mathcal{A}$ are simulated one step into the future. For the observations encountered during these determinized simulations, a new node is created one level deeper and the corresponding state-outcomes are recorded to form a particle belief at every node. As a result, the tree considers all action-branches but only the sparse set of observation-nodes encountered under the prefixed scenarios.

As the number of scenarios increases, this sparse representation has been shown to converge to the true belief tree.

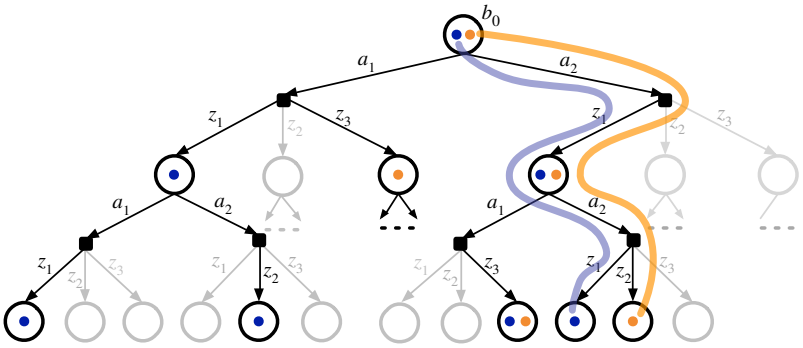


Figure 2.2: A partially constructed DESPOT $\mathcal{D}$ rooted at the current belief, b_0 [SomaniEtAl13]. The tree is obtained under two scenarios highlighted orange and blue.

Anytime Heuristic Search on a DESPOT

The sparse belief tree representation presented in Section 2.2.1 can be significantly more compact than a standard belief tree if the number of scenarios K is small. Therefore, it constitutes a potentially suitable representation for policy search. Since a DESPOT $\mathcal{D}$ obtained under a finite number of scenarios only considers a subset of potential outcomes, in general the optimal policy on $\mathcal{D}$ may be suboptimal for the dense formulation. In practice, however, a limited number of scenarios may sufficiently approximate the true belief tree such that the optimal decision computed on $\mathcal{D}$ is near-optimal for the full problem. In fact, it has been shown that for a policy optimized on $\mathcal{D}$ the error of its value when transfered to the dense problem stays bounded.

Therefore, in theory a near-optimal policy $\hat{\pi}$ could be computed by constructing the entire DESPOT in advance and performing dynamic programming to optimize $\hat{\pi}$ on $\mathcal{D}$. However, a fully constructed DESPOT obtained under K scenarios still contains $\mathcal{O}(|A|^D K)$ belief nodes. Therefore, the authors of [SomaniEtAl13] propose a *forward search* strategy to incrementally construct the sparse belief tree. The search method uses a heuristic to guide the exploration and performs regularization to prevent overfitting to the sampled scenarios. In the following we present the high level idea of this *anytime heuristic search* on a DESPOT. This section aims to provide intuition for the core concepts of this procedure. For a detailed discussion of the algorithm and precise mathematical justification of the individual steps the reader may refer to [SomaniEtAl13].

We aim to derive a policy from $\mathcal{D}$ that maximizes the *regularized empirical value*

$$\hat{V}^* = \max_{\pi \in \Pi_{\mathcal{D}}} \left(\hat{V}_\pi(b_0) - \lambda |\pi| \right) \tag{2.3}$$

under the sampled scenarios within a maximum planning time, $T_{\max}$. Here, b_0 is the current belief; π is a policy in $\Pi_{\mathcal{D}}$, the set of all policies on DESPOT $\mathcal{D}$; $\hat{V}_\pi$ is the empirical value of that policy when evaluated on $\mathcal{D}$; and $\lambda \in \mathbb{R}_{>0}$ is a constant to regularize the empirical value based on the size of the policy, $|\pi|$. Intuitively, the regularization alleviates overfitting by making sure that complex policies with a lot of flexibly to adopt to different scenarios need to perform significantly better than policies with a smaller representation size.

On an abstract level we determine a policy optimized according to Eq. (2.3) by incrementally constructing a DESPOT under K scenarios while maintaining upper bounds $U(b)$ and lower bounds $L(b)$ on the regularized empirical value at very node, b. An initial estimate of these bounds (L_0 and U_0) is provided by the user and may be used to encode domain knowledge. There exists a wide range of strategies to compute these bounds. A common strategy to obtain a lower bound is the simulation of a default policy. An upper bound can be computed by reasoning over relaxed versions of the problem. In practice, tight initial bounds on $\hat{V}^*$ are often crucial for the success of DESPOT when operating under limited planning time. A practical example for the design of suitable upper and lower bounds for a specific application domain is provided in Section 3.3.2. The DESPOT is constructed until the value gap $\epsilon(b_0)$ at the root b_0 satisfy the target gap ε_0 or the maximum planning time $T_{\max}$ is exceeded. Once either of these terminal conditions is met, we extract the optimized action at the root that maximizes the lower bound estimate on regularized empirical value.

While on an abstract level this algorithm is very simple, a key element of this search strategy is the incremental construction of the DESPOT. We therefore proceed by explaining the two main steps – *exploration* and *backup* – that are iteratively applied to enlarge the tree.

Exploration The partially constructed DESPOT $\mathcal{D}$ is explored starting from the root node, b_0. Initially, $\mathcal{D}$ only contains the root belief – an unweighted particle collection featuring a state particle for every scenario. The bounds at the root are initialized with $U_0(b_0)$ and $L_0(b_0)$ as provided by the user. Each exploration aims to reduce the current gap $\epsilon(b_0)$ at the root of the tree by a factor of $0 < \xi < 1$. If node b is a leaf node, the tree is expanded one level deeper by simulating all actions under each scenario and initializing the bounds for each child node. Otherwise, the tree is traversed according to the following tree policy: First, we greedily select the action branch a^* maximizing the upper bound estimate on the value. On this action branch we select the observation branch

$$o^* = \underset{o \in \mathcal{O}_{b,a^*}}{\operatorname{argmax}} E(b') = \underset{o \in \mathcal{O}_{b,a^*}}{\operatorname{argmax}} \left\{ \epsilon(b') - \frac{K_{b'}}{K} \cdot \xi \epsilon(b_0) \right\} \tag{2.4}$$

that features the highest *excess uncertainty* at the associated child node b'. Here, $\epsilon(b')$ is the current gap at the belief node b'; $K_{b'}/K$ is the fraction of the number of scenarios passing through that node; and $\xi\epsilon(b_0)$ is the target gap at the root for this exploration. Thus, the excess uncertainty at b' measures the difference between the current gap at the node and the expected gap at b' if the target gap $\xi\epsilon(b_0)$ at the root is satisfied. By this means, the tree policy seeks to greedily reduce uncertainty to quickly tighten the bounds at the root. Exploration at a node b is terminated under three conditions. First, the maximum depth D of the tree is reached. Second, the excess uncertainty is negative, indicating that further exploration will not significantly reduce the gap at the root. Finally, b is blocked by an ancestor node. Since the objective features regularization through $\lambda|\pi|$ (cf. Eq. (2.3)), the *regularized* utility $\hat{V}^*$ at an ancestor b' may be reduced if its policy subtree is enlarged by expanding b. Thus, a node is considered blocked if expanding this node reduces the $\hat{V}^*$ at an ancestor. In particular, this is the case if an insufficient number of scenarios passes through b' and further exploration entails the risk of overfitting. Blocked nodes are pruned from the DESPOT.

Backup Once exploration terminates according to the conditions mentioned above, we trace the path back to the root and update the bounds at all ancestors using Bellmans's principle. By this means, information obtained from expanding leaf nodes of $\mathcal{D}$ is propagated up the tree. Therefore, this step is often referred to as *backpropagation* or *backup*.

2.2.2 POMCPOW

Partially Observable Monte-Carlo Planning with Observation Widening (POMCPOW) is an approximate online POMDP solver that combines the idea of Monte-Carlo Tree Search (MCTS) – as utilized in Partially Observable Monte-Carlo Planning (POMCP) [SilverVeness10] – with the idea of Double Progressive Widening (DPW) [SunbergKochenderfer18].

Before we discuss the algorithmic procedure of POMCPOW we briefly review the general idea of MCTS. This section is a concise recap of the fundamental concept and seeks to provide a baseline for understanding how this technique is adopted in the context of POMCPOW; a thorough discussion of MCTS is beyond the scope of this work. For a detailed survey of a wide range of MCTS algorithms, the reader may refer to [BrowneEtAl12]. An extensive comparison of POMCPOW with other state-of-the-art solvers is provided in [SunbergKochenderfer18]. Beyond that, [Sunberg18b] provides a well structured and generic implementation of POMCPOW.

Monte-Carlo Tree Search

A theoretically valid solution method for sequential decision making problems is the construction of the entire policy tree. This tree consists of alternating layers of states and actions, enumerating all possible future trajectories. Given this object, extracting the optimal strategy from is trivial. However, for problems relevant in practice the curse of dimensionality makes simulating all possible futures intractable.

Monte-Carlo Tree Search (MCTS) is a method that breaks the curse of dimensionality by utilizing Monte-Carlo simulations to incrementally construct only important parts of the policy tree in an asymmetric fashion. As the policy is constructed, the samples are used to approximate the *state-action value function*, $Q(s, a)$, also referred to as the *Q-function*. The approximate Q-function is used to focus further exploration of promising parts of the tree, allowing the algorithm to approximate the optimal policy without attempting the intractable task of constructing the entire policy tree. This behavior is achieved by invoking a suitable *tree policy* that balances exploration and exploitation, the trade-off between looking at new parts of the state-action space (widening the tree) and further exploring promising trajectories (deepening the tree). One reason for the popularity of this method is that it only requires access to a *generative model*, G, used to sample new states given a state-action a pair. That is, no explicit model of the state dynamics is required, as long as a black box simulator of the environment is available.

To date, MCTS has been applied to a wide range of problems [BrowneEtAl12]. While research has put forth many variants of this algorithm, conceptually they all share the idea of iteratively running Monte-Carlo simulations until a terminal condition is reached. This terminal condition may be derived from a convergence criterion, a fixed number of iterations or a limited computational budget. One iteration of the general MCTS algorithm is visualized in Fig. 2.3. Each cycle comprises four steps:

Search The tree policy is invoked to traverse the policy tree constructed up to this point. That is, at each state node the tree policy selects an action based on the current Q-function approximation. Once an action has been chosen, the next state is selected by sampling from the G. This search procedure is repeated in an effort to reach the most urgent leaf node of the policy tree.

Expansion Once an action node is reached that does not have any children, a new node is appended to the policy tree by sampling a state from G and adding the corresponding actions.

Rollout Starting from the newly added node, a reward trajectory is simulated by selecting actions according to a *rollout policy*. This rollout policy may be as trivial as randomly selecting actions until a terminal state is reached or the future cumulative reward becomes negligible due to the compounding discount factor. Alternatively, domain specific knowledge may be utilized to design a more structured rollout strategy. The rollout yields a value estimate for the newly added leaf node.

Backpropagation The value estimate obtained from the rollout simulation is passed back up the tree to update the Q-function estimate on the path to the root node.

An important factor for the success of of this algorithm is the choice of the tree policy as this policy must focus the expansion to the relevant part of the tree. A widely used tree policy is the Upper Confidence Bound for Trees (UCT). This policy is a greedy policy with respect to the *upper confidence bound*,

$$UCB(s,a) = Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}. \tag{2.5}$$

Here, $N(s,a)$ is the number of times the action node (child) has been visited, $N(s) = \sum_{a\in\mathcal{A}} N(s,a)$ is the number of times the corresponding state node (parent) has been visited, and c is a problem specific parameter to balance the exploration/exploitation trade-off. Intuitively, the first term encourages the algorithm to select actions that in the past showed promising future reward trajectories while the second term urges the tree policy to expand nodes that have not been sufficiently explored. For a thorough discussion of convergence properties as well as guidance for the tuning of c, the reader may refer to [KocsisSzepesvári06, BrowneEtAl12].

Double Progressive Widening

For problems with large or continuous the state and action space, standard MCTS as presented in the previous section will produce very shallow trees. In fact, if the action space is infinite (e.g. continuous or countably infinite), MCTS using UCT will never choose a previously expanded action at the same state node a second time. This is evident from Eq. (2.5). For any unexplored action ($N(s,a) = 0$) the second term is understood to be unbounded, causing the tree policy to favor untried actions. Additionally, if the state space is continuous and the transition probability density is finite, the event of G generating the same state twice has probability zero. As a result, simulated trajectories will never pass through the same state node, causing the policy tree to never expand beyond the first layer.

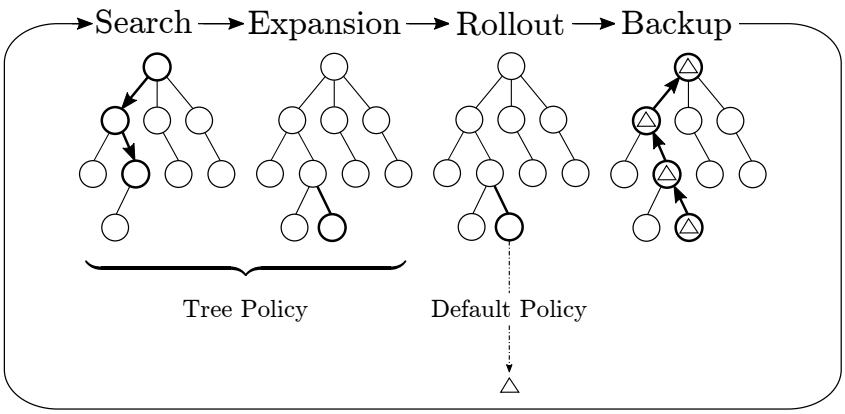


Figure 2.3: One iteration of the general MCTS approach [BrowneEtAl12].

Progressive widening addresses this issue by artificially limiting the number for children of a node to kN^{α}, where N is the number of times the parent has been visited and k and α are problem specific tuning parameters [CouëtouxEtAl11]. Originally, this strategy has been applied to the action space, as described in Algorithm 1. Double Progressive Widening (DPW) extends this idea to additionally apply to $\mathcal{S}$. In either case, if the number of children at a node exceeds the progressively incremented limit (Line 2), instead of generating a new child, one of the previously generated nodes is selected. By this means, MCTS expands to deeper layers earlier and widens the tree as additional iterations are run. This results in less myopic policies for problems with large state and action spaces [CouëtouxEtAl11].

Algorithm 1 Progressive widening applied to $\mathcal{A}$ [SunbergKochenderfer18].

1: **procedure** ACTIONPROGWIDEN(h)
2: **if** $|C(h)| \leq k_a N(h)^{\alpha_a}$ **then**
3: $a \leftarrow$ NEXTACTION(h)
4: $C(h) \leftarrow C(h) \cup \{a\}$
5: **return** $\underset{a \in C(h)}{\operatorname{argmax}}\, Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$

POMCPOW Algorithm

The algorithmic procedure for POMCPOW fundamentally resembles the idea of MCTS with a set of modifications for application in the POMDP domain. Instead of reasoning over state-action sequences, the policy tree spans over action-observation trajectories starting from the initial belief, b_0. In contrast to POMCP,

POMCPOW uses DPW to focus search on a progressively expanded set of actions and observations [SilverVeness10, SunbergKochenderfer18]. This enhancement produces deeper and more richly sampled policy trees even in the case of large action and observation spaces. Finally, since DPW constraints the set of considered observations, the algorithm uses a weighted particle collection to represent the outcome statistics at observation nodes. By this means, the outcome of the generative model can be fixed to fall inside the allowed set to avoid sample rejection, while the weighted belief update prevents an erroneous distribution shift.

A detailed pseudo code representation of the algorithmic procedure is provided in Algorithm 2. With the preceding introduction of MCTS and the aforementioned outline of the algorithms key ideas in mind, we proceed by discussing each subroutine in greater detail.

In addition to the variables defined in the previous sections, we introduce the following notation to describe the POMCPOW algorithm: h represents a history, $(b_0, a_1, o_1, a_2, \ldots, a_k, o_k)$, as introduced in Section 2.1.2, ha and hao are shorthand for concatenation of h with a or (a, o), respectively. d denotes the remaining depth to explore, with $d_{\max}$ the maximum depth; $C(h)$ is the list of children of node h; $N(h)$ denotes the number of visits of node h; $M(hao)$ is a count for the number of times the observation terminated history hao has been sampled by the generative model, G. $B(h)$ represents a list of states associated with node h, with $W(h)$ the corresponding weights.

Plan

The PLAN procedure represents the outer loop of the algorithm. Input to the function is the current belief, b, as obtained from a belief updater. Typically, b is the output of a particle filter. Starting at b, the algorithm performs n iterations of MCTS to obtain a local approximation the action value function, Q. The procedure is parametrized with $d_{\max}$, the maximum search depth of the policy tree. Finally, the planner selects the action, that maximizes the obtain Q-function approximation.

Simulate

The SIMULATE procedure resembles a recursive implementation of a single MCTS iteration. Inputs to the function are a sampled state, s, an action-observation history as introduced in Section 2.1.2, h, as well as the remaining search depth, d. Unless the maximum depth is reached, the algorithm proceeds by selecting an action a using action progressive widening. Given this action, a state-observation-reward tuple is sampled from the generative model. Subsequently, observation progressive widening (Lines 11 to 14) is applied to either generate a new observation and increment the corresponding counter, or sample one of the previously generated observations, $o \in C(ha)$. The generated state, s', is added to the state

list at the observation node, $B(hao)$. In order to resemble a weighted particle belief, the corresponding weight is chosen according to the observation model, $\mathcal{Z}(o \mid s, a, s')$, and is stored in the associated weight vector, $W(hao)$.

If the observation is not a child of this action terminated history node, a new observation node is created and a rollout in the spirit of MCTS is performed. Otherwise, search on the partially constructed policy tree continues one level deeper by sampling a state, s', from the weighted particle collection to proceed with the next SIMULATE recursion.

Finally, backpropagation is performed by passing the simulated *total* reward up the recursion stack and updating the Q-function approximation accordingly.

Algorithm 2 POMCPOW [SunbergKochenderfer18].

procedure PLAN(b)
  for $i \in 1:n$ **do**
    $s \leftarrow$ sample from b
    SIMULATE($s, b, d_{\max}$)
  return $\operatorname{argmax}_a Q(ba)$

procedure SIMULATE(s, h, d)
  if $d = 0$ **then**
    return 0
  $a \leftarrow$ ACTIONPROGWIDEN(h)
  $s', o, r \leftarrow G(s, a)$
  if $|C(ha)| \leq k_o N(ha)^{\alpha_o}$ **then**
    $M(hao) \leftarrow M(hao) + 1$
  else
    $o \leftarrow$ select $o \in C(ha)$ w.p. $\frac{M(hao)}{\sum_o M(hao)}$
  append s' to $B(hao)$
  append $\mathcal{Z}(o \mid s, a, s')$ to $W(hao)$
  if $o \notin C(ha)$ **then**
    $C(ha) \leftarrow C(ha) \cup \{o\}$
    $total \leftarrow r + \gamma$ROLLOUT$(s', hao, d-1)$
  else
    $s' \leftarrow$ select $B(hao)[i]$ w.p. $\frac{W(hao)[i]}{\sum_{j=1}^{m} W(hao)[j]}$
    $r \leftarrow \mathcal{R}(s, a, s')$
    $total \leftarrow r + \gamma$SIMULATE$(s', hao, d-1)$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $Q(ha) \leftarrow Q(ha) + \frac{total - Q(ha)}{N(ha)}$
  return $total$

2.3 Tools and Software Framework

Solving a POMDP poses a challenging problem due to several aspects. POMDP solvers, like those presented in Section 2.2, typically are complex algorithms that are not straight forward to implement correctly. Furthermore, the computational complexity of these algorithms requires sophisticated optimization in order to achieve sufficiently fast execution times. Therefore, suitable software tooling is

required that provides a simple but expressive way of describing problems and solvers while leveraging the full computational power of the hardware involved.

This section briefly discusses the tooling choices made for the experiments conducted throughout this work. This discussion is not thought to be a detailed comparison of POMDP frameworks but rather seeks to provide insight into how the tools used here can help to accelerate the solution process.

To date, there exist a number of frameworks that are capable of solving sequential decision making problems under uncertainty, some of which also accommodate partially observable domains. In this work we make use of the novel framework *POMDPs.jl*, an open-source interface for solving MDPs and POMDPs [EgorovEtAl17]. Unlike other popular frameworks like APPL ([KurniawatiEtAl08]), AI-Toolbox ([Bargiacchi13]), and ZMDP ([Smith05]), *POMDPs.jl* is not written in *C++* but in the *Julia* programming language. This enables the user to flexibly build prototypes while the high-performance nature of the language allows to scale ideas to run extensive simulations on large multi-node clusters. Benchmarks in the past have shown that despite being a high-level language that abstracts a lot of complexity, the execution time of code written in *Julia* can still compete with *C++* implementations. In fact, experiments we ran to evaluate the runtime for one of the problems studied in Chapter 4 suggest that the highly-optimized code produced by the *Julia* compiler may even outperform a *C++* implementation, unless the latter is carefully tuned by an experienced software engineer.

In addition to the advantages generated by the choice of programming language, *POMDPs.jl* provides a simple yet expressive interface to describe, solve and simulate POMDPs. The toolbox specifies abstract interfaces for three conceptual elements: *problems*, *solvers* and *experiments*. This design decouples these subdomains, thus ensuring that code can be written for each of them separately while not sacrificing interchangeability. Finally, the associated *JuliaPOMDP* community provides a number of state-of-the-art MDP and POMDP solvers as well as a rich library of tools for implementing and evaluating solvers and solutions.

Chapter 3

Simultaneous Localization and Planning

Typically, robot localization and motion planning are treated as mostly independent tasks. Often, the problem provides the agent with a focused initial belief or allows the agent to quickly reduce uncertainty by first collecting a series of measurements that do not require elaborate motion planning. This is particularly the case if the state space is well observed, i.e. there exists a sensor that provides the robot with a (noisy) position reading (e.g. GPS). For these well behaved problems, a simple yet effective approach is to first collect sensor data while executing a default motion, i.e. rotating about the yaw axis to collect measurements in all directions. Once state uncertainty is reduced to a focused, unimodal belief, the problem is treated as fully observed problem and motion planning is performed on expectation. That is, trajectories are planned based on the assumption that the expectation of the belief, $E[b]$, is a sufficient approximation for the true position of the robot. Furthermore, characteristic properties of the belief (e.g. the entropy) may be evaluated to trigger additional information gathering. In even simpler problems, plan execution may implicitly yield sufficient information gathering, rendering explicit consideration of this aspect obsolete.

However, more challenging problems are obtained when information gathering happens less naturally. In these cases, a simple approach as outlined above may not perform well. Instead, persistently good performance requires the agent to consider the full belief distribution as well as future observations in order to identify informative action sequences.

An instance of such problem is studied in this chapter. We begin by stating the problem details in Section 3.1. Section 3.2 then formalizes this problem as a POMDP. Section 3.3 presents solution strategies based on POMDP solvers

introduced in Section 2.2, as well as two heuristic baseline approaches. Finally, we evaluate the performance of each solver and discuss the results in Section 3.4.

3.1 Problem Statement

In this chapter we study an instance of simultaneous localization and planning. Consider a robot operating in a planar, previously mapped room as depicted in Fig. 3.1. In this figure the red marker denotes the true position of the robot, while the blue particles represent the robot's belief of its current location in the room. The agent is tasked to reach the exit (green) while avoiding falling down the stairs (red). Additionally, unnecessary collisions are to be avoided. The scene shows the simulation in its initial configuration: The robot starts at a position drawn uniformly at random from the set of all possible position-orientation tuples. This initial position is not known by the agent, thus the particles representing the initial belief are spread uniformly across the entire room. The robot is equipped with a single sensor that provides information about collisions with walls of the room. The robot's dynamics are governed by an underactuated first-order model that allows the robot to translate forward and/or rotate about its vertical axis. The agent holds control authority to choose translation and rotation rates.

This problem has originally been published under the name *"Escape Roomba"* by the *Stanford Intelligent Systems Laboratory* as part of the class *Decision Making under Uncertainty*[1]. The experiments presented in the following have been conducted using the *POMDPs.jl* implementation of the problem[2].

[1] `https://web.stanford.edu/class/aa228/cgi-bin/wp/`
[2] `https://github.com/sisl/AA228FinalProject`

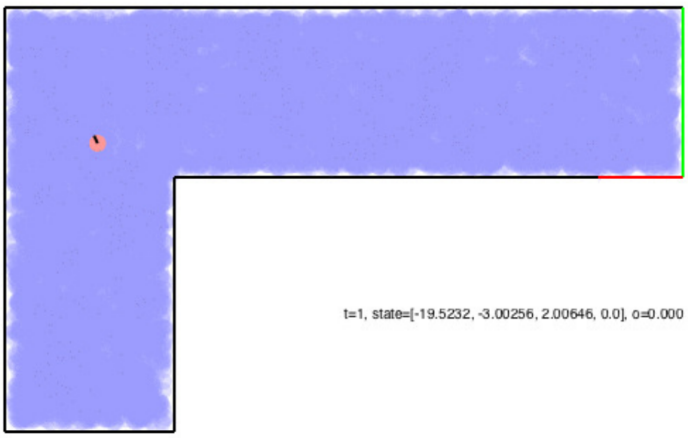


Figure 3.1: A screen shot of the simulation environment for the *"Escape Roomba"* problem provided by the *Stanford Intelligent Systems Laboratory* as part of the class *Decision Making under Uncertainty*.

3.2 POMDP Formalization

The problem is formalized as a POMDP with the following properties introduced in Section 2.1.2:

State Space $\mathcal{S}$. The state of the robot is defined as the position, orientation tuple, $s = ((x, y), \alpha)$. Respectively, $\mathcal{S}$ is the set of all positions-orientation tuples inside the room.

Action Space $\mathcal{A}$. An action is defined as a translation-rotation rate tuple, $a = (v, \omega)$. The provided implementation of the generative model only allows sampling of state transitions at significant computational cost. Thus, we reduce the policy search space by discretizing the action space to three actions: *"left"*, *"right"* and *"straight"*. The set of these three actions constitutes the action space, $\mathcal{A}$.

Transition Model $\mathcal{T}$. The robot transition follows the dynamics of a simple differential drive model. The details of this transition are defined implicitly by the simulator and are treated as a generative black box model.

Reward Function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The reward model is defined as:

$$\mathcal{R}(s, a, s') = r_{\text{time}} + \delta_{\text{collision}}(s')r_{\text{collision}} + \delta_{\text{goal}}(s')r_{\text{goal}} + \delta_{\text{stairs}}(s')r_{\text{stairs}}. \quad (3.1)$$

Here, $\delta_{\text{c}}(s)$ denotes the *Kronecke delta* function under the condition, c, such that

$$\delta_{\text{c}}(s) = \begin{cases} 1, & \text{if condition } c \text{ holds for } s \\ 0, & \text{otherwise.} \end{cases} \quad (3.2)$$

The reward model is parametrized by the following quantities: r_{time} is a living penalty encouraging the robot to minimize the time until the goal is reached; $r_{\text{collision}}$ is a penalty for collision with walls; r_{goal} is the reward obtained when reaching the goal, and r_{stairs} the penalty for falling down the stairs.

Discount Factor γ. Rewards are discounted with $\gamma = 0.99$.

Observation Space $\mathcal{O}$. The sensor provides a binary output, stating whether the robot is currently in contact with the wall or not. That is, there are only two observations: o^0 and o^1.

Observation Model $\mathcal{Z}$. The collision sensor is deterministic. Therefore, the observation model is a Dirac delta function, evaluating non-zero for a state, s, if and only if the collision feature evaluated on s matches the observation, o.

Summarizing, the problem is a POMDP with continuous state space and discrete action and observation space. The reward model chosen for this problem is sparse. By this means it only encodes high level preferences but avoids biasing the solution through additional transition dependent rewards (e.g. rewards for reducing the distance to the goal).

3.3 Solution Strategies

We solve the POMDP described in Section 3.2 using DESPOT and POMCPOW, respectively. For each of these solvers we propose two domain specific adaptations to guide the policy search. Additionally, we present two baseline policies that use a heuristic policy to control the agent. In order to consider efficiency of each algorithm we limit the computation time per planning step to $T_{\max} = 1\,\mathrm{s}$. Each policy uses the same particle filter to estimate the root belief.

Maybe point to code attached or to repo.

3.3.1 Baseline Policies

In order to obtain a baseline we make a common simplifying assumption: Instead of planning with the full root belief, b, the policy uses only the *most likely state*, $s_{\mathrm{ml}} = \mathrm{mode}(b)$. Using this approximation, the problem is then treated as fully observable. That is, at every time step the control action is selected based on the assumption that the true position of the robot matches the most likely state. The loop is closed by applying only the first action of the policy at each time step to then update the belief and re-plan from the next most likely state, $s'_{\mathrm{ml}} = \mathrm{mode}(b')$. Based on this assumption we propose two heuristic policies:

Most Likely Reflex Agent (MLRA) For this baseline, the agent uses a simple, handcrafted feedback policy that does not involve any active reasoning about future states. Instead, it uses a proportional controller to track the center line of the room to reach exit. As a result, evaluating MLRA is computationally inexpensive. MLRA is not an optimal policy for the fully observable problem.

Most Likely Model Predictive Control (MLMPC) This planning agent computes the optimal state trajectory for the fully observable problem starting from s_{ml} using Model Predictive Control (MPC). That is, it uses the negative reward model as a cost function to obtain the objective for the MPC.

Unlike MLRA, MLMPC is an optimal policy for the fully observed problem. However, it should be noted that this result does not necessarily translate to the partially observed domain. Appendix TODO compares the performance of both baseline policies for the fully observable case.

cref to appendix

3.3.2 POMDP Solvers

We use DESPOT and POMCPOW to solve the POMDP. For each solver we propose two strategies in which domain knowledge can be utilized to guide the policy search for the localization and planning problem.

DESPOT

As presented in Section 2.2.1, DESPOT uses upper and lower bounds, U_0 and L_0, on the value function to guide the policy search and prune the determinized policy tree. Intuitively, the tighter these bounds are on the true value function, the more efficiently the algorithm will focus on exploring the relevant trajectories. On the other hand, if these estimates violate the true bounds on the optimal cost to go, DESPOT with perform sub-optimally. However, as proven in [SomaniEtAl13], the algorithm is robust against approximation errors. That is, similar to informed graph search algorithms like $A*$, the performance of the algorithm will degrade gracefully with violations of U_0. In fact, similar to bounded relaxations of $A*$, non-admissible heuristics may help to find good solutions earlier. Therefore, even if U_0 is not a true upper bound on the optimal value, it may help to improve performance when limited planning time does not allow to thoroughly explore the search space. Furthermore, it should be noted that finding tight bounds at the expense of high compute also defeats the purpose. In the limit, finding the tightest upper and lower bound on the optimal value requires solving the entire POMDP. Therefore, choosing suitable upper and lower bounds is an important design choice when adapting DESPOT for a specific problem.

There exist a variety of strategies to compute these upper and lower bounds. In the following we present two of the strategies we found most effective for the problem studied here.

Analytic Bounds In this strategy we compute the bounds using an analytic, closed form approximation for U_0 and L_0. Due to the structure of the reward model, we can directly compute the cumulative reward for an arbitrary n-step, collision-free trajectory. Let $\hat{\Sigma} : \mathbb{N} \to \mathbb{R}$ be this mapping. We compute the upper bound, U_0, on the optimal value by considering a strict relaxation of the problem: First, we assume that the robot is located at the closest

position to the goal out of all particles in the current belief; Second, we assume that the robot can immediately move on a straight line to the target. With this relaxation we compute the minimum remaining number of steps, $n_{\min}$, from the minimum Euclidean distance over all particles to the goal and the maximum translational speed. For the lower bound, L_0, we proceed in a similar fashion. In this case, however, we consider the worst case particle by using the ℓ_1 norm as a distance metric combined with an under-approximation on the velocity to obtain $n_{\max}$. In either case, we use $\hat{\Sigma}$ to map the step estimate to a corresponding return. Using these approximations, U_0 is a true upper bound on the optimal value. While for L_0 there is no strict guarantee to be a true lower bound, we found that most cases it is well behaved.

Rollout Lower Bound This strategy uses the same upper bound as the first approach. However, for L_0 we use a default policy rollout to obtain a true lower bound. In order to generate a structured motion, we use a *fixed-action* policy that always selects the forward action, instead of the common random policy rollout. By this means, the rollout policy provides a tight lower bound once evaluated from a state that already faces the exit of the room.

POMCPOW

POMCPOW uses a value estimate to approximate the remaining reward to be collected from a leaf node of the policy tree. Due to the design of the algorithm, this value estimate is only ever invoked on observation nodes when visited for the first time (cf. Algorithm 2). As a result, the corresponding belief consists of only a single state and the problem is de facto fully observable. Hence, in order to adapt this algorithm for a specific problem one can design the value estimate based on the corresponding MDP. Hereafter, we present two strategies strategies to compute this value estimate for the localization and planning problem.

Analytic Value Estimate We compute an analytic value estimate for state, s, with a strategy similar to the one used to obtain analytic bounds for DESPOT. We relax the planning problem by assuming that the robot can move immediately from s to the goal on a rectilinear trajectory. Using this relaxation we can compute the estimate on the remaining number of steps to the goal, $\hat{n}$. We then obtain the value estimate for an $\hat{n}$-step, collision-free trajectory by propagating $\hat{n}$ through $\hat{\Sigma}$, introduced above.

Rollout Estimate A commonly used strategy in MCTS is a random policy rollout [BrowneEtAl12]. For this problem, however, random play-outs do

not provide a good value estimate since random trajectories will often fail to reach a terminal state within reasonable time. Instead, we use the *fixed-action* policy proposed for the DESPOT lower bound approximation in the preceding paragraph.

3.4 Evaluation and Discussion

We evaluate the performance of each strategy by running $N_{\text{sim}} = 1000$ simulations for every setup. In an effort to reduce sampling variance, we initially fix N_{sim} random seeds. These seeds are shared across the different setups for a fixed run. By this means, the ith run of each setup will see the same random outcome (i.e. same initial conditions and noise trajectories, etc.). In order to handle cases in which the agents fails to reach a terminal state in a reasonable amount of time, each simulation is run for a maximum of number of time steps, $T_{\text{max,sim}} = 300$. The choice of this upper bound on simulation time is justified by evaluation of the optimal policy for the fully observable problem. In the corresponding MDP with the same set of initial conditions, MLMPC reached the goal within a maximum number of 56 over all scenarios. Therefore, it is reasonable to assume that a well behaved policy for this POMDP should be able to reach a succeeding terminal state within a time horizon more than five times longer than the worst case fully observable run.

We begin by comparing the performance of each strategy by examining the cumulative discounted reward. Since maximizing the sum of discounted rewards is the objective function for this optimization problem, this is the most natural benchmark. In order to account for the reward discarded in cases where $T_{\text{max, sim}}$ is exceeded, we penalize the value of these runs by extrapolating the living penalty over infinite horizon. Since the sum over the infinite sequence of discounted rewards is a geometric series, we obtain the closed form solution,

$$r_{T_{\text{sim,max}}} = \sum_{t=1}^{\infty} r_{\text{time}} \gamma^{t-1} = r_{\text{time}} \frac{\gamma}{1-\gamma}. \tag{3.3}$$

Since this reward is collected at the end of the simulation horizon it must discounted accordingly, entering the objective function as

$$\Delta V_{T_{\text{max,sim}}} = r_{T_{\text{sim,max}}} \gamma^{(T_{\text{max,sim}}-1)} \tag{3.4}$$

Following this argument, with the parameters chosen for this problem an agent that does not reach a terminal state within the simulation horizon is penalized with $\Delta V_{T_{\text{max,sim}}} = -0.49$.

Figure 3.2 shows the mean and Standard Error of the Mean (SEM) of the cumulative discounted reward for 1000 experiments over each policy. The SEM

approximates the standard error between the empirical mean obtained from the sampled experiments and the true mean of the distribution, obtained hypothetically for $N_{\text{sim}} \to \infty$. The results shows that under this metric all POMDP planners perform significantly better than the baseline policies. Within the baselines the planning agent (MLMPC) performs better than the proposed reflex agent (MLRA). Out of all POMDP planners POMCPOW with analytic value estimate achieves the best results. Furthermore, notice that DESPOT shows approximately the same performance for both bound approximations while for POMCPOW the analytic value estimate shows a yields improvement.

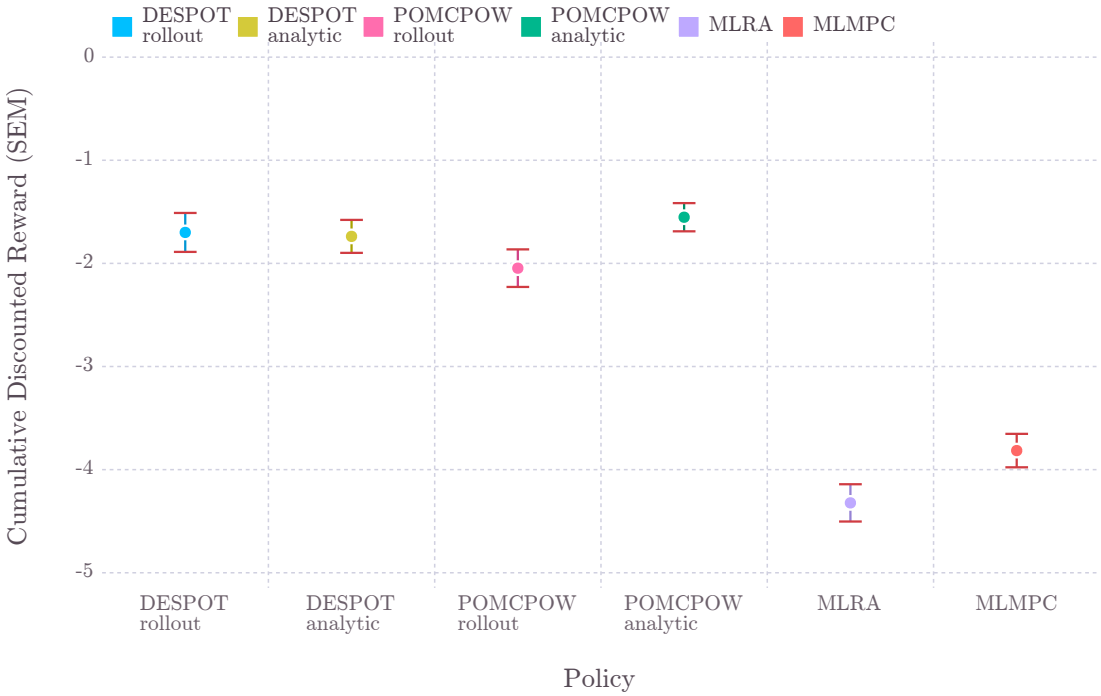


Figure 3.2: Mean and Standard Error of the Mean (SEM) of the cumulative discounted reward for each policy introduced in Section 3.3.

Further insight is gained by looking at the distribution of returns for each policy. Figure 3.3 shows an approximation of the probability density function for the value distribution based on the 1000 samples for each policy. From this visualization another important difference becomes apparent. While the mean performance of all POMDP planners is approximately the same, those versions of DESPOT and POMCPOW using an analytic strategy for bound and value approximation manage to reduce the lower tail end of the distribution. As a result, these planners show less variance in performance than solvers using the corresponding rollout estimates.

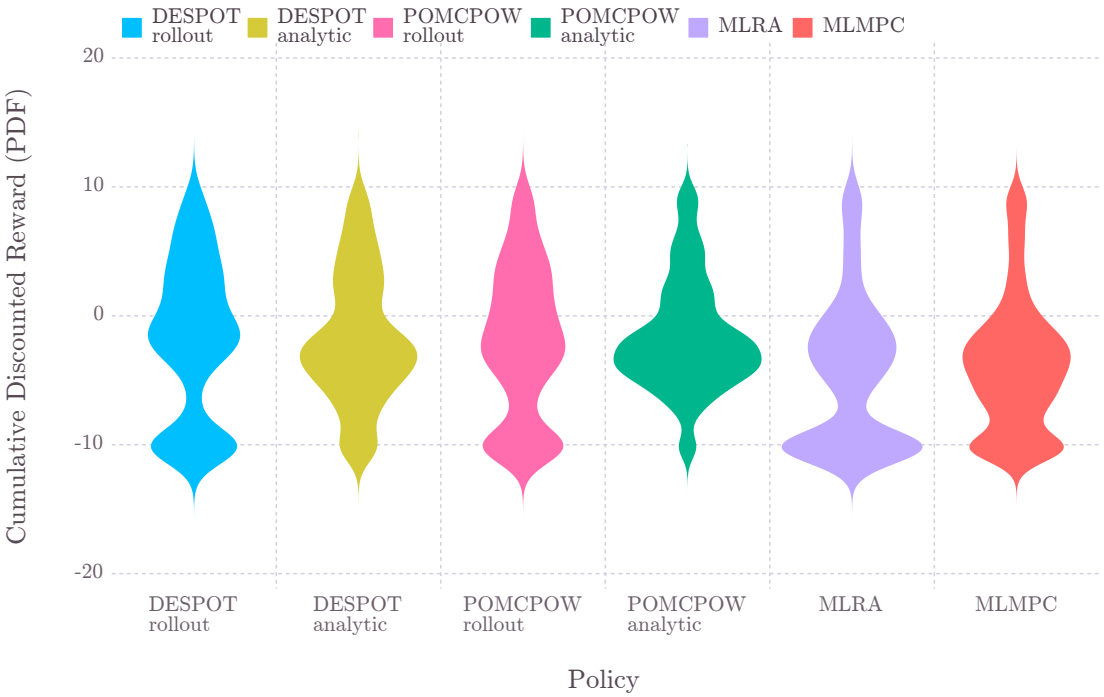


Figure 3.3: Approximation of the Probability Density Function (PDF) for the discounted cumulative reward for each policy.

These quantitative results also manifest in the qualitative performance of each policy. When looking at trajectories generated by each solver, we notice that the POMDP planners typically generate trajectories that efficiently eliminate a lot of uncertainty. The baseline strategies on the other hand only passively reduce uncertainty by colliding with wall if the mode of the belief is located in cluster far away from the true position of the robot. For many cases this kind of passive information gathering is strong enough to help the robot eventually reach its goal. However, we notice that for certain typologies of the belief the baseline policies fail to reduce uncertainty, e.g. if the belief is highly symmetric. In these cases, often slight modifications of the trajectory would have been sufficient to break symmetry and completely eliminate a cluster. While DESPOT and POMCPOW are able to actively identify information gathering strategies for these cases, the baseline policies often oscillate until repeated re-sampling in the particle filter breaks the symmetry to an extent that allows to stabilize decisions.

Furthermore, one can observe that all POMDP planners are significantly better at avoiding failures (falling down the stairs). If the planner is presented with a belief that features a cluster near the stairs, the agent typically chooses a sequence of actions that is safe for this hypothesis and reduces uncertainty in this regime

to avoid entering the failing terminal state. Since the baselines consider only the most likely state, these policies fail when the topology of the reward is such that this state is facing the goal while the true position – guided by particles with a smaller likelihood weight – is facing the stairs.

Finally, we observe that the POMDP planners using a rollout strategy to guide the search behave more myopically. While shortsighted planning still allows the agent to be robust against falling down the stairs, these policies tend to get stuck when a long sequence of actions is required to eliminate uncertainty. A possible reason for this is the fact that rollouts are more computationally expensive than analytic bound and value estimates. Additionally, in these regimes the fixed-action rollout typically does not provide a tight lower bound or accurate value estimate for DESPOT and POMCPOW, respectively. In the case of DESPOT, even though the sparse approach most likely allows some branches of the tree to be constructed to full depth, the loose lower bound causes policy search to be poorly guided while the limit planning time prevents sufficient exploration of more promising branches. In the case of POMCPOW the computationally expensive rollout performed for every iteration of the algorithms hinders the policy tree to reach sufficient depth within the computation budget. Since rewards are sparse, on a short horizon all futures appear to provide similar reward. Hence, DESPOT and POMCPOW fail to identify a promising strategy within their mostly shallow policies trees.

Summarizing the results of this qualitative analysis, *DESPOT-analytic* and *POMCPOW-analytic* generate highly efficient and robust behaviors. All other policies show weaker with different kinds of failure modes: The rollout versions of the POMDP planners more often fail to reach the exit due to myopia but still show good performance when it comes to avoiding failure states; MLMPC rarely fails to reach the goal state but often takes very long or becomes unlucky when greedily tyring to reach the exit; MLRA shows an overall worse performance than all other policies.

These results are additionally confirmed when evaluating metrics other than the objective of the optimization problem. Figure 3.4 shows a histogram of the outcome statistics for each policy. We distinguish between three classes of outcomes: *success (+)*, the robot reached the exit; *failed to exit (0)*, the agent did not make it to the exit within the simulation horizon; and *failure (-)*, the robot fell down the stairs. Notice that *POMCPOW-analytic* succeeds in than 96.9% of all runs. Given the fact that about 3% of the perimeter is covered by stairs and initially the robot has to take a random guess to receive information, this policy can be considered near optimal with respect to safety.

Additional evidence for the findings discussed above is provided by the statistics of the cumulative *undiscoutned* reward. While this metric does not match the

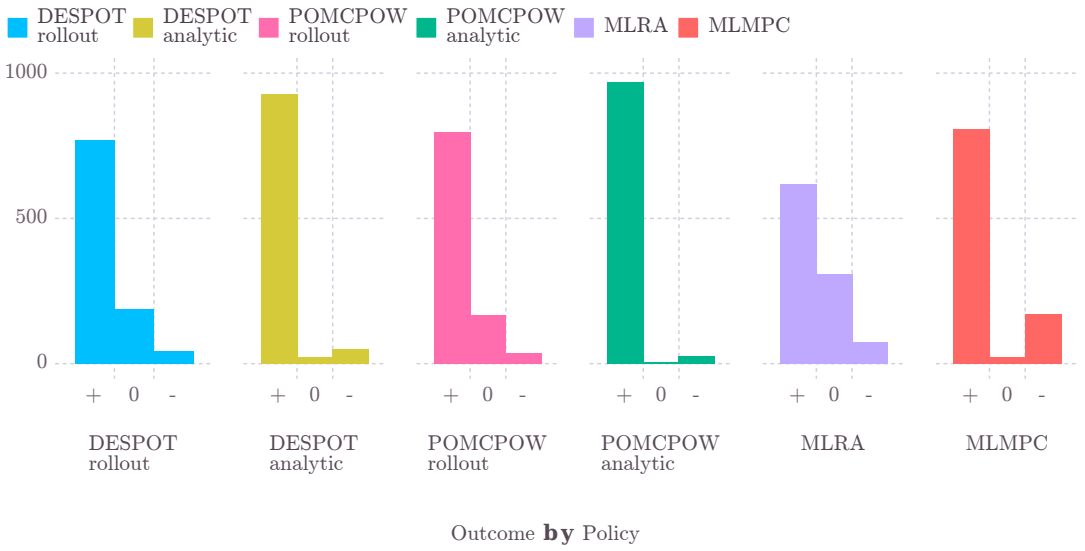


Figure 3.4: Histogram of outcome frequencies grouped by policy. Outcome classes: *success (+)*, the robot reached the exit; *failed to exit (0)*, the agent did not make it to the exit within the simulation horizon; and *failure (-)*, the robot fell down the stairs.

objective for the infinite horizon formulation, it may align more closely with the perceived *qualitative performance* since one can only observe the agent for a finite time horizon. Figure 3.5 depicts the SEM and Probability Density Function (PDF) approximation for the undiscoutned reward sequence. Using this metric, it becomes clear that the analytic versions of the POMDP planners outperform all other strategies. Here, we can additionally identify a subtle performance gap between *DESPOT-analytic* and *POMCPOW-analytic*. The PDF approximation of the cumulative undiscounted reward show that *POMCPOW-analytic* manages to eliminate large parts of the low value tail.

In summary we can state that for this problem we were able to generate robust and efficient behaviors through POMDP planning. We show that by carefully integrating domain knowledge in a computationally efficient manner, we obtain policies significantly safer and more efficient than the common baseline approaches.

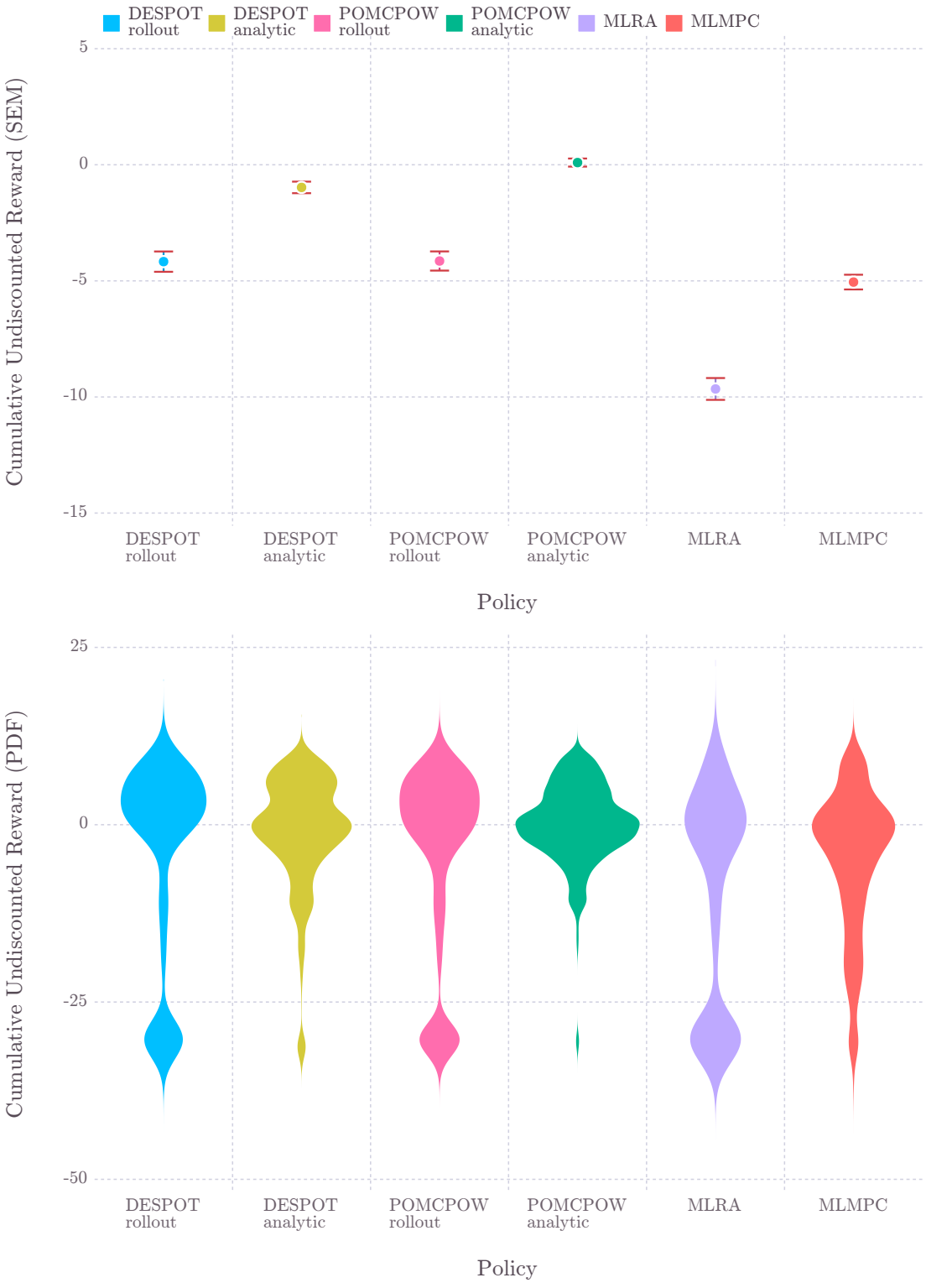


Figure 3.5: Statistics of the cumulative *undiscounted* reward for each policy. Top: Mean and Standard Error of the Mean (SEM). Bottom: Approximation of the Probability Density Function (PDF).

Chapter 4

Motion Planning with Latent Human Intentions

Robots that are designed to assist humans almost inevitably have to operate in a shared environment. Human Robot Interaction (HRI) requires autonomous agents to navigate safely in domains that typically do not feature safety barriers to physically separate them from humans. Therefore, robots must ensure human safety through careful planning and robust behaviors. At the same time, trajectories of humans are hard to predict as they follow complex behaviors whose dynamics are only partially understood. For this reason, research in the past has moved from simple rule-based and deterministic models [HelbingMolnar95, BursteddeEtAl01] towards data driven probabilistic predictions that approximate the future as a distribution over trajectories [KretzschmarKudererBurgard14, AlahiEtAl16, GuptaEtAl18].

Incorporating stochasticity in the prediction pipeline allows the planner to model uncertainty over both, high-level intentions (e.g. *where* does the human want to go), and low-level motion behavior (e.g. *how* does the human want to go there). Planning strategies that take this uncertainty into account promise to provide robustified and potentially more efficient policies for navigation around humans. While the properties gained from this kind of planning are desirable for many HRI problems, reasoning over distributions of possible futures may pose a challenging POMDP. Therefore, a lot of research has focused on recovering some of these properties by proposing domain specific simplifications for these applications [FernEtAl07, SadighEtAl16, JavdaniEtAl18, FisacEtAl18].

On the other hand, recent research in this field suggests that through increased performance of modern solvers, POMDP approaches for motion planning problems with HRI are becoming increasingly practical and that this kind of reasoning can help to generate robustified and more efficient behaviors for this domain.

[BaiEtAl15] use the DESPOT to control the speed of an autonomous golf cart for navigation in a crowd. They show that by reasoning over future observations the system is able to maneuver more safely while reaching the goal faster and increasing passenger comfort through smoother trajectories than the proposed greedy baseline. [SunbergHoKochenderfer17] examine the value of inferring the internal state of traffic participants for autonomous freeway driving. The results presented in their work show that the problem allows for a significant increase in performance when the agent is omniscient to the individual internal state of other vehicles compared to planning with a static *normal* behavior for all traffic participants. They demonstrate that inference of the latent behavior parameters in combination with POMDP planning allows to greatly reduce the gap to the omniscient upper bound.

While a lot of work in motion planning under uncertainty has focus on either *problem specific simplifications* on the one hand or *full POMDP solutions* on the other, few results exist on the direct comparison of the two. In this chapter we consider a case of motion planning in the presence of humans for which a sophisticated domain specific strategy has been proposed in [FisacEtAl18]. This strategy simplifies the planning problem by neglecting future observations. Instead, human prediction are performed on expectation from the current belief, providing series of probability maps for future human motion. The authors propose a method that allows planning with these probabilistic predictions by means of conventional motion planning algorithms. We implement this strategy in the *POMDPs.jl* framework to compare its performance with behaviors generated by POMCPOW.

We begin by stating the details of the motion planning problem in Section 4.1. Section 4.2 then formalizes this problem as a POMDP. Section 4.3 briefly presents the domain specific approximate planner proposed in [FisacEtAl18], as well as the POMCPOW solver adapted for this problem. In Section 4.4 we describe the implementation of a software model for interacting humans and robots. Finally, we evaluate the performance of both solvers and discuss the results in Section 4.5.

4.1 Problem Statement

In this chapter we study a motion planning problem for a robot in a shared environment with a human actor. The assumptions made for this planning problem aim to closely match the setup described in [FisacEtAl18].

High Level Problem Consider a human and a robot navigating in a common environment. As a running example, this problem may be envisioned as an indoor

navigation problem. In this problem, the robot aims to efficiently reach a preset goal location inside the room while trying to avoid collisions with the human. At the same time, the human does not pay attention to the robot. That is, the decisions of the human actor do not depend on the location of the robot. Furthermore, the human has time-varying intentions that are not known by the robot. At every time step the human aims to reach a certain goal inside the room. Once arrived at this goal, the human may either stay at this goal or proceed to another goal location. Furthermore, the human has a small likelihood of changing the intended walk target mid way. The robot has a model for some of the locations humans might want to go to. However, the planner neither knows the exact path the human will take, nor is its model of human goals complete. That is, the person might aim to reach an unmodeled goal location. At every time step the robot receives a perfect observation of the current position of both itself and the human. The robot ultimately needs to plan and execute a trajectory to a goal state according to some notion of efficiency, while avoiding collisions with the human.

Human Behavior Model This problem poses a challenging planning problem as it requires the agent to carefully reason about the intentions of the human in order to avoid collisions while trying to reach its own goal. Furthermore, in order to be robust against cases in which the person tries to approach an unmodeled goal, the agent needs to make safe fall-back predictions when the human behaves unexpectedly. At the same time, planning with the worst case assumption by trying to avoid the humans forward reachable set is too conservative.

In this problem setting, the robot is assumed to have access to suitable human reward function. In practice such reward model can be learned offline from prior human demonstrations, e.g. using inverse reinforcement learning. By means of a noisy-rationality model used in cognitive science [BakerTenenbaumSaxe07], the planner can make probabilistic predictions of human actions given the state,

$$P\left(a_{\mathrm{H}}^{t} | s_{\mathrm{H}}^{t}; \beta, \theta\right) \propto e^{\beta Q_{\mathrm{H}}\left(s_{\mathrm{H}}^{t}, a_{\mathrm{H}}^{t}; \theta\right)}. \tag{4.1}$$

Here, a_{H}^{t} is the human action taken at time t; s_{H}^{t} is the state of the human at this time; $Q_{\mathrm{H}}(s_{\mathrm{H}}, a_{\mathrm{H}}; \theta)$ is the state-action value function of the human, with θ being the parameters of the Q-value encoding latent human intentions (e.g. the goal location); β is the so called *rationality coefficient*.
This model encodes the assumption that humans are more likely to select actions that provide a higher Q-value. The rationality constant, β, determines the degree to which the robot expects the human to align with the provided reward model. For $\beta = 0$ the model in Eq. (4.1) degenerates to a uniform distribution over all

actions. For $\beta \to \infty$ the human acts perfectly rational with respect to the utility model, Q_{H}.

Here, we treat both θ and β as latent parameters. That is, the agent needs to reason over the parameters of the human reward model (e.g. its current goal) as well as the accuracy of this model.

4.2 POMDP Formalization

The problem is formalized as a POMDP from the robot perspective. It is characterized by the following properties introduced in Section 2.1.2:

State Space $\mathcal{S}$. The state of the environment is defined as the joint state of the human and the robot. For clarity, we introduce the subscripts H and R to denote the affiliation of a part of the state to the human or the robot, respectively. Furthermore, we differentiate between *external states* (i.e. state variables that encode physical properties of the environment) and *internal states* (i.e. state variables that determine the behavior dynamics of an agent), denoted with subscript E and I, respectively. Using this notation, a state s is composed as:

$$s = [s_{\mathrm{E,R}}, s_{\mathrm{E,H}}, s_{\mathrm{I,H}}]^T \tag{4.2}$$

It is important to note that, in order to make s Markovian, we need to incorporate the internal state of the human. That is, in contrast to a game theoretic approach, humans are not explicitly modeled as players taking actions but rather in terms of dynamics of the environment. In the model of the robot, the human internal state consists of the reward model parameters, θ, and the model confidence, β. The external state of each actor is defined by its current position.

Using this definition of the state, $\mathcal{S}$ is the set of all possible combinations of positions the human and the robot and the human internal state.

Action Space $\mathcal{A}$. An action in this problem refers *exclusively* to the robot action. At every time step the robot can select one of the following six actions: *"north", "east", "south", "west", "to goal", and "stay put"*. The latter keeps the robot at its current position. All other actions will move the agent with constant velocity in the corresponding direction. Here, *to goal* is a state dependent action, that is directed straight at the robot's goal location. The set of these six actions constitutes the action space, $\mathcal{A}$.

Transition Model $\mathcal{T}$. The dynamics of the robot and the human are decoupled. Therefore, we can compose the joint state transition from the dynamics of

each actor. The transition of the robot, s_{R} to s'_{R}, depends on the action as described above. The transition of the human obeys Eq. (4.1). In our model, the human has a discretized action space that allows it to stay put or move in one out of eight preset directions (i.e. *north*, *north-east*, *east*...). Hence, at every time step the human action is generated by sampling from the generalized Bernoulli distribution obtained from Eq. (4.1) and the human is moved in the corresponding direction. Additionally, the transitions of both actors are corrupted by Gaussian process noise. A terminal state is reached if either the robot collides with the human or it reaches the goal.

Reward Function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The reward model of the robot is defined as:

$$\mathcal{R}(s, a, s') = r_{\mathrm{time}} + \delta_{\mathrm{collision}}(s') r_{\mathrm{collision}} + \delta_{\mathrm{close}}(s') r_{\mathrm{close}}. \tag{4.3}$$

The reward model is parametrized by the following quantities: r_{time} is a living penalty encouraging the robot to minimize the time until the goal is reached; $r_{\mathrm{collision}}$ is a penalty for collision with the human or the walls of the room; r_{close} is the penalty obtained when entering the one-step forward reachable set of the human.

Discount Factor γ. Rewards are discounted with $\gamma = 0.99$.

Observation Space $\mathcal{O}$. At every time step the robot receives an exact measurement of the human and itself. That is, the robot gets to observe the external state, s_{E} of both actors. As a result, $\mathcal{O}$ is strict subset of the state space.

Observation Model $\mathcal{Z}$. Since the position sensor is exact, the observation model is a Dirac delta function, evaluating non-zero for a state, s, if and only if the external state component matches the observation, o.

It must be noted that, in order to match the problem statement provided in Section 4.1, we consider two slightly different versions of this POMDP. One version is used to represent the dynamics of the simulated environment, the other is used as a planning model for the robot. This clear distinction is necessary to encode the modelling assumption that the robot does not know about all human goals. That is, the robot does not have access to the true behavior model of the human.

By inspection of the problem formalized above it becomes apparent that the assumptions made in [FisacEtAl18] poses a Mixed Observability Markov Decision Process (MOMDP) – a special case of the more general POMDP. That is, we can factor the state into a fully observed part (s_{E}, the external state) and a partially observed part (s_{I}).

4.3 Solution Strategies

We solve the POMDP described in Section 4.2 using two different strategies. First, Section 4.3.1 presents Probabilistically Safe Robot Planning (PSRP), a method proposed in [FisacEtAl18] that exploits the special structure of the problem to reduce the computational effort of solving the full POMDP. Thereafter, Section 4.3.2 presents an adaptation of POMCPOW for this problem.

4.3.1 Probabilistically Safe Robot Planning

Probabilistically Safe Robot Planning (PSRP) is a method for motion planning with probabilistic predictions. While an exhaustive discussion of the details of this method is beyond the scope of this work, we aim to provide insight into the general idea of this method as a basis for further discussion. For additional information the reader may refer to the original paper [FisacEtAl18].

PSRP is designed for a special class of POMDPs with the following properties: The problem must be a MOMDP in which the notion of *immediate safety* can be evaluated solely on fully observed states. Furthermore, all such state components affecting immediate safety must be either *actuated* (i.e. directly controlled by the agent) or *fully decoupled* from the robot dynamics to allow for predictions independent of the agent's decisions. Both requirements are satisfied for the problem studied in this chapter: Immediate safety depends only on the position of the human and the robot, both contained in the *external* (fully observed) part of the state; Predictions of the human state can be made independently of the robots trajectory as she ignores the robot position in her decision making process.

The methods exploits these properties to significantly reduce the complexity of policy search by essentially converting it to an equivalent non-probabilistic formulation. The converted problem can then be solved using conventional motion-planning techniques. On an abstract level the algorithmic procedure of PSRP performs the following computations to select the next action:

Inference / Belief Update At every time step, the robot receives an exact observation of the external state of the environment. These observations are used to maintain a belief over the latent parameters of the human model presented in Eq. (4.1). These are the goal location of the human, θ, and the model confidence, β. The exact update of the joint belief takes the form

$$b'(\theta, \beta) \propto P(s_{\mathrm{E,H}} | s_{\mathrm{H}}; \theta, \beta) b(\theta, \beta). \tag{4.4}$$

Since the exact Bayesian update is computationally demanding, we perform Monte-Carlo integration to run the inference. Note that this is a subtle

difference to the original implementation, where the authors decide the discretize the parameter and state space to obtain a tractable update rule.

Prediction Using the belief provided by the inference, the agent computes a probabilistic prediction of the human position for K future time steps. This is done by recursively propagating the particles belief through a generative model matching the statistics of Eq. (4.1). Since the human is modeled to act noisily rational and additionally the particle belief represents a distribution over possible human behaviors, the prediction introduces uncertainty to the future trajectory of the human. That is, while at the current time step the exact position of the human is known from the most recent observation, at future times the robot only has a probabilistic estimate of the person's location. An example of such open-loop predictions of the human location is depicted in Fig. 4.1. These predictions are used to synthesize a probabilistic obstacle map for the next K time steps.

Planning The agent uses the K probabilistic obstacle maps obtained from the preceding step to compute the optimal sequence of actions for the next K time steps. Since the action-space is finite, we use A^* to solve this policy search problem. We use the negative reward model as a cost function to obtain the cost function for the graph search problem. Additionally, in line with the idea of [FisacEtAl18] we prune search nodes for which the collision likelihood exceeds a preset threshold, ε.

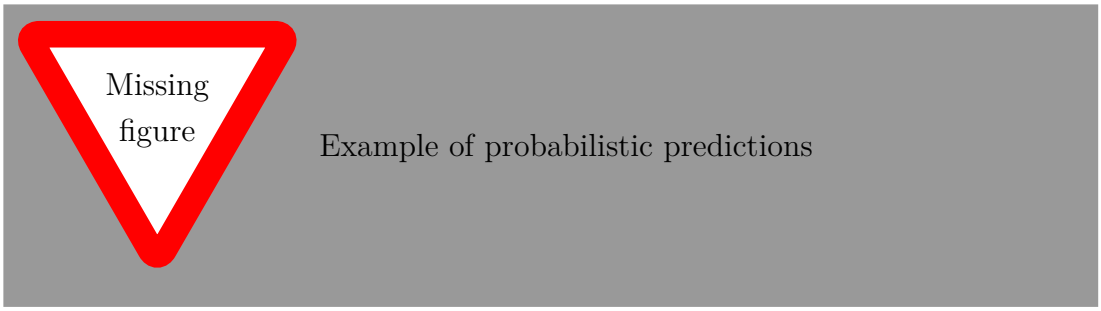
The result of this computation is applied in the fashion of a K-step receding horizon MPC. That is, at every time step the robot only executes the first action of the optimal action sequence computed above. At the next time step, all steps are repeated closing the loop over the received observation.

We implement this solver using the *POMDPs.jl* interface. By this means we can share the procedures for inference and sampling, thus allowing for a direct comparison of the solver performance. Our implementation of this algorithm is carefully optimized for computational efficiency. In particular, we use k-d trees to perform nearest-neighbor search when checking for collisions of particles in the belief prediction. Furthermore, we implement an optimized lightweight graph search library to efficiently solve the policy search problem. The latter has been published as an open-source Julia package[1] to allow for usage in other projects.

4.3.2 POMCPOW

The POMDP formalization presented in Section 4.2 allows to directly apply POMCPOW to the problem. To ensure a fair comparison, the planner uses

[1] `https://github.com/lassepe/GraphSearchZero.jl`



.

Figure 4.1: Example predictions of the human future location in Probabilistically Safe Robot Planning (PSRP).

Refine caption once figure is finished.

the same particle filter as proposed for PSRP in Section 4.3.1 for inference of the latent parameters of the human model.

Considering the insight obtained from the solver comparison discussed in Chapter 3, we choose to embed domain knowledge through an analytic value estimate. Using the same kind of reasoning as discussed in Section 3.3.2 we approximate the value at the leaf of the policy tree using a relaxed version of the problem in which the robot can move on a straight, collision-free line to its goal location. By means of this relaxation, the value estimate is given as the discounted sum of living penalties over the remaining steps to the goal.

4.4 A Software Model for Motion Planning with Latent Human Intentions

We implement a generic software model in the Julia programming language to simulate robotic navigation problems in the presence of humans. With the reuse for other HRI motion planning problems in mind, our implementation is designed to accommodate convenient interchangeability of the human behavior model. The loosely coupled architecture in combination with the expressiveness of the Julia programming language allows to implement additional human models within a few lines of code. Furthermore, our implementation builds upon the *POMDPs.jl* framework to allow for direct interaction with a wide range of POMDP solvers.

Additionally, we provide a simple visualization of the state of the environment as well as the evolution of the belief and reward trajectory. This visualization can be easily augmented to display additional meta data. An example of such visualization is depicted in Fig. 4.2. The left half of this figure displays the current

state of the environment. Here, the robot position is marked in pink while the human position is marked in red. The goal location of the robot is marked in green. The belief over the one step ahead prediction of the human is visualized by blue particles. Here we use the goal conditioned Boltzmann model for prediction of the human trajectory. The inferred goal of each particle is highlighted in blue and connected with a line. The right half of the figure shows additional meta data. Here, the top half displays a histogram over the current belief of the model confidence, β. The lower half shows the evolution of the cumulative reward up to current time step.

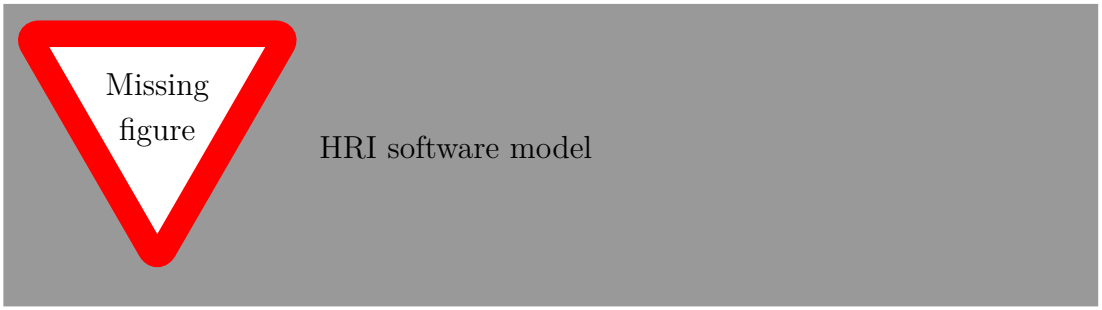


Figure 4.2: Example visualization of the environment state (left) as well as additional meta data (right).

4.5 Evaluation and Discussion

We evaluate the performance of each strategy by running 5000 simulations for each policy using the software model presented in Section 4.4. Again, each policy is simulated on the same set of random seeds to reduce sampling variance. Additionally, in an effort to obtain a comparison that focuses on safety critical situations, we only consider scenarios in which a naive *straight-to-goal* policy is not successful. We create these scenarios by sampling initial conditions from uniform prior and reject all samples for which the simulated naive policy allows the robot to reach its goal.

In contrast to the experiments presented in Chapter 3, we don't explicitly set an upper bound on the decision time per step for POMCPOW but fix the number of MCTS iterations instead. We choose the number of iterations by evaluating the performance of POMCPOW over a sequence of parameters and select the number of iterations beyond which saturation is observed. Furthermore, for the experiments discussed we are able to simulate all policies until a terminal state is reached. Hence, we don't need to adjust the returns to account for rewards beyond the simulated horizon. Additionally, we limit the planning horizon for both solvers to ten steps to avoid any bias due to unbalanced prescience.

We begin by comparing the performance of each policy by examining the cumulative discounted reward – the metric whose maximization is the objective of this optimization problem. Figure 4.3 shows the mean and SEM of the cumulative discounted reward for 5000 experiments over both policies. These results show that POMCPOW performs significantly better than PSRP.

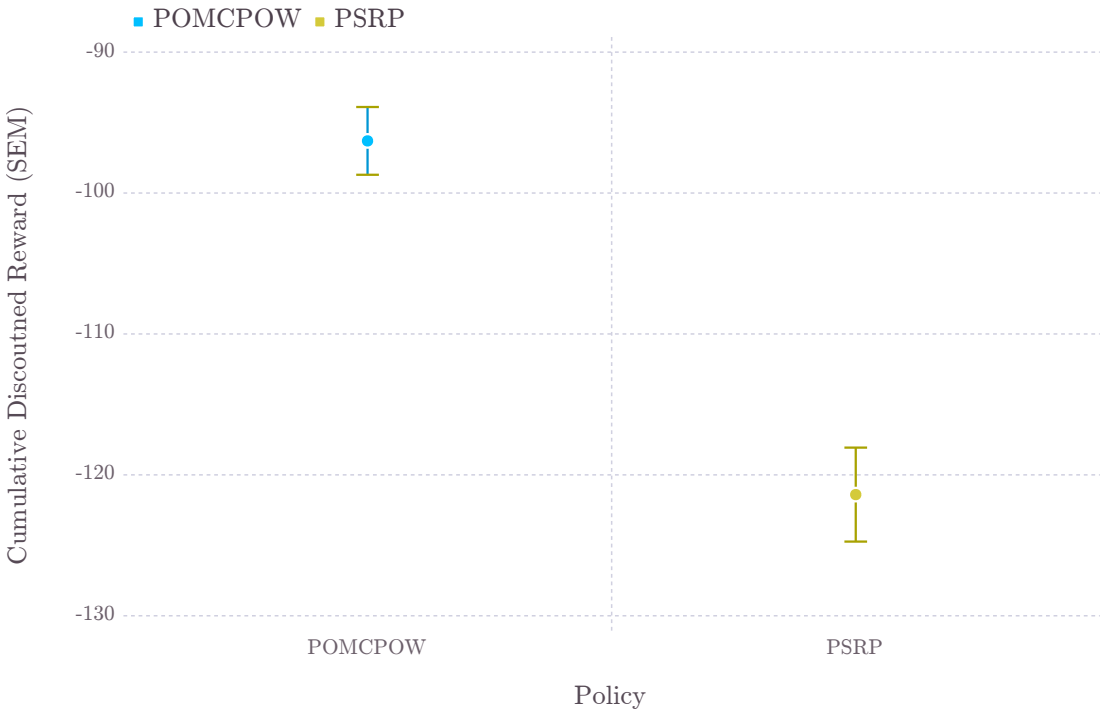


Figure 4.3: Mean and Standard Error of the Mean (SEM) of the cumulative discounted reward for POMCPOW (blue) and PSRP (yellow) as introduced in Section 4.3.

Further insight is gained by evaluating the outcome statistics for each policy. Figure 4.4 depicts the frequencies of outcomes for each policy. Since both policies reach a terminal state within the simulated horizon for all sampled scenarios, here we only consider two classes of outcomes: *success*, the robot reached the goal location; and *failure* the robot collided with a wall or the human. The outcome statistics show that both policies are able to successfully navigate the robot to its goal location in the vast majority of all scenarios. However, POMCPOW still performs significantly better with respect to safety. In fact, the collision rate in the sampled scenarios for POMCPOW (3.40%) is only half has high as for PSRP (7.08%).

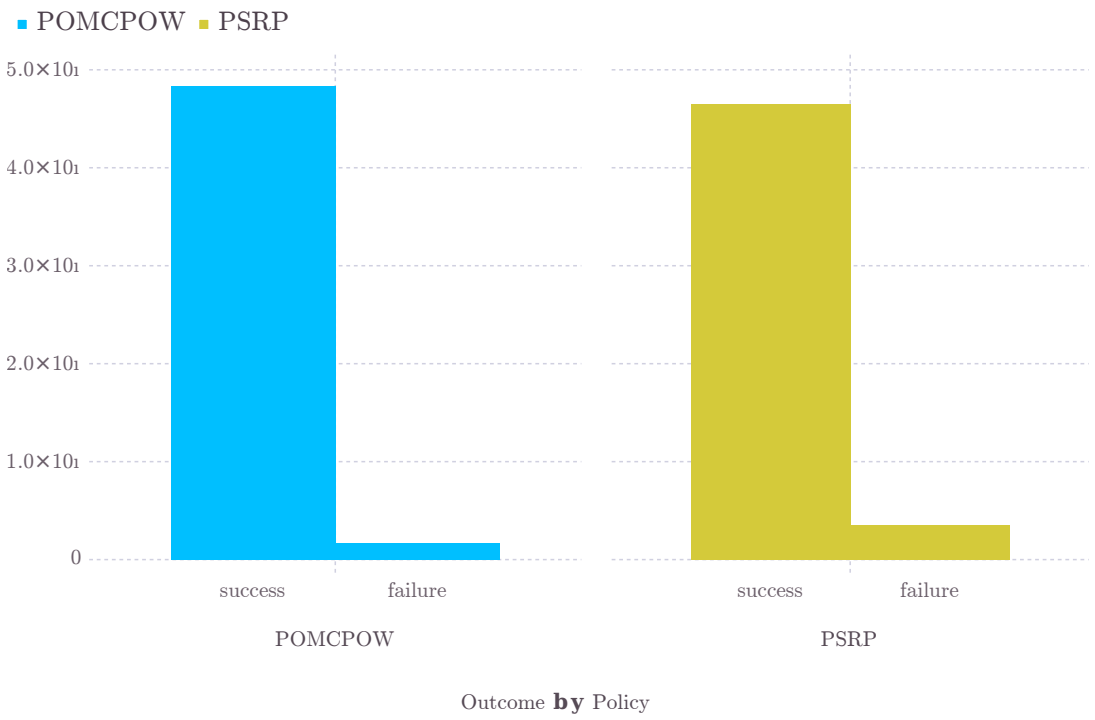


Figure 4.4: Histogram of outcome frequencies grouped by policy. Outcome classes: *success*, the robot reached the goal location; and *failure* the robot collided with a wall or the human.

While POMCPOW provides quantifiable better behaviors, this advantage comes at the cost of higher computational effort. On average, decision making in POMCPOW takes 0.279 s while for PSRP a decision is computed within 0.042 s.

A qualitative analysis of the behaviors generated by each solvers provides further insight into the reasons for this gap in performance. We observe that in many cases both solvers follow very similar strategies. When the human walks towards one of the goal locations known by the robot, her decisions are well explained by the behavior model used by the planner. In these cases, the model confidence, β, increases and the planner is able to make confident predictions of the future human position. Hence, for sufficiently high values of β the problem essentially approaches a deterministic planning problem. Under these conditions PSRP performs equally well as POMCPOW since optimal planning with probabilistic obstacles reliably finds suitable trajectories to avoid collisions and reach the goal efficiently. In fact, in the limit of $\beta \to \infty$ observation branching as employed by POMCPOW is rendered obsolete.

Furthermore, in the problem discussed here the human does not react to the robot. Therefore, observations whose statistics depend on latent variables are

independent of the robot action. As a result, in contrast to the application domain discussed in Chapter 3 this problem does not enable active information gathering. This property induced by the problem structure further reduces the effect of observation branching.
However, there still is a decisive difference between both strategies that causes the performance gap observed above. In cases where the human appears to act irrationally, the model confidence decreases and Eq. (4.1) essentially becomes a uniform distribution. As as a result, the sequence of probabilistic predictions used in PSRP features rapidly expanding uncertainty and the planner has to avoid a large region of the state space. We observe that during these maneuvers the agent often gets trapped in a corner of the room. When using POMCPOW on the other hand, the agent is aware that it will get further information to reduce the uncertainty in the future. Hence, the agent is able to approach the human more confidently, enabling the robot to navigate around the human in a less reactive manner.

In summary we can state that for the application example discussed in this chapter the difference between the domain specific solver, PSRP, and the generic POMDP solver, POMCPOW, is less dramatic. Since by design the problem does not allow for active information gathering, in many cases both solvers generate similar behaviors. However, when faced with significant uncertainty over the human intention, POMCPOW is still able to navigate more safely, since observation branching allows to keep uncertainty over future human positions bounded.

Chapter 5

Summary

Write once everything else has converged.

Mention contribution again.

Chapter 6

Formatting - TODO

- genitive ”s” (robots vs. robot’s, its vs it’s)
- Introduction must be page one (see TOC)
- (done) ix acronym capitalization
- maybe have acronyms rendered in italic
- section references in caps?
- look for common spelling mistakes
- look for ”will”, most things should be ”present”
- avoid ”don’t”, ”won’t”, ”doesn’t” etc.
- flows → follows
- make sure that there is only one version of writing ”Monte-Carlo” (hyphen)
- policy-tree vs belief-tree
- maybe avoid using ”That is,” at the beginning of a sentence but it is not that important

”its” is analogous to ”his” or ”hers”; it’s is a contraction. You’ll probably never use it’s in technical writing.

Bibliography

[AlahiEtAl16] Alahi, A.; Goel, K.; Ramanathan, V.; Robicquet, A.; Fei-Fei, L.; Savarese, S.: Social lstm: Human trajectory prediction in crowded spaces. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 961–971, 2016.

[Altman99] Altman, E.: Constrained Markov decision processes. CRC Press, 1999.

[AmatoEtAl15] Amato, C.; Konidaris, G.; Cruz, G.; Maynor, C.A.; How, J.P.; Kaelbling, L.P.: Planning for decentralized control of multiple robots under uncertainty. In 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 1241–1248, IEEE, 2015.

[BaiEtAl15] Bai, H.; Cai, S.; Ye, N.; Hsu, D.; Lee, W.S.: Intention-aware online pomdp planning for autonomous driving in a crowd. In 2015 ieee international conference on robotics and automation (icra), pp. 454–460, IEEE, 2015.

[BakerTenenbaumSaxe07] Baker, C.L.; Tenenbaum, J.B.; Saxe, R.R.: Goal inference as inverse planning. In Proceedings of the Annual Meeting of the Cognitive Science Society, Vol. 29, 2007.

[BandyopadhyayEtAl13] Bandyopadhyay, T.; Won, K.S.; Frazzoli, E.; Hsu, D.; Lee, W.S.; Rus, D.: Intention-aware motion planning. In Algorithmic foundations of robotics X, pp. 475–491. Springer, 2013.

[Bargiacchi13] Bargiacchi, E.: AI-Toolbox. `https://github.com/Svalorzen/AI-Toolbox`, 2013.

[Bertsekas05] Bertsekas, D.: Dynamic Programming and Optimal Control. Athena Scientific, 2005.

[BrowneEtAl12] Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.;

Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, Vol. 4, No. 1, pp. 1–43, 2012.

[BursteddeEtAl01] Burstedde, C.; Klauck, K.; Schadschneider, A.; Zittartz, J.: Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Physica A: Statistical Mechanics and its Applications, Vol. 295, No. 3-4, pp. 507–525, 2001.

[ChoudhuryKochenderfer19] Choudhury, S.; Kochenderfer, M.J.: Dynamic real-time multimodal routing with hierarchical hybrid planning. CoRR, Vol. abs/1902.01560, 2019.

[CouëtouxEtAl11] Couëtoux, A.; Hoock, J.B.; Sokolovska, N.; Teytaud, O.; Bonnard, N.: Continuous upper confidence trees. In International Conference on Learning and Intelligent Optimization, pp. 433–445, Springer, 2011.

[EgorovEtAl17] Egorov, M.; Sunberg, Z.N.; Balaban, E.; Wheeler, T.A.; Gupta, J.K.; Kochenderfer, M.J.: Pomdps.jl: A framework for sequential decision making under uncertainty. The Journal of Machine Learning Research, Vol. 18, No. 1, pp. 831–835, 2017.

[FernEtAl07] Fern, A.; Natarajan, S.; Judah, K.; Tadepalli, P.: A decision-theoretic model of assistance. In IJCAI, pp. 1879–1884, 2007.

[FisacEtAl18] Fisac, J.F.; Bajcsy, A.; Herbert, S.L.; Fridovich-Keil, D.; Wang, S.; Tomlin, C.J.; Dragan, A.D.: Probabilistically safe robot planning with confidence-based human predictions. arXiv preprint arXiv:1806.00109, 2018.

[GuptaEtAl18] Gupta, A.; Johnson, J.; Fei-Fei, L.; Savarese, S.; Alahi, A.: Social gan: Socially acceptable trajectories with generative adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2255–2264, 2018.

[HelbingMolnar95] Helbing, D.; Molnar, P.: Social force model for pedestrian dynamics. Physical review E, Vol. 51, No. 5, p. 4282, 1995.

[JavdaniEtAl18] Javdani, S.; Admoni, H.; Pellegrinelli, S.; Srinivasa, S.S.; Bagnell, J.A.: Shared autonomy via hindsight optimization for teleoperation and teaming. The International Journal of Robotics Research, Vol. 37, No. 7, pp. 717–742, 2018.

[KaelblingLittmanCassandra98] Kaelbling, L.P.; Littman, M.L.; Cassandra, A.: Planning and acting in partially observable stochastic domains. Artificial Intelligence, Vol. 101, pp. 99–134, 1998.

[Kochenderfer15] Kochenderfer, M.: Decision Making Under Uncertainty: Theory and Application. MIT Lincoln Laboratory Series. MIT Press, 2015.

[KochenderferHollandChryssanthacopoulos12] Kochenderfer, M.J.; Holland, J.E.; Chryssanthacopoulos, J.P.: Next-generation airborne collision avoidance system. Tech. rep., Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[KocsisSzepesvári06] Kocsis, L.; Szepesvári, C.: Bandit based monte-carlo planning. In European conference on machine learning, pp. 282–293, Springer, 2006.

[KretzschmarKudererBurgard14] Kretzschmar, H.; Kuderer, M.; Burgard, W.: Learning to predict trajectories of cooperatively navigating agents. In 2014 IEEE international conference on robotics and automation (ICRA), pp. 4015–4020, IEEE, 2014.

[KurniawatiEtAl08] Kurniawati, H.; Hsu, D.; Lee, W.S.; Bai, H.: Approximate POMDP Planning Toolkit. `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`, 2008.

[KurniawatiYadav16] Kurniawati, H.; Yadav, V.: An online pomdp solver for uncertainty planning in dynamic environment. In Robotics Research, pp. 611–629. Springer, 2016.

[LevinsonEtAl11] Levinson, J.; Askeland, J.; Becker, J.; Dolson, J.; Held, D.; Kammel, S.; Kolter, J.Z.; Langer, D.; Pink, O.; Pratt, V.; et al.: Towards fully autonomous driving: Systems and algorithms. In 2011 IEEE Intelligent Vehicles Symposium (IV), pp. 163–168, IEEE, 2011.

[PapadimitriouTsitsiklis87] Papadimitriou, C.H.; Tsitsiklis, J.N.: The complexity of Markov decision processes. Mathematics of Operations Research, Vol. 12, No. 3, pp. 441–450, 1987.

[PetersenUgrinovskiiSavkin12] Petersen, I.; Ugrinovskii, V.; Savkin, A.: Robust Control Design Using H-∞ Methods. Communications and Control Engineering. Springer London, 2012.

[PineauEtAl03] Pineau, J.; Montemerlo, M.; Pollack, M.; Roy, N.; Thrun, S.: Towards robotic assistants in nursing homes: Challenges and results. Robotics and autonomous systems, Vol. 42, No. 3-4, pp. 271–281, 2003.

[RoyEtAl99] Roy, N.; Burgard, W.; Fox, D.; Thrun, S.: Coastal navigation-mobile robot navigation with uncertainty in dynamic environments. In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C), Vol. 1, pp. 35–40, IEEE, 1999.

[SadighEtAl16] Sadigh, D.; Sastry, S.S.; Seshia, S.A.; Dragan, A.: Information gathering actions over human internal state. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 66–73, IEEE, 2016.

[SchaeferEtAl05] Schaefer, A.J.; Bailey, M.D.; Shechter, S.M.; Roberts, M.S.: Modeling medical treatment using markov decision processes. In Operations research and health care, pp. 593–612. Springer, 2005.

[SilverVeness10] Silver, D.; Veness, J.: Monte-Carlo planning in large POMDPs. In Advances in Neural Information Processing Systems (NIPS), 2010.

[Smith05] Smith, T.: ZMDP software for POMDP planning. `https://github.com/trey0/zmdp`, 2005.

[SomaniEtAl13] Somani, A.; Ye, N.; Hsu, D.; Lee, W.S.: DESPOT: Online POMDP planning with regularization. pp. 1772–1780, 2013.

[Sunberg18a] Sunberg, Z.: Ardespot.jl: Implementation of the ar-despot pomdp algorithm for the pomdps.jl framework. `https://github.com/JuliaPOMDP/ARDESPOT.jl`, 2018.

[Sunberg18b] Sunberg, Z.: Pomcpow: Online solver based on monte carlo tree search for pomdps with continuous state, action, and observation spaces for the pomdps.jl framework. `https://github.com/JuliaPOMDP/POMCPOW.jl`, 2018.

[SunbergHoKochenderfer17] Sunberg, Z.N.; Ho, C.J.; Kochenderfer, M.J.: The value of inferring the internal state of traffic participants for autonomous freeway driving. In 2017 American Control Conference (ACC), pp. 3004–3010, IEEE, 2017.

[SunbergKochenderfer18] Sunberg, Z.N.; Kochenderfer, M.J.: Online algorithms for pomdps with continuous state, action, and observation spaces. In Twenty-Eighth International Conference on Automated Planning and Scheduling, 2018.

[ThrunBurgardFox05] Thrun, S.; Burgard, W.; Fox, D.: Probabilistic Robotics. MIT Press, 2005.

Appendix

A.1 Contents Archive

There is a folder **PRO_XXX_Peters/** in the archive. The main folder contains the entries

- **PRO_XXX_Peters.pdf**: the pdf-file of the thesis PRO-XXX.
- **Data/**: a folder with all the relevant data, programs, scripts and simulation environments.
- **Latex/**: a folder with the *.tex documents of the thesis PRO-XXX written in Latex and all figures (also in *.svg data format if available).
- **Presentation/**: a folder with the relevant data for the presentation including the presentation itself, figures and videos.

Erklärung

Ich, Lasse Peters (Student der Mechatronik an der Technischen Universität Hamburg, Matrikelnummer 21486931), versichere, dass ich die vorliegende Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Unterschrift	Datum