Interactive Planning under Uncertainty

Interaktives Planen unter unsicheren Bedingungen

Master-Thesis von Janine Hölscher aus San Bartolome de Tirajana/Spanien

Juli 2017

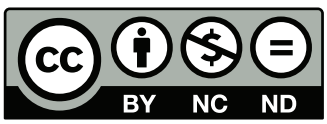
TECHNISCHE UNIVERSITÄT DARMSTADT

Interactive Planning under Uncertainty
Interaktives Planen unter unsicheren Bedingungen

Vorgelegte Master-Thesis von Janine Hölscher aus San Bartolome de Tirajana/Spanien

1. Gutachten: Jan Peters, Ph. D.
2. Gutachten: Dr. Joni Pajarinen
3. Gutachten: Dorothea Koert, M. Sc.

Tag der Einreichung:



Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 10. Juli 2017

(Janine Hölscher)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, July 10, 2017

(Janine Hölscher)

Abstract

In recent years, robots have increasingly gained importance for a wide variety of auxiliary tasks. Yet, adaptation to many household objectives remains challenging as task planning is significantly hindered by the partial observability of the environment. The resulting uncertainty demands a balance between environment exploration and the exploitation of the such gained knowledge. Partially Observable Markov Decision Processes (POMDPs) allow for intuitive modeling of sequential planning tasks in face of uncertainty. However, the computational load required for the solution of a POMDP rises rapidly with both the size of the state space and the number of potential actions. In this study, we propose a generalized POMDP model applicable to robot manipulation tasks under uncertainty. In addition, we introduce means of human robot interaction: The robot is enabled to gather additional information about its environment by direct inquiry of a human. This significantly reduces the number of exploratory actions and thus the planning horizon. Furthermore, we suggest a method for expressing user preferences on a task's outcome. We reformulate the underlying model as a constraint POMDP and propose a modified solution algorithm. Here, human robot interaction is designed with emphasis on operation by non-expert users. The resulting model for interactive planning under uncertainty is evaluated on a box stacking task in simulation and on a real 7 DoF robot arm.

Zusammenfassung

In den vergangenen Jahren ist die Wichtigkeit von Robotern für verschiedenste Hilfsarbeiten deutlich gestiegen. Allerdings bleibt es eine Herausforderung, sie auch im Haushaltsbereich einzusetzen, da bei solchen Aufgaben die Umgebung häufig nicht vollständig beobachtbar ist. Daher muss in solchen Fällen abgewogen werden zwischen einerseits dem Erkunden der Umgebung und andererseits dem Ausnutzen des so gewonnenen Wissens zur Erfüllung der eigentlichen Aufgabe. Partially Observable Markov Decision Processes (POMDPs) eignen sich gut zur Modellierung von sequentieller Planung unter unsicheren Bedingungen. Allerdings steigt die Komplexität dieser Prozesse mit der Größe des Zustandsraumes und der Menge potentiell ausführbarer Aktionen deutlich an. In dieser Arbeit stellen wir ein allgemeines POMDP-Modell vor, das auf verschiedene Manipulationsaufgaben für Roboter anwendbar ist. Außerdem ermöglichen wir in unserem Modell die Interaktion des Roboters mit Menschen in seiner Umgebung: dem Roboter wird die Möglichkeit gegeben, Menschen zu befragen, um auf diese Weise zusätzliche Informationen zu sammeln. Auf diese Weise muss er insgesamt weniger Aktionen durchführen, da weniger Erkundungsaktionen vonnöten sind. Somit kann der Planungshorizont verringert werden. Außerdem schlagen wir eine Methode vor, mit der Nutzer eigene Wünsche über den genauen Ausgang einer Aufgabe äußern können. Dafür wird das zugrundeliegende Modell zu einem beschränkten Prozess umdefiniert und ein Lösungsalgorithmus entsprechend angepasst. Insgesamt ist unser Modell auf Nutzer zugeschnitten, die über keinerlei Vorwissen im Umgang mit Robotern verfügen. Wir evaluieren das Modell, indem wir sowohl Simulationen ausführen als auch Experimente mit einem physischen 7 Roboterarm durchführen.

Figures and Tables

List of Figures

Contents

1 Introduction

1.1 Overview

In today's world, robots have increasingly gained importance for a wide variety of auxiliary tasks. Robotic support is in particular well-established in industrial applications such as manufacturing or logistics. Here, the operating environment can be characterized by some unifying features: Usually, the setting is precisely monitored and fully observable while it only contains few differing but predefined types of objects. In addition, object locations are mostly known and tasks are highly defined. Robotic tasks such as product assembly or packaging have a repetitive character, and hence, solutions are always very similar and can therefore often be decomposed into a small number of predefined subtasks. Therefore, high-level task planning under uncertainty is neither necessary nor desired. As such, industrial environments provide a perfect setting for the employment of robotic technologies.

In contrast, the rather recent introduction of auxiliary robots for typical household tasks has to cope with much more diversified operating conditions: Differing objects might be present with the potential appearance of new (uncharacterized) ones at any given time. Object properties might be either unknown or immeasurable. In addition, environments are in general cluttered as well as partially observable. Another complicating factor is given by the presence of humans in the environment, who tend to change the robot's surroundings. This uncertainty about the robot's current state is combined with rather sparse task definitions, not containing many details. Therefore, the desired strategy for task completion might exhibit great variability depending on the current environment. As these conditions might change substantially, decision making under uncertainty is often required for task planning.

Yet, the demand for household robots has considerably been rising; the first commercial products are already available, such as vacuum cleaners and lawnmowers. Nevertheless, planning under uncertainty remains challenging in face of more complicated tasks, such as doing laundry or tidying up. These tasks have a large variety of actions to choose from and the environment model has to be complex in order to represent all necessary information about the current world state. This is not only true for household tasks, but also for other kinds of robotic tasks such as autonomous driving or rescue missions. For such tasks, robotic procedures outside of a controlled environment are — at present — largely limited to direct operation by a human. In order to address this challenge, the Partially Observable Markov Decision Process (POMDP) framework provides an applicable structure by allowing to model an agent in a sequential decision process in presence of uncertainty. Thus, it can be utilized for the planning of action sequences which lead to the fulfillment of a task in an uncertain environment and which are optimal with respect to a defined criterion. Here, missing information not only about the current world state, but also about uncertain state transitions and observations is taken into account by definition. Thus, this framework is highly appropriate for modeling household problems. However, the intricacy of POMDPs lies in their computational expense and potential lack of applicability to real-world problems with large state and action spaces. Moreover, defining a POMDP for a particular task might necessitate time consuming efforts as it demands significant prior analysis in order to determine an appropriate world model incorporating uncertainty [2].

As already stated above, the presence of humans is a unifying feature of household environments. Here, the effect is of two-fold nature: For once, humans are the cause for the majority of the discussed difficulties, for instance by changing the environment frequently and by not always acting rationally [3]. On the other hand, humans can also be regarded as a valuable source of information as they are likely to perceive additional information about the environment that is inaccessible to the robot's sensors. In particular, human perception tends to provide a more complete overview about the current situation. In addition, human performance in pattern recognition [4] and intuitive decision making by applying common sense (without knowing each detail of a particular situation) [5] can be quite advantageous. As such, those skills can supplement a robot's planning procedure which might be very precise by assessing every detail of a world state, but always remains limited to the underlying world model. Here, humans can supplement details that are not perceived by the robot while providing high-level recommendations concerning the overall task.

1.2 Contributions

In this work, we address a particular type of real-world problems which are defined as robotic tasks in a highly uncertain environment involving interaction with a non-expert human user. Therefore, we propose a general POMDP model for planning the optimal sequence of actions in order to solve this type of task.

On the one hand, human input is employed to reduce the complexity of solving the POMDP by limiting the necessary planning horizon. Here, the number of required exploration actions can be reduced significantly when a human oracle provides additional information and thus reducing uncertainty.

On the other hand, non-expert users are enabled to utilize such a framework as parameter tuning, such as discount factor and reward selection, is avoided. Instead, a user can either choose not to interact at all with a robot or to adjust several more intuitive settings. The latter include, for instance, the number of questions that a robot is allowed to ask or preferences regarding task completion. User interfaces are provided for these settings as well as for answering the robot's questions. To incorporate user preferences into the task solving process, we suggest an alternation of the Policy Graph Improvement algorithm [6]. Due to these changes, the algorithm is able to handle multiple objectives such as fulfilling additional constraints requested by users in the planning process. All of the proposed interaction methods can be easily adapted for most robotic object manipulation tasks.

One common task for robotic operation is object assembly, which involves long-term planning to find a compromise between information gain and task performance. In this thesis, we demonstrate our approach for a robot arm stacking paper boxes in order to create a tower structure. During these experiments, a human user is both answering the robot's questions and expressing his preferences about the resulting tower.

1.3 Structure

This section provides an overview and outline of this study's content.

In **Chapter 2**, we introduce POMDPs as well as different solution techniques for determining an optimal policy. We also review existing methods for integrating human knowledge into a POMDP planning process.

In **Chapter 3**, we introduce our approach for the optimal planning problem of a general manipulation task under uncertainty by employing POMDPs for modeling. In addition, we propose methods for human robot interaction during such a task: On the one hand, robots are enabled to ask humans for information about their environment. On the other hand, humans are given the possibility to declare preferences about a task's solution.

Chapter 4 describes an example application for our POMDP planning technique: a robot arm stacking paper boxes. We demonstrate in detail the application of each feature in our approach, and we furthermore introduce our experimental setup.

Chapter 5 presents results from various experiments involving the box-stacking task. We evaluate our approach in both simulation and experiment (on a physical robot).

In **Chapter 6**, we summarize our results and give an outlook about future work.

2 Foundations and Related Work

2.1 Planning under Uncertainty

A sequential process can be modeled as a discrete sequence of consecutive states. If the Markov property holds, i.e. the next state after a transition only depends on the current state, such a succession of states is called a *Markov chain* (MC).

In a sequential decision process, an agent actively pursues a defined objective and therefore executes certain actions in order to influence its environment. Thus, state transitions depend not only on the current state, but also on the selected action. The agent's performance is evaluated according to the rewards that are particularly collected in the visited states. One way of modeling such a mechanism is via a *Markov decision process* (MDP). Here, an MDP can be used to develop a policy for selecting the optimal action in each world state. Although an MDP assumes that all relevant information about the current state can be observed by the agent, it allows for the incorporation of state transition uncertainties.

The generalization of MDPs, *partially observable Markov decision processes* (POMDPs) additionally models the current state as uncertain as well as only accessible through unreliable observations. The relation between MCs, MDPs and POMDPs is summarized in Figure 2.1.

The development of a decision policy for an agent in an uncertain environment is a difficult task, especially when sequential decisions are required. In the following sections, we introduce both MDPs and POMDPs as well as strategies for finding the corresponding optimal policies for planning tasks in such environments.

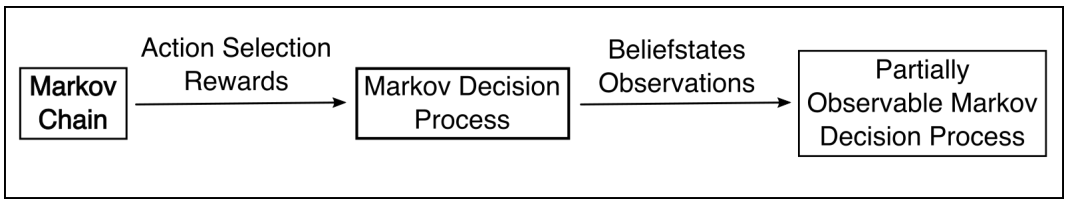


Figure 2.1.: MDPs and POMDPs are generalizations of Markov chains: Both are decision processes in which the planning agent seeks to optimize its rewards. In addition, POMDPs allow for uncertainty about the current state.

2.1.1 Markov Decision Processes

The MDP framework is naturally suited for modeling sequential decision processes in a fully observable system. It was developed in the 1950s mainly by Ronald A. Howard in [7] and Richard Bellman in [8]. The following detailed description is inspired by the comprehensive account provided by Kaelbling in [9] and by Thrun in [10].

2.1.1.1 Definition

An MDP is defined as the tuple $\langle S, A, T, R, s_0 \rangle$ with the following properties:

- S is a finite set of states. A state $s_t \in S$ describes the world at a specific point in time t.
- A is a finite set of actions. At each discrete instant, one action $a_t \in A$ is chosen.
- $T : S \times A \to S$ is a state-transition function. $T(s, a, s') \in [0, 1]$ denotes the probability for a transition to state s' given state s and action a:

$$T\left(s, a, s'\right) = Pr\left(s_{t+1} = s' \;\middle|\; s_t = s, a_t = a\right).$$

- $R : S \times A \to \mathbb{R}$ is a reward function that returns the immediate reward gained at the current time for reaching state s after choosing action a.
- s_0 : denotes the initial state of the process.

The MDP model describes a system's current state s as deterministic for all t. States are interpreted as fully observable, i.e. every information about the environment relevant for an action decision is available to the agent.

Yet, transitions between states are probabilistic and Markov: The next state s_{t+1} depends only on the current state s_t and the current action a_t selected by the agent:

$$Pr\big(s_{t+1} = s'' \;\big|\; s_t = s', a_t = a', \ldots, s_0 = s, a_0 = a\big) = Pr\big(s_{t+1} = s'' \;\big|\; s_t = s', a_t = a'\big) \,.$$

Terminal states are defined as states that do not admit any transition to a different state. Hence,

$$Pr\big(s' \;\big|\; a, s_{\text{terminal}}\big) = \begin{cases} 0 & \text{if } s' \neq s_{\text{terminal}} \\ 1 & \text{if } s' = s_{\text{terminal}} \end{cases}$$

holds for any action $a \in A$. If a terminal state is reached, the process ends prematurely, and no further actions are possible.

A decision process always pursues a goal which determines the optimality criterion for the required decisions. This goal is expressed by means of a reward function R: The current performance is evaluated by returning positive rewards for desirable state/action combinations while negative rewards (costs) arise for situations that should be avoided. The purpose of an MDP model consists of finding an optimal policy (selected actions) maximizing the sum of rewards collected during action execution. Figure 2.2 illustrates the relation between the different MDP components.

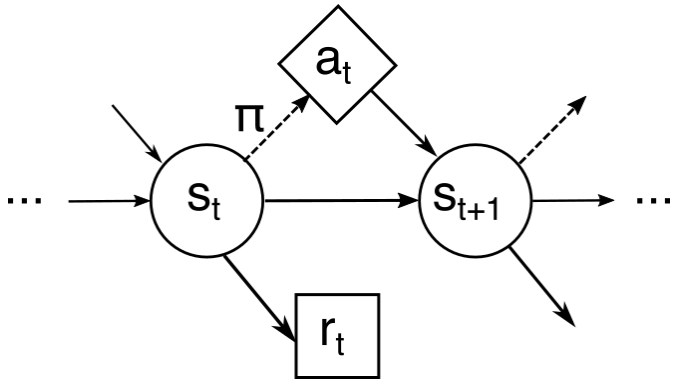


Figure 2.2.: In each state s_t the agent gains a reward r_t and chooses its subsequent action according to policy π, which leads to a transition to the next state s_{t+1}.

2.1.1.2 Solution Techniques

The process of finding the optimal policy denoted as $\pi^*(s) = a^*$ such that the reward is maximized is called *solving an MDP*.

As transitions are probabilistic, all possible outcomes of a particular action have to be taken into account. In addition, not only the immediately following state and its reward are important for an optimal decision, but also all subsequent states. Hence, the cumulative expected reward for future actions has to be considered:

$$\arg\max_{\pi} E\left\{\sum_{t=0}^{t_f} R\big(s_{t+1}, \pi(s_t)\big)\right\}, \text{ with } \pi(s_t) = a_t \,. \tag{2.1}$$

The variable t_f, the policy horizon, denotes the total number of steps taken into account for the policy optimization. Optimal policies for finite time processes are in general *non-stationary*. Here, $\pi^*(s,t)$ does not only depend on the state, but also on the particular time at which an action is chosen or on the number of steps remaining. For instance, the optimal action might differ for an identical state, depending on whether it is the initial state or the second to last one.

However, this is not the case for processes with an infinite number of iteration. As the remaining time horizon does not change, the optimal policy for *infinite horizon* problems results as *stationary*, i.e. it does not depend on the current time. To find a solution for such a process, it is convenient to restrict the cumulative reward, in order to make it comparable. The introduction of a discount factor γ ensures, that the sum in Eq. (2.1) is finite. It amplifies rewards collected in temporal proximity to the current instant and devalues contributions in the distant future:

$$\arg\max_{\pi} \mathscr{E}\left\{\sum_{t=0}^{\infty} \gamma^t \, R\big(s_{t+1}, \pi(s_t)\big)\right\}, \text{ with } \gamma \in [0,1] \, .$$

This above equation is called the *infinite-horizon discounted model* [9]. The smaller the weight γ, the less is the impact of future gains with respect to the cumulated reward and hence, the influence on policy optimization.

To determine an optimal policy, a value function V can be defined. This function recursively calculates the value of a policy π at time t by forming the sum of the immediate reward and the discounted average value of all possible future states. Here, all possible future states are additionally weighted with their corresponding transition probability:

$$V_{\pi,t}(s) = R\big(s, \pi_t(s)\big) \; + \; \gamma \sum_{s' \in S} T\big(s, \pi_t(s), s'\big) \; V_{\pi,t-1}(s') \, .$$

Recursive value functions of this form are referred to as a Bellman equation. The basic case for this recursion is given for $t = 1$ when the value function reduces to the immediate reward.

The value function for the optimal policy π^* has the following form:

$$V_t^*(s) = \arg\max_{a \in A} \left[R(s,a) \; + \; \gamma \sum_{s' \in S} T\big(s, a, s'\big) \; V_{t-1}^*(s') \right] .$$

One method for the determination of π^* is provided by the value iteration algorithm introduced by Bellman in 1957 in [8]. As such, this algorithm is a particular application of Bellman's principle of optimality (dynamic programming). The algorithm recursively calculates $V(s)$ for all states, starting with the last actions' immediate reward. In case of an infinite model, only an approximate value can be determined. The recursion terminates, when $|V_t(s) - V_{t-1}(s')| < \varepsilon$ holds for all $s \in S$. If the difference between two consecutive iterations is less than a given tolerance ε, all successive iterations are considered irrelevant because of the discount factor. It has been shown that the remaining *Bellman error* does not exceed $2\varepsilon \frac{\gamma}{1-\gamma}$ [9].

Other algorithms for solving an MDP include Policy Iteration [7] and Prioritized Sweeping [11]. In addition, MDPs can be extended in various ways, including continuous time approaches as well as infinite action and state spaces. The latter extension leads to POMDPs as introduced in section 2.1.2.

2.1.2 Partially Observable Markov Decision Processes

As MDPs assume a fully known world state, they are not sufficient to model decisions in partially observable environments. POMDPs take the resulting uncertainty into account by treating observations as probabilistic and by considering a random distribution of all possible current states. The necessary model extensions from Markov Decision Chains to POMDPs are depicted in figure 2.1.

2.1.2.1 Definition

The formal definition of POMDPs extends the MDP definition in the following way:

$$\langle S, A, T, R, O, \Omega, b_0 \rangle$$

- S, A, T and R are defined as before for MDPs.
- O is a finite set of observations, i.e. after reaching a new state s', the agent makes observation $o \in O$.

- $\Omega : S \times A \rightarrow O$ is called the observation function providing the probability distribution of an observation, depending on the current action and the next state. $\Omega(s', a, o)$ denotes the probability of making observation o after performing action a and reaching state s', i.e.

$$\Omega\big(s', a, o\big) = Pr\big(O = o \;\big|\; S_{t+1} = s', A_t = a\big)\,.$$

- b_0 is the initial belief at $t = 0$, hence, a distribution over states $s \in S$.

Note that POMDPs can be reformulated such that the definition of Ω is based on the current state, not the following one.

In the POMDP model, observations o are allowed to be partial as well as noisy. As it is illustrated in figure 2.3, observations depend on both the current state and the performed action. Different states could result in the same observation, and in addition, the observation function is probabilistic. Therefore, it is in most cases impossible for the agent to determine the current state. Instead, the agent keeps an internal belief state b based on its previous experience. The variable b arises as a probability distribution over all possible world states at the present point in time. Using the Markov property, it can be shown that a belief state is a sufficient statistic for a process' history [9]. Storing more information during policy determination (for instance, already performed actions) does not provide additional information at any time. Thus, the expected reward cannot be increased by collecting more data [12].

Similar to the transition between states, the transition between beliefs, called the *belief update*, can be formulated for given action a and observation o:

$$\begin{aligned} b'(s')_{a,o} &= Pr\big(s' \;\big|\; o, a, b\big) \\ &= \frac{Pr(o \mid s', a, b) \cdot Pr(s' \mid a, b)}{Pr(o \mid a, b)} \\ &= \frac{Pr(o \mid s', a) \sum_{s \in S} Pr(s' \mid a, b, s) Pr(s \mid a, b)}{Pr(o \mid a, b)} \\ &= \frac{\Omega(s', a, o) \sum_{s \in S} T(s, a, s') b(s)}{Pr(o \mid a, b)}. \end{aligned}$$

Thus, a belief update can be directly calculated by applying the model-specific observation function Ω and the transition function T.

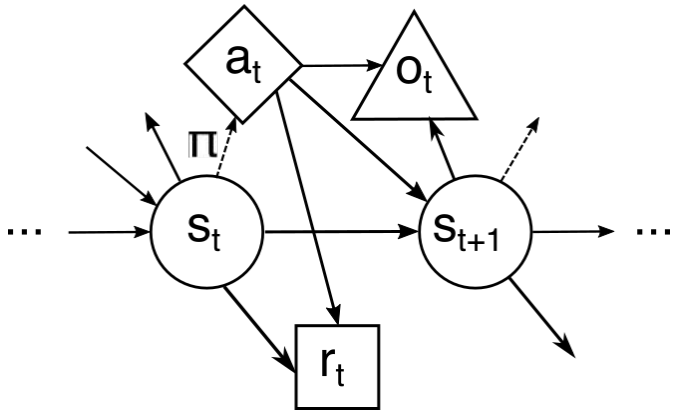


Figure 2.3.: In each state s_t, the agent gains a reward r_t. The current state is unknown, but can be deduced from observations o_t. Based on this information, action a_t is selected according to policy π.

2.1.2.2 Solution Techniques

Exact Solutions

A POMDP can be solved by reformulation as an MDP with a continuous state space, a so-called *belief state MDP*, $\langle B, A, \tau, \rho, b_0 \rangle$ admitting the following properties:

- B is a continuous belief space where each $b \in B$ is a distribution over states $s \in S$.
- $\tau : B \times A \rightarrow B$ is the belief transition function, yielding

$$\begin{aligned} \tau(b, a, b') &= Pr(b' \mid a, b) \\ &= \sum_{o \in \Omega} Pr(b' \mid a, b, o) Pr(o \mid a, b), \end{aligned}$$

$$\text{where } Pr(b' \mid a, b, o) = \begin{cases} 1 & \text{if } b_{a,o} = b' \\ 0 & \text{otherwise} \end{cases} .$$

- $\rho : B \times A \rightarrow \mathbb{R}$ is the reward function, i.e.

$$\rho(b, a) = \sum_{s \in S} b(s) R(s, a) .$$

All other properties remain identical to the MDP definition in section 2.1.1. Here, observations and their probabilities are incorporated into the transition function τ and do not have to be considered explicitly. The value function can now be adapted to a belief:

$$V(b) = \arg\max_{a \in A} \left[\rho(s, a) \; + \; \gamma \sum_{b' \in B} \tau(b, a, b') V_{\pi}(b') \right] .$$

The continuous state space remains challenging as the value function contains a sum of (belief) states. Yet, it can be shown [13] that the above value function is convex and piecewise linear in the belief. Therefore, it can be written as

$$V(b) = \arg\max_{\alpha \in V_{\pi}} \left[\sum_{s \in S} \alpha(s) b(s) \right]$$

where $V_t = \{\alpha_0, \ldots, \alpha_m\}$ is a set of $|S|$-dimensional α-vectors providing a partitioning of the belief space at time t. Here, one α-vector represents the value V within each partitioned section and is called the *dominant* vector. A particular action a can be associated with each α_i. This action a is optimal for beliefs within corresponding section. Figure 2.4 illustrates α vectors partitioning a belief space.

The update relation for α at time $t + 1$ is defined as

$$\alpha_{i,t}(s) = R(s, a) \; + \; \gamma \sum_{o \in \Omega} \sum_{s' \in S} P(o \mid s', a) P(s' \mid s, a) \alpha_{j,t+1}(s')$$

After an iteration of the value, some α might be completely dominated by other vectors and thus can be pruned. It is computationally expensive to determine these vectors, but usually worth the effort given the solution's exponential growth in each iteration.

Algorithms that provide exact solutions include Monahan's Enumeration algorithm in [14], Sondik's One-Pass algorithm in [15] and Incremental Pruning in [16]. These approaches mostly vary in their method for the selection and update of the α-vectors.

Approximate Solutions

As exact solutions to POMDP problems are, in general, very expensive to compute, applicability is questionable for most applications. Thus, a variety of approximation methods has been proposed. Here, a prominent category among the suggested algorithms consists of *point-based* methods. Algorithms within this group, such as Point-Based Value Iteration (PBVI) [12], exploit the fact that many points of the belief space will never be reached, most likely. Hence, it suffices to solve the POMDP for a finite amount of likely to occur belief points. The belief state is reduced to a finite subset, and as a consequence, the number of α vectors representing the value function is limited. Various point-based methods exists, differing predominantly in the selection of belief points and of the value function update. Some exemplary and popular algorithms are Forward Search Value Iteration [17], Heuristic Search Value Iteration [18] and the Perseus algorithm [19].

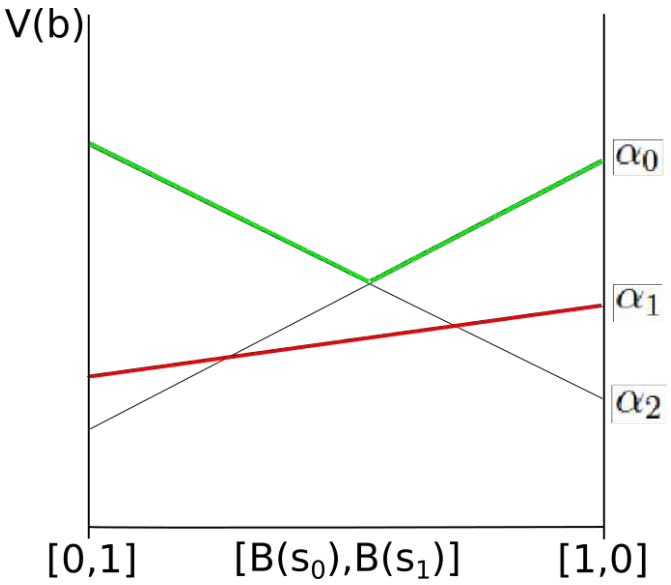


Figure 2.4.: The α-vectors partition a belief state consisting of two states s_0 and s_1. The belief can be represented in one dimension as the probabilities for both states add up to 1. Both α_0 and α_2 are piecewise dominant over all other α vectors and thus represent V (green line) in an interval. α_1 (red line) is completely dominated by other vectors and can therefore be pruned.

The Policy Graph Improvement Algorithm

The Policy Graph Improvement Algorithm (PGI) algorithm [20] is closely related to the PBVI approach, but operates directly on a policy graph (such as in Figure 2.5) rather than on value functions. In a policy graph, each node represents an action, and the edges connecting the nodes correspond to observations. An agent begins at the start node, executes the first action, and subsequently follows an edge to the next node, according to its observation. The current node provides sufficient information about the agent's situation; it is thus redundant to determine the agent's exact state in order to follow the policy. For a reactive agent, this policy depiction is ideal as it does not require any further processing [21]. Its intuitive description together with the resulting compactness also provides perceptibility for human beings. Thus, it is relatively effortless to review such a policy.

In case of the PGI Algorithm, the policy graph emerges with a predefined, fixed planning horizon. Each node represents an action at a particular time, and the number of nodes for each time is likewise predetermined. Therefore, the problem of POMDPs becoming intractable as policies grow exponentially is avoided while long planning horizons are yet possible.

The policy is established by applying a particle based approximation with an invariant number of particles [22]. Each particle constitutes a particular state while the complete entity of particles can be utilized to approximate the belief state at a certain time. Hence, the algorithm is capable to numerically cope with high dimensional state spaces.

In detail, the algorithm can be described as follows: A single iteration of the algorithm consists of two main stages as depicted in Figure 2.6. During the forward pass, the initial belief is projected ahead by applying the current policy. Here, all particles start in the initial belief state before following the current policy through the policy graph. The transition to the following state and the subsequent observation (according to the current policy) are sampled for each particle and time until all particles have reached the planning horizon. The backward pass updates the policy and value function estimates at each node. Therefore, it starts at the last policy graph layer and, for each node, determines a new policy maximizing the expected rewards, by reassigning best actions and best forward edges. The particles at each node are weighted according to their likelihood. Their weighted average value reflects the current value of the node.

In case of redundant nodes representing identical policies, compression is applied, i.e. one of the nodes is replaced by a newly sampled belief. In [20] it has been proven that the PGI algorithm guarantees the overall policy value in a policy graph to be monotonically decreasing. The algorithm can be utilized for both offline and online planning.

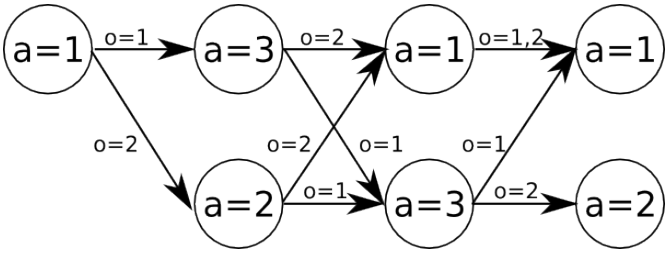


Figure 2.5.: A policy graph depicting a POMDP policy with a planning horizon of 4 and a node width of 2 is illustrated. At each time, an action a is selected depending on the previously made observation o.

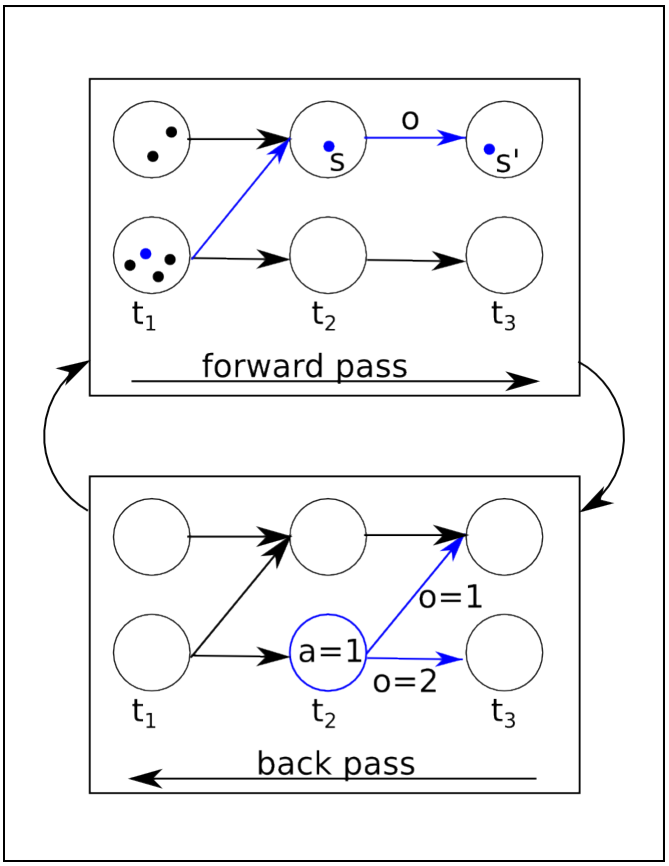


Figure 2.6.: The Policy Graph Improvement Algorithm consists of two phases: During the forward pass, the most likely observation o as well as the most likely subsequent state s' is sampled for each particle at every point in time. When executing the backward pass, the best action with the best upcoming nodes are selected for each policy graph node, depending on the observation.

2.1.2.3 Constrained POMDPs

POMDPs are well-suited to obtain a trade-off between exploration and exploitation in uncertain environments. However, they do not incorporate hard constraints directly into the state space although they might be desirable in certain risky situations. Instead, constraints can only be applied by assigning large costs to violations. It can be easily shown that this approach does not always lead to a good policies, even in a rather simple setting [23].

A Constrained POMDP (CPOMDP) supplies an extension to the standard POMDP model allowing for hard constraints. Thus, decision making in face of multiple objectives is enabled while still optimizing the reward [24]:

$$\arg\max_{\pi} \mathscr{E}\left[\sum_{t=0}^{T} R(s_t, a_t)\right]$$
$$s.t.\ \mathscr{E}\left[\sum_{t=0}^{T} C(s_t, a_t)\right] \leq C_{max}$$

where $C(s,a)$ denotes a cost function for violated constraints, and C_{max} is the upper bound for accumulated constraint penalties.

Various methods for solving the resulting CPOMDPs have been proposed: Isom et al. proposed an offline dynamic programming method for finding deterministic optimal policies in [25] which is based on value iteration. In return, Kim et al. show in [26] that deterministic policies may remain suboptimal for some cases, and hence, an approximate, randomized algorithm is introduced. The approach proposed by Undurti et al. in [23] encompasses two parts: First, the expected penalty resulting from violated constraints is estimated offline. Then, the previous estimate is employed as an upper bound in a standard online algorithm maximizing the rewards. Beside constraining POMDP states, actions also can be subject to limitations. Chanel et al. [27] introduced the idea of an Action-Constrained POMDP (ACPOMDP). In this model, actions are limited to certain states and can only be performed when the agent's current state belongs to the action's state subset. Application examples for CPOMDPs can be found in spoken dialogue systems [28] or opportunistic spectrum access in wireless communications[29]. CPOMDPs provide a method for including additional planning limitations into a POMDP. We employ this approach for embedding user-specific preferences into a planning process. To solve the resulting CPOMDPs, we adapt the PGI algorithm described in section 2.1.2.2. Our modifications are introduced in section 3.3.

2.1.2.4 Challenges

Complexity

Despite recent advances in algorithms solving POMDPs, the determination of an optimal policy for a POMDP problem remains a hard to conquer problem. It has been shown that solving a POMDP is PSPACE-complete for finite-horizon problems [30]. Infinite-horizon POMDPs are even undecidable [31].

There are two main factors contributing to the complexity of a POMDP solution. One is the size of the state space, i.e. for n states, the solving algorithm operates on an $(n-1)$ dimensional continuous belief space. The other factor emanates from the size of the set containing the action-observation histories that have to be considered for the simulation of the POMDP. The cardinality of this history set grows exponentially with the planning horizon [12]. The scalability of POMDPs is therefore limited. Kurniawati et al. [32] demonstrated that an automated downsizing of the state space reduces the complexity of the model — and hence the necessary solution time — significantly. However, this approach is difficult to generalize to different types of problems as a specific state structure has been exploited.

Model definition

Another issue complicating the application of POMDPs lies in the model definition. Both state transitions and observation probabilities have to be determined for each state/action combination and resulting policies depend heavily on these values. It is a common approach to determine POMDP specifications through extensive prior experiments although this might be rather time-consuming [33], [34], [35]. An alternative technique commences with an underspecified model and improves during task execution. In [36], Rosenthal et. al propose an algorithm for the learning task of observation and transition uncertainties through observations while Jaulmes et al. supply an active learning approach [37] that requires an oracle. Both methods require feedback that might potentially be expensive and not available in every setting.

In addition, appropriate immediate rewards have to be verified in order to obtain a suitable solution. The tuning of reward values is crucial for an efficient policy that balances exploration and exploitation. Cai et al. suggest a dual-policy method that combines traditional exploration and active learning for obtaining optimal actions [38]. In another active learning approach, Doshi et al. define a "model-uncertainty" POMDP that expresses unknown parameters as an additional hidden state [39].

2.2 Human-Robot Interaction in POMDPs

POMDPs have been employed within a wide range of different fields, including conservation of threatened species [40], speech recognition [41], online commerce [42], or dynamic pricing [43]. In addition, POMDPs are appealing for task planning in autonomous robotics: States represent a robot's location or its current status during task execution in an intuitive manner. Partial observations describe well noisy sensor data. The robot's actuators not always performing in the intended way is reflected by a probabilistic state transition model. Finally, rewards emanate as a natural approach in expressing a task's goals and pitfalls [44]. A particular area exhibiting a significant potential is found in household environments where states appear rather complex by monitoring the positions of many relevant objects. Recently, compelling results in this area have been emerging including the execution of assignments such as fitting a dishwasher [6], separating clothes [45], or fetching requested objects [46]. In general, the available precision in object manipulation tasks can be improved using a POMDP to plan grasping actions as shown in [47].

Many applications for robotic planning address problems concerned with human environments as they are usually associated with a large amount of uncertainty. Already in 1998, Burgard et al. [48] developed a mobile museum guide robot that is able to provide tours and to interact with visitors. The task planner utilizes a hierarchical approach employing first order logic to develop conditional high-level plans. The planned actions are then translated into predefined sequences of elementary commands. Shiomi et al. [49] designed a robot offering assistance to passengers in a train station. The robot's behavior control consists of specific planning modules for different situations that might be encountered. Each of these modules consists of predefined transition rules which are followed by the robot. Both approaches are built upon highly complex sets of situation-specific action sequences, which makes them work well in their specific environment. Yet, this type of planning system is difficult to adapt to other situations or tasks.

Here, POMDPs might exhibit higher flexibility because of their situation dependent automatic adaptability. It has already been established that POMDP agents are able to fulfill complex tasks such as assisting elderly people [50] or escorting visitors through a building [51]. Nevertheless, the application of POMDPs is restricted by their complexity as explained in section 2.1.2.4. As state spaces of real-world tasks tend to be large, the number of actual application still remains small.

In many of these tasks, humans can be found in proximity of a robot, as for example in a household environment. These humans are familiar with the robot's task and its solution. It has been shown that humans are excellent at intuitively evaluating well-known situations from their experience [5]. Therefore, they should be considered as a valuable information source.

In the following section, we describe different existing methods to decrease the uncertainty — and thus complexity — by including human knowledge into the model. Thereby, we focus on non-myopic information gathering, i.e. a human does not physically participate in a task solving process, but only shares knowledge in different ways. The presented approaches and their relationship to the newly introduced technique in this study are further detailed, analyzed, and criticized in chapter 3.

2.2.1 Information Gathering Actions with Human Input

The easiest method to incorporate new information into a POMDP is via an observation as observations are already external input parameter. By nature, the most immediate way to obtain desired information from a human is by directly asking. In addition, it is also possible to observe a human's behavior and deduce data in this fashion. The latter is a common approach typical for collaboration tasks. Here, we do not intend to study collaboration with a human; instead, we desire that a robot fulfills a task on its own, but with occasional support. Nevertheless, both approaches exhibit very similar methods for including knowledge, such that we do not distinguish between them. We rather focus — in this section — on the different types of information that can be included into POMDPs through observations and their interpretation within the POMDP.

(a) A robot arm is separating a pile of clothes [45].

(b) Pearl is shown, a robot assistant for the elderly [50].

Figure 2.7.: Robots can provide help for many household tasks. In these environments, there is usually a lot of uncertainty due to clutter. Hence, planning a robot's actions is rendered rather challenging. POMDPs provide an intuitive way of solving such planning tasks. The two examples presented in (a) and (b) demonstrate the particular kind of tasks that can already be solved by applying POMDP techniques.

In the following analysis of existing approaches, we focus on two main questions: how can we modify POMDPs such that the planning agent can actively choose to request information? How high is the potential with information gain and how easy the knowledge integration? And how can we design a planning process such that non-expert users can affect it according to their preferences and much influence do these methods give them about the outcome?

In this section, we refer to a human with an interest in a robotic task as the 'user' and to the robot and its planning process as the 'agent'.

2.2.1.1 Collecting State Information

In this section, a method for the planning agent to actively collect state information through questioning a human oracle is described. We first present the basis approach and then describe a further development.

Oracular POMDP

An *Oracular POMDP* (OPOMDP) as introduced by Armstrong et al. in [52] supplies the possibility to ask a human oracle for observations. Therefore, an additional action a_{oracle} is included into the model. When this particular action is executed, the world state is returned as an observation. At each given time, the agent can choose to either perform a_{oracle} or one of the previously introduced actions. Note that Armstrong's model — in its purest form — strongly distinguishes between information gathering and world state changing actions, i.e. the oracle action is the only way to collect information about the current state, other actions do not result in any observations. At any given point in time, the agent has to decide whether to pursue his current plan or to ask for new information, thus weighing exploration versus exploitation. To incorporate the observed information, the transition function has to be adapted such that the current state is adjusted to the observations, after the oracle was questioned. Oracle observations can be requested at any time and are always correct and complete, i.e. after inquiry, the agent has access to all information about the current world state, and there is no uncertainty about it. Hence, the observed world state is immediately adopted as the current state.

As the other actions are not deterministic, a single question is not sufficient to determine the unknown initial state. In a good policy, the agent should question the oracle before selecting an action when a risky situation arises. But the costs (negative rewards) assigned to executing a_{oracle} consist of such a large amount that the action is only chosen if much information gain is expected. The selection of the assigned costs is crucial for an optimal policy [36].

Although the OPOMDP model's restrictions simplify planning, they challenge its application to real-world problems. Most humans would probably not be willing to describe the complete world state to a robot over and over again; or they might not even have access to all parameters of the current state. In addition, humans might not be available for questions at

every time or might not always provide correct answers. In contrast, some information could be more precise if the robot measures it directly with its own sensors (for instance, the robot's position).

Human Observation Provider POMDP

The *Human Observation Provider POMDP* (HOP-POMDP) by Rosenthal et al. in [53] attempts to resolve some of these issues. Similarly to the OPOMDP, it supplies additional actions a_{ask} to request help in the form of observations. Yet, answers include only partial information, not complete state descriptions, in this model. In addition, the agent has to select one of multiple oracle actions to specify in which particular information it is interested in. Furthermore, not only does a_{ask} lead to observations but also every original action. Executing a_{ask} offers additional help for situations not observable by the robot.

Likewise, humans are not supposed to be continuously available. The response rate α of each human for each state is assumed known through prior experiments (see section 2.2.2). Nevertheless, it is required that human's answer is correct if a question is answered. It is therefore not necessary to take noise into account and hence to model observations probabilistically. The current state can immediately be adapted to a received observation. To prevent a robot from unnecessarily waiting for a human to become available, there an additional observation o_{null} is added that is automatically generated if no human has responded within 30 seconds after a query. The policy algorithm then suggests a different action. Each human is only available in one state. HOP-POMPDPs are especially suited for spatially-situated tasks, i.e. for tasks in which a robot has to move through an uncertain environment. Here, humans are expected to only be located in one particular position (state), e.g. in an office. Hence, the robot specifically plans on whom to ask based on the person's availability and proximity [54]. The resulting policy exhibits a trade-off between moving to states leading directly to the goal and states having a greater availability of help. The costs for a query are determined a priori. They vary for each person as they are supposed to reflect the cost of each individual user's time spent with answering a question. If nobody is available, and o_{null} is received, there are no costs.

In comparison to previous methods, this model is far more suited for real-world tasks as it considers availability and the necessity to actively approach humans whenever the agent wants to inquire. However, the underlying human user model requires a large amount of prior information; the given results exhibit that balancing costs and response rates is crucial for reliable policies. Thus, the creation of a sufficiently reliable model might be problematic for more complex tasks. Furthermore, this model does not account for possibly incorrect human suggestions, i.e. it is potentially dangerous to immediately adapt the current world state to a user's suggestion.

2.2.1.2 Asking for the Next Optimal Action

In this section, we introduce various methods for gaining information about optimal actions. In all approaches, a human oracle is questioned about the next best action in a certain state, but the process of utilizing this knowledge to further improve POMDP models varies.

Model-Uncertainty POMDP

Doshi et al. propose a *Model-Uncertainty POMDP* in [55], in which transitions, observation probabilities Ω and rewards R are uncertain. Therefore, these variables are added to the state-space: $S' = T \times \Omega \times R$. A belief distribution represented by samples is maintained to track the current belief in model parameters. The POMDP model is improved over time through both exploration actions and query. In the latter, an expert is questioned about the best action at a particular point in time. Therefore, a set of policy queries is included into the set of actions. These are selected by the agent whenever the situation is deemed too risky to safely select an action. A Bayes risk criterion is employed to determine this case. If a policy query action is selected, the agent asks a user a series of yes/no questions or presents a list of actions from which the user is expected to select the best one. The action is performed, and the belief distribution is updated accordingly. Therefore, it is necessary to solve each of the candidate models in order to evaluate the model policies being consistent with the query answer. Between trials, models are re-sampled periodically.

Thereby, this method allows for model improvement as well as online adaptation as user priorities might change over time. It also accounts for users making mistakes and suggesting non-optimal actions. Nevertheless, the procedure of maintaining multiple sample models is very expensive computationally.

Learning the Observation Function

A method for learning the observation dynamics Ω of a POMDP is described by Atrash and Pineau in [56]. Here, transitions and rewards are assumed to be known. Similarly to the Model-Uncertainty approach as previously described, a set of candidate models is sampled and updated whenever new oracle information about an optimal action becomes available. Instead of adding model parameters to the state space, a Dirichlet distribution over possible observations is associated with each state. For a newly created POMDP model, the multivariate distribution representing observation probabilities is sampled for each state. If an oracle inquiry is executed, the returned optimal action (for a certain situation) is compared to the actions each POMDP model suggested. For those agreeing with the oracle, the Dirichlet parameters are updated via a gradient ascent method.

The presented technique has been tested for the navigation task of an autonomous wheelchair while control is applied via high-level commands. In this scenario, it is crucial to adapt the underlying planning system to a specific user as needs might differ. This approach presents an efficient way for such a user dependent improvement of an existing prior model. However, sustaining and solving multiple POMDP models might not be feasible in more complex settings.

Human Help Provider MDP

Côté et al. introduce a *Human Help Provider MDP* (HHP-MDP) for human-robot collaboration in [57]. Their goal is to establish an adjustable autonomy for the agent, such that the human can intervene and control whenever necessary. The user is able to stop the autonomous robot at any point in time and is allowed to suggest one or multiple actions. Then, the agent will perform these actions before returning to autonomous action. At the same time, the agent learns from these interventions: The policy for the states in question is immediately adapted to the suggested actions. Transitions and rewards are modified accordingly. This update process is incorporated into an effective algorithm by avoiding the calculation of rewards and transitions for states in which the policy is explicitly supplied by the human. Moreover, the space of reachable states can be reduced.

This approach allows to for directly influencing the current POMDP policy and integrates the new knowledge in an efficient way. Yet, the user is assumed to be permanently attentive to identify possible errors and to be continuously willing to intervene. Unless a very specific goal requiring precise steps in a specific order is given, it is challenging for the user to determine on-the-fly at what point the agents behavior deviates from the optimal sequence of actions. For many tasks, multiple sequences of actions equivalently lead to the goal. Therefore, it is demanding to evaluate one state without knowing the outcome of future actions in more complex settings. Even an attentive user will most likely not be able to identify suboptimal actions in time without subsequent observations of the robot's actions. As the user can only make action suggestions for the current state, he will thus be advising on a state that should not have been reached in the first place. In addition, the user has no information available about the length of the suggested sequence of actions. In summary, it can be stated that this approach is a suitable method to integrate teleoperation while at the same time updating the agent's plans. Yet, it might not be the best option for model improvement with the goal of an autonomously acting agent.

All three examples exhibit the lack of an immediate technique to include information about optimal actions into a POMDP. One solution consists of evaluating several possible models according to the action suggested by an oracle. Another discussed method directly includes an action for a specific state into the POMDP policy. It has been demonstrated that it is necessary to adapt both transitions and rewards while at the same time preventing the adapted parts of the policy from overwriting in the next policy update. Both approaches are computationally expensive.

2.2.1.3 Gathering Goal Specifications

In this section, we describe different approaches for utilizing of goal information. All presented methods were designed for human-robot collaboration and the agent is meant to watch the user to determine his intentions. But this setting can be simplified such that only the agent can change the world state and the user serves as an oracle by directly answering questions instead of performing actions accordingly. This facilitated version of the problem corresponds to our knowledge integration approach.

Goal State as Part of the State Space

In this approach by Taha et al. [35], it is demonstrated how goal information can be gained from observations. Here, human input is necessary at each time in the form of observations. Those observations are interpreted as directions towards the destination state. Thereby, the state space is defined as the cross product of the current state C and the desired destination state D (terminal state):

$$S = C \times D = s_1 d_1, s_2 d_1, \ldots, s_n d_m \qquad \text{with } D \subseteq C$$

The current state is fully observable whereas the possible destination states are characterized by a probability distribution, i.e. the destination is not fully known. The transition changes in C solely depend on the current action, updates on the expected destination D are exclusively influenced by collected observations. A positive reward is gained if the current state matches the destination state. Hence, a user is able to implicitly select the task's goal.

In the provided example by [35], a wheelchair is navigated throughout a building while the user's input (up, down, left, right) is employed to compute the most likely destination (e.g. the kitchen). Although intended for spatial movement tasks, this model can be adapted for other purposes. The main complication arises from the compact modeling of both current and desired states. If a state consists of more than location coordinates, it might not be feasible to handle the resulting state space due to its size. Moreover, it might be challenging to determine observations that allow for direct translation into a desired state for most task models. Yet another difficulty lies within the creation of an accurate observation model that assigns which observation in each state is an indicator for a particular destination. In the presented case, the observation model has been constructed based on long-term statistics. Note that observation input is necessary at each time. Hence, convenience for a user is questionable depending on the task. In summary, the user is supplied with a large amount of control over the task execution and the task's result, but a very detailed model is crucial to make this possible.

Integrating Human Intentions into States

Karami et al. present a very similar POMDP model in [58]. Again, the state space is extended by an additional variable representing human intentions; observations provide the necessary information to update those intentions. But the interpretation differs slightly: Instead of choosing a single long-term goal, subtasks are selected through observations gathered at every time instant. The reward function is defined such that actions corresponding to these subtasks receive higher rewards.

In the specific setting provided in [58], a robot and a human are both collaborating on a task consisting of several steps that can be performed independently. By observing the human, the robot discovers his current intention and selects a different subtask to avoid redundancy. This setting can be reformulated such that the user directly selects the robot's upcoming subtask. Once more, a prerequisite for this technique demands a precise model of a task and its decomposition into subtasks. Moreover, a transition function that translates observations into intentions has to be defined. This might be difficult for more complex tasks and might require a large amount of a priori analysis. In addition, the definition of a reward function reflecting the usability of each action at all times with respect to every subtask is challenging.

Assistance POMDP

Fern et al. introduce a general *Assistance POMDP* model in [59]. Once more, a new state variable reflecting the task's goal is introduced. But in this approach, it is assumed that observing the user does not necessarily provide enough information for the agent to act optimally. Instead, a "noop" action is introduced. The agent select this action when its current state is too uncertain. In response, a user performs one action for the agent to point it into the right direction. The noop action can be modeled equivalently to an oracle question. The only difference is that it could be extended to collaboration tasks in which the user is able to perform actions the robot can't do. In this case, collaboration would not only be necessary for knowledge gain but also because the agent cannot solve the task on its own. Also, it is possible to introduce different costs for an action depending on who performs it. This can be applied to further encourage collaboration. Other than the previous two papers, this one proposes a method for learning the goal distribution and the users policy based on observations. As learning from observations requires a lot of data and is therefore not applicable for many tasks, the authors propose to speed this process up by bootstrapping the available observation data.

Even though this might make the learning process more tractable, in many tasks it is still cumbersome for a user to demonstrate them to a robot several times before it is of any help. Furthermore, the agent is assumed to be rational and act optimally according to its own policy. But in many cases, human behavior cannot be simplified this way.

An essential part for including human knowledge into POMDP models lies within the underlying model of a human. In the discussed work, human influence ranges from supplying simple observation to immediate goal selection. Nevertheless, all previous models share a common feature although sometimes implicitly: They incorporate assumptions about the uncertainty that is introduced by human information providers. Of course, it could be assumed that users always act in a nearly optimal fashion [59]. But the user information significantly influencing the planning outcome likely leads to non-optimal results.

A simplified manner of modeling uncertainty emanates from including a constant accuracy probability for human input while the observation function is modeled accordingly. This approach can be extended by allowing for a varying probability value for each human (if more than one is involved) potentially depending on a particular state, action, and type of question asked by the agent. In [35], for instance, the probability that a user moves a joystick into the desired direction and thus provides the intended observation is assigned differently for individual joystick directions. By nature, the more specific such a model is defined the better new information can be evaluated appropriately. Yet, it becomes increasingly expensive to create such elaborated models as mentioned in section 2.1.2.4. Parameter values describing human participants are usually determined a priori; they are not adapted throughout the process.

In [53], another concept is furnished by analyzing the underlying reasons for human unreliability: Human availability is considered as a combination of presence and willingness to help. Then, these two resulting values are combined in a single reliability variable α_s for each human. The observation function is modeled accordingly, yielding

$$O(s, a_{\text{ask}}, o) = \begin{cases} \alpha_s & \text{if } o = o_s \\ 1 - \alpha_s & \text{if } o = o_{\text{null}} \\ 0 & \text{if } o = o'_s, s \neq s' \end{cases} .$$

For completeness, an additional default observation o_{null} emerges if no other information is available, i.e. if no human responds. Note that availability is the subject of probability analysis in this approach. However, human accuracy is assumed perfect if an answer is provided.

In [54], this assumption is not maintained anymore by the authors. Instead, the model is extended by an additional accuracy variable η_s that is assigned to each human, i.e.

$$\eta_s = \frac{p(o_s \mid s, a_{\text{ask}})}{\alpha_s} = \frac{p(o_s \mid s, a = \text{ask})}{\sum_{o \neq o_{\text{null}}} p(o \mid s, a_{\text{ask}})} .$$

Here, the accuracy is defined as the probability relationship of a correct observation in comparison with a wrong one under the condition that the human is available. The such introduced accuracy variable η supplies another probability value assigned to each involved human that has to be determined a priori. The same a priori requirements hold for asking costs (interruptibility) and availabilities. Therefore, complex experiments are cardinal in order to estimate these values and decisive for the performance of the agent [36].

The last discussed modeling concept for a single as well as multiple humans participating in a process relies on defining individual POMDPs for each human member. These POMDPs then reflects their respective current participation state [58]. The combination of these individual POMDPs results in a so-called multi-agent POMDP with a shared state space:

$$S = S_{\text{Robot}} \times S_{\text{Human},0} \times \cdots \times S_{\text{Human},n} .$$

Here, humans may implicitly provide observations by changing their state at any given point in time, for instance by moving or pressing buttons. The emanating state transitions are deterministic and are communicated to the planning agent via observations. However, it might be rather problematic to model appropriate state transitions for more complex tasks and to solve the resulting enlarged state space POMDP.

Modeling humans and the uncertainty they may cause in terms of e.g. reliability or responsiveness is an important issue in human-robot interaction and collaboration. However, we focus on including information into planning processes in this work and we refer to our future work for handling uncertainty regarding this information.

3 Concepts for Interactive Planning under Uncertainty

In this chapter, we introduce our approach for planning robot manipulation tasks under uncertainty. Therefore, we propose a general template for a POMDP model that can be applied to a wide range of robot assignments involving object manipulation. We solve this POMDP by applying the PGI algorithm (see section 2.1.2.2). This algorithm has been proven to handle large state spaces, which are likely to occur in tasks involving multiple objects, very well.

We also propose methods for involving humans surrounding the robot in the task: First, we show how to include human knowledge by enabling the planning agent to actively request information the human can easily give. Second, we offer an intuitive approach for non-expert users to express their preferences about the outcome of a task and thus influence the resulting policy. Therefore, we suggest adaptations to the PGI algorithm for solving constrained POMDPs.

3.1 Defining a POMDP

In this section, we describe how POMDP properties of a generic manipulation task can be modeled. Thereby, we take into account the specific properties of this kind of task: a robot manipulates a certain number of objects in order to reach a predefined goal. The robot is presumably not able to directly perceive all information necessary to plan such a task. The planning process has to incorporate the resulting uncertainty about the current state. In the following subsections, we propose a way of modeling states and observations as well as terminal states.

3.1.1 States

We assume that in the tasks of concern, there exist multiple involved objects. These objects can be characterized by the same set of task relevant properties. For example, books in a sorting task are characterized by a size (Do they fit into a book shelf?) and a current position (Already on a shelf?) property. The set of possible objects may be defined as:

$$\text{Obj} := P_1 \times \cdots \times P_J$$

where P_i is the set of possible values for a particular property and J the total number of object properties. As object manipulation is concerned with the state of each object, all information necessary in assessing the progress of a task can be accessed through the object properties. Therefore, the current state s is modeled as:

$$s \in S, \quad S := \text{Obj}^N$$

where S is the state space with a total number of objects N. We assume that the number of objects remains unchanged during one manipulation trial. However, objects may be altered in between trials.

3.1.2 Observations

We distinguish between two main types of properties: Those that are changing during a task (dynamic) and those who contain static information:

Dynamic properties $P^{(d)}$ are affected by action execution through object manipulation, for example object positions. Therefore, they can be interpreted as monitoring the task's progress. We assume that they are directly observable, i.e. measurable, and that, in addition, the associated observations are deterministic. Thus, there is no uncertainty in the task's progress; for instance, it is known how many books are already sorted on shelves as their positions are known.

Static properties $P^{(s)}$, on the other hand, are not immediately linked to the task's progression. Yet, the information contained is considered relevant for planning. Unlike dynamic properties, these parameters are not necessarily observable in a direct way. As depicted in Figure 3.1, they may additionally differ in their observability:

It is assumed that the planning agent is able to directly observe some of the static parameters after each action. For example, a book's color can be perceived with an RGB camera. However, there are other object properties the agent cannot immediately perceive, so-called *hidden* parameters. Nevertheless, the agent can learn about them via knowledge gaining methods. For example, if a book's size is not available immediately, it could be deduced by either comparison to other books with known sizes or by gained information through fitting attempts on a shelf. The properties in the remaining category are *unobservable* ones, such as a book's author (e.g. for alphabetical sorting). The only way to determine values for those unobservable parameters consists of asking a human for help (see below in Section 3.2). The inclusion of properties within this category might potentially lead to a large number of agent inquiries, so that we recommend to avoid it entirely.

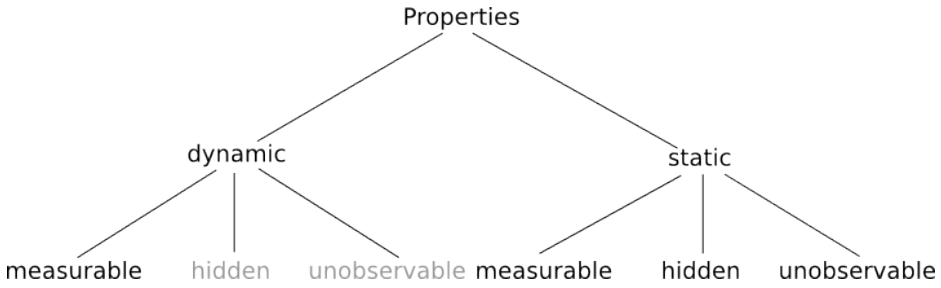


Figure 3.1.: Properties can be divided into two categories: dynamic ones that can change throughout a task and static ones that keep the same values. Properties also vary in their observability: They can be measurable, hidden (indirectly observable) or unobservable. In this approach, we assume that dynamic properties are always measurable.

Taking both property types into account, the object space can be rewritten as

$$\text{Obj} := P_1^{(s)} \times \cdots \times P_K^{(s)} \times P_1^{(d)} \times \ldots P_L^{(d)}$$

where K is the number of static properties and L the number of dynamic properties. Note that $K+L=J$ i.e. each property belongs to exactly one of the two categories. In order to access an object's properties, we define projections

$$p_j : \text{Obj} \to P_j \quad \forall j \in [1,\ldots,J]\,.$$

These projections map from an object onto a specific property's value.

3.1.3 Goal Definition

In a way, every POMDP model is based on human intention, i.e. the goal that the creator had in mind is reflected in the model. This might be a very broad definition of human influence. Yet, it is of value to understand by which means such a goal is modeled.

One way to reinforce goals is by an appropriate reward function. Rewards for various actions and observations are chosen to recompense achieving certain states and to punish others: Primarily, a planning agent is expected to solve a task while minimizing the number of actions (at least in the most common applications). This can easily be arrived at by punishing each action with a minor negative reward. Then, the goal state should be attractive because of its high positive reward. Furthermore, there might be subgoals that should be fulfilled during the process, and whose identification might facilitate the planning. These subgoals should likewise be tagged with a high reward. Of course, some states should be avoided entirely which can be implemented via high costs. Hence, a policy is expected to only choose actions with low certainty of transitioning to this state. In addition, it is possible to further assess how well the goal has been reached to acknowledge for good, but non-optimal policies. Therefore, the particular terminal state as well as the additional history of states passed by is evaluated. This procedure is of particular importance for tasks that are not always solvable for certain initial states.

In summary, the more specific the rewards are chosen for each state, the more the average reward for a plan results in a meaningful and reliable success metric [10]. However, balancing multiple rewards for one model becomes increasingly demanding. It is far from trivial to alter rewards for including user preferences. As we design our system for non-expert users, we cannot expect them to manipulate the reward function directly. Instead, a method for indirectly translating user

expectations into rewards would be needed. This is a challenging implementation and might, for example, be realizable through Bayesian Reinforcement Learning (which requires a lot of training [60]).

For our approach, we decided to avoid defining rewards, due to complications in definition and balancing. The only possibility to gain a reward is by fulfilling the task and reaching a goal state. We assume that there is only a single task goal to be achieved, even though there are multiple objectives. For instance, it could simultaneously be required to solve the task as fast as possible and collecting information on the way (e.g. a vacuum robot creating a map of its surroundings as well as cleaning the floor). For manipulation tasks this assumption is meaningful as they are usually defined with respect to one specific goal, such as attaching all parts in an assembly task. As we assume that the tasks are fully solvable, rewards terminal states with only partial solutions are unnecessary.

Nevertheless, multiple equivalent manifestations of a particular goal might exist, and therefore multiple goal states with the same reward might result. In any case, only one goal state can be reached in each trial with associated single rewards collection as goal states are modeled as terminal. By nature, each of the objects has to meet certain requirements in a goal state, such that the associated set G can be defined as

$$G = \bigcap_{h\in[1,\dots,H]} \left\{ s \in S \;\middle|\; f_h(\mathrm{obj}_1, \dots, \mathrm{obj}_N) = x_h,\ \mathrm{obj}_n \in s \right\}$$

where f_h is a particular constraint function for which a certain value x_h is required. There can be multiple constraints — and hence functions $f_h(\mathrm{obj}_1, \dots \mathrm{obj}_N)$ — modeling all conditions to be met for a goal state. The total number of constraints defining a goal state is denoted as H.

A negative reward for each transition to minimize actions is not necessary because of the discount factor applied on future rewards in the PGI algorithm: The more steps it takes to reach the goal, the smaller the discounted expected reward becomes. Hence, our reward function is defined as

$$R(s,a) = \begin{cases} 100 & \text{if } s \in G \\ 0 & \text{else} \end{cases} .$$

Note that the rewards can solely be gained by reaching a goal state.

3.1.4 Risk Avoidance

Terminal states are not limited to defining a goal, but also allow for avoidance enforcement by tagging risky states. As no transition is possible in a terminal state, and therefore, no rewards can be collected, an additional punishment through negative rewards is redundant. Here, the reward maximizing planning agent will automatically try to avoid these states. Thus, situations that are deemed harmful for objects or negative for the task's progression, can be excluded. A terminal state is already reached, if a single object condition considered 'at risk' appears. Therefore, the set of undesired terminal states T can be defined as

$$T = \bigcup_{i\in[1,\dots,I]} \left\{ s \in S \;\middle|\; g_i(\mathrm{obj}_1, \dots \mathrm{obj}_N) = y_i,\ \mathrm{obj}_n \in s \right\}$$

where the functions g_i represent the collection of all states associated with a high risk. Although, there is a total of I 'at risk' states, i.e. risk functions $g_i(\mathrm{obj}_1, \dots \mathrm{obj}_N)$, it fully suffices that one constraint is fulfilled to render s undesired, hence terminal. For user convenience, an outside parameter v can be added to the predefined terminal constraints, i.e. $g_i(\mathrm{obj}_1, \dots, \mathrm{obj}_N, v)$, which expresses a limit value for a specific task if desired so by the user. For example, the cumulative weight of all books on a single shelf should not surpass 3kg. This additional parameter can be varied at any time in between trials. Hereby, a non-expert user does not have to deal with POMDP definitions, but is still able to adjust the model to his needs.

3.2 Adding Oracle Actions

As described in section 2.2.1, adding an oracle action a_{oracle} provides an intuitive way for the agent to actively gather specific information. There are different types of information that can be asked for, such as best actions, goal specifications, or state information.

3.2.1 Action Recommendations

In the case of oracle recommendations for the next action, the user instantaneously a single or multiple particular actions. Therefore, he has a lot of influence on a specific part of the trial. Yet, a non-expert user might experience difficulties in understanding, how his action selection will affect the task's outcome. There is no immediate user control reaching further than the next action(s).

Information gain is extensive if the agent is uncertain about the next task. However, it is challenging for a user to select the best action in case of a manipulation task. Here, there are most likely several similar objects that have to be considered simultaneously while the order of manipulation might not be of importance. In addition, it has been shown in [61] that the human brain tries to operate with high efficiency in decision making. Therefore, it does not analyze all possibilities in detail in order to find the absolute best solution. Instead, it quickly selects an option that is 'good enough'. Thus, it might appear easy for the user to suggest a good action, but it is very demanding to choose the best one.

One benefit of oracle implementation seeking action recommendation emanates in situations which are assigned a high risk by the agent. In this case, human advice can be of utmost help, for instance in situations in which the agent is uncertain about the object being damaged. In a setting familiar to the user, he would most likely be able to understand the risks and advise the agent on how to resolve the situation.

3.2.2 Goal Specifications

For tasks with implicitly embedded goals (i.e. tasks seeking goal specifications), solving techniques can be implemented by adding additional parameters to the state space. By nature, these additional parameters are characterized as according to the previous classification and they reflect user preferences on the task's outcome. In order to solve a task with implicitly embedded goals, a set of possible goals has to be predefined, and a process for mapping user reactions to goal selection has to be established. As we want to offer the user a flexible selection of preferences, concerning a wide range of object properties, finding a solution to this model might be very expensive. Establishing the goal at the beginning of each trial results in additional actions to be performed by the agent. As goals might be difficult to distinguish, the number of necessary actions likely becomes very large.

The key assumption underlying this approach consists of allowing the user to desire a different outcome every time the task is performed. Therefore, the user is supposed to watch the process closely and to point the agent into the right direction. In contrast to this interpretation, we assume that our users expect the same outcome in general and only occasionally change their preferences. As a consequence, our conceptual user would not be willing to observe the agent in order to suggest alternations every time a task is performed. Although the just discussed technique would be a possible approach for preference integration, we decided for an alternative solution being computationally less expensive. This approach is described in section 3.3.

3.2.3 State Information

As stated in section 3.1, some of the static object properties are modeled as or even unobservable. If those properties are of importance for the planning process, an oracle action inquiring about the current state provides a convenient method for collecting this additional information. To maximize the information gain and to limit the user's effort at the same time, the agent can ask about a single parameter of a particular object in each oracle action. Thus, an oracle question is defined as

$$a_{\text{oracle}}(s) = < \text{obj}_n, P_j >,\ \text{obj}_n \in s\ .$$

Here, the agent is able to precisely select the needed information; the user is simply required to enter a single parameter value per question.

By nature, we assume that the user has an interest in the agent solving a task correctly. Furthermore, the required answers are considered easy to obtain by a human in case of a familiar environment such as a household. Therefore, providing an answer to the agent's questions should be intuitive and should not require substantial cognitive effort. This is in contrast to other approaches previously discussed. In particular, selecting the next best action might be significantly more difficult (depending on the current state).

As the user selects the number of questions a priori, we expect him to be available for all of them. If he has no interest in training the agent, he may change the settings to '0 questions'. As neither user availability nor user uncertainty are considered as influencing factors, the observations provided by the user are modeled as deterministic. This assumption might not always be strictly maintainable, but incorporating user uncertainty would require a disproportional effort with respect to the resulting inaccuracies (see section 3.4).

In contrast to other previously discussed methods, the POMDP model does not require significant effort in order to include this new information. As already described in section 2.2.1, it is merely necessary to expand the observation function such that it reflects potential answers. In addition, the transition function has to be adapted: After an oracle action, new information has to be incorporated into the current belief state.

The user's effort can additionally be limited by keeping the number of questions small. Therefore, existing approaches suggest the assignment of a high cost to oracle actions. But for this study, we prefer to avoid reward balancing and consequently introduce a question counter instead. The user is enabled to select the number of questions he is willing to answer in each trial a priori via the system settings. The counter is added as a parameter to the state space and is automatically reduced after each question. As soon as the counter reaches 0, the agent is prevented from asking any more questions. As every inquiry corresponds to an additional action further discounting the expected reward, the agent queries a user only if indeed necessary; it does not always utilize the full number of allowed questions.

3.3 Including User Preferences

It has already been detailed how user preferences can be included, i.e. by defining undesired terminal states. In this section, we present an alternative technique allowing non-expert users to specify their preferences in the form of logical constraints. Therefore, we reformulate the task as a Constrained POMDP (CPOMDP). At first, we describe our solution to the CPOMDPs problem before illustrating the definition of constraints in detail.

3.3.1 Solving Constrained POMDPs

In order to solve CPOMDP problems, we modify the PGI algorithm illustrated in section 2.1.2.2. Therefore, we introduce a constraint function C to be computed besides the previous rewards in the algorithm. This function's evaluation determines whether a particular state fulfills all given constraints, i.e.

$$C(s) := \begin{cases} 1 & \text{if } \forall\ m \in [1,\dots,M] : c_m(s) \\ 0 & \text{else} \end{cases}$$

where $c_m(s)$ corresponds to a single constraint as defined below. During the backward pass of the PGI algorithm, the expected reward value for a specific node is determined by simulating particles following the policy graph until the planning horizon. For each specific particle b_i, the associated reward sum is calculated by adding the discounted rewards collected at each time t. Then, the average discounted reward for all particles is obtained.

In a very similar way, we determine the constraint value for each node. Here, we multiply the constraint values received at each time instead of computing the sum:

$$C(b_i) = \prod_{t=\tau}^{T} C(b_{i,t}), \quad b_i \in B,$$

where B is the set of particles at a policy graph node, and τ denotes the associate node's time. The resulting value is equal to 1 if the particle meets all constraints for all times, and it emanates as 0 if at least one constraint is violated at any time. Hence, the average constraint value over all particles reflects for a particular node the percentage of particles that will meet all constraints in the future, i.e.

$$C(B) = \sum_{i=0}^{N} C(b_i) \cdot w_i$$

where N is the number of particles in B, and w_i are particle weights. Now, the constraint value C is employed instead of the reward value R in order to select best actions and best next nodes. If the constraint value arises as 0 for all possible

choices, the algorithm simply reverts to the reward values for evaluation. This contingency mechanism is included to avoid a planning failure for non-solvable constraints.

To bypass a discount factor, a counter is included for every state. This counter limits the number of actions the agent is allowed to perform before reaching the goal. In order to ensure termination, one particular constraint c_s always has to be part of C: c_s is fulfilled if a terminal state is reached before the maximum number of actions is reached. The limit value can be altered by the user. As the maximum number of actions in a trial is an intuitively understandable property, non-expert users should be capable of select meaningful values.

3.3.2 Constraint Definition

Constraints always consist of two parts: First, either a single or multiple objects of concern are selected via one of the static properties $P^{(s)}$ (for instance, all objects having a certain id or a specific color). Then, the desired constraint for those particular objects is incorporated by utilizing one of the dynamic properties $P^{(d)}$ (such as the object positions). This procedure can be formalized in the following way:

$$c_m(s) := \left[\forall \text{obj}_n \in s : p_k^{(s)}(\text{obj}_n) = \alpha \implies p_l^{(d)}(\text{obj}_n) \,\square\, \beta \right]$$

where $\alpha \in P_k^{(s)}$ and $\beta \in P_l^{(d)}$. The statement about the dynamic property $P_l^{(d)}$ allows for alteration in order to satisfy the specific user demands:

$$\square \in \{>, \geq, <, , \leq, =, \neq\} \,.$$

The such defined constraints are required to hold for all objects within a state that fulfill $p_k^{(s)}$, for instance, *'all yellow books have to be on a certain shelf'*. In addition, the user might likewise desire to express preferences concerning only a single object. For example, he would like to have one yellow book on a particular shelf, but does not care which one. Hence, we need to additionally formalize that only a single object has to fulfill the dynamic property, even if several ones meet the static property:

$$c_m(s) := \left[\exists \text{obj}_n \in s : p_k^{(s)}(\text{obj}_n) = \alpha \wedge p_l^{(d)}(\text{obj}_n) \,\square\, \beta \right]$$

where $\alpha \in P_k^{(s)}$ and $\beta \in P_l^{(d)}$. Here, the value 1 is returned as soon as the requirements for properties $P_k^{(s)}$ and $P_l^{(d)}$ are met by at least one object.

We can further distinguish constraints by the time duration for which they should be applied: Constraints can either be relevant continuously (as assumed above) or only at certain times.By nature, constraints for goal states have merely to be verified if a terminal state is reached. Therefore, we extend the function $C(s)$ to

$$C(s,t) := \begin{cases} 1 & \text{if } \forall m \in [1,\ldots,M] \;:\; t \in [\tau_{\text{min,m}},\ldots,\tau_{\text{max,m}}] \implies c_m(s) \\ 0 & \text{else} \end{cases}$$

where $\tau_{\text{min, m}}$ and $\tau_{\text{max, m}}$ characterize the time interval for which constraint $c_m(s)$ has to be fulfilled. If $\tau_{\text{min, m}}$ and $\tau_{\text{max, m}}$ are equal, the constraint is only relevant for one specific point in time $t = \tau$. Hence, a terminal constraint, i.e. a constraint only holding in terminal states, is a special case of this formulation. Note that defining such a terminal constraint is equivalent to adding undesired states to T, as discussed in section 3.1. Nevertheless, we believe that selecting constraints is more intuitive for non-expert users than constructing POMDP states to be avoided. An overview of all possible constraint types is depicted in Figure 3.2.

The constraints as outlined above can be easily translated into a graphical user interface: Here, necessary elements that have to be selected are the constraint type, both associated static and dynamic properties, corresponding property values as well as the relation between β and $P_l^{(d)}$, i.e. the above denoted relation $\square$. Those selections can be realized through drop down menus or radio buttons, for example. The possibility of creating multiple constraints should be accounted for, likewise.

Note that not every constraint type might be useful for each context. It is advisable to limit the number of selections to maintain intuitive usability.

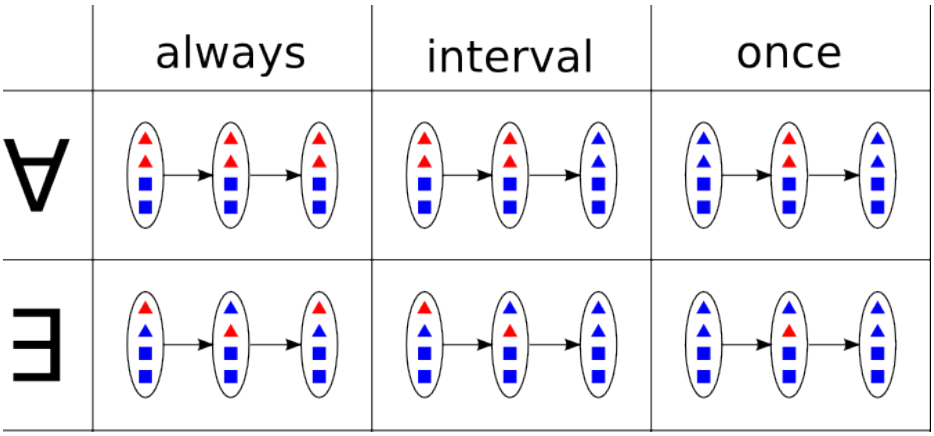


Figure 3.2.: In this figure, each oval symbolizes a state that changes over time. The shapes within each state represent the objects belonging to the particular state. Differing shapes depict distinct values for a static property $P_k^{(s)}$. In the given example, $p_k^{(s)}$ selects the triangle objects. Here, either a single or all of the triangles have to fulfill the constraint on their dynamic property $P_l^{(d)}$. We can further distinguish between constraints that hold continuously and constraints that are limited to a certain time interval.

3.4 POMDP Uncertainty

In reality, user input can be prone to uncertainty: As discussed in section 2.2.2, users may be unavailable, not willing to provide help to the robot, or simply give wrong answers.

In our model, we attempt to minimize the influence of these factors. For once, we let the user select the number of questions he is willing to answer in one trial. Furthermore, we have designed the oracle questions in such a way that it should be effortless to provide the answer. In addition, we have incorporated a fallback principle into the algorithm by reverting to reward evaluation if user-defined constraints cannot be met.

However, disturbances may still occur as a user might be very busy — and therefore unavailable —, or as he might mistakenly provide the wrong answer. Yet another uncertainty factor arises from the robot and sensor hardware itself: On the one hand, the agent might change or move an object in an unintended way during the execution of an action. On the other hand, many sensors are imprecise in cluttered environments (in which we intend to utilize our model), and hence, observations after each action might be noisy.

Nevertheless, it is not trivial to estimate uncertainty for POMDP models as described in section 2.1.2.4. Both previously proposed methods, i.e. performing a large number of prior experiments and learning the necessary values over time, are very complex and therefore exceed the scope of this work.

3.4.1 Preserving Knowledge

In section 3.2.3, a method for collecting information about hidden or unobservable properties has been described. The agent is not able to directly obtain this kind of information. Instead, either an inquiry for help has to be sent to a user or additional exploration actions have to be performed. As such obtained knowledge on properties is costly, yet highly valuable, it should be preserved throughout the execution of a task.

This is best illustrated by an example: Let there be books of two different sizes as well as two shelves with corresponding heights. The agent has attempted to place a book on the smaller shelf in the most recent trial. As he has not been able to succeed, the agent has deduced the size of the book to be of the larger kind. This knowledge is preserved and available for utilization in the next trial.

In our approach, all information about static properties is stored after each successful trial: The particles in the terminal belief state $\mathbf{b}_T$ reflect the probability distribution over all possible values for each property. These obtained values are employed in the next trial to sample a new initial belief $\mathbf{b}_0$. Thus, uncertainty about the state space is reduced over time,

and planning becomes more efficient. Thereby, the number of actions necessary for each trial is likewise reduced as less exploration actions become necessary.

4 Box Stacking under Uncertainty

To demonstrate the applicability of our methods, we employ them to solve a specific planning task: utilizing a robot arm for stacking boxes in a not fully observable environment with potential help by human non-experts. Such object manipulation challenges are popular showcases for robotic task planning while stacking paper boxes is a common representation for this class of tasks (see [62] and [63]). In fact, planning box manipulation is one of the most famous artificial intelligence problems, first introduced by Winograd in 1972 in [64] as "blocks world". Here, the domain's strength lies in the associated ease of creating minimal examples for non-trivial planning tasks that are not solvable by a simple, greedy approach, but afford non-myopic planning. This is exemplary depicted in Figure 4.1. Furthermore, this particular class allows for the simple introduction of specific properties for both the environment and the boxes that have to be additionally taken into account for the planning process.

Figure 4.1.: In this task, only one box can be moved at a time, and the goal is to stack the boxes in a specific order. To reach this goal state, the partially existing tower has to be destroyed first. This example is taken from [1].

In our case, a robot arm is required to build a tower from a given set of boxes. There is a hidden weight that potentially causes tower instability hidden in each box. The center of mass of each box is considered to be a point mass with an unknown position inside the box. These hidden weights potentially cause instabilities in the tower. The only approach for the agent to determine a box's weight position consists of placing it on the tower and of observing whether it stays on top or not.

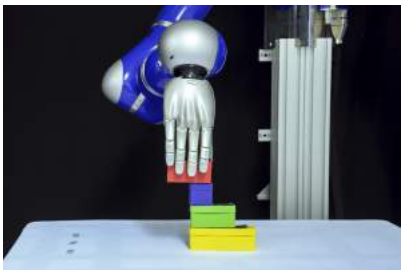


Figure 4.2.: In an experimental setup, a robot arm is stacking paper boxes. Because of the position of the hidden weight inside the red box, the current stacking action will result in the box falling down.

This setup contains all characteristics of a typical assembly planning task in an uncertain environment: The state space is large, and there are partially unknown objects whose properties have to be determined by gathering actions that also effect the progress of the assembly. Also, it is likely that the manipulated objects have additional properties that are not perceivable by a robot, but a human. On the one hand, it would be helpful if a human could provide this information if asked. On the other hand, humans might complicate the situation as they might be influenced by preferences regarding a particular way of fulfilling the outcome requirements of the task (which might be based on access to additional knowledge). Therefore, the task goal allows for freedom to include supplementary specifications.

It should be noticed that our approach as demonstrated in this example of box stacking can directly be transferred to other tasks, such as putting dishes into a cabinet, organizing laundry or putting groceries into the refrigerator.

4.1 Box Stacking Task Specification

The boxes in our setup have different properties including length l, height h and the hidden weight's position pos_w. These properties serve as examples on for different requirements for planning in real-world scenarios. Another example would be material and color of clothes, that are sorted for laundry. As described in Chapter 3, the underlying model features can directly be adapted for other related object manipulation tasks, such as other household functions. Our experimental setting is detailed in the following section: We will first explain the paper boxes in detail and subsequently describe our assumptions on the box stacking process.

4.1.1 Boxes

In general, boxes can have different sizes. However, in the experiments conducted for this thesis we will from here on assume that all box sizes are multiples of a unit value. This simplifies the POMDP planning and keeps the state space at a reasonable size. In theory, this value can be chosen as an arbitrarily small number to match any set of boxes. In our case, a value of 5 cm was selected. As an additional feature, boxes also admit different colors, which signify different object properties. Figure 4.3a shows an example for such a box.

(a) Paper boxes have different lengths and heights expressed in the unit value as well as varying colors.

(b) A coin is used to shift the center of mass of each box. It is attached to one of several predefined weight positions.

Figure 4.3.: Each paper box has specific observable dynamic properties: length, height and color. It also has a single hidden property: an interior weight whose position cannot be perceived directly.

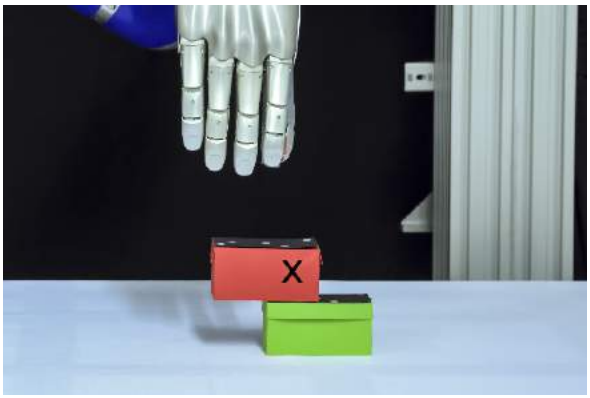
Inside each box, a hidden weight is attached to one of multiple predefined positions and we assume that the position will not change during the experiment once it was chosen during initialization. (see Figure 4.3b). Coins are utilized as weights. Thus, the boxes' centers of mass are altered, and this has to be taken into account during the tower-building process. As the robot cannot perceive the weight positions inside the boxes, uncertainty about the world state is introduced in this setting. The initial uncertainty can be reduced via exploratory actions where the weight positions decide about the outcome. This is illustrated in picture 4.4.

The modeled problem is considered as two-dimensional, i.e. boxes can be positioned in x-direction while the tower grows into z-direction. Both axes are partitioned utilizing discrete positions of unit length. There is a designated 'construction area' where boxes can be placed and a 'storage area' where currently non-integrated boxes are positioned (see Figure 4.5).

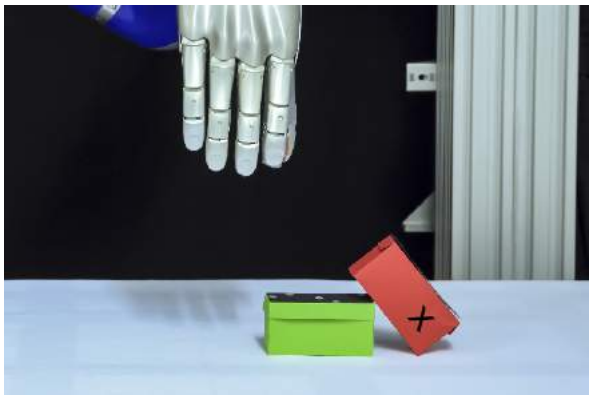
4.1.2 The Process of Building a Tower

A trial starts in the initial state which consists of one box positioned in the construction area as the tower foundation. The robot must not move this box as it would simplify some of our planning tasks. The remaining boxes are spread adjacent in the storage area.

For each iteration, the agent performs one single action. An action in the basic experimental setting always involves moving one box to a new position. As the robot grasps the boxes from above, only a single box can be carried each



(a) As the weight's position is on the right side of the red box, stacking it on the left side of the green box is stable.



(b) Positioning the red box at the right edge of the green box is unstable as the weight is on the right.

Figure 4.4.: The tower's stability depends on the positions of the hidden weights. Here, the ×-symbol marks the weight position in (a) and (b).

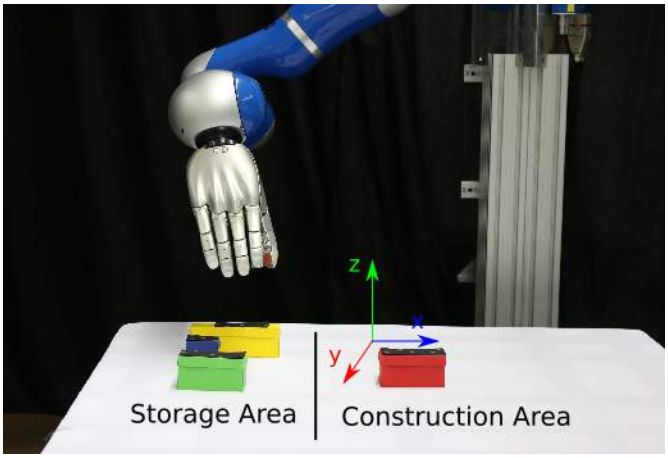


Figure 4.5.: The task environment consists of two different areas: the storage area where unused boxes are positioned, and the construction area where the tower is built. The box coordinate system has its origin between the two areas.

iteration, yet, any box can be selected. If a particular box is chosen that is part of the construction but not the top one, all boxes above the selected one are pushed aside by the robot hand before the robot grasps the target box (This behavior should appear in any good policy, though.). The boxes brushed aside are falling down, by nature, and are available in the storage area for the next iteration.

The selected box is then moved to a new position. This position can either be part of the tower structure (see Figure 4.7a) or be within the storage area (see Figure 4.7b). It is not admissible to position boxes anywhere else than on top of the existing tower structure or within the storage area. In particular, it is prohibited to create a secondary tower (see Figure 4.7c). Furthermore, it is not possible to place two boxes at the same height within the tower (see Figure 4.7d). For each performed action, there are two possible outcomes: either the tower remains stable or at least one box is toppling. As in the case of brushing aside, dropped boxes are assumed to move back to the storage area and are available for the next iteration. A trial is completed when a terminal state is reached. Depending on the specific experiment, different terminal states may occur. Here, the desirable end state of a tower containing all boxes is, of course, included. Nevertheless, a trial can converge to other states as well, depending on the specific box properties. For instance, a box might break due to falling from a certain height. As this box is rendered unusable, the task can not be completed, and the trial is therefore terminated. This process is depicted in Figure 4.6.

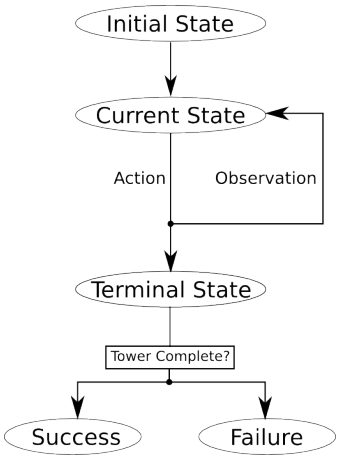
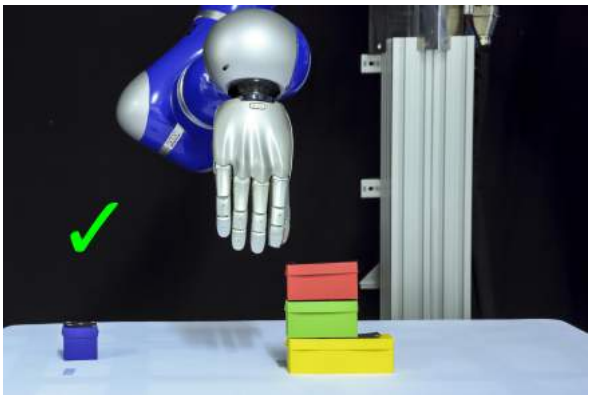
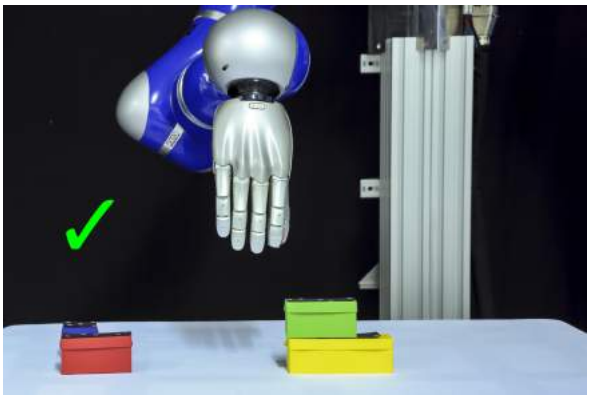


Figure 4.6.: Starting in the initial state, a sequence of actions is performed until a terminal state is reached. Here, it is verified if the tower is complete, and the trial has been a success, or if an early termination has occurred.

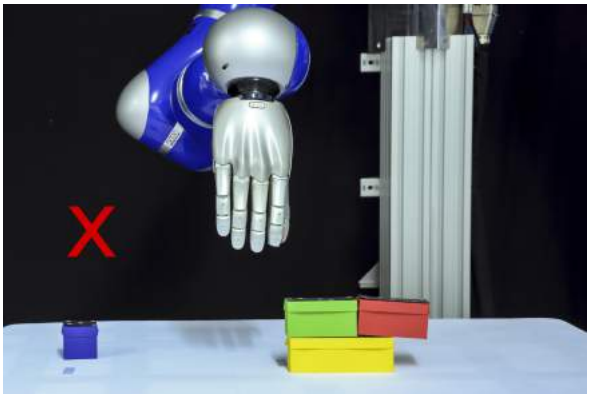


(a) The box can be positioned on top of the tower.



(b) The box can be moved back to its storage position.

(c) The box must not form the foundation of a secondary tower.



(d) The box must not be placed adjacent to another box in the tower at the same height.

Figure 4.7.: In the current action, the red box is moved. Admissible and prohibited new positions for the red box are illustrated in (a), (b), (c), (d).

4.2 POMDP Model

The previously described box-stacking application with the incorporated uncertainty about weight positions can be considered as a planning problem under uncertainty. As explained before, POMDPs are designed to solve such problems. This section first describes how to frame the task as a POMDP and then suggests various extensions that showcase our approach for user integration.

4.2.1 POMDP without Human Influence

As introduced in section 2.1.2, a POMDP is defined as a tuple $\langle S, A, O, R, T, \Omega, b_0 \rangle$. Within this framework, the box-stacking application can be modeled as follows:

- $S = \{\text{box}_1, \dots \text{box}_n\}$: Here, N is the number of boxes present in the experiment. The variable box_i is a tuple describing a specific box within the set S: $\text{box} = \left\langle id, \text{pos}_x, \text{pos}_z, \text{pos}_w \right\rangle$, color. Here, $id \in [1, n]$ identifies a particular box while $\text{pos}_x \in [-1, m]$ and $\text{pos}_z \in [-1, n]$ are integers specifying the box's current position. The values $\text{pos}_x = 0$ and $\text{pos}_z = -1$ is assigned if the box's position is in the storage area. The parameter $\text{pos}_w \in [1, w]$ denotes the position of the hidden weight inside the box.
- $A = \left\langle id, \text{pos}_x \right\rangle$: The variable id identifies a box, and pos_x is the x-position to which the box is moved to. The z-position is predetermined by the current positions of the other boxes. Note that more than one box can be moved in one action as boxes might drop from the tower structure.
- $O \in [0, n]$: The variable O denotes the number of boxes toppling during or after the execution of an action.
- T : Transitions are assumed to be deterministic. A physics simulation (see Section 4.3.1.3) is utilized to determine the box positions (stable or unstable tower structure) after an action.
- R : The reward gathered for a certain action is defined as

$$R = \begin{cases} +100 & \text{if a terminal state is reached} \\ 0 & \text{otherwise} \end{cases} .$$

  In a terminal state, the tower structure employs all boxes, and there is only one box at each height.
- Ω : Observations are modeled as deterministic. Similarly to the transition model, the gravity simulation is employed to determine if a particular action yields a stable tower.
- b_0: At time $t = 0$, a single box is located in the middle of the designated construction area, the remaining ones are positioned aside in the storage section. pos_x and pos_z as well as their assigned variable id are deterministic for all boxes. The combined possible weight configurations for the system of boxes is uniformly distributed.

4.2.2 Additional Terminal States

A refinement allowing to specify desired behavior is provided by defining additional undesired states as terminal ones that do not result in any reward. Thus, the application becomes more realistic. In our example, the boxes can be assumed fragile, i.e. they break and are rendered useless if they are dropped from above a certain height. In order to avoid this behavior, all states which occur after a box drop higher than a certain limit are therefore added to the list of undesired terminal states. If the algorithm reaches such a point, the task cannot be completed anymore, and the stacking process is aborted. The transition function has to be adapted accordingly: transitions from these states to any other states are removed. The extension via an additional negative reward is not necessary as it is a sufficient condition that the goal state with a positive reward becomes unreachable.

The maximum drop height can be selected by the user for each box before the beginning of each trial. That way he can ensure that more fragile objects are handled with more precaution and therefore avoid critical situations.

4.2.3 Oracle POMDP

As defined in section 3.2, an oracle action allows the agent to gather specific information from a user via a question. In our box-stacking application, the agent is allowed to inquire about the position of the hidden weight in a single box. It is assumed that the user possesses this information or is able to further investigate the box in question.

Thus, the following modifications within the POMDP model are necessary: An additional parameter 'oracle' is added to each state. This variable supplies a counter for the remaining number of oracle questions. In b_0, 'oracle' is set to the

value predefined by the user, i.e. it can be chosen how many questions are permissible. For each box, an action $a_{\text{oracle},i}$ is added. In case of selection, the user is asked about the weight position in box i. If the value of the oracle counter has reached 0 at the current state, no questions remain and nothing happens anymore with respect to the oracle action. In addition, it is necessary to prescribe transitions from each state s to a new state s' if an oracle action is performed. These transitions emanate immediately: The oracle counter in s is reduced by one, and the user's answer is assigned as the resulting observation. Here, the observation is a weight position in contrast to the previous number of boxes dropped. Thus, it has to be interpreted in a different way.

4.2.4 User Preferences

In our approach, users are allowed to influence the outcome of a planning task according to their preferences. Therefore, they may select specific constraints in the system settings regarding the resulting tower structure, as described in section 4.3.1.4.

It has been defined in section 3.3 that values of dynamic object properties can be subject to constraints while an object affected by a constraint is selected via its static properties. In case of our box stacking model, this translates into selecting boxes by their (unique) id or by their color. The latter might potentially result in a non-unique choice (multiple boxes). All boxes have to be part of the tower structure in order to accomplish the task's goal, but it is — in general — not predefined in which order the boxes are stacked. The user may constrain the value assigned to the parameter pos_z for a single or multiple boxes.

While the system attempts to fulfill all requests, it may be impossible to find an adequate solution in certain situations, e.g. if only yellow boxes are available. Then, the planning agent selects a solution scheme as close as possible to the fulfillment of the user requests.

For this proof of concept, we select two different constraint types: The first type has to hold for all boxes selected through the static property at all times. This way, potentially dangerous states can be prohibited. For example,

$$\forall b \in \text{Boxes} \ : \ \text{color}(b) = \text{yellow} \implies \text{pos}_z(b) < 2$$

yields a tower structure with no yellow boxes above the first two levels as these boxes are especially fragile.

The second constraint type concerns only terminal states and only has to hold for one box, such as

$$\exists b \in \text{Boxes} \ : \ \text{color}(b) = \text{yellow} \wedge \text{pos}_z(b) = 4$$

which signifies that the box on level 4 of the resulting tower has to be yellow. Utilizing this second type, the user may specify the final tower.

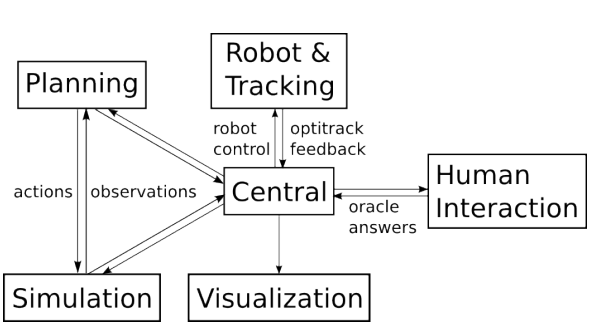
4.3 Software and Hardware Setup

In this section, we describe the experimental setup's software and hardware components. The software includes segments for planning, simulation of the box-stacking, visualization and user interfaces. The hardware consists of a robot arm equipped with a hand as well as a tracking system.

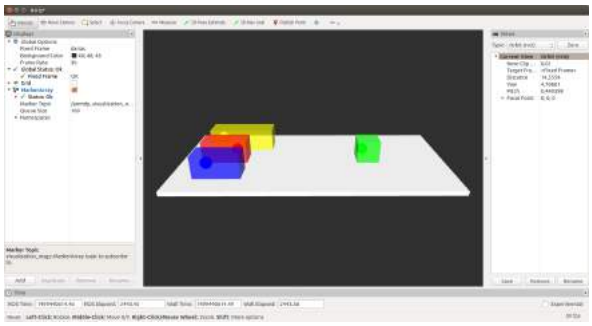
4.3.1 Software

The experimental setup is implemented in ROS (Robot Operating System). ROS is a middleware that enables several heterogeneous, communicating processes to run. These processes are represented in a graph architecture where each process corresponds to a node and each message-passing connection (between processes) to an edge. This architecture renders projects very modular as it is easy to replace one node with another (see [65]).

Our structure of ROS nodes is illustrated in Figure 4.8a. As depicted, there is a central node handling all communication between components. The planning node receives an observation of the current state; then, the planning agent decides for the best action to execute, based on the information about the current world state. Subsequently, this action is either simulated or performed by the robot. An observation of the action's outcome is returned for both cases. For the



(a) The application is partitioned into several communicating ROS nodes.



(b) The rViz application is used to visualize boxes and their hidden weights.

Figure 4.8.: The experimental setup consists of several software components including visualization.

simulation of an action, it is necessary to determine whether the tower remains stable after an action. Hence, a gravity simulation is necessary based on the positions of the hidden weights. The simulation's outcome is illustrated with rViz, a 3D visualization tool in ROS (see Figure 4.8b). For the real-world action case, an Optitrack camera system tracks the box position and thus supplies the outcome of the action. For an oracle action, neither simulation nor robot movements are required. Instead, the user information is provided through a dialog window which is controlled by the input node. In this section, all components are described in detail. In this subsection, we describe the software components utilized for planning, simulation, and user interaction.

4.3.1.1 POMDP Planner

In the planning node, all current information about the world state is utilized to determine the best subsequent action. This is done via the solution to the POMDP model defined in section 4.2: Here, the PGI algorithm (see Section 2.1.2.2) is employed to determine a solution for the POMDP, i.e. to find an optimal policy that reaches the task goal (a finished tower) as fast as possible.

In Figure **??**, an example for such an optimal policy is shown: Three boxes with different sizes must be used to build a tower. Box 1 is positioned in the construction area for the initial state and must not be moved by the robot. There are two possible weight positions in Box 2 and 3. The optimal policy, as depicted in the policy graph in Figure A.1, starts at time $t = 1$ by placing box 2 on top of box 1. Depending on the observation resulting from the action, box 3 is subsequently positioned et cetera. The observations and their probabilities are denoted on the graph edges. Terminal nodes are assigned two circles. Note that there are nodes which are terminal only for some cases, depending on the action outcome. The rectangle placed below each node contains the probabilities for the weight position in each box. As the same node may be reached repeatedly in different states, these probabilities do not always reflect the current knowledge. For example, the weight position in box 3 is already known at $t = 2$ in the upper node. But as this node can be reached after two different observations, i.e. box 3 stayed on the tower or box 3 fell, the overall probability in this node is 0.5 for each position.

Once the offline planning phase is completed, the resulting policy graph is utilized to determine the best action for the current state: As soon as a new observation arrives, the agent advances in the policy graph accordingly and returns the best action with respect to the new node. There are several additional parameters that need to be supplied for the PGI algorithm:

- γ: discount factor, $\gamma \in [0, 1]$
- N: policy length
- M: policy width
- B: number of belief states/particles

In order to provide comparable results, these values are indicated for each experiment.

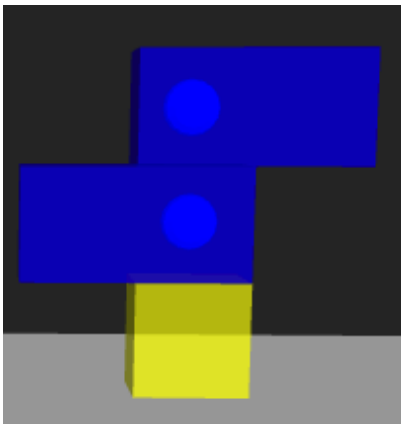


Figure 4.9.: In this example, 3 boxes have to be stacked: The yellow box serves as the tower foundation, and the two blue boxes are positioned on top.

4.3.1.2 Heuristic Planning

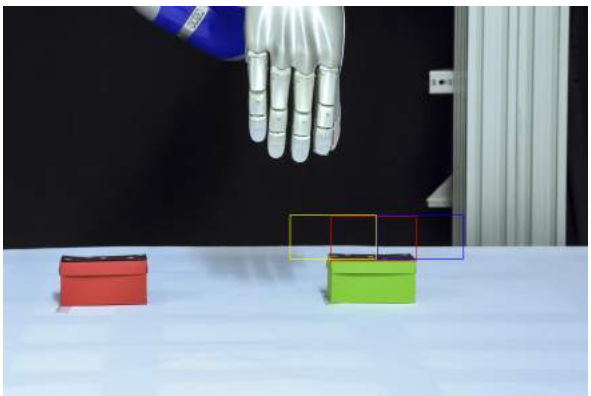
In order to evaluate our POMDP policies, we compare their results to those of a random sequence of actions. Planning should always lead to a significantly better outcome than these baseline results.

In an identical setting of boxes with unknown weight positions, random actions are selected and performed. There is a positive reward for completing the tower. Depending on the specific experiment, there are also terminal states without a reward. A trial continues until one terminal state is reached. Depending on this last state, the trial is counted as a success or failure. We also keep track of the number of actions performed in each trial. From this information, the discounted reward for one trial can be calculated:

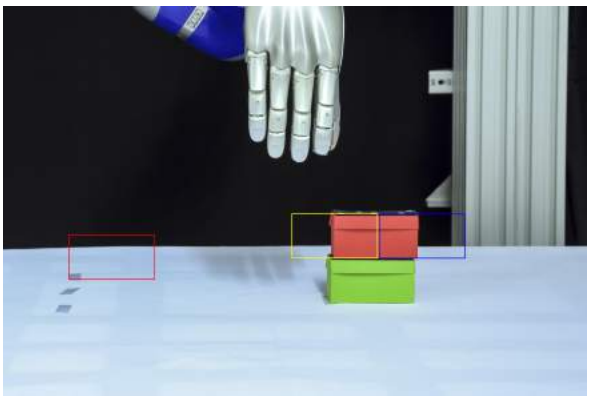
$$R = \begin{cases} \gamma^{n_a - 1} \cdot 100 & \text{if tower complete} \\ 0 & \text{otherwise} \end{cases}$$

where γ is the discount factor, s_T is the terminal state of a trial and n_a is the number of actions that were performed before reaching s_T. Finishing the tower is the only way to gain a reward, and a trial is over in that instant (the tower can not be destroyed again). That is why the reward for a failed trial is always $R = 0$.

In order to make the random policy more competitive, only "useful" actions are taken into account, i.e. only boxes from the storage area or from the top of the tower can be moved. As depicted in Figure 4.10, they can only be moved to positions that might be beneficial for the task's progress. Furthermore, actions that move a box from a certain position back to the very same position, i.e. do not move the box at all, cannot be selected.



(a) From the storage area, a box can only be moved to the top of the tower.



(b) From the top of the tower, a box can either be moved to a different position at the same height or to the storage area.

Figure 4.10.: This example shows allowed random actions for the red box, depending on its position. The transparent shapes illustrate possible new positions.

4.3.1.3 Gravity Simulation

In order to simulate the problem of tower construction in this study, the outcome of a box moving action has to be provided. While this is a trivial task for transporting a box to the storage area, it is necessary to analyze (based on a physics model), whether a box newly added to the top of the tower renders the structure stable or not. Here, instability is defined as the toppling of the just placed top box or of any additional number of boxes. As will be seen in the following discussion, it is crucial to monitor a stability criterion for each layer of the tower, based on the combined center of mass (CM) of the corresponding substructure above. Following the discussion, algorithm 1 suggests the implementation of the resulting gravity simulation.

Only two situations are of interest and, therefore, have to be distinguished: Either the tower structure remains stable or collapses after the box has been added. Here, the collapse can be partial or complete. While it is demanding to determine the exact equations of motion (EoM) for the actual toppling process (friction assumptions, changing resultant of the normal forces, translational and angular accelerations, etc.), it is rather simple to consider the dynamic equilibrium. In the context of the tower structure, the latter means that no motion, and hence no acceleration, occurs after a new block has been placed on top. In general, EoMs for dynamic systems can be determined via force-mass-acceleration, impulse-momentum, or work-energy methods. For the given two-dimensional structure of rigid bodies, the standard force-mass-acceleration approach suffices, such that the EoMs for a single block result as

$$\sum F_x = m \cdot \ddot{x} = m \cdot a_x, \qquad \sum F_y = m \cdot \ddot{y} = m \cdot a_y, \qquad \sum M_A = \sum (\mathbf{r}_A \times F) = I_z \cdot \ddot{\varphi} = I_z \cdot \alpha$$

where F_x and F_y are external forces in the x- and y-direction, m is the mass of a block, a_x and a_y are the translational accelerations in x- and y-direction, M_A are the resulting moments with respect to an arbitrary point A in the structure, r_A is the radius vector from the point A to the acting point of a force, and I_z is the moment of inertia component in z-direction. Gravitation is always acting at the CM of the block. In equilibrium, i.e. in rest, there are no forces acting in the y-direction. In addition, the resultant normal force, i.e. the resultant of the distributed surface pressure, can be thought to act along a line perpendicular to the surface and passing through the box's CM. In equilibrium, the resultant forces of gravity and surface pressure must have the same magnitude, the same line of action, and opposite sense. As a consequence, the sum of moments caused by these two different types of forces remains zero with respect to any point A in the structure as $\mathbf{r}_A \times F_g$ and $\mathbf{r}_A \times F_n$ will be of equal magnitude and opposing sign. No angular acceleration α is induced, and the tower structure remains stable. This stable equilibrium always exists if the projection on the x-axis of the block's CM lies within the x-domain of the block below. In more descriptive words, the location of the CM of the top block has to be within the x-limits of the block below.

Is the previous condition violated, the resultant forces of gravity and surface pressure cannot maintain the same line of action anymore as the normal force has to stay perpendicular to the the x-axis, initially. Hence, the sum of resulting moments will not remain zero and a rotational motion about the edge of the block below is initiated. As soon as this angular rotation starts, the CM of the top block will experience translational accelerations in x- and y-direction; the surface pressure will be concentrated as a single support force at the edge of the block below, orthogonal to the surface of the moving top block. The angular acceleration is determined by the mass of the block, the y-coordinate of its CM and the moment of inertia about the z-axis. At one point, the static friction force between edge and surface will be exceeded by the gravitational component orthogonal to the normal force, and the top block will start to undergo a translational sliding motion, in addition to the existing angular one. The last phase of the toppling process consists of the free fall of the top block superposed with a rotation of constant angular velocity. As previously mentioned, the exact EoMs for the general collapsing process are challenging. Yet, the conclusion on stability of the top block simply depends on its CM to remain immediately above the block below.

However, this stability condition for the top block is necessary, but not sufficient for the overall integrity of the tower structure. If this stability criterion is met, the combined system of the two top blocks has to be checked. Here, the surface forces between both blocks are treated as internal, and the free body diagram of the combined formation is considered. The previous discussion applies similarly if the combined CM of the two top blocks is utilized. If the mass of the blocks is treated as concentrated at the individual CMs, then, the coordinates of the resulting combined CM of this system of particles is given by

$$\mathbf{r}_{CM}^{(n)} = \frac{1}{n} \sum_{i=1}^{n} m_i \cdot \mathbf{r_i}$$

where n is the number of particles, i.e. blocks, m_i are the individual masses, and r_i the corresponding coordinates of the individual CMs. Now, the y-component of $\mathbf{r}_{CM}^{(2)}$, i.e. the y-component of the combined CM of the two top blocks has to

lie within the y-domain of the 3$^{\text{rd}}$ block from the top. If this is the case, the substructure of the two top blocks fulfills the stability criterion. This argument can be extended to every level of the tower, i.e. the substructure of the top n blocks remains stable if the y-coordinate of $\mathbf{r}_{CM}^{(n)}$ lies within the y-domain of the $(n+1)^{\text{th}}$ block from the top.

If this procedure is repeated for every level of the tower structure, a sufficient condition for stability results, and the simulation is enabled to determine tower collapse based on a physics model. Figure 4.11 exemplary illustrates the importance of analyzing every single level of the tower structure every time a new box is placed on top. Algorithm 1 shows the embedding of the discussed physics model via pseudo-code. More detailed explanations can be found in [66].

```
nrBoxesFalling = 0;
if tower.size > 1 then
    for i = 1 to tower.size do
        centerOfMass = 0.0;
        weight = 0.0;
        for j = i + 1 to tower.size do
            centerOfMass += tower[j].weightPos * tower[j].weight;
            weight += tower[j].weight;
        centerOfMass = centerOfMass / weight;
        if centerOfMass >= tower[i].rightBorder or centerOfMass <= tower[i].leftBorder then
            nrBoxesFalling = tower.size - i;
            break;
    return nrBoxesFalling;
```

Algorithm 1: Algorithm for stability verification at each level of the tower

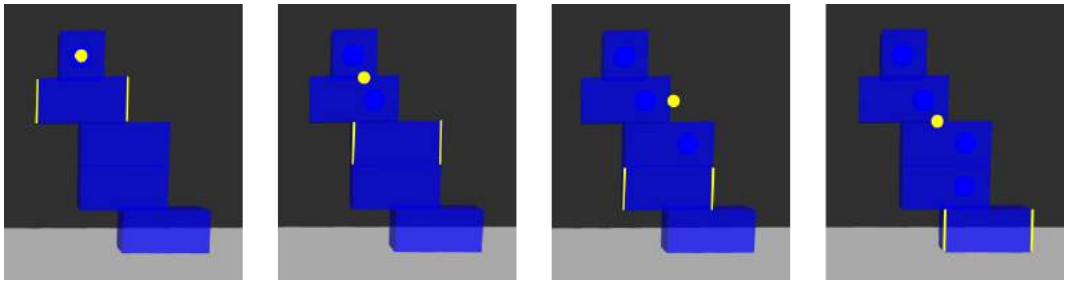


Figure 4.11.: This example demonstrates the necessity to monitor each level of the tower separately in order to determine whether the CM of the subsystem above is within limits: The small box has just been added to the tower. The blue circles depict the individual masses, the yellow circles the corresponding CMs of the subsystems while the yellow lines show the borders of the associated limiting box. Note that the stability criterion is only violated at the lowest level of the tower, and thus, all boxes will topple.

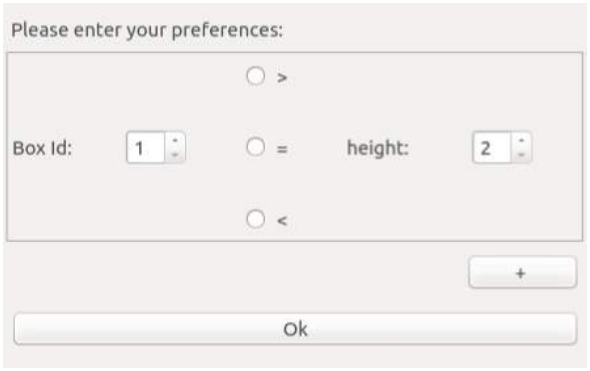
4.3.1.4 Human Feedback

To establish communication with a user, two intuitive user interface windows are provided. They open for the two situations in which user feedback is required: at the beginning of the task and for the oracle action.

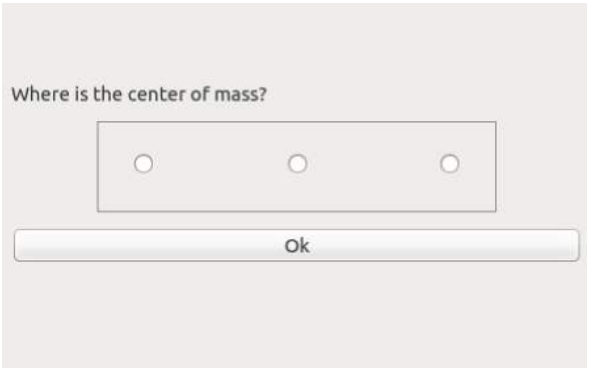
Before the planning process begins, the user is given the opportunity to define preferences concerning the outcome of the task. The corresponding dialog window is shown in Figure 4.12a. Here, the constraint choices match those defined in section 4.2.4. For instance, the height of a certain box in the tower structure can be selected. In addition, it is possible to add multiple preferences. Yet, the user's preferences are ignored and the building process nevertheless commences if the selected preferences contradict each other rendering the planning unsolvable. By nature, it is also permissible not to add any preferences at all.

The other situation in which information is inquired from the user occurs during an oracle action. Here, the planning agent asks about the position of the hidden weight inside a specific box. If the agent selects an oracle action, the robot

points at the box in question and a dialog window as illustrated in Figure 4.12b pops up. In the dialog window, the box is depicted while the answer possibilities (weight positions) are shown as radio buttons inside the box. The user selects one of the possible positions followed by clicking on the "OK"-button. Then, the given information is released to be employed by the agent for further planning.



(a) In the preference settings, the user may enter constraints for a single or for several boxes prescribing their height position in the tower.



(b) This dialog window opens when the agent asks an oracle question. The radio buttons symbolize possible locations of the hidden center of mass.

Figure 4.12.: The depicted dialog windows are examples for interaction with non-expert users.

4.3.2 Hardware

In this section, we describe the robot components, the utilized camera system as well as their corresponding usage during a trial.

4.3.2.1 Robot

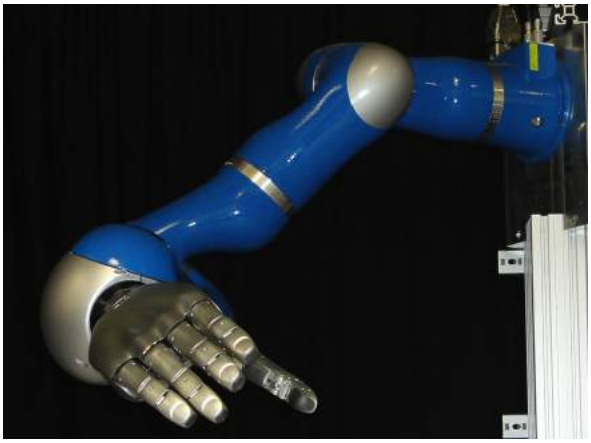
The robotic setup consists of a 7 DoF Kuka LBR and a DLR Hit Hand II. The Kuka LBR (see Figure 4.13a) is equipped with seven joints in an anthropomorphic configuration: three shoulder joints, one elbow, and three wrist joints. The abbreviation LBR stands for the German term 'LeichtBauRoboter' translating as light weight robot. It allows for torque- or position control with a frequency of 1kHz. The arm supplies a gravity compensation mode in which the joints can be effortlessly moved.

The DLR Hit Hand II (see Figure 4.13b) is a human-like dexterous hand with five fingers. For modularity reasons, all fingers are identical. Each one is equipped with four joints providing three degrees of freedom: proximal flexion, distal flexion, and spread. The fingers are operated using joint impedance control and deliver an active fingertip force of 10N. The features of the hand are described in detail in [67].

4.3.2.2 Optitrack

The Optitrack system is a real-time motion tracking system consisting of multiple synchronized cameras; for the employed setup, six cameras are mounted to the ceiling. The system is able to track reflective surface markers with a frame-rate of up to 360 Hz. Each camera has an LED ring, such that the light emitted by the LEDs is reflected by the markers. These reflections are captured by the camera where each pixel with a brightness above a certain threshold as considered as belonging to a marker. Then, the marker positions in the different overlapping 2D camera images are the utilized for triangulation in order to determine 3D positions.

With the help of the MotiveTracker software, multiple markers can be grouped and assigned to a single rigid object. Thus, distinction between objects tagged with unique marker patterns is enabled. Under ideal conditions, it is possible to simultaneously track more than 25 rigid bodies with a sub-millimeter accuracy. Apart from robotics applications, the Optitrack system is widely employed in human movement analysis.



(a) The Kuka lightweight arm has 7 degrees of freedom.

(b) The DLR Hit Hand II is equipped with five identical fingers.

Figure 4.13.: The robotic configuration consists of a KUKA lightweight arm and a DLR Hit Hand II.

(a) The optitrack camera has an LED ring and captures light reflections.

(b) Unique patterns of reflective markers identify the boxes.

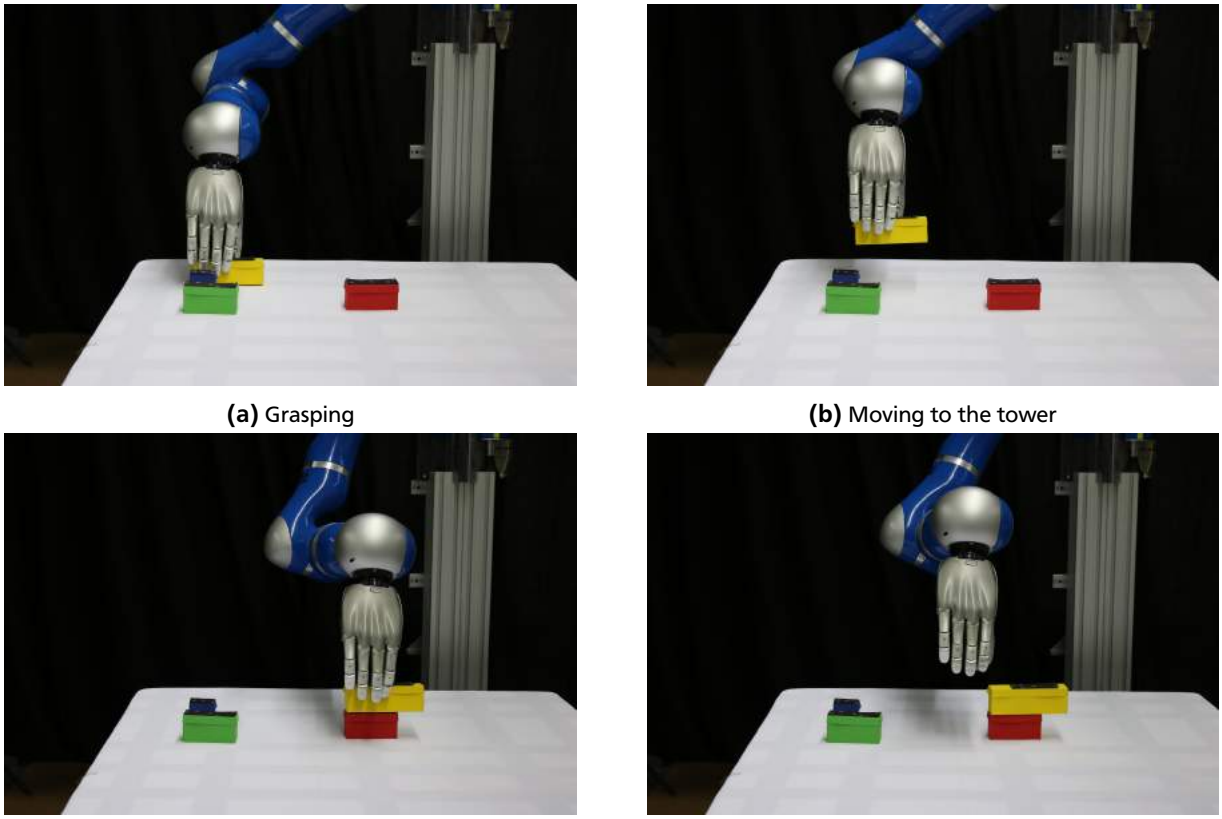
Figure 4.14.: The Optitrack system consists of multiple cameras which capture reflections from markers attached to the boxes.

4.3.2.3 Hardware Components in the Experiment

When the robot receives a new action, it has to handle the instruction appropriately. The engaged procedure consists of four steps as depicted in Figure 4.15: First, the robot hand is positioned above the particular box that is to be moved. The box's location is known precisely by means of the Optitrack system; each box can be individually recognized by its unique marker pattern. Then, the robot picks up the box by a two-finger grasp with thumb and the index finger. Now, the box is transported towards its new position. If this new position is on top of the existing tower, Optitrack data is again required to determine placement of the tower. As the box positions are all deterministic, the particular box on top of the tower and its position can be ascertained immediately. For the case of placement in the storage area, box position are known and no additional position information is necessary. Finally, the box is released at its new position.

To supply the observation value, the top box of the tower structure is identified via the Optitrack system. If this result is compatible with the action (by either being the new box or the box previously positioned below the top in case of removal), the action is deemed successful as zero boxes have dropped. Otherwise, the new top box is determined, and hence, the resulting completion level of the tower structure. Thus, the number of toppled boxes can be deduced.

As only discrete positions are employed in the POMDP framework, the position values have to be translated into real-world coordinates. Here, the fixed tower foundation serves as a reference point.



(a) Grasping

(b) Moving to the tower

(c) Stacking

(d) Releasing

Figure 4.15.: One robot action consists of several steps, i.e. grasping, moving, stacking, and releasing a box as depicted in (a), (b), (c) and (d).

5 Experiments and Results

In this chapter, we evaluate our approach for manipulation task planning under uncertainty with human involvement, as previously described in chapter 3. Therefore, we execute the stacking application as introduced in chapter 4 with different box configurations. To begin with, we present and discuss results from simulations: The PGI algorithm's performance on a basic box configuration is shown and compared to random actions. We further exhibit the dependence of the resulting plan's complexity on both the number of hidden weight positions and the particular box serving as the tower foundation. Then, we introduce additional terminal states as well as oracle actions in order to investigate the emerging effects on the solution. The first part of this chapter concludes with a demonstration of the algorithm's performance when adapted to constraints and when subject to user-defined preferences. The second part

5.1 Basic Box Problem

In this section, we detail the basic configuration of boxes used throughout all experiments. The selection of the PGI algorithm's parameters is described before its performance is compared to a random policy algorithm.

5.1.1 Setting Algorithm Parameters

The initial basic configuration consists of n boxes. One single box has a length of 1 and is employed as the tower foundation. The remaining ones are of length 2 and are required to be stacked on top of the foundation box. For simplicity, all boxes are assigned a height of 1. Two possible weight positions in each box are permitted, but all boxes share the same weight. As depicted in Figure 5.1, one solution for a stable finished tower consists of positioning all weights on top of the foundation. Other configurations are possible depending on the distribution of weight positions

(a) The yellow box serves as the tower foundation. There are 2 possible positions for the hidden weight in each of the $n-1$ blue boxes.

(b) For each layer of the tower, there are up to 3 possible box positions, depend on the previous layer. Yet, stability has to be maintained.

Figure 5.1.: In this setting, $n-1$ blue boxes of size 2 are stacked on top of a yellow foundation box of size 1.

amongst the boxes. Therefore, if a previously unknown box is positioned on top of the tower, the probability that it will stay on top is at least 0.5. That is why an optimal policy should need a minimum of $n-1$ actions a maximum of $(n-1)*2$ actions until tower completion. As we experiment with up to 6 boxes, a policy length of 15 offers enough buffer for non-optimal policies. Experiments also have shown that with the maximum number of boxes, a policy width of 15 was sufficient, as the number of possible actions in one time step never exceeded 9. As shown in 5.2, the PGI algorithm reaches convergence for all numbers of boxes after less than 15 iterations. A discount factor of 0.9 has shown a good tradeoff between fast convergence and solutions optimal in terms of policy length. However, as rewards can only be gained by the end of each trial, the selection of the discount factor is not crucial for good results. We simulate each resulting policy 1000 times. Among the initial states values for unknown properties are uniformly distributed.

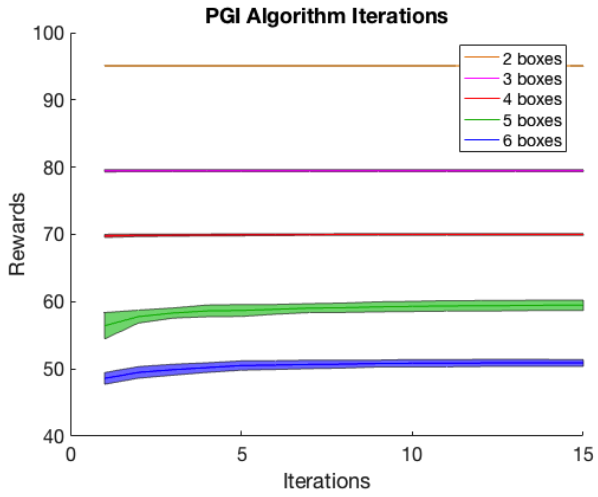


Figure 5.2.: Rewards converge in a similarly rapid fashion for different box counts.

Figure 5.3 shows the timespan that is necessary for planning an offline policy for different numbers of boxes. As both state space and action space grows with the number of boxes, it is not surprising that the amount of time necessary to find an optimal solution increases quickly. The large standard deviations can be explained with the randomness factor in the solving algorithm. In some trials, the initial random policy might already be good, in other cases several algorithm iterations are necessary before the resulting policy converges. Planning the task for 6 boxes takes on average 200 seconds. Each algorithm iteration takes about 30 seconds and in our trials, the optimal solution was found a minimum of 2 and a maximum of 9 iterations. Thus, the timespan necessary for planning varies a lot. Note, that in order to make different policies comparable, we kept a fixed policy graph size and number of particles, as described above. Therefore, planning for a smaller number of boxes could be faster, as a shorter planning horizon is required. As we assume that most household tasks are usually not very urgent, this is a reasonable amount of time.

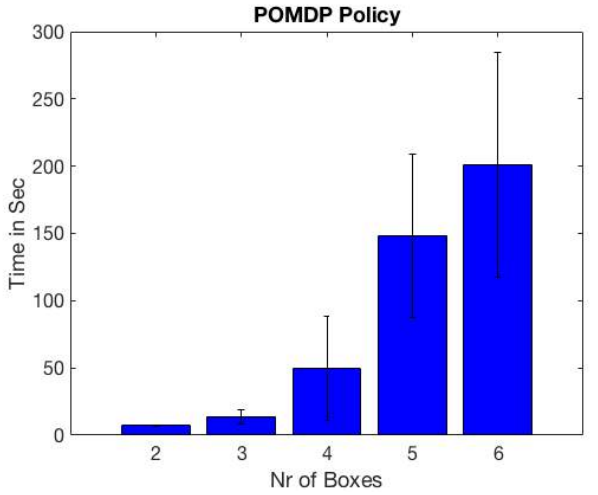


Figure 5.3.: Planning time increases with the number of boxes involved.

5.1.2 Comparison to Random Policy

Now that we have determined all parameters for the algorithm, we want to assess the algorithms performance by comparing its resulting policies to a random policy. Trial success is not a good performance metric in this case: In this basic setting, there is no terminal state except for the goal state. Therefore, there is no way that the trial ends before the tower is finished, assuming that the planning horizon is selected appropriately. The random policy may perform an unlimited number of actions. In conclusion, the success rate for both policies is equal to 1.

Instead, we compare the average number of actions necessary to finish the tower and the expected discounted rewards. Note, that these values are equivalent for comparison in this case, because the discounted reward depends on both the number of actions and the amount of the gained reward. As already established, the reward has always the same amount and is gained in every trial. Thus, it only depends on the number of actions in this particular case.

In this experiment, we varied the number of boxes from 2 to 6. We created 10 policies with both planning algorithms — PGI and random. The results can be seen in Figures 5.4 and 5.5: For 2 boxes, the number of actions is almost equivalent. In this case, as the foundation cannot be moved, only one box has to be placed. Therefore, there are only two possible actions to choose from. Thus, the probability of completing the tower with the first action is 0.5 for both policies. The difference between the results can be explained with the fact that the PGI policy learns from its observations. When the first action failed, the only solution can be the remaining action. Therefore, the average number of steps is always 1.5. However, the random policy has no memory and therefore might choose the same action consecutively. That is why the average number of actions slightly increased.

The average number of actions grows linearly with the number of boxes for the PGI policy. As stated in the previous section, the maximum number of actions in an optimal policy is $(n-1)*2$ in this setting. This limit is met for all numbers of boxes. Yet, the number of actions for the random policy is growing exponentially with the number of boxes. For 6 boxes, an average of 1941 actions result in tower completion. An exemplary policy graph for 3 boxes can be found in A.

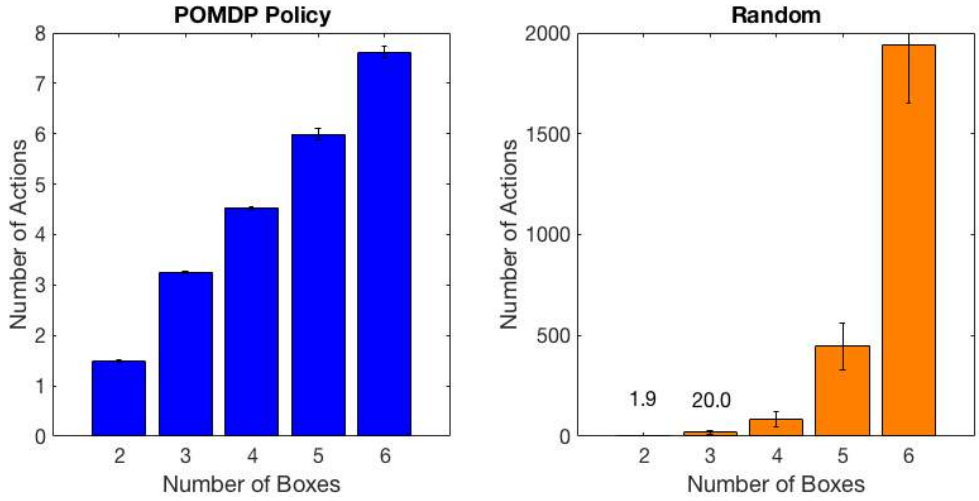


Figure 5.4.: The average number of actions needed for tower completion is increasing with the box count. Note that this quantity is growing significantly faster in case of the random policy in comparison to the POMDP framework.

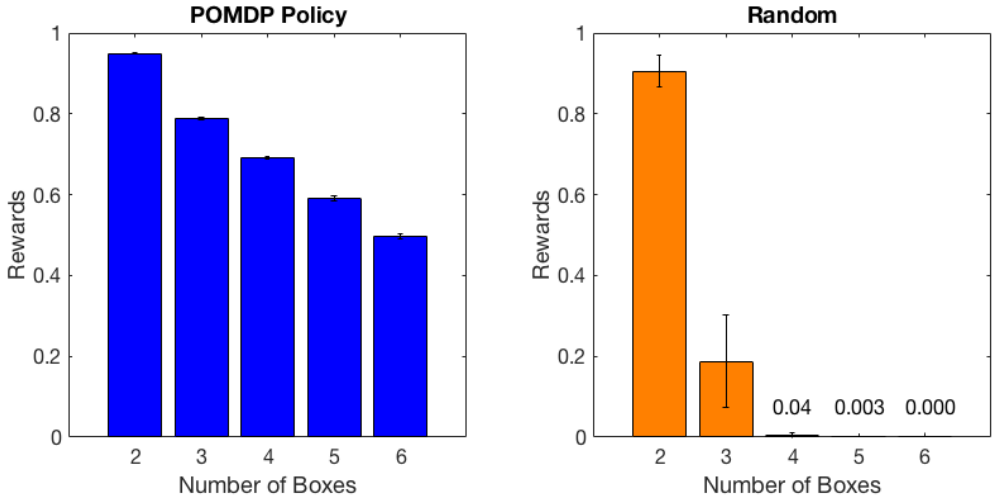


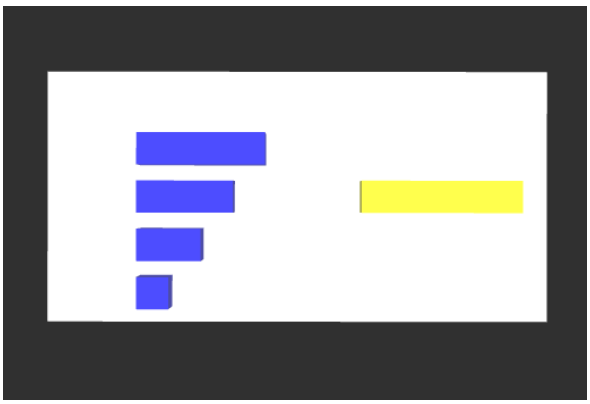
Figure 5.5.: The average discounted reward is decreasing with the box count. A comparison of the two policies reveals a similar relationship as exhibited for the average number of actions necessary for tower completion.

5.2 Influence of Box Properties

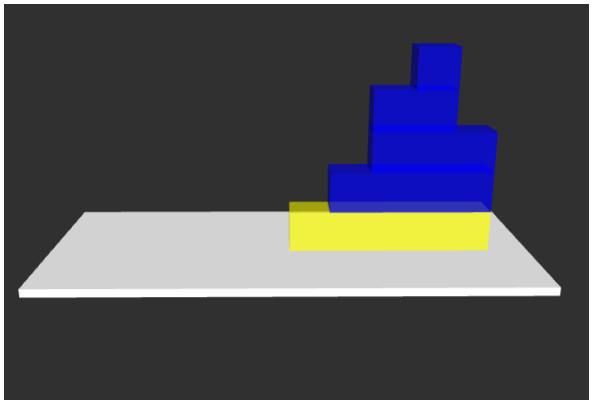
In this section, different properties of the boxes and the tower and their influence on the resulting policies are explored. Properties include box sizes, tower foundations, the number of possible positions of the hidden weights and the number of unknown weight positions.

5.2.1 Tower Foundation

In this experiment, the set of boxes consists of n boxes of sizes 1 to n, as depicted in Figure 5.6a. The number of boxes is varied between 2 and 6 boxes and for each of these numbers, 10 policies are created by the PGI algorithm. The largest box always serves as the foundation. Therefore, the remaining boxes can be stacked in reverse order of their sizes (see Figure 5.6b. As each box is thereby placed on top of a bigger one, the positions of the hidden weights are irrelevant in this case. This characteristic of the problem is reflected in the policy determined by the planning algorithm: the boxes can directly be stacked and there is no uncertainty about observations after each action. Also, there is no need for additional exploratory actions. Therefore, the number of actions is equivalent to $n-1$, as can be seen in Figure 5.7.

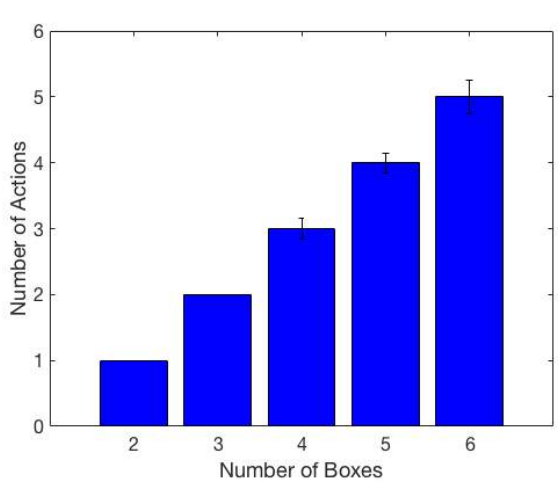


(a) The yellow box of size n serves as the tower foundation. The blue boxes consist of sizes from 1 to $(n-1)$.

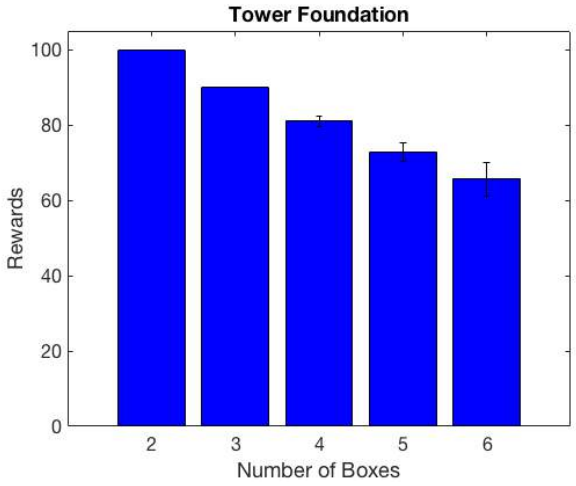


(b) For the resulting tower structure, the ordering of the boxes by their corresponding size is apparent.

Figure 5.6.: In this setting, n boxes of increasing sizes are stacked. The largest box serves as the foundation.



(a) The number of necessary actions corresponds to the count of boxes outside the foundation.

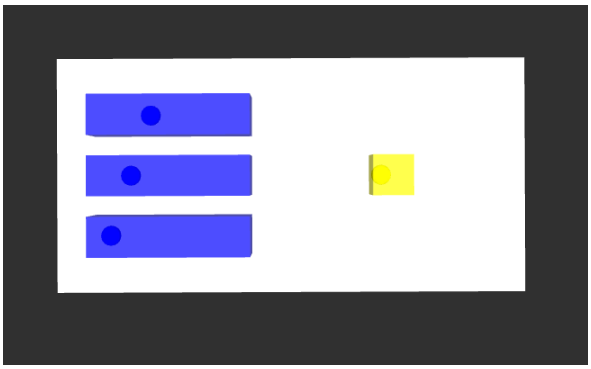


(b) The rewards directly reflect the number of performed actions and therefore decrease with the box count.

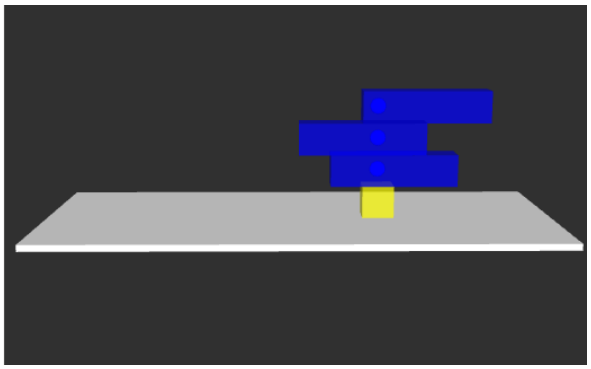
Figure 5.7.: In this setting, no exploration actions are performed. Instead, a stable solution is arrived at by immediately stacking the boxes in a decreasing order of their corresponding size.

5.2.2 Number of Hidden Weight Positions

In this simulation, 3 blue boxes are stacked on a yellow box of size 1. The size of the blue boxes as well as their numbers of possible positions for the hidden weights is varied between 1 and 5. In Figure 5.8a, the setting with blue boxes of size 5 is illustrated. As shown in Figure 5.9, when the blue boxes have size 1, no additional actions are necessary in order to explore weight positions. However, with increasing box size and an increasing number of possible weight positions, the amount of actions necessary to complete the tower is growing. For each box, multiple actions to explore the exact weight position become necessary. Note, that the observations do not reflect, in which direction a box is tilting when put in the wrong position. Therefore, when a box has 5 possible weight positions, up to 5 actions are necessary to determine that position.

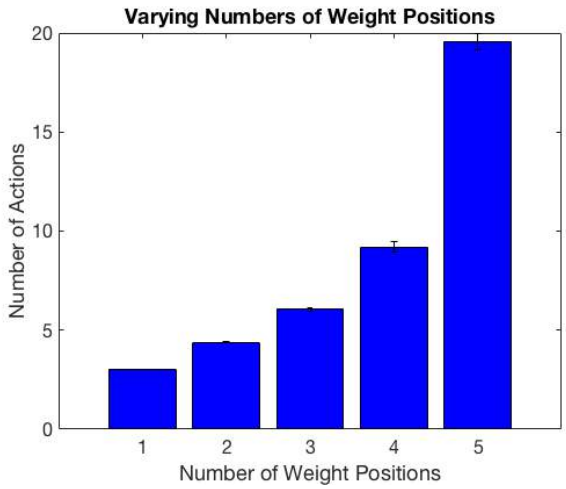


(a) The yellow box serves as a tower foundation. The hidden weights in the remaining 3 blue boxes have a varying number of possible positions.

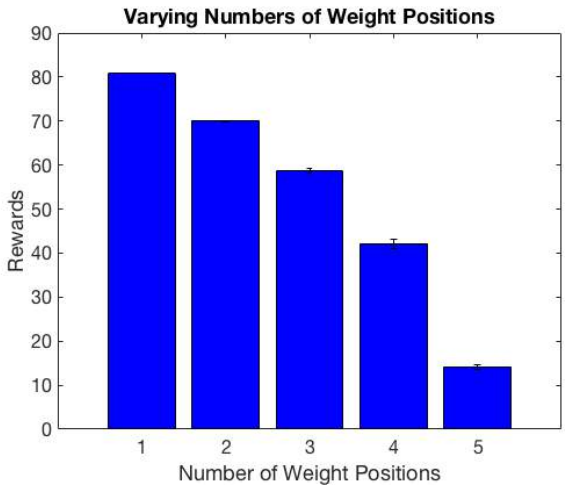


(b) With an increasing number of possible weight positions, balancing the blue boxes on top of the yellow box becomes more complex.

Figure 5.8.: For this test setting, the number of possible weight positions in 3 boxes is varied.



(a) The number of actions increases rapidly with the number of weight positions.



(b) Conversely, the expected rewards quickly vanish for a growing number of weight configurations.

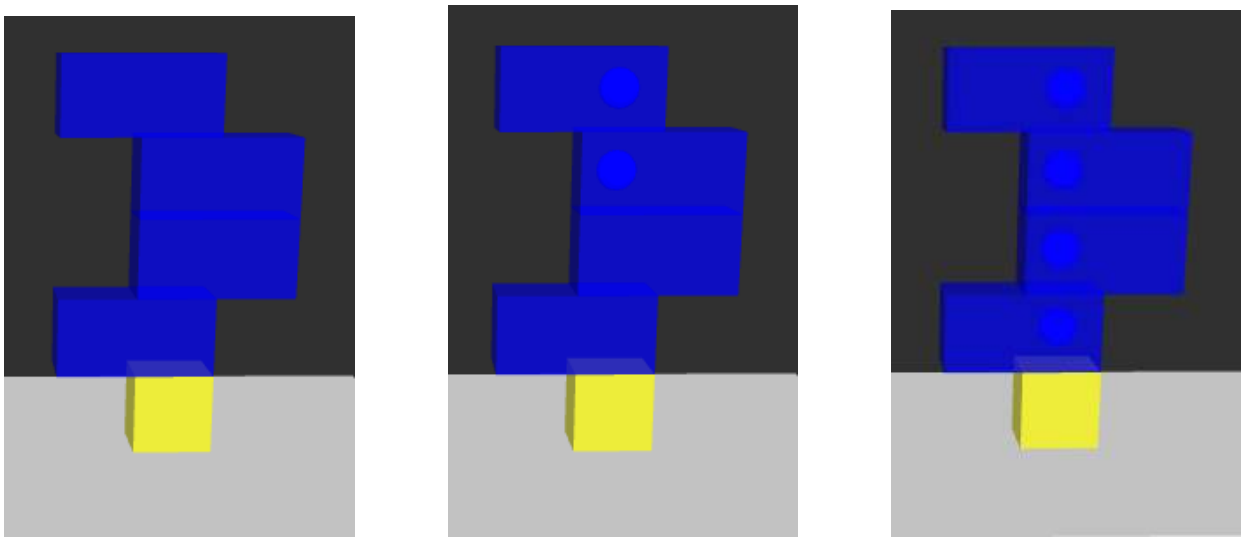
Figure 5.9.: Inflating the number of possible weight positions increases the planning complexity significantly.

5.2.3 Number of Unknown Objects

As described in section 3.4.1, knowledge obtained during a trial is preserved for future use. Therefore, the positions of hidden weights might already be known for some boxes. In this experiment, we explore how the number of unknown boxes affects the planning results. As depicted in Figure 5.10, the boxes consist of one foundation box of size 1 and 4 boxes of size 2 with 2 possible weight positions. The number of known weight positions is varied between 0 and 5. Figures 5.11a and 5.11b show that it does not make a difference whether 0 or 1 weight positions are known. This

can be explained by the fact that the process for selecting known boxes started with box id 1, which corresponds to the foundation box. As this box is never moved, the increase of information does not lead to any planning advantage. However, weight position information about the other boxes proves to be valuable, as the average number of actions decreases. Yet, it can be observed that there is no difference between a setting with 1 unknown box and a setting with no unknown boxes. This is due to the specific setting of boxes: All but one blue boxes are stacked such that their weights are directly on top of the yellow box, as depicted in 5.10c. Then, the last box can be placed in the same x-position as the second-to-last box. Its weight does not have enough effect on the tower's center of mass to render it unstable. Therefore, the last box's weight position does not matter to the planning algorithm.

Another interesting finding is that the number of iterations until the planning algorithm converges and thus the planning time decreases significantly with a decreasing number of unknown boxes, as can be seen in Figure 5.11c. This fact is important for our application, as it can be assumed that in a typical household task most objects are already known from previous trials. Preserving knowledge about these objects results in a significant planning performance increase.



(a) Weight positions for all 4 boxes not serving as a foundation are unknown.

(b) Weight information on 2 boxes is available while 2 remain unknown.

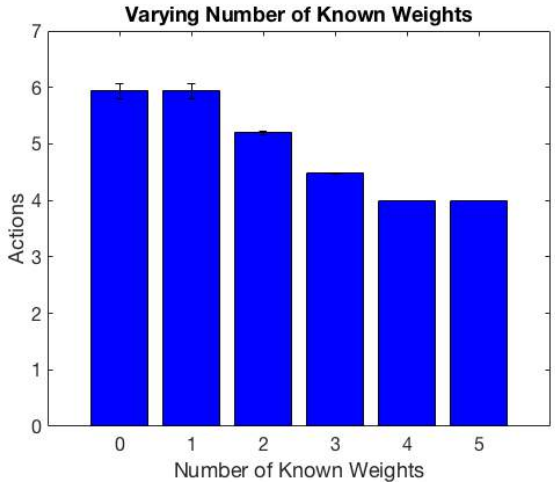
(c) Full information on the weights is available as no weight position is unknown.

Figure 5.10.: For this test setting, the number of boxes with unknown weight positions is varied.

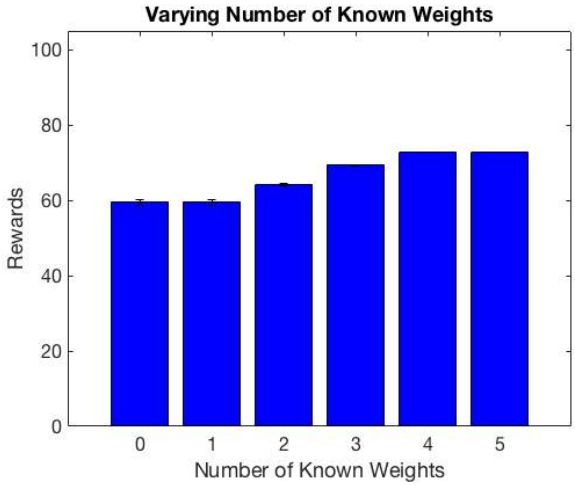
5.3 Additional Terminal States and Oracle Actions

In this section, additional terminal states are introduced. As described in section 4.2.2, we assume that boxes are fragile and therefore break, if they fall down from a certain height and the policy reaches a terminal state. In this series of experiments, we have a fixed number of 5 boxes with the same properties as the ones in section 5.1 (2 weight positions, size 2). The height limit, as depicted in Figure 5.12, is changed throughout the experiments. Figure 5.13 shows the simulation results height limits between 0 and 4. A limit of 0 means that as soon as one box falls, the trial is over. Therefore, there is no possibility for exploratory actions and stacking success is completely random. This fact also explains why the average number of actions for limit 0 (see Figure 5.13b is so low. As no exploration is possible, the tower is either directly stacked or the agent fail during the attempt. Therefore, a success is only possible with the minimum number of actions. Starting from height limit 1, it is possible to solve the task by first testing all boxes below the height limit. As can be seen in A.2, boxes that are already in the tower might have to be removed again in order to safely test other boxes. With an increasing height limit, success becomes more likely and less additional exploration actions are necessary.

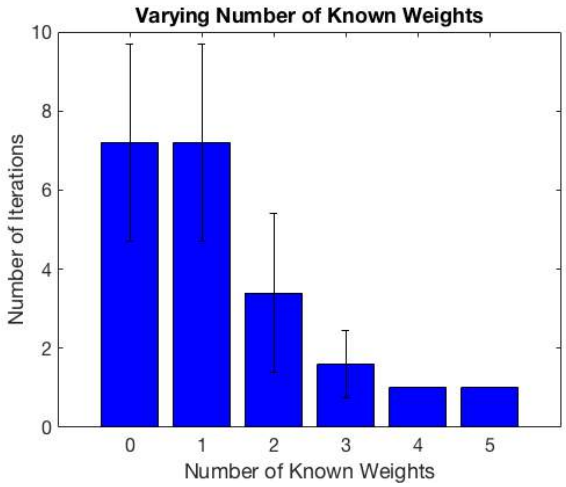
In order to support the planning agent in this potentially dangerous situation, oracle actions as defined in 4.2.3 are introduced. With one oracle question, the agent can ask a human user about the weight position in one box. The answers are assumed to be correct. In the following part, we compare simulation results for an agent with 0 oracle questions and one with 5 oracle questions at its disposal per trial. As can be seen in 5.14, questions are only asked when necessary: When the falling height is so low (i.e. the maximum height is 1) that each box weight has to be determined before the actual stacking can start, the oracle question is beneficial, as it costs only one action. However, testing one box costs on average 1.5 actions: When a box is put on the tower and stays on top, an additional action is necessary to move it back



(a) The number of actions decreases with a rising quantity of known weight positions.



(b) The expected rewards exhibit an increase with the number of known positions.



(c) Algorithm iterations reduce significantly with a decreasing number of unknown boxes.

Figure 5.11.: A small number of unknown objects leads to a reduced number of actions as well as higher expected rewards and faster planning.

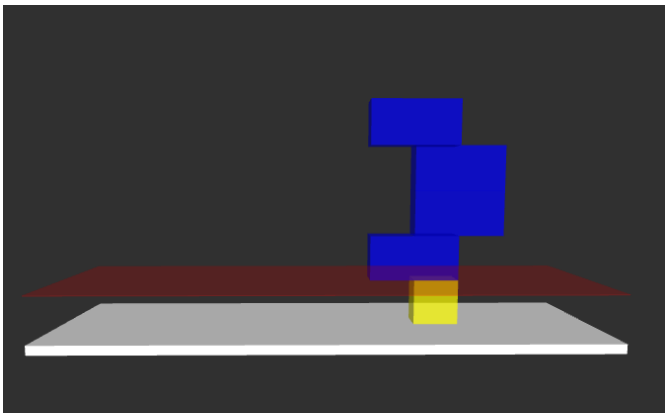
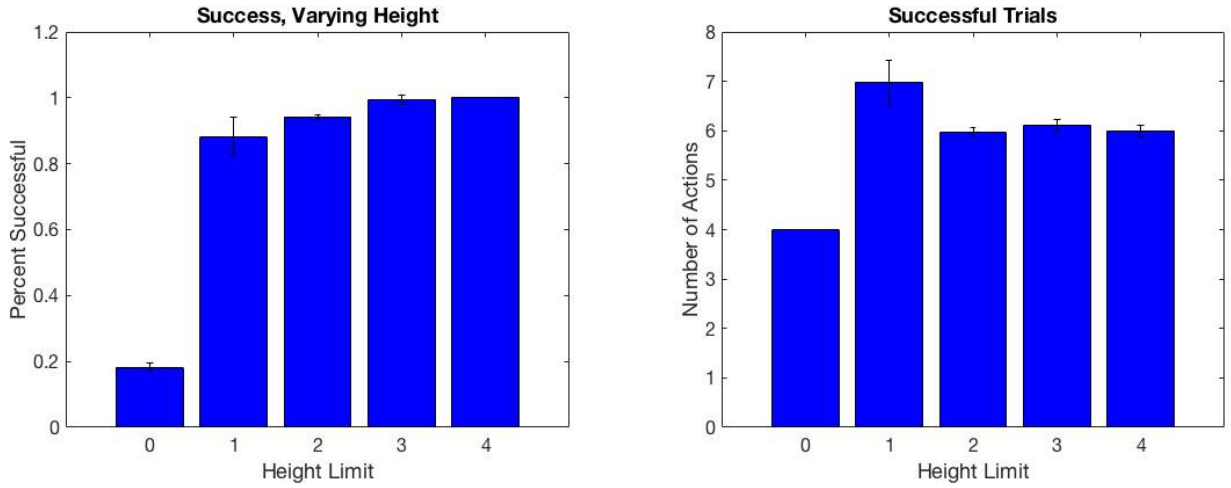


Figure 5.12.: The height limit is set to 1, i.e. boxes are at most allowed to fall from a height immediately above the foundation.



(a) Successful trials become significantly more probable with an increased height limit.

(b) Here, only successful trials are considered for the count of necessary actions.

Figure 5.13.: A low height limit renders successful planning very difficult.

to the storage area, because the space is needed to test the next box below maximum falling height. Asking the oracle is therefore on average faster. In other cases, if the falling height limit is higher, boxes do not have to be removed if they stay on top after an action. Then, asking the oracle is more expensive, because it does not potentially result in an additional box on top of the tower. In this setting, the oracle action would be selected only when the tower is already high and there is a lot of uncertainty about all remaining boxes. In this case, the risk of destroying the whole tower outweighs the cost of one additional action for an oracle query.

Even though the user may choose to answer as many questions as he likes, it is important for task performance that only necessary questions are being asked. As depicted in Figure 5.15, this is the case. Even with an increased supply of questions, only a small, beneficial number is used. The planning algorithm balances well between information gain and task progress. In addition, the planning complexity does only slightly increase when questions are offered at all and does not further grow with the number of questions. An example for a policy selecting an oracle question can be found in A.3.

5.4 Constraints

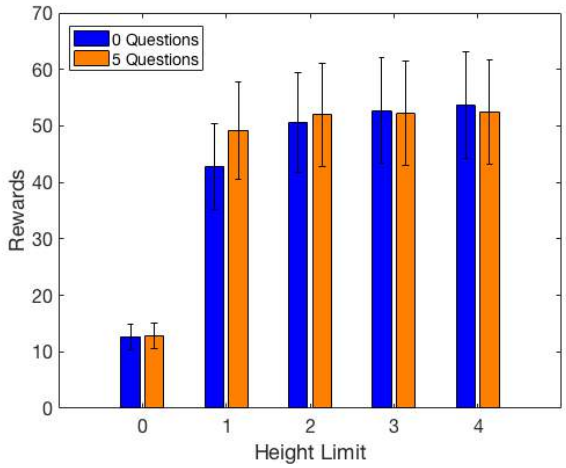
In this section, we introduce constraints and therefore apply the adapted PGI algorithm we first compare its performance to the original algorithm and then evaluate it on two different types of constraints.

5.4.1 Constraints in Comparison to Rewards

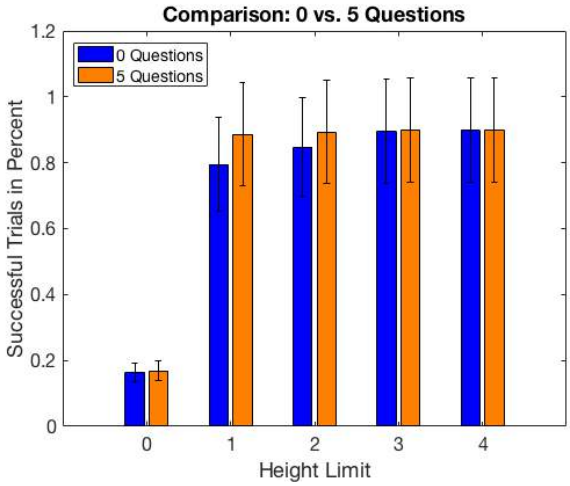
In this experiment, we compare constrained PGI algorithm to the original PGI algorithm. Therefore, we apply both algorithms to the basic box problem introduced in section 5.1 . As can be seen in Figure 5.16, the number of actions for the adapted algorithm is higher than for the original one. That is due to the fact that there is no discount factor in the adapted algorithm. Therefore, it has no incentive to terminate as fast as possible. Instead, the tower only has to be completed before the maximum number of actions is reached. In this experiment, this maximum was set to $(n-1)*2$, the maximum number of actions in an optimal policy. The resulting number of actions is slightly less than this maximum for all numbers of boxes. That means that in some cases, the policy reached the goal state early by chance. The constrained algorithm also leads to faster convergence for all numbers of boxes, as depicted in 5.16b.

5.4.2 Color Constraints

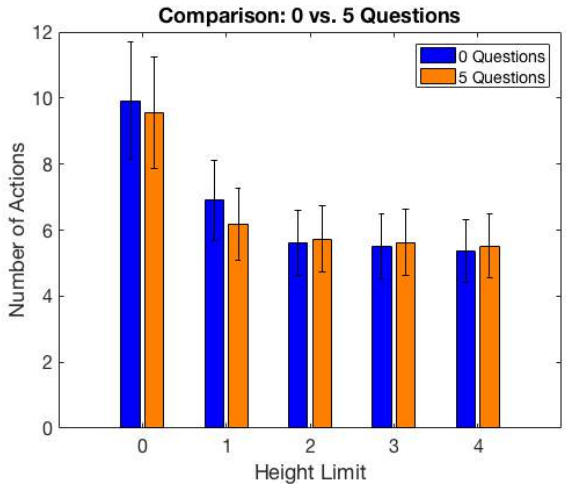
In this example we include constraints into the basic box problem: it has to hold at all times that the box at height 1 is not yellow. It is studied how a changing number of yellow boxes as depicted in 5.18b affects the planning results. As can be seen in 5.18, the constraints are always met for 0 - 4 yellow boxes. However, when all boxes are yellow, it is impossible



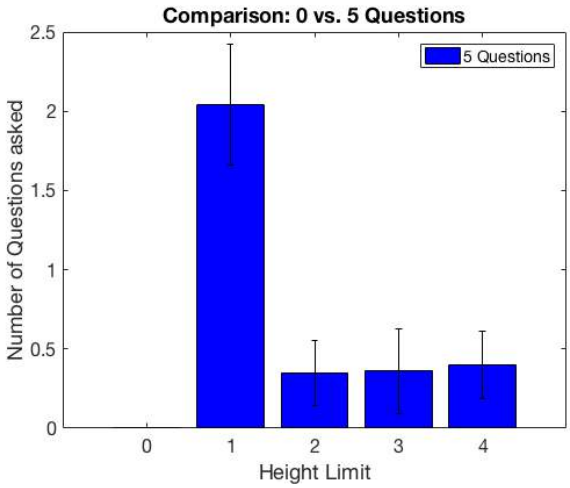
(a) The availability of oracle questions leads to an initial significant increase of rewards in face of uncertain situations before convergence.



(b) Oracle questions can be in general considered as beneficial, a larger effect can be observed in situations with increased uncertainty such as for a low permitted falling height.

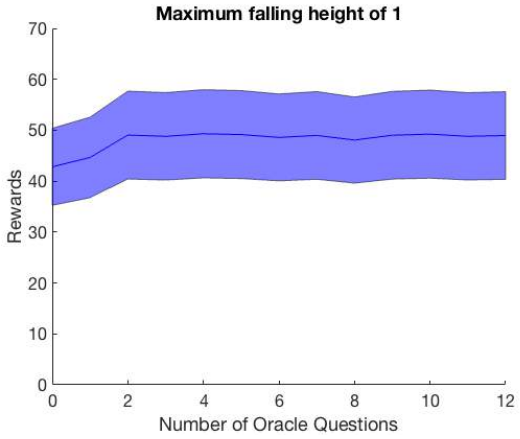


(c) A significant decrease in the number of necessary actions when employing oracle questions is only exhibited for uncertain situations. However, the resulting increase in effort for more certain configuration can be considered negligible.

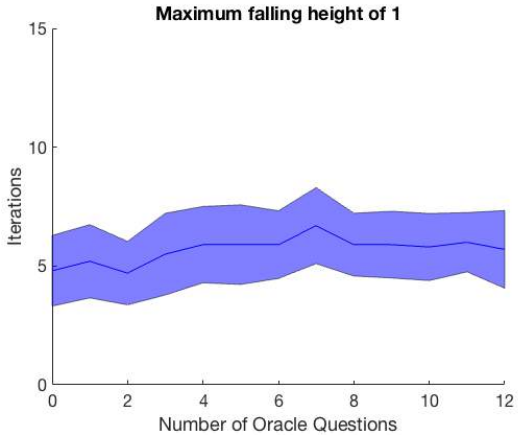


(d) For a falling height of 1, the oracle action is always utilized. For other height configurations, balancing between oracle action and tower collapse occurs.

Figure 5.14.: Planning results are compared for 0 and 5 oracle questions as well as for different height limits.

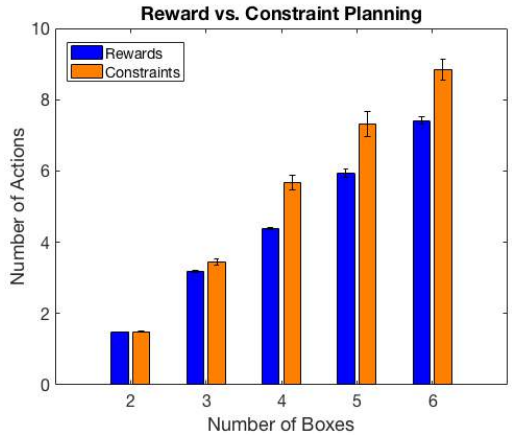


(a) For a maximum falling height of 1, no more than 2 oracle questions are actually employed.

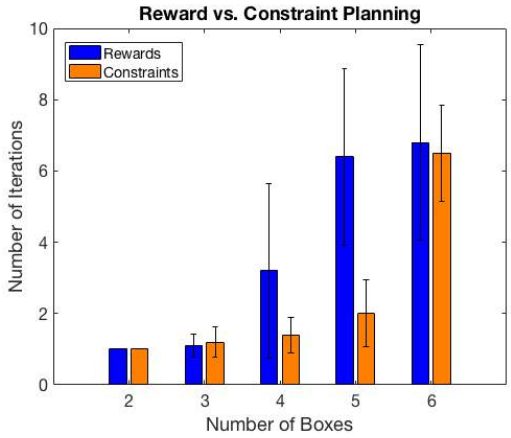


(b) Required planning iterations until convergence merely rise with an increase in oracle questions.

Figure 5.15.: Allowing too many user inquiries does not exhibit a negative influence on the planning performance.



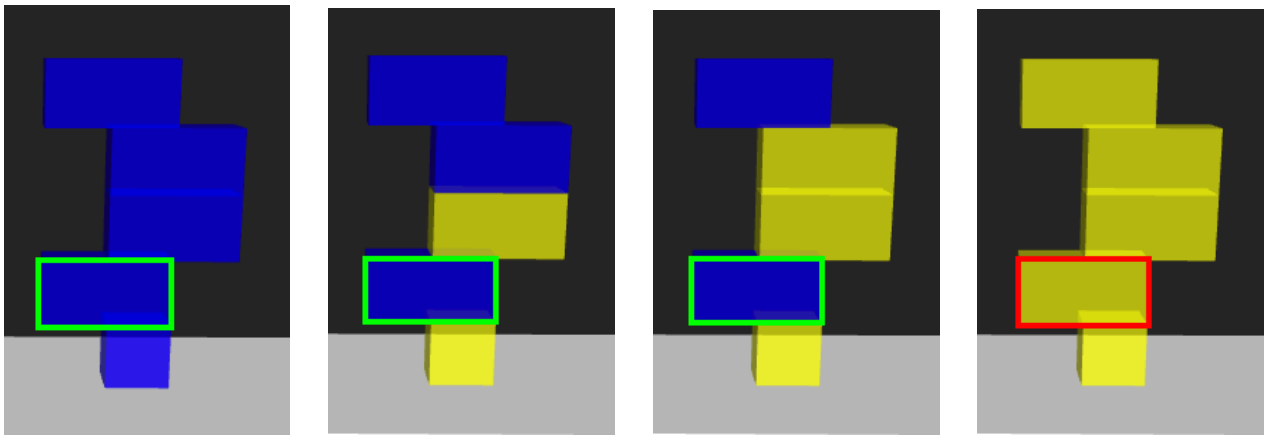
(a) The number of actions is slightly higher for the constraint algorithm.



(b) The constraint algorithm converges faster.

Figure 5.16.: The original PGI algorithm and its adaptation to constrained POMDPs are compared.

to meet the constraints. Therefore, the algorithm applies its fallback reward evaluation. This fallback is successful, as can be seen in 5.18. The number of actions is reduced due to the fact that reward planning tries to limit the number of actions, whereas constraint planning does not, as already explained in section 5.4.1.

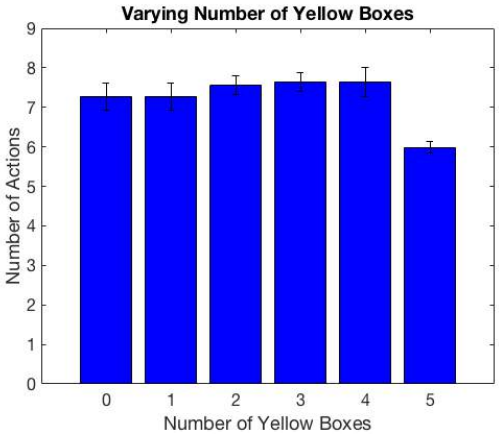


(a) No yellow boxes **(b)** Two yellow boxes **(c)** Three yellow boxes **(d)** Five yellow boxes

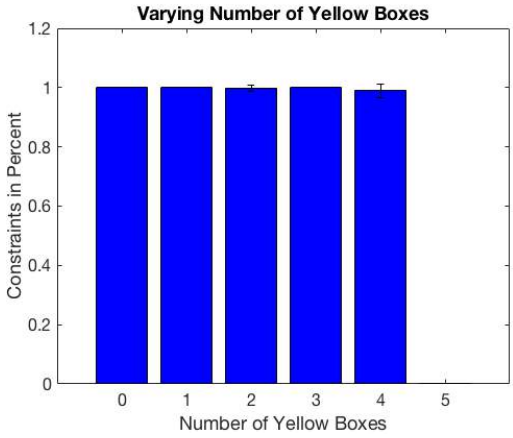
Figure 5.17.: The number of yellow boxes is varied for each trial in an increasing fashion. The green shape illustrates the constraint to be met. For five yellow boxes, a solution fulfilling the constraint is rendered impossible.

5.4.3 Terminal Constraints

In this example we include terminal constraints into the basic box problem: a certain number of boxes is assigned fixed positions for the terminal state. In Figure 5.19, different colors symbolize these unique positions. The number of these boxes is varied between 0 and 5. As can be seen in Figure 5.20, the constraint are only always met for 0 or 1 fixed positions. While the percentage of fulfilled constraints is decreasing with the number of fixed positions, the number of actions necessary to reach a terminal state is increasing, because early successful exploratory actions sometimes have to be reversed in order to maintain the imposed order of boxes. One problem that leads to the constraints not being met, is that the algorithm alternates between constraint evaluation and reward evaluation, as can be seen over several algorithm iterations in 5.20d. This issue should be resolved by fixing one objective for a whole iteration or even several iterations.

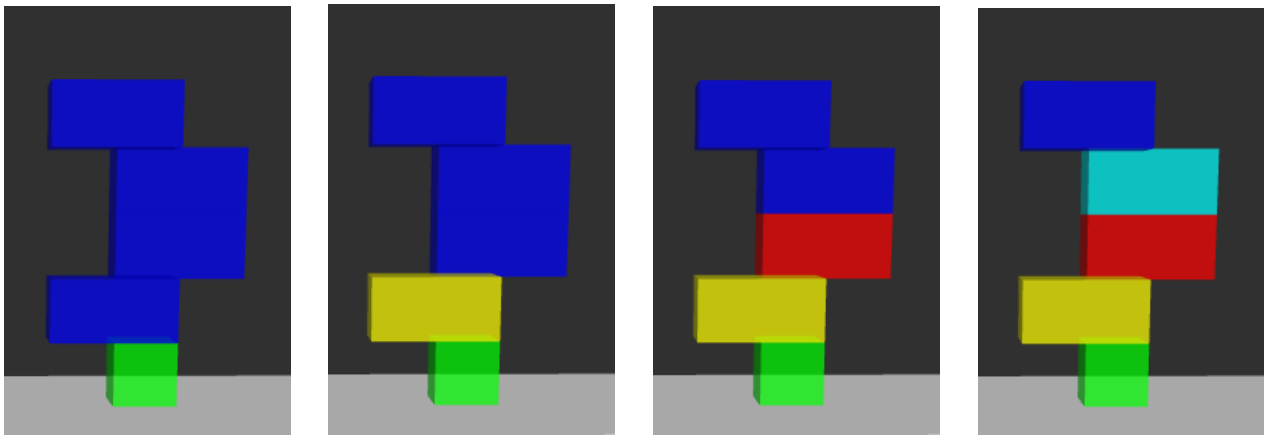


(a) The necessary number of actions slightly increases as more configurations are not permitted by the constraint.



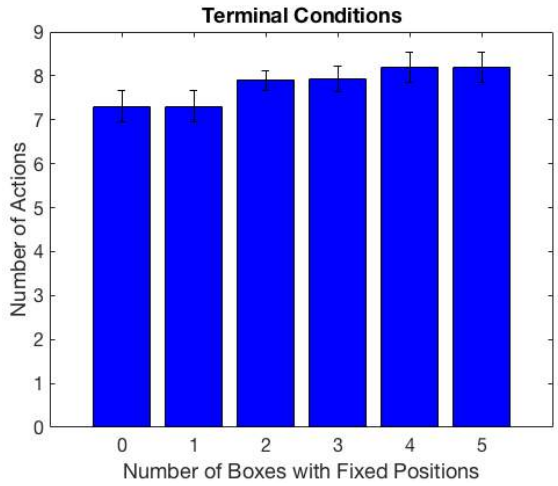
(b) The constraint is always met if a solution exists.

Figure 5.18.: The contingency fallback on reward planning is a suitable approach as illustrated for the color constraint with a varying number of yellow boxes.

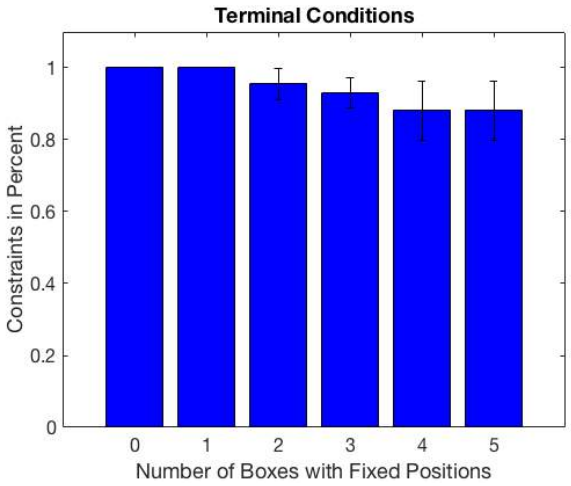


(a) A single position is assigned. **(b)** Two positions are assigned. **(c)** Three positions are assigned. **(d)** All positions are assigned.

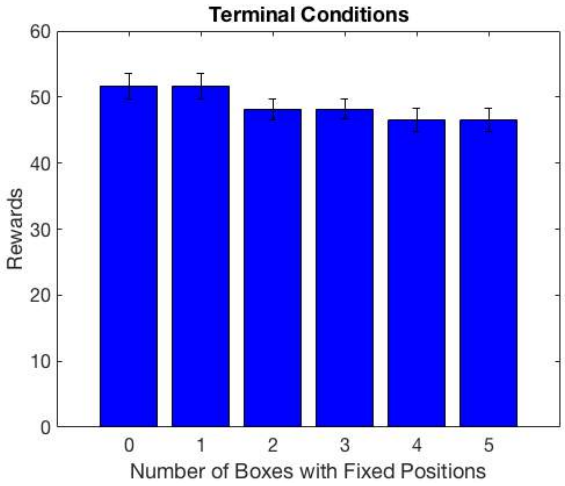
Figure 5.19.: The different colors illustrate the requirement of fixed positions for certain boxes. The resulting number of constraints increases as more boxes are assigned to a particular height.



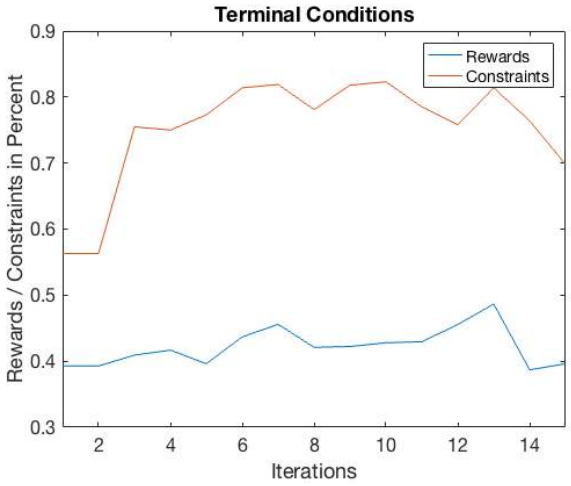
(a) The number of necessary actions increases when constraints are added via the number of fixed positions.



(b) Fulfilling all constraints becomes increasingly demanding as less configurations are permitted.



(c) Expected rewards decrease as more constraints are required to be fulfilled.



(d) A particular example involving 15 algorithm iterations exhibits the failure of the contingency fallback in case of terminal constraints. Here, neither fulfillment of constraints nor gathering of rewards are improved after the second iterations as the algorithm begins to alternate between the two objectives.

Figure 5.20.: The test setting employing terminal constraints exhibits mixed results.

5.5 Robot Experiments

In this section, we describe experiments performed not only in simulation, but also on a physical robot, as described in section 4.3.2. Therefore, we combine the previously introduced features of the box stacking problem: The boxes consist of one box of size 1 two boxes of size 2 and one box of size 3, all with 2 possible weight positions (see Figure 5.21. Each box has a unique color. A constraint on the blue box's position in the tower is introduced. In each of the four performed experiments, the blue box has to be positioned at a different fixed height, as can be seen in Figure 5.22. In addition, there is a height limit of 1 for all boxes and 1 oracle question can be asked in each trial.

Each of the resulting sample policies is performed 10 times with the robot. Note, that worst-case policies were selected, such that the robot had to perform a maximum number of actions. A detailed example for such a policy can be seen in Figure 5.22: After the first action with the red box has been successful, the box has to be removed again because the space below the height limit is needed for additional exploratory actions. The third action is an oracle action and therefore, no box movement takes place. Instead, a user reveals the weight position in the green box. After the yellow box has fallen in action 4, it is returned to the storage area and successfully positioned on the tower in the fifth action, as its weight position is now known. Now all necessary information is available to finish the tower.

In Figure 5.22 is depicted how many trials of each policy were successfully performed. It can be seen that balancing other boxes on top of the small blue box leads to less success. That is due to the fact that the paper boxes are not very stable and are toppling easily. Another issue is the gravity simulation. The assumption that all weight is in the center of mass oversimplifies the problem, as paper also has a certain weight. This leads to situations in which the simulation forecasts a falling box, but the box stays on top of the tower. A third problem is the occasional confusion of boxes by the tracking system, especially when the robot arm blocks the cameras' view of the boxes.

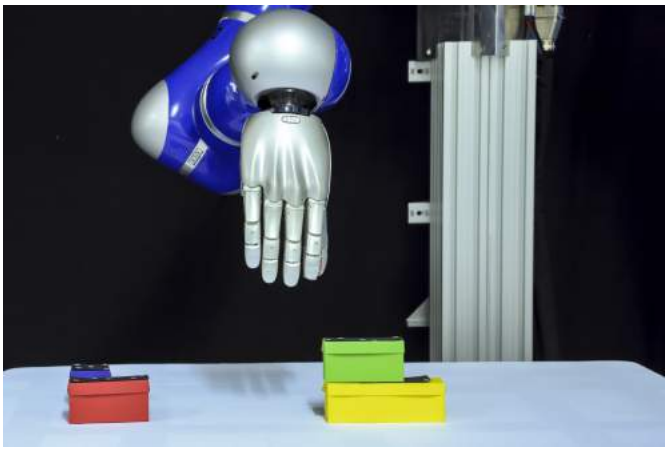


Figure 5.21.: In this setup, the robot arm is stacking 4 paper boxes of different sizes.

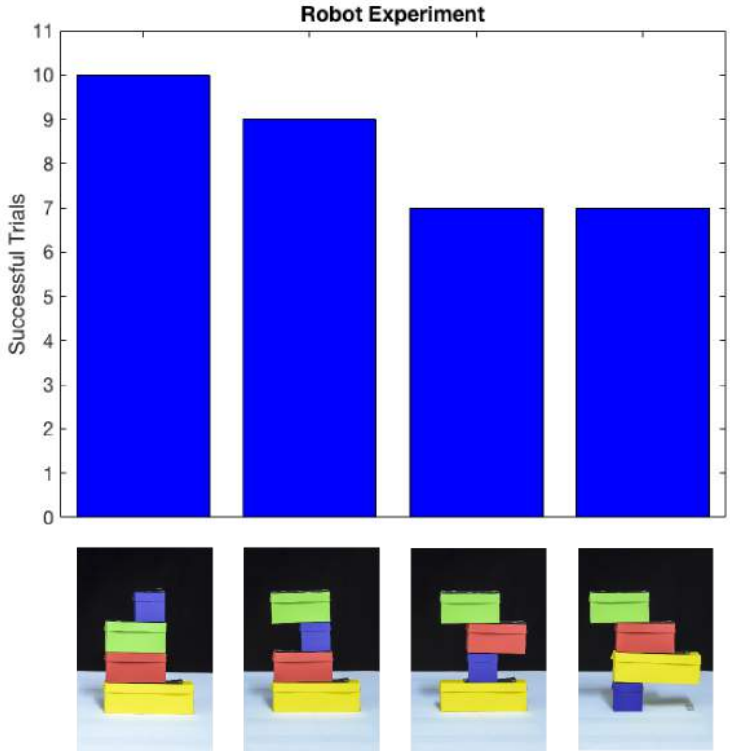


Figure 5.22.: The smallest box (blue) is required to be at a certain varying height level in the completed tower structure. The share of successful trials is reduced as balancing an increasing number of boxes on top of the smallest one becomes more and more demanding.

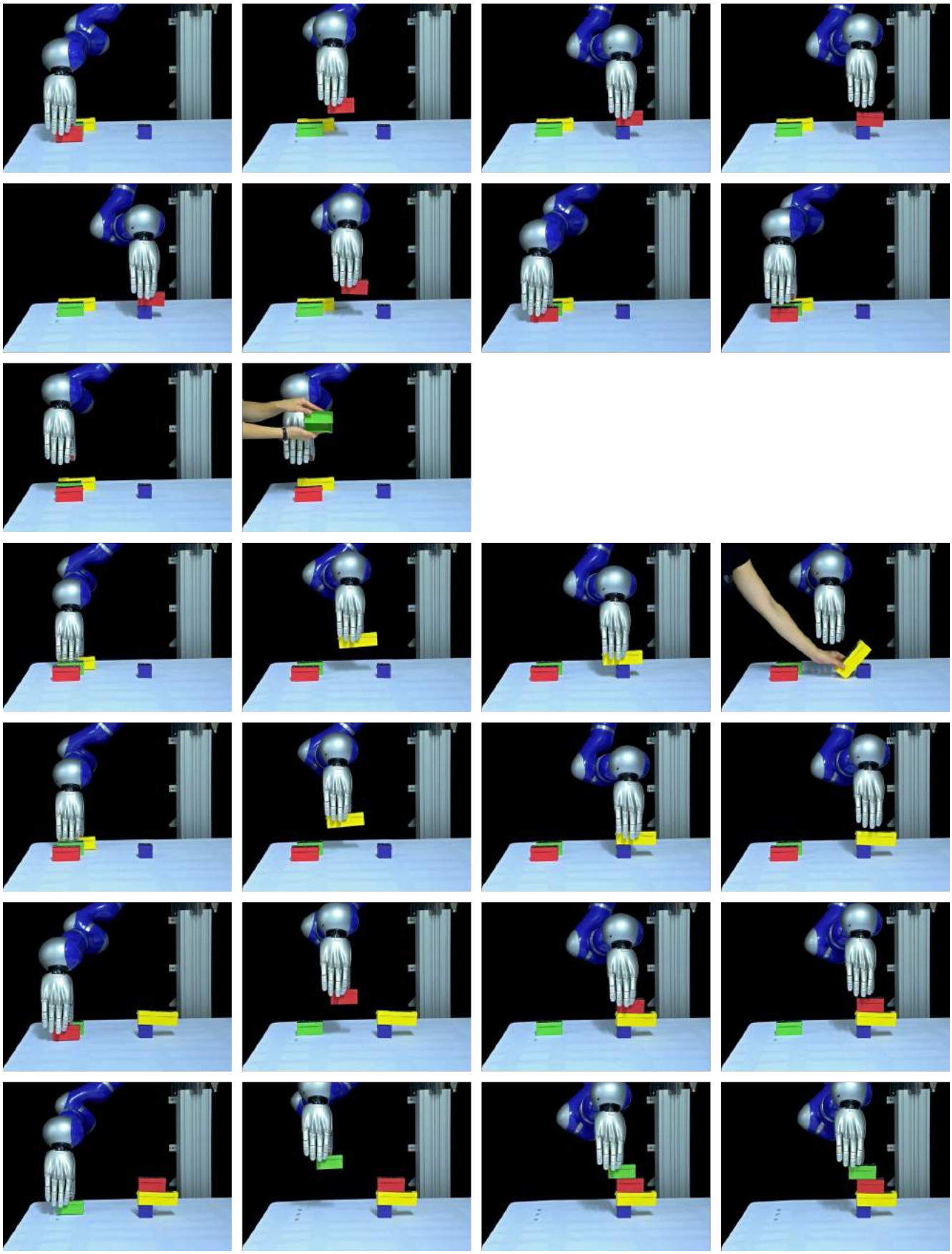


Figure 5.23.: This sequence of images illustrates a complete trial run in order to stack four paper boxes. Each row of photographs corresponds to a single action. Note that this is a worst-case trial due to the a priori unknown hidden weights, i.e. the maximum necessary number of actions has to be performed .

6 Conclusion and Future Work

In this study, we introduce an advanced model for planning robot manipulation tasks under uncertainty in a human environment. The developed system allows for interaction between a robot and humans present in its environment. On the one hand, the planning agent is enabled to request human help in case of insufficient information about the current world state, including situations associated with a high risk by the agent. On the other hand, human users are provided with the opportunity to express preferences about the manipulation task's outcome.

The suggested POMDP model can be further employed as a template for a wide range of robot applications: It can be utilized to model any task that can be defined on a similar set of objects with adapted properties. Significant and expensive prior analysis to tune the model is not necessary. The task policy is determined prior to execution by applying the PGI solving algorithm to the POMDP. This algorithm produces a policy graph storing a specific task policy in a very compact arrangement. Therefore, the system becomes very reactive during task execution rendering it highly convenient for real-world applications. In order to demonstrate the potential of our POMDP model, it has been exemplary adapted to, and employed for the task of a robot's arm constructing a tower consisting of paper boxes. In order to equip the problem with increased uncertainty, the centers of mass have been shifted for each box via hidden weights. The simulation results confirm that the planning process works well for up to six boxes of different shapes while being completed in a reasonable amount of time. Repeated executions of the (stochastic) planning task result in almost identical policies admitting a low standard error. Thus the algorithm performs reliable and can be easily be modified for application to real-world challenges. Planning time is significantly decreased if prior information about the boxes is available. In this case, the number of objects can be increased while maintaining the computational load. This is a particularly important feature as only few objects in a household task are usually unknown a priori. Yet, the number of objects remains limited within the initial analysis of this study. By nature, a significant increase of handled object is desired and shall be the subject of future work. Based on the presented study, we consider our approach well suited for such an expansion. As discussed, a higher number of objects is achievable by exploiting domain knowledge and thus further decreasing the state and action space. Another suggestion for efficiency improvement in the future could utilize a combination with hierarchical planning: As we perform offline computation prior to task execution, the system could, for instance, be extended to prepare a variety of readily available policies adjusting for a differing numbers of newly added objects. The presented results involving experimental evaluation demonstrate that the model is immediately transferable from a simulation environment to a physical robot. In our test setting, we were able to solve the stacking task for up to four boxes in a variety of different configurations. This number has been mainly limited by the structural instability of the paper boxes. Other observed causes for failure include an oversimplification of the physics model and the occasional confusion of boxes by the tracking system. The presented robotic example is intended as a proof of concept. Hence, boxes are moved back manually after falling down. While these limitations should certainly be be eliminated for a real application and would require rather complex improvements, they do not constitute a loss of generality in the presented results.

In addition, we have proposed methods enabling human involvement: On the one hand, we have introduced an additional action for the agent, allowing an information request about the current world state by inquiring a human. As our experimental results exhibit, the incorporation of this action increases the expected reward significantly for certain situations. While the user may select the maximum number of questions he is willing to answer (in each trial), the agent does not necessarily exploit all permitted questions. Experiments reveal that the oracle action is merely selected for states with associated high uncertainty. Here, obtaining information would require many non-oracle actions, and therefore, would be expensive. Yet, in situations easily solvable without additional costs, the oracle is not queried at all. In particular, it has been observed that planning does not require significantly more effort, even though the size of the action space is increased. In summary, the concept of oracle action has been established as a suitable technique to incorporate external help if advantageous while minimally increasing user effort and maintaining planning time.

In a modified perspective, communication and information exchange are reversed when users are empowered to specify their preferred outcome of a task. In this advancement, we have enabled the Policy Graph Improvement algorithm to handle constrained POMDPs: Instead of optimizing expected rewards, the algorithm seeks to satisfy multiple objectives expressed in the form of constraints. As a restricted objective inherently admits the risk of an unsolvable problem, contingency has been incorporated by allowing the algorithm to revert to rewards. Experiments have exhibited the performance of the constrained PGI algorithm to at least remain equivalent to the unconstrained version. While this

efficiency has been demonstrated for continuously valid constraints, challenges in face of terminal restrictions have been exposed. Here, the algorithm tends to revert to rewards in an accelerated way and thus begins to alternate between the two potentially conflicting objectives. In a future improvement, the algorithm should be forced to abstain from rewards until no other solution can be found. In addition, it has been observed, that the number of actions in the resulting policies tends to be higher than in the unconstrained case. The reason for this shortcoming emanates from the lack of incentives for early termination (before the maximum number of iterations is reached). Thus, the maximum has to be select with great care. It is suggested to equip prospective improvements of the presented method with a discounted constraint function evaluation penalizing late completion.

For both employed concepts of communication, emphasis has been placed on convenience for non-expert user operation: The agent's inquiries have been limited to information easily obtainable by the user without requiring any prior knowledge on POMDPs. Furthermore, the user can designate his level of involvement into the planning process, from refusal of any to expressing his preferences about a task's outcome. All communication has been enabled by the provision of modular graphical user interfaces that allow for quick adaptation to different manipulation tasks. By nature, future user studies with non-experts are desired and encouraged to further evaluate our approach.

Bibliography

[1] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 1995.

[2] J. Pineau and G. J. Gordon, "Pomdp planning for robust robot control," in *Robotics Research*, pp. 69–82, Springer, 2007.

[3] W. B. Arthur, "Inductive reasoning and bounded rationality," *The American economic review*, vol. 84, no. 2, pp. 406–411, 1994.

[4] H. A. Simon, "Invariants of human behavior," *Annual review of psychology*, vol. 41, no. 1, pp. 1–20, 1990.

[5] G. A. Klein, *Sources of power: How people make decisions*. MIT press, 1999.

[6] J. Pajarinen and V. Kyrki, "Robotic manipulation of multiple objects as a pomdp," *Artificial Intelligence*, 2015.

[7] R. A. Howard, *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[8] R. Bellman, "A markovian decision process," tech. rep., DTIC Document, 1957.

[9] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[11] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine learning*, vol. 13, no. 1, pp. 103–130, 1993.

[12] J. Pineau, G. Gordon, S. Thrun, *et al.*, "Point-based value iteration: An anytime algorithm for pomdps," in *IJCAI*, vol. 3, pp. 1025–1032, 2003.

[13] E. J. Sondik, "The optimal control of partially observable markov processes.," tech. rep., DTIC Document, 1971.

[14] G. E. Monahan, "State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[15] E. J. Sondik, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs," *Operations research*, vol. 26, no. 2, pp. 282–304, 1978.

[16] A. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact method for partially observable markov decision processes," in *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pp. 54–61, Morgan Kaufmann Publishers Inc., 1997.

[17] G. Shani, R. I. Brafman, and S. E. Shimony, "Forward search value iteration for pomdps.," in *IJCAI*, pp. 2619–2624, 2007.

[18] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 520–527, AUAI Press, 2004.

[19] M. T. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for pomdps," *Journal of artificial intelligence research*, vol. 24, pp. 195–220, 2005.

[20] J. K. Pajarinen and J. Peltonen, "Periodic finite state controllers for efficient pomdp and dec-pomdp planning," in *Advances in Neural Information Processing Systems*, pp. 2636–2644, 2011.

[21] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman, "Acting optimally in partially observable stochastic domains," in *AAAI*, vol. 94, pp. 1023–1028, 1994.

[22] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, "Monte carlo value iteration for continuous-state pomdps," in *Algorithmic foundations of robotics IX*, pp. 175–191, Springer, 2010.

[23] A. Undurti and J. P. How, "An online algorithm for constrained pomdps," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 3966–3973, IEEE, 2010.

[24] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[25] J. D. Isom, S. P. Meyn, and R. D. Braatz, "Piecewise linear dynamic programming for constrained pomdps.," in *AAAI*, pp. 291–296, 2008.

[26] D. Kim, J. Lee, K.-E. Kim, and P. Poupart, "Point-based value iteration for constrained pomdps," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[27] C. P. C. Chanel and F. Teichteil-Königsbuch, "Properly acting under partial observability with action feasibility constraints," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 145–161, Springer, 2013.

[28] J. D. Williams and S. Young, "Partially observable markov decision processes for spoken dialog systems," *Computer Speech & Language*, vol. 21, no. 2, pp. 393–422, 2007.

[29] Q. Zhao, L. Tong, A. Swami, and Y. Chen, "Decentralized cognitive mac for opportunistic spectrum access in ad hoc networks: A pomdp framework," *IEEE Journal on selected areas in communications*, vol. 25, no. 3, 2007.

[30] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[31] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems," in *AAAI/IAAI*, pp. 541–548, 1999.

[32] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.

[33] J. Luo, S. Zhang, X. Dong, and H. Yang, "Designing states, actions, and rewards for using pomdp in session search," in *European Conference on Information Retrieval*, pp. 526–537, Springer, 2015.

[34] S. Rosenthal, M. Veloso, and A. K. Dey, "Is someone in this office available to help me?," *Journal of Intelligent & Robotic Systems*, vol. 66, no. 1-2, pp. 205–221, 2012.

[35] T. Taha, J. V. Miró, and G. Dissanayake, "Pomdp-based long-term user intention prediction for wheelchair navigation," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3920–3925, IEEE, 2008.

[36] S. Rosenthal, M. M. Veloso, and A. K. Dey, "Learning accuracy and availability of humans who help mobile robots.," in *AAAI*, 2011.

[37] R. Jaulmes, J. Pineau, and D. Precup, "Active learning in partially observable markov decision processes," in *European Conference on Machine Learning*, pp. 601–608, Springer, 2005.

[38] C. Cai, X. Liao, and L. Carin, "Learning to explore and exploit in pomdps," in *Advances in Neural Information Processing Systems*, pp. 198–206, 2009.

[39] F. Doshi, J. Pineau, and N. Roy, "Reinforcement learning with limited reinforcement: Using bayes risk for active learning in pomdps," in *Proceedings of the 25th international conference on Machine learning*, pp. 256–263, ACM, 2008.

[40] S. Nicol and I. Chadès, "Which states matter? an application of an intelligent discretization method to solve a continuous pomdp in conservation biology," *PloS one*, vol. 7, no. 2, p. e28993, 2012.

[41] E. O. Selfridge, I. Arizmendi, P. A. Heeman, and J. D. Williams, "Integrating incremental speech recognition and pomdp-based dialogue systems," in *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 275–279, Association for Computational Linguistics, 2012.

[42] A. A. Irissappane, F. A. Oliehoek, and J. Zhang, "A pomdp based approach to optimally select sellers in electronic marketplaces," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pp. 1329–1336, International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[43] Y. Aviv and A. Pazgal, "A partially observed markov decision process for dynamic pricing," *Management Science*, vol. 51, no. 9, pp. 1400–1416, 2005.

[44] A. R. Cassandra, "A survey of pomdp applications," in *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, vol. 1724, 1998.

[45] P. Monsó, G. Alenyà, and C. Torras, "Pomdp approach to robotized clothes separation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1324–1329, IEEE, 2012.

[46] S. R. Schmidt-Rohr, S. Knoop, M. Lösch, and R. Dillmann, "Reasoning for a multi-modal service robot considering uncertainty in human-robot interaction," in *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pp. 249–254, ACM, 2008.

[47] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping pomdps," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 4685–4692, IEEE, 2007.

[48] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Aaai/iaai*, pp. 11–18, 1998.

[49] M. Shiomi, D. Sakamoto, T. Kanda, C. T. Ishi, H. Ishiguro, and N. Hagita, "A semi-autonomous communication robot - a field trial at a train station," in *Human-Robot Interaction (HRI), 2008 3rd ACM/IEEE International Conference on*, pp. 303–310, IEEE, 2008.

[50] M. E. Pollack, L. Brown, D. Colbry, C. Orosz, B. Peintner, S. Ramakrishnan, S. Engberg, J. T. Matthews, J. Dunbar-Jacob, C. E. McCarthy, *et al.*, "Pearl: A mobile robotic assistant for the elderly," in *AAAI workshop on automation as eldercare*, vol. 2002, pp. 85–91, 2002.

[51] S. Rosenthal and M. M. Veloso, "Mobile robot planning to seek help with spatially-situated tasks.," in *AAAI*, vol. 4, p. 1, 2012.

[52] N. Armstrong-Crews and M. Veloso, "Oracular partially observable markov decision processes: A very special case," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 2477–2482, IEEE, 2007.

[53] S. Rosenthal and M. Veloso, "Modeling humans as observation providers using pomdps," in *2011 RO-MAN*, pp. 53–58, IEEE, 2011.

[54] S. Rosenthal, M. M. Veloso, and A. K. Dey, "Task behavior and interaction planning for a mobile service robot that occasionally requires help.," in *Automated Action Planning for Autonomous Mobile Robots*, 2011.

[55] F. Doshi-Velez, J. Pineau, and N. Roy, "Reinforcement learning with limited reinforcement: Using bayes risk for active learning in pomdps," *Artificial Intelligence*, vol. 187, pp. 115–132, 2012.

[56] A. Atrash and J. Pineau, "A bayesian method for learning pomdp observation parameters for robot interaction management systems," in *The POMDP practitioners workshop*, 2010.

[57] N. Côté, M. Bouzid, and A.-I. Mouaddib, "Integrating human recommendations in the decision process of autonomous agents," in *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02*, pp. 195–200, IEEE Computer Society, 2013.

[58] A.-B. Karami, L. Jeanpierre, and A.-I. Mouaddib, "Partially observable markov decision process for managing robot collaboration with human," in *Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on*, pp. 518–521, IEEE, 2009.

[59] A. Fern, S. Natarajan, K. Judah, and P. Tadepalli, "A decision-theoretic model of assistance.," in *IJCAI*, pp. 1879–1884, 2007.

[60] A. Atrash and J. Pineau, "A bayesian reinforcement learning approach for customizing human-robot interfaces," in *Proceedings of the 14th international conference on Intelligent user interfaces*, pp. 355–360, ACM, 2009.

[61] H. A. Simon, *Models of man; social and rational.* Wiley, 1957.

[62] L. Trilla and G. Alenya, "Planning stacking operations with an unknown number of objects.," in *ICINCO (2)*, pp. 348–353, 2010.

[63] Z. Sui, O. C. Jenkins, and K. Desingh, "Axiomatic particle filtering for goal-directed robotic manipulation," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 4429–4436, IEEE, 2015.

[64] T. Winograd, "Understanding natural language," *Cognitive psychology*, vol. 3, no. 1, pp. 1–191, 1972.

[65] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[66] W. Demtröder, "Experimentalphysik 1," *Mechanik und Wärme*, vol. 7, 2006.

[67] H. Liu, K. Wu, P. Meusel, N. Seitz, G. Hirzinger, M. Jin, Y. Liu, S. Fan, T. Lan, and Z. Chen, "Multisensory five-finger dexterous hand: The dlr/hit hand ii," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3692–3697, IEEE, 2008.

A Appendix

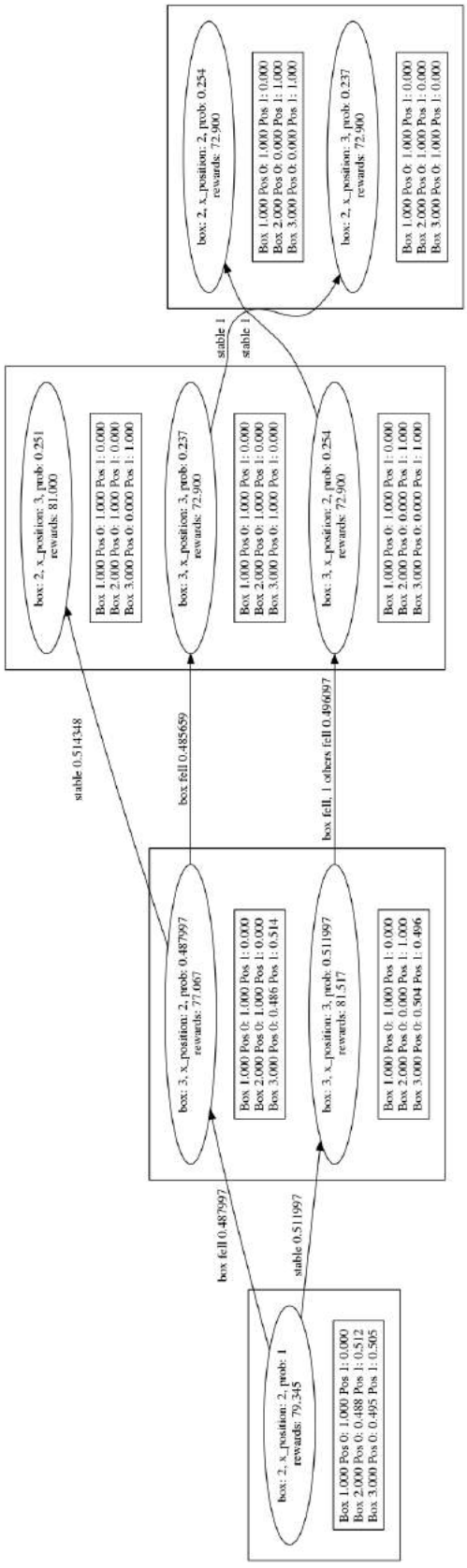


Figure A.1.: In the basic box problem, three boxes are stacked. Each box has two possible weight positions.

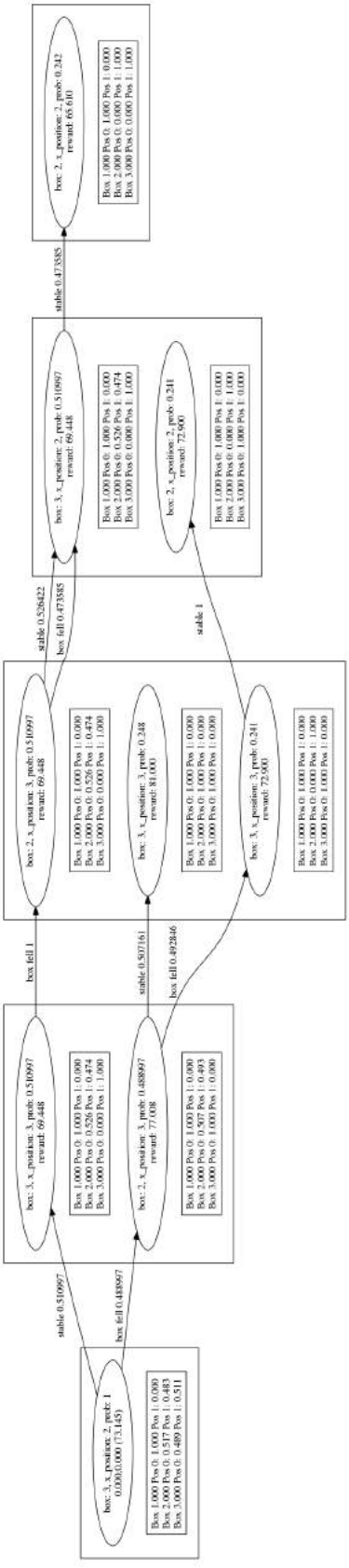


Figure A.2.: In this setting a height limit of 1 is introduced.

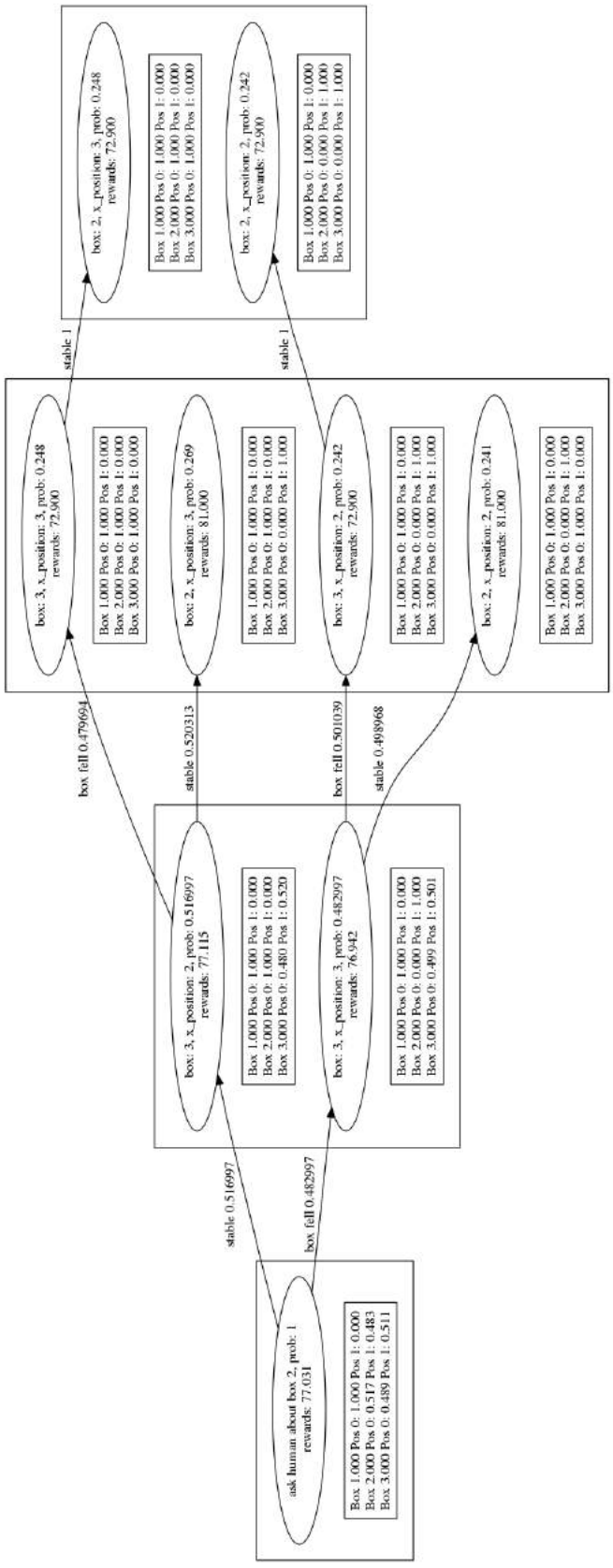


Figure A.3.: In this policy for three boxes, the agent is able to ask one oracle question.

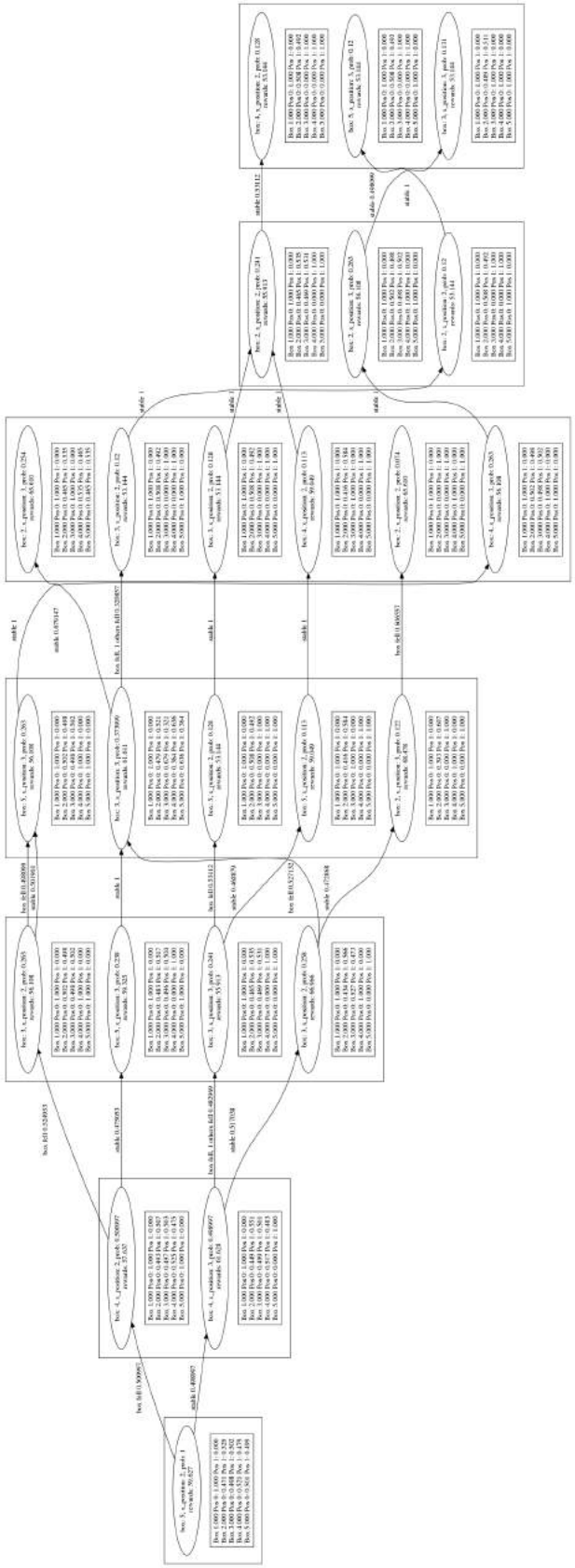


Figure A.4.: In this policy, boxes are being stacked.