Learning deep policies for physics-based robotic manipulation in cluttered real-world environments

Wissam Bejjani

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computing

January 2021

Some parts of the work presented in this thesis have been published in the following articles.

Bejjani, W., Papallas, R., Leonetti, M., and Dogar, M. R. "Planning with a Receding Horizon for Manipulation in Clutter using a Learned Value Function". In: IEEE-RAS 18th International Conference on Humanoid Robots, 2018.

Bejjani, W., Dogar, M. R., and Leonetti, M. "Learning Physics-Based Manipulation in Clutter: Combining Image-Based Generalization and Look-Ahead Planning". In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019.

Bejjani, W., Agboh, C. W., Dogar, M. R., and Leonetti, M. "Occlusion-Aware Search for Object Retrieval in Clutter". In arXiv preprint arXiv:2011.03334, 2020.

Bejjani, W., Leonetti, M., and Dogar, M. R. "Learning Image-Based Receding Horizon Planning for Manipulation in Clutter". In: Robotics and Autonomous Systems, 2021.

The above publications are primarily the work of the candidate.

Acknowledgements

I would first like to express my gratitude to my advisors, Mehmet R. Dogar and Metteo Leonetti, whose guidance is best captured by a quote from Kahlil Gibran: “The teacher who is indeed wise does not bid you to enter the house of his wisdom but rather leads you to the threshold of your mind.”.

I wish to acknowledge the technical support and administrative help provided by the School of Computing staff at the University of Leeds. I would like to thank Rafael Papallas and Wisdom C. Agboh with whom I collaborated in several publications leading to a better understanding of the interconnections between the different simulation environments and the real world. I am particularly indebted to Logan Dunbar for his help with my countless queries regarding containerization with Singularity. Thanks to Alexia Toumpa, Francesco Foglino, and Luis Figueredo for their interesting and fruitful discussions which resulted in many valuable ideas. In addition, I am grateful for the moral and emotional support of all my friends and colleagues during the many lock-downs of 2020, especially Giulia Sindoni and Mark Houghton.

Thanks to my parents Rached Bejjani and Linda Abi Habib, who supported me during my entire life, always believing in my strengths. I also would like to thank my fiancee Nathalie Charbel for her love. Without her patience and support, I wouldn’t have been able to finish this work.

Finally, I would like to thank the examiners for David C. Hogg and Edward Johns for reading my manuscript, and I would like to acknowledge the School of Computing at the University of Leeds for funding my research.

Abstract

This thesis presents a series of planners and learning algorithms for real-world manipulation in clutter. The focus is on interleaving real-world execution with look-ahead planning in simulation as an effective way to address the uncertainty arising from complex physics interactions and occlusions.

We introduce VisualRHP, a receding horizon planner in the image space guided by a learned heuristic. VisualRHP generates, in closed-loop, prehensile and non-prehensile manipulation actions to manipulate a desired object in clutter while avoiding dropping obstacle objects off the edge of the manipulation surface. To acquire the heuristic of VisualRHP, we develop deep imitation learning and deep reinforcement learning algorithms specifically tailored for environments with complex dynamics and requiring long-term sequential decision making. The learned heuristic ensures generalization over different environment settings and transferability of manipulation skills to different desired objects in the real world.

In the second part of this thesis, we integrate VisualRHP with a learnable object pose estimator to guide the search for an occluded desired object. This hybrid approach harnesses neural networks with convolution and recurrent structures to capture relevant information from the history of partial observation to guide VisualRHP future actions.

We run an ablation study over the different component of VisualRHP and compare it with model-free and model-based alternatives. We run experiments in different simulation environments and real-world settings. The results show that by trading a small computation time for heuristic-guided look-ahead planning, VisualRHP delivers a more robust and efficient behaviour compared to alternative state-of-the-art approaches while still operating in near real-time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

“AI has by now succeeded in doing essentially everything that requires ‘thinking’ but has failed to do most of what people and animals do ‘without thinking.”

- Donald Knuth

Autonomously manipulating every-day objects in cluttered environments has long been a target milestone in robotics [1, 6]. The last 40 years of research in this field have mostly focused on fetch-and-place applications that involve little to no contact with obstacles in the environment [96]. In this approach, the robot is expected to reach a desired object along a collision-free trajectory and find a grasp pose that ensures a stable grasp. Once the object is grasped, it is treated as an extension of the robot’s kinematics. Consequently, long-term reasoning over the physics of the robot-environment interactions can be avoided. Contacts with the environment, also referred to as ’collisions’, which carries a more negative connotation, are often dealt with by making the robot more compliant as a safety measure.

The 2015 and 2016 Amazon Picking Challenge (APC) were competitions where robots were tasked with retrieving items from cluttered shelves. The height of the shelves prevented robots from approaching objects from the top, and they had to rely on getting into the shelf from the side. The competition showcased state-of-the-art approaches for motion and grasp planning. All participants adopted collision-free based approaches to the problem [31, 24]. Despite impressive results, avoiding contacts inherently limited the scenarios that the robots could deal with [42]. In the following years, the challenge was redesigned to allow for top-down bin picking, a setup that is less sensitive to sequential physics interactions [8]. Redesigning the environment to better suit robotics application is one way of achieving autonomous object retrieval. However, as argued in [1, 71], future robots are envisioned to adapt to environments designed for humans and not the other way around.

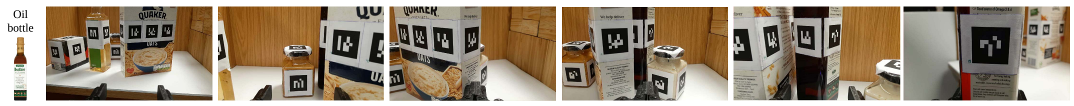


Figure 1.1: Retrieving the oil bottle. Images are from a hand-mounted camera.

Many real-world applications necessitate sequential reasoning over complex physics interactions between multiple objects. Consider the example illustrated in Fig. 1.1. The robot is tasked with retrieving the oil bottle from the kitchen cabinet. The cabinet shelf is cluttered with jars, cereal boxes, and other bottles. The oil bottle is not reachable due to occlusion and possibly the lack of a collision-free trajectory. While decluttering the shelf one reachable object at a time might be possible, time constraints and the possible lack of a temporary holding space eliminates this approach as a solution. The robot is left with having to navigate its way through the clutter to search for the oil bottle, and then reach it and pull it out without dropping any of the other objects off the shelf. While the oil bottle is small enough to be grasped, other objects, such as a large cereal box, might be best manipulated with non-prehensile actions. Accomplishing this task involves executing a sequence of prehensile and non-prehensile actions that must adapt in real-time to unforeseeable real-world physics-based interactions.

In this thesis, we investigate methods to solve manipulation tasks in cluttered real-world environments with occlusions. We develop heuristic-based planners that enable robots to physically interact with the environment. We focus on using Imitation Learning (IL) and Reinforcement Learning (RL) to learn the planners' heuristics. Within this context, we address several key requirements common to many physics-based manipulation tasks.

- Sequential decision making: Efficiently solving the manipulation task, that is with minimal number of actions, requires the manipulation actions to be performed in the right order. Accounting for the space that the robot will be traversing and the location objects will be occupying in the future is critical for avoiding unnecessary corrective actions or undesirable object configurations.

- Reactive behaviour: Predicting the exact results of multiple objects interacting over a long sequence of actions is a notoriously hard problem. Instead, the robot must adapt its motions in closed-loop during execution to unmodeled real-world dynamics.

- Generalization and transferability: The robot is expected to acquire manipulation skills that apply to a wide variety of real-world environments such as different environment settings and objects to manipulate. Although generalization and transferability are often used interchangeably in the robotics literature, in this thesis, we distinguish the nuances between them. By generalization, we refer to the robot's ability to act in environments with different number of objects, different object shapes, and different initial objects and target layouts on the manipulation surface. By transferability, we refer to the robot's ability to target different desired objects in different manipulations tasks. For example, from a physics and geometry perspective manipulating an orange fruit is not very different from manipulating an apple. The acquired manipulation skills must be seamlessly transferable to a multitude of objects that the robot is expected to interact with.

- Occlusion-aware behaviour: In many real-world shelf setups, objects can be partially or totally hidden behind one another. The robot must account for this fact and in turn reason over past observations to efficiently explore and retrieve a desired object.

We investigate solutions that can simultaneously address all these key requirements. In this process, we explore novel planning frameworks and learning algorithms which leads us to a number of new opportunities and challenges.

1.1 Main Themes

The research involved in this work evolves around several fundamental ideas in the field of motion planning and policy learning. In this section, we introduce the main themes explored in this thesis.

In the real world, object-to-object and object-to-robot physics interactions can vary widely depending on a number of parameters such as mass, inertia, friction, geometry, etc. Yet, physics predictions remain fundamental for planning sequential actions in clutter. Exactly predicting the outcome of an interaction is not always necessary to solve a manipulation in clutter problem. Many planning-based approaches embrace environment contacts by leveraging simple physics models and a strategy that takes uncertainty into account. This could be done for example by continuously re-planning or exploiting uncertainty reducing actions [29]. Physics models, however, introduce two major drawbacks. They are computationally expensive to query and inaccurate over long-term predictions. In this thesis, we look at overcoming these two drawbacks by combining two paradigms. The first is on learning a direct mapping from state to actions without explicitly relying on a physics model. The second is on using a physics engine to run a short horizon look-ahead planner such that physics transition queries are minimized and physics prediction errors are not compounded.

We investigate the use of a learned heuristic to accelerate the search of the short horizon planner to run in near real-time time. In the context of planning, a heuristic is used to estimate the cost-to-go from the current state to the goal. This estimate allows the heuristic to guide a search algorithm to explore, using a forward model, promising directions to reach the goal. The closer the heuristic estimates are to the optimal cost-to-go the shorter the solution would be. In a large and continuous state space, such as the case for manipulation in clutter, handcrafting efficient heuristics is not a straightforward task. In addition to estimating the cost-to-go, a suitable heuristic should compute this estimate relatively fast for the planner to run in near-real time. Further, the heuristic should be applicable to different environment settings for the planner to generalize. Instead handcrafting a heuristic for physics-based manipulation in clutter, this thesis delve into learned heuristics modeled by deep neural networks. The focus will be on learning the optimal cost-to-go while satisfying the real-time and generalizability requirements.

Another theme that we explore is related to the challenges posed by complex physics environments and long action sequences to existing learning algorithms. Learning-based approaches, particularly the ones based on deep learning, rely on an iterative process which gradually improves the learned behaviour. The convergence of the learning process is highly sensitive to changes in the observed interactions with the environment. In a complex physics environment, even a small change in how the robot interacts with the environment can lead to drastically different observations, potentially derailing the convergence to a useful behaviour. We re-examine some of the most common learning algorithms applied to robotic manipulation and improve on them to better mitigate premature convergence to sub-optimal solutions and unstable learning.

Further, we elaborate on the idea that a simple representation of the environment that carries relevant information across different tasks is fundamental for generalizability and transferability. A manipulation task can be represented over different granularity levels to match the requirements of the decision maker. High level abstract representations can cope with a wide range of tasks. At planning time, states and actions are modelled on a descriptive level with symbolic preconditions and effects. Actions are assumed to have instantaneous and measurable effects [35, 78]. For example, an "open_door" action could have preconditions such as "door_is_closed" and "robot_hand_is_free". If satisfied, the action can be assumed feasible and the door state would be updated to "door_is_open". In contrast, physics-based manipulation in clutter requires a substantial amount of low level reasoning over the physics [69]. A representation that, at the same time, captures low level state features, such as object shapes and relative poses, and the high level task definition, such as the type of the desired object and its target destination, would allow for a unified decision making process to perform a variety of manipulation tasks. We explore the use of an image-based representation that relies on colour labelling to describe the manipulation task while also preserving the geometrical properties of the environment.

Further, acting in an environment under occlusions would benefit from retaining information from past observations to plan future actions. Engineering what information is relevant to retain, how to represent it, and how it could be evolving due to physical interactions are not easy to answer. We investigate the possibility of delegating these questions to the learning process, whereby, the history of past observations is directly mapped to actions.

1.2 Contributions

This section outlines the contributions produced by this work.

- Closed-loop control scheme for near real-time physics and occlusion-aware manipulation enabled by a learned heuristic. The control scheme steps are to observe, represent, plan, execute the first action of the plan, and then loop-back to the observation step. The developed control scheme forms the main framework to interleave real-world execution with planning in simulation.

- Abstract image-based representation of the real-world rendered from the simulator state. The representation is able to capture an arbitrary number of objects and it uses colour-labelling for identifying the type of the objects in the scene. These two features enable greater generalization and transferability.

- Formulation of physics-based manipulation as a short horizon heuristic search problem. We show how a heuristic can be used to shorten the planning horizon. Consequently, the planner can run in near real-time and compounding modelling errors are reduced.

- Formulation of the planner's heuristic as a learnable function. It enables the planner to efficiently perform a local search and estimate the cost-to-go of a state. Additionally, in a partially observable environment, we also learn to predict the most likely poses of a hidden desired object.
- A novel formulation of IL and RL to acquire the planner's heuristic. IL is used to initialize the heuristic from sub-optimal demonstrations of how to solve the manipulation task. The heuristic learns to replicate the behaviour observed in these demonstrations. RL further optimizes the heuristic to generate a more robust and efficient behaviour. Algorithms were developed for learning in a discrete action space and in a continuous action space.
- Implementation of the aforementioned approaches on a simulated and real UR5 robot platform with a Robotiq gripper. The robot demonstrates the ability to search and retrieve a desired object from a cluttered shelf without dropping any of the other objects off the edges.

1.3 Thesis Outline

Chapter 2 presents the background theory behind our work. Chapter 3 locates our work with respect to the state-of-the-art. Chapter 4 introduces the framework of the closed-loop control scheme for manipulation in environments without occlusions. The heuristic learning algorithms and how the heuristic is used to guide the planner is also laid out in this chapter. Chapter 5 presents our approach to manipulation in environments with occlusions. Chapter 6 presents a road-map for future work on how to extend the contributions of this thesis to more complex 3D environments.

1.4 Publication Note

This thesis builds on the findings of two published conference papers [15, 10], one journal paper that is accepted for publication [11], two published workshop papers [14, 13], and one conference paper that is currently under review [9]. Most of the work in Chapter 4 on discrete action space appeared in our conference papers [15, 10], while work on continuous action space appeared in our journal paper [11]. Most of the work in Chapter 5 is under review for the conference paper [9].

Chapter 2

Background

This chapter presents a background on the learning literature that this thesis builds upon. The chapter starts by introducing Markov chains and follows the literature up to state-of-the-art deep Reinforcement Learning (RL) algorithms.

Although this thesis presents several novel concepts related to planning, profound background knowledge of the planning literature is not required to follow this work. Where required, relevant planning concepts and algorithms will be presented in subsequent chapters along with the corresponding contribution.

2.1 Markov Chains

A Markov Chain is a memory-less stochastic process for describing state transitions. A state, $s \in S$, is a known variable or set of variables describing the environment. The state can also describe a partially observable environment as in a Hidden Markov Model. A process is called Markovian if it satisfies the Markov property: the probability distribution over the next state only depends on the current state.

$$P(s_{t+1}|s_0, s_1, \dots, s_t) = P(s_{t+1}|s_t). \tag{2.1}$$

2.2 Markov Decision Process (MDP) and the Bellman Equations

An MDP is a mathematical framework for decision making in a Markov process with discrete time step. Compared to a Markov Chain, an MDP introduces a decision maker for selecting an action $a_t \in A$ at a Markov state s_t and thus influencing the distribution over the next state s_{t+1}, such that

$$T(s_t, a_t, s_{t+1}) = P^a_{ss'} = P(s_{t+1} = s'|s_t = s, a_t = a), \tag{2.2}$$

with T being the probability transition function. At each time step, the decision maker selects an action according to a policy π, where π is a mapping from states to actions:

$$a_t \sim \pi(.|s_t) \tag{2.3}$$

In return, an immediate scalar reward is received:

$$r_{t+1} = r(s_{t+1} = s', s_t = s, a_t = a). \tag{2.4}$$

The optimal MDP solution is a policy π^* that maximises the discounted sum of future rewards, also called the *return* R, at any instance t:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=t} \gamma^{k-t} r_{k+1}, \tag{2.5}$$

where $\gamma \in [0, 1)$ is the discount factor used to prioritize immediate rewards over the reward signal from distant future actions.

The value of a state provides an estimate of the *return* if a policy π were to be followed from the current state. It is described by the expectation over the *return* given by the value function v^π. Similarly, it can also be described by the action-value function q^π which is the expected *return* from the current state s assuming action a was selected. The value function and action-value function are given by:

$$v^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=t} \gamma^{k-t} r_{k+1} \middle| s_t = s \right] \tag{2.6}$$

and

$$q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=t} \gamma^{k-t} r_{k+1} \middle| s_t = s, a_t = a \right]. \tag{2.7}$$

The value function can also be expressed in terms of the action-value function:

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a). \tag{2.8}$$

A certain policy π_1 is considered better than another policy π_2 if policy π_1 yields higher *return* than π_2, that is $v^{\pi_1} > v^{\pi_2}$ $\forall$ $s \in S$. Hence, the optimal policy is the policy that yields the highest *return*:

$$\pi^* = \arg\max_\pi v^\pi(s) \;\; \forall \; s \in S. \tag{2.9}$$

The optimal value function v^* corresponding to following the optimal policy π^*. It can be written as:

$$\begin{aligned} v^*(s) = v^{\pi^*}(s) &= q(s, \pi^*(.|s)) \\ &= \max_{a \in A} q^*(s, a) \;\; \forall \; s \in S. \end{aligned} \tag{2.10}$$

Hence, the optimal policy can also be formulated in terms of the action-value function:

$$\pi^* = \arg\max_{a \in A} q^*(s, a) \;\; \forall \; s \in S. \tag{2.11}$$

Dynamic programming can be used to solve the search for the optimal policy in a discrete time-step MDP. It solve the problem by recursively updating the expected *return* and storing it over for all the MDP states, a not so computationally efficient technique for large MDPs. We derive the Bellman equations used in dynamic programming to compute the value function in a stochastic processes:

$$\begin{aligned} v(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s] \\ &\text{using Eq. 2.5} \\ &= \mathbb{E}_\pi[r_{t+1} + \sum_{k=t+1} \gamma^{k-t} r_{k+1} | s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s] \\ &\text{we rewrite it as conditional probability over the next state } s_{t+1} = s' \\ &= \sum_{s' \in S} P(s'|s) \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s, s'] \\ &\text{we rewrite it as conditional probability over the action } a \text{ at state } s \\ &= \sum_{a \in A} P(a|s) \sum_{s' \in S} P(s'|s, a) \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s, s', a] \\ &\text{The addition in an expectation can be separated such that} \\ &= \sum_{a \in A} P(a|s) \sum_{s' \in S} P(s'|s, a) \left[\mathbb{E}_\pi[r_{t+1}|s, s', a] + \gamma \mathbb{E}_\pi[R_{t+1}|s, s', a]\right] \\ &\text{then using the Markov property and Eq. 2.4, we reformulate it as} \\ &= \sum_{a \in A} P(a|s) \sum_{s' \in S} P(s'|s, a) \left[r_{t+1} + \gamma \mathbb{E}_\pi[R_{t+1}|s']\right] \end{aligned} \tag{2.12}$$

We can now express the value function using the transition model:

$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} T(s', s, a)[r_{t+1} + \gamma v^{\pi}(s')] \tag{2.13}$$

Similarly, the the Bellman equation can also be expressed in terms of the action-value function. Using Eq. 2.8, we get:

$$q^{\pi}(s, a) = \sum_{s' \in S} T(s', s, a)[r_{t+1} + \gamma \sum_{a' \in A} \pi(a'|s') q^{\pi}(s', a')] \tag{2.14}$$

The Bellman equations are used to learn the value of a policy. By iteratively updating the the value function and the policy, the optimal policy can be retrieved. Solving these equations requires a known a MDP, which is not the case for most real-world problems.

2.3 Model-Free Reinforcement Learning

Model-free RL offers a framework for approximating the optimal policy of MDPs with unknown structure from experience. The policy is either indirectly learned with value iteration or directly with policy gradient approaches.

Monte-Carlo: Monte-Carlo (MC) RL learns the value of a state using the experienced *return* from following the acting policy from that state. It mostly applies to episodic MDP, where an episode is a sequence of state-action-reward $\langle s_t, a_t, r_{t+1} \rangle$ ending at an MDP terminal state. At the end of an episode, the values for each visited state are updated only based on the final *return.* That is in contrast to updating it based on the neighbour states as in the Bellman equations Eq. 2.13 and Eq. 2.14. In a discrete state space, if a state is visited more than once in an episode, it can either be updated based on the *return* from the first visit, or the *return* can be averaged over the multiple visits. In a continuous state space, it is very unlikely to visit the same exact state twice in an episode. The updated state value formula is:

$$v^{\pi}(s_t) \leftarrow v^{\pi}(s_t) + \alpha(R_t - v^{\pi}(s_t)), \tag{2.15}$$

where $\alpha \in (0, 1]$ is the learning rate hyper-parameter, typically close to 0. A drawback of MC methods is that they can only be updated after the episode has terminated and they suffer from high variance, particularly in long time length episodes. Hence, MC methods require a large number of samples compared to alternative low variance approaches

Temporal Difference: Temporal difference (TD) methods allow for value update at every time step during execution in MDP with unknown transition function. TD combines the *return* sampled from experience, as with MC methods, with the Bellman equations to bootstrap the estimate value updates. The value estimate at the current state s_t is updated based on the immediate reward received from the model and the value estimate at the next state s_{t+1}:

$$v(s_t) \leftarrow v(s_t) + \alpha(r_{t+1} + \gamma v(s_{t+1}) - v(s_t)), \tag{2.16}$$

where

$$\begin{aligned} r_{t+1} + \gamma v(s_{t+1}) & \quad \text{is the TD target value,} && (2.17) \\ r_{t+1} + \gamma v(s_{t+1}) - v(s_t) & \quad \text{is the TD error.} && (2.18) \end{aligned}$$

Using the action-value function, we can rewrite 2.16 as:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t)). \tag{2.19}$$

This update rule results in an on-policy method know as SARSA, which is short for: $\langle s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} \rangle$. The learned action-value function corresponds to the acting policy controlling the agent.

Alternatively, “q-learning” is a TD algorithm in the discrete action space that allows to directly learn an estimate of the optimal action-value function q^* instead of q^π [114]:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a q(s_{t+1}, a) - q(s_t, a_t)). \tag{2.20}$$

The *max* operator in the TD target, $r_{t+1} + \gamma \max_a q(s_{t+1}, a)$, makes the q-learning algorithm an off-policy algorithm. It makes possible to learn an estimate of the optimal action-value function irrespective of the acting policy. A major benefit of an off-policy approach is that $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ transition samples collected from previous experiences using an older version of the currently acting policy can still be used to update the action-value function. Several exploration strategies have been developed for q-learning. The most common and simple strategy is ε-greedy. With probability ε a random action is selected and with probability $1 - \varepsilon$ a greedy action is selected, i. e., $a = \arg\max_a q(s, a)$. At convergence, the agent executes the greedy policy.

TD methods, which only use the first experienced reward in the update rule, overcome the high variance in MC methods. The low variance comes at a cost of a high bias introduced by the leaned estimate the *return* a the next state. A middle ground exists between MC and single-step TD. It is possible to construct the update target based on the n-step rewards received from the model. The TD target value becomes:

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n v(s_{t+n}), \tag{2.21}$$

where n is a hyper-parameter that is task dependent. $v(s_{t+n})$ is replaced with $q(s_{t+n}, a_{t+n})$ and $\max_a q(s_{t+n})$ for SARSA and q-learning, respectively.

Policy Gradient: Policy gradient methods present another paradigm for approximating the optimal policy. Instead of first learning a value function then retrieving the policy, policy gradient methods aim at modelling and optimizing the policy directly. The policy must be parametrized by a differentiable function with respect to θ and is it learned in an on-policy fashion. Policy gradient methods are better suited for continuous action spaces compared to value iteration method. Take for example q-learning. In a continuous action space, it must scan the *max* operator over a continuous space, suffering from the curse of dimensionality. In policy gradient, on the other hand, the policy model can directly learn the parameters of Gaussian policy in the continuous action space.

The objective of policy gradient methods is to learn the policy that maximizes the expected *return*,

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0} \gamma^t r_{t+1}], \tag{2.22}$$

by following the gradient of the expected *return*. The parameters of the policy are updated according the following update rule:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta). \tag{2.23}$$

According to the *policy gradient theorem* [106], the gradient of Eq. 2.22 is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\Psi_t \nabla_\theta log \pi_\theta(a_t|s_t)], \tag{2.24}$$

where Ψ_t can have different formats depending on the implementation. Some of the most common are [100]:

$$\Psi_t = \sum_t \gamma^t r_{t+1} \quad \textit{return}, \tag{2.25}$$

$$\Psi_t = q^\pi(s_t, a_t) \quad \text{action-value ft.}, \tag{2.26}$$

$$\Psi_t = Adv^\pi(s_t, a_t) \quad = q^\pi(s_t, a_t) - v^\pi(s_t) \quad \text{advantage ft.}, \tag{2.27}$$

$$\Psi_t = Adv^\pi(s_t, r_{t+1}, s_{t+1}) = r_{t+1} + v^\pi(s_{t+1}) - v^\pi(s_t) \text{ adv. ft. with TD residual.} \tag{2.28}$$

Following the gradient formulation in Eq. 2.24, the policy update will increase or decrease the probability of experienced trajectories according to their yielded *return*. Eq. 2.25 and Eq. 2.26 scale the probability of a trajectory irrespective if the trajectory received much smaller or larger rewards than others. Eq. 2.27 and Eq. 2.28 uses the advantage function where they subtract a θ independent term, called *baseline*, such that changes in the probability of a trajectory becomes relative to the average expected *return*. In other words, the advantage function estimates how much better or worse was the action a_t that the agent took at state s_t compared to the average *return* expected at s_t. In practice, Eq. 2.28 is preferred over as Eq. 2.27 as only the value function needs to be learned as opposed to learning the action-value function and the value function. Approaches where the gradient is bootstrapped using a learnable v^π are called "Actor-Critic" Algorithms. The Actor is the policy $\pi(a|s)$ and the Critic is the value function v^π.

2.4 Deep Reinforcement Learning

Deep RL refers to using a neural network (NN) as a non-linear, differentiable function approximator to model the value function or the policy. A deep RL cycle consists of (*i*) collecting transition samples by interacting with the environment following a certain policy, then (*ii*) run stochastic gradient descent on the NN parameters via back-propagation. NNs can generalize over large and continuous state space by inherently learning relevant abstract features to estimate the value function or for the policy to act on. However, compared to other function approximators such as radial basis function or tile coding, NNs have high sample complexity and require the data, i. e., the transition samples, to be independent and identically distributed (i.i.d.). This is particularly challenging in RL since the data is generated from a sequential process making it highly correlated and hence causing stability problems to the learning process. There exist a plethora of algorithms and hacks to stabilize Deep RL. We present the ones most relevant to this thesis.

Deep Q-Learning (DQN): The DQN algorithm, proposed in [80], is the q-learning algorithm with additional features to stabilize the learning of an action-value function modelled by a NN with parameters θ. As the generated data in RL are highly correlated, the DQN algorithm proposes collecting and storing transition samples in a large replay buffer D^{replay}. In what is known as "Experience Replay", the NN is updated over batches of random samples from the replay buffer. The benefit of the replay buffer is twofold. It decorrelates the sequential experiences by random sampling. Also since the q-learning algorithm is off-policy, the replay buffer makes learning more sample efficient. Data in the replay buffer, even if collected with an older acting policy (inferred from an older action-value function), can still be used to update the current action-value function. Hence, the transition samples are used more than once. Another feature of DQN is to freeze the action-value function parameters θ_{frozen} in the TD target and only updates it periodically. This features aims at reducing the short-term oscillation in action-value function updates (avoid chasing a moving target) such that:

$$\mathcal{L}_q(\theta) = \mathbb{E}_{\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle \sim D^{replay}}[(r_{t+1} + \gamma \max_a q_{\theta_{\text{frozen}}}(s_{t+1}, a) - q_\theta(s_t, a_t))^2] \quad (2.29)$$

Mnih et al. [80] demonstrate DQN over Atari games with images-based state representation. They concatenate the last 4 frames of the game to have a Markovian state and use the TD updates with n-step reward, as in Eq. 2.21.

Synchronous Advantage Actor-Critic (A2C): Asynchronous Advantage Actor-Critic (A3C) is a variation of Actor-Critic algorithms with NN function approximators designed to parallelize data collection [79]. A3C uses multiple agents each with its own local policy parameters and running in its own environment. The gradient computed from each agent is used to update a set of global parameters asynchronously. After every certain number of interactions, an agent updates its local parameters from the global parameters.

Synchronous A2C is a variation of A3C where all the agents use the same parameters [87, 116]. A2C waits for a certain number of transition samples to be collected from all the agent, then computes the gradient and update the global parameters using all the collected transition samples. The global parameters are then shared with the agents for a new round of data collection.

Proximal Policy Optimisation PPO: In Actor-Critic methods, the policy and value function are updated based on the stream of transition samples produced by the agent. This means that the stability of the learning process is highly sensitive to changes in the policy and by consequence the generated transition samples. A large update to the policy, far outside the range where the transition samples were collected, can entail a new stream of bad transition samples, i. e., transition samples with faint or no reward signal from the environment. The estimate of the advantage function becomes less accurate which further alters policy making it hard to recover. This phenomenon is sometimes referred to as *catastrophic forgetting*, i. e., loosing all the previously acquired knowledge [61].

A naive solution would be to lower the learning rate at the expense of an even higher sample complexity. Several alternative solutions have been proposed to tackle this issue, mostly focusing on having conservative policy updates [99]. PPO offers an intuitive first order optimisation method to limit the policy updates by clipping the policy ratio, $\frac{\pi_\theta}{\pi_{\theta_{old}}}$. PPO suggest reformulating the policy loss function (Eq. 2.24) as follows:

$$\mathcal{L}_{CLIP}(\theta) = \mathbb{E}_t \left[min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} Adv_t,\ clip \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon \right) Adv_t \right) \right], \tag{2.30}$$

where $\nabla_\theta log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. This formulation of the objective function ensures that actions that had a better effect than the expected *return* ($Adv > 0$), and are becoming more probable under π_θ, their probability will increase but not by much compared to the previous policy π_θ, i. e., no more than $\frac{\pi_\theta}{\pi_{\theta_{old}}} > 1 + \varepsilon$. In the case where actions that had a worse effect than the expected *return* ($Adv < 0$) and are becoming less probable under π_θ, their probability will decrease but not by much compared to the previous policy π_θ, i. e., no less than $\frac{\pi_\theta}{\pi_{\theta_{old}}} < 1 - \varepsilon$.

2.5 Partially Observable Markov Decision Process (POMDP)

A generalization of a Markov Process is a Hidden Markov Model (HMM) where the state is not fully observable. Acting in a HMM is modelled by a POMDP. The agent has restricted information on the state via partial observations $o \in O$. At each discrete time step, the state s is manifested by an observation o following the observation probability distribution $\Omega(o|s)$.

When a model is available, decision making takes place in belief space b [52], that is, on a probability distribution over the state space $b(s) = P(s)$. The belief is updated after every interaction with the environment. Given the previous belief $b(s)$ and after executing action a and observing o', the updated belief becomes:

$$b(s') = \frac{1}{P_\Omega}\Omega(o'|s')\sum_{s\in S} T(s,a,s')b(s), \tag{2.31}$$

where P_Ω normalizes the entries of the belief state:

$$P_\Omega = \sum_{s'\in S}\Omega(o'|s')\sum_{s\in S} T(s,a,s')b(s). \tag{2.32}$$

A POMDP solution is a policy π that maximizes the *return* by mapping a belief b to an action a over the entire state space S. However, maintaining a belief over a large state space requires a high computational and memory cost [84]. Instead, value iteration and policy gradient methods with function approximators have been developed for computing the policy directly from observations [41]. Nevertheless, efficiently scaling POMDP solutions to large and continuous state and action spaces remains an active field of research.

Chapter 3

Related Work

"We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard."
- John F. Kennedy

3.1 Related Work in Environments without Occlusions

3.1.1 Open-Loop Based Approaches

Manipulation in cluttered spaces has long been approached with planning-based techniques. Planners, such as [39, 62, 60, 70, 12], adopt an approach of motion planning followed by open-loop execution to solve the task. Kitaev et al. [62] propose a trajectory optimization based approach for retrieving an object from a cluttered shelf. By evaluating roll-outs in a physics simulator, they iteratively optimize a trajectory which minimizes a weighted cost function. The cost function is designed such that the resulting trajectory is less likely to topple objects, avoid objects falling over, and have obstructing objects moved orthogonally to the grasping direction. A simplified geometric model of the robot made of primitive shapes is used to speed up the physics roll-outs. After convergence, the resulting trajectory is further refined with the complete mesh model of the robot and executed in open-loop. Leidner et al. [70] motivate a physics-based semantic planner for wiping cluttered surfaces. To reason about the effect of a wiping motion, they use an abstract representation of particles distributed on a surface. The physics of interactions between the particles and the wiping tool is defined following the semantic goal of the task. In an skim task, for example, the particles are pushed by the tool along the direction of motion. In an absorb task, the particles are deleted when they come in contact with the tool, simulating a sponge absorbing a liquid. Using this representation, the planner generates Semantic Directed Graphs (SDG) in the Cartesian space that, when traced by the robot end-effector, produce the desired wiping effect such as collect, skim, and absorb. SDGs are resolved into the robot joint motion and executed in the real world. When executed in open-loop, the plan might fail to produce the desired effect due, for example, to unaccounted for interactions. When faced with such situation, the planner is queried for another solution as presented in [69]. Haustein et al. [39] use sampling-based planning to solve manipulation in clutter problems. They propose reducing the search space of Kino-dynamic Rapidly exploring Random Trees (RRT) by planning over statically stable environment states while allowing for physical interaction in-between these states. By considering a quasistatic model of the physics, object velocity can be eliminated from the search space. Planning can then take place in the configuration space and actions are projected to a constraint manifold parallel to the manipulation surface. When operating at relatively low velocities in a cluttered environment, it is safe to assume a quasistatic physics model [53]. We develop a similar open-loop Kino-dynamic RRT planner to [39]. We use it to generate demonstrations on how to solve different instances of the manipulation in clutter task.

There are open-loop planners which also take uncertainty into account before the generation of the motion trajectory. Dogar et al. [27] introduce a framework for planning with uncertainty reducing actions. The framework utilizes the funnelling effect of pushing actions to reduce a large object pose uncertainty into its actual pose. It works by moving the robot's gripper towards, then past, the object pose distribution while the fingers of the gripper are open. This motion forces the object to slide against the fingers towards the centre of the gripper. Mahayuddin et al. [83] extend a sampling-based Kino-dynamic planner in multi-contact environments to account for physics-based uncertainty. They propose evaluating a set of particle motions at every node propagation step in a search tree. The particles are sampled from a probability density function over the different sources of uncertainty. The most robust motion is chosen and the uncertainty is propagated to the subsequent node in the tree. Koval et al. [66] formulate manipulation planning under uncertainty as a selection problem. Using a Kino-dynamic RRT planner, they generate multiple goal-leading trajectories. To estimate the chances of a trajectory to reproduce the desired outcome in the real world, each trajectory is evaluated multiple times in simulation by forward propagating the control sequence through stochastic dynamics. The trajectory that scores the highest estimated success probability is selected for execution on the real robot. To minimize the total number of roll-outs to evaluate, they also propose eliminating candidates that start to show a relatively low estimated success probability from being further evaluated. These open-loop planners generate a provably conservative sequence of actions, limiting the robot from exploiting the complete dynamics of the domain.

Alternatively, to avoid the uncertainty associated with multiple objects interacting in a cluttered environment, planning approaches have been developed to avoid contact with obstacles altogether. Finding a collision free trajectory has been the common theme in many of the approaches presented at the Amazon Picking Challenge [42, 82]. Kimmel et al. [59] motivate a two-step planning approach, first in the task space and then in the robot joint configuration space, to find a collision-free trajectory to a stable grasp pose and for retrieving the desired object. In many cluttered environments, however, a collision-free trajectory is not always available, and with that comes the necessity of a reactive system.

3.1.2 One-Shot Top-Down Manipulation

The research on object retrieval is witnessing a growing interest in image-based systems particularly for top-down bin picking. The work by Zeng et al. [122] approaches object picking from a bin using a custom designed gripper capable of push, grasp, and suction actions. Working with RGB-D images, they use several Neural Networks (NNs), one per action primitive, to evaluate pixel-wise affordances for the corresponding actions, then execute the action at the location and orientation of the highest affordance. Once an object is retrieved, an independent object identification module is engaged to detect the type of the grasped object. Also using RGB-D images, Shome et al. [102] use a suction cap end-effector to perform one of three manipulation primitives, namely toppling, pulling, and pushing for bin packing tasks of cuboidal objects. Johns et al. [51] address uncertainty in top-down picking tasks. They train a NN over synthetic depth images to predict a grasp function over an object shape. The grasp function is composed of discretized grasp candidates for every possible top-down grasp pose. The NN learns to classify each grasp candidate by a discrete grasp score. To account for inaccuracies from noisy sensors or inaccurate calibration, a gripper's pose uncertainty function is used to smooth grasp function. The smoothing eliminates good grasp candidates that are surrounded by low score candidates in favour of regions with high average grasp scores. Their approach is applied to picking an isolated object. For grasping in a tightly packed cluster of objects, Zeng et al. [121] learn synergetic push and grasp actions over pixel-wise action-value heat map to disperse the clutter then grasp one of the objects in the scene. These approaches, however, require specific high-level manipulation primitives to be defined. On-shot execution of a high-level primitive limits the robotic manipulator from dynamically correcting its behaviour in response to unexpected changes in the environment. Viereck et al. [111] addresses this problem by using a controller that continuously updates the target grasp pose during execution. The controller relies on a learned distance-to-grasp function that is evaluated over depth information. These approaches, however, are object agnostic, i. e., no specific object to be manipulated can be selected apriori and it is left to the system to select an object that is feasible to grasp. In many real-world scenarios, it is desirable to be able to specify the object to be grasped. Moreover, in many real-world scenarios a top-down picking approach might not be feasible due to space limitations.

3.1.3 Closed-Loop Pushing Tasks

Non-prehensile manipulation is mostly dominated by pushing based actions. Pushing is an effective strategy for manipulating objects that are too large to grasp or too heavy to lift. Further, manipulating small and light objects can also benefit from pushing-based strategies as they also offer action efficient manipulation strategies. A pushing action is highly sensitive to the friction characteristics which can vary significantly across the manipulation surface and the surface of the object [119]. Analytical models have been developed to compute fast prediction of an object's behaviour under a push action [77, 27]. Hogan et al. [45] apply a model-based feedback control scheme that can alternate between different interaction modes to control a tool pushing a slider on a planar surface. As uncertainty on the physics interaction increase with higher pushing velocities, Agboh et al. [4] argue for using a task-adaptive trajectory optimizer. The control scheme modulates the pushing velocity to adapt the accuracy of the trajectory to the task requirements. Pushing planners, however, require careful calibration of the parameters of the underlying analytical model. They must ensure that the assumption on the physics parameters in simulation accurately conforms to that of the real world.

Data-driven approaches have been proven effective in overcoming the simplifying assumptions that engineered analytical models tend to rely on. To learn a robust behaviour for real-world applications, Kloss et al. [63] combine an image-based learning approach with an analytical model of a pushing task. They train a NN, on visual sensory input, to output the appropriate physical parameters to the analytical model that is controlling the robot. Clavera et al. [22] argue for a modular approach where sensing, policy, and controller are designed separately to ease the skill transfer from simulation to the real world.

The policy is trained with reinforcement learning in simulation over a handcrafted set of features and following an engineered reward function to improve convergence. These approaches have proven capable in real-world manipulation. However, they are designed for pushing a single object and their extension to multi-object environments is yet to be explored.

3.1.4 Closed-loop Planar Manipulation in Clutter

Closed-loop planar manipulation, that is, approaching the manipulation surface from a plane parallel to that surface, is regaining interest with the advent of relatively fast physics simulators. Papallas et al. [90] propose closing the loop with feedback from a human operator. Instead of solving a relatively long sequential problem with the goal of reaching a desired object in clutter, the human operator sets sub-goals that are intuitively easier to solve by an open-loop planner. Sub-goals include which and where to move an obstructing object. After the execution of a sub-task, the human operator reassesses the scene and either sets a new sub-goal or the main task goal. To minimize the workload on the human, Papallas et al. [89], propose a trajectory optimization strategy which only queries the human for a sub-goal when failing to solve the current task. With a human in the loop, a single operator can simultaneously assist multiple robots to improve the success rate and execution time. Agboh et al. [5] propose parallelizing trajectory optimization based algorithms to more efficiently generate a sequence of optimal controls.

Learning policies that can generate actions for planar tasks without access to an explicit physics model has proven to be substantially more difficult to train compared to top-down bin picking or single object pushing. Albeit, closed-loop policies remain an attractive option for reactive planar manipulation. By acting in closed-loop, they can correct in real-time for uncertainty in the dynamics and modelling errors.

Laskey et al. [67] uses an IL approach by relying on expert human demonstrators and a NN to control a 2-DOF robot arm to reach a desired object on a cluttered surface. The NN learns a policy from plans generated online using the Dataset Aggregation algorithm (DAgger). DAgger requires all states that the robot visits during training to be labelled by an action from a demonstrator. However, querying an expert demonstrator, whether a human or a near-optimal planner, for every visited state is impractical. They propose to alleviate the cost associated with this drawback using a hierarchy of supervisors with different skill levels, ranging from simple motion planners to PhD students.

In environments with sparse rewards and long horizon, RL based approaches are challenged by high sample complexity and the balance between exploration and exploitation [30, 58]. To address the exploration problem, Pinto et al. [92] propose an informed strategy to sample initial states for RL training episodes. They, first, use an open-loop physics-based planner to generate solution plans to planar manipulation tasks. Then, the visited state along the generated plans are used to initiate RL training episodes.

The approaches stated so far for solving planar manipulation tasks have proven robust to high clutter environment. However, they are oriented towards tasks that can be solved with with non-prehensile action. They do not consider prehensile actions which are sometimes necessary to efficiently manipulate the desired object to a target destination.

Lee et al. [68] explore solving planar manipulation tasks with prehensile actions. They put forth an algorithm for clearing the clutter using collision free fetch-and-place actions. The algorithm recursively selects an obstacle object to grasp and relocate. The process repeats until a collision-free trajectory to the desired objects is found. The algorithm is shown to operate in near real-time. Nevertheless, a potentially restrictive assumption is that an empty space for relocating objects must be available. Simultaneously using prehensile and non-prehensile manipulation action to reach and manipulate a desired object in multi-contact environments remains underexplored.

3.1.5 State Representation and Domain Randomization for Policy Learning

The state representation, that is, the features on which decisions are made, plays a major role in shaping the learned behaviour. When deciding on the state representation, the key design choice lies in balancing the trade-off between having a representation that is *expressive* enough to capture the state, while also providing a space that is *efficiently searchable* [20].

Feature engineering is commonly used to provide a concise representation of the environment. However, feature engineering often entails that human biases that are carried through the representation with possible performance degradation [17]. Particularly, engineering a low dimensional representation for a manipulation in clutter task is not trivial. Physics interactions are highly dependent on the objects' shape. For instance, consider a state representation that omits modelling the objects' shape while only capturing information on the Cartesian pose of the objects. A resulting policy acting on such a representation may yield a robust yet conservative behaviour that does not leverage the geometry of the objects [15].

Raw sensory representations, such as realistic images, carry an abundant amount of information, i. e., highly expressive, and do not require feature engineering. Image-based representations can take advantage of the spatial generalization of Convolutional Neural Networks (CNN) to implicitly learn spatial features that allow for greater task generalization. Domain randomization is often used in conjunction with image-based representations to guide the learning process to capture relevant features that are regular across different tasks. Tobin et al. [109] show that domain randomization is particularly useful for sim-to-real transfer. They focus on object pose detection in clutter from a single image. They study how different randomized elements such as clutter density, object texture, object configuration, background colour, etc., affect the pose detection accuracy. Their results show that the accuracy improves with higher observed variation in the imaged-based state representation. James et al. [48] demonstrate the robustness of a NN-controlled manipulator to bridge the sim-to-real gap when only trained in simulation with highly randomize environment characteristics. The NN learns from demonstrations how to drive a robot to solve a long-horizon multi-stage task of reaching for a cube object, grasping it, then transferring into a basket. Having been trained in the simulation environment over different background colours, cube and basket poses, camera and light sources, distractor objects, etc., the image from the real world would appear as just another variation that the NN would be able to handle. Peng et al. [91] show that domain randomization of parameters that are not observed in the state representation also helps in learning robust manipulation behaviour. They train, in simulation, a robot to perform a pushing action while randomizing the physics parameters such as mass, friction, time-step between actions, etc. In this context, the randomization also helps in avoiding a behavior that exploits the idiosyncrasies of the physics model in the simulation environment. The learned skill is then used to push an object in the real world to a target location. In our work, we use domain randomization to ensure generalization over object shapes and configurations in the image space.

In many applications where the attributes of the desired object and environment conditions are known a prior, end-to-end systems have been successfully deployed to map real-world images to actions [95]. However, the use of end-to-end systems for multi-task manipulation remains limited. For instance, a policy that is trained for gasping apples cannot be tasked to grasp an orange instead without necessitating further training or domain adaptation [32, 50]. In this thesis, we overcome this limitation by using a common representation for manipulation tasks with different objects types.

3.1.6 Closed-Loop with Look-Ahead Planning

In environments with sparse reward functions, remarkable results have been achieved by combining RL based approaches with planning. Planning compensates for the suboptimality of a policy with informed model-based conjectures. This combination is dominated by tree search planners to guide RL policy search [7, 103]. A large body of work rely on Monte Carlo Tree Search (MCTS) to guide the RL search policy. The vast majority of MCTS implementations uses Upper Confidence Bounds (UCB) to balance between exploitation of experienced rewards and exploration of un-visited states [56, 94]. Anthony et al. [7] use MCTS to generate plans leading to the goal. They suggest making the searches more efficient by biasing the search process with a NN-based value function that is recursively trained on the previous iteration of the generated plans.

Song et al. [105] apply this concept to physics-based manipulation domains for solving rearrangement tasks that necessitate a long sequence of actions. Albeit the use of NNs that are recursively trained over previous iterations of the generated plans to speed up MCTS [18, 64], the computation cost associated with simulating the physics for state transitions remains prohibitively expensive for this process to run in closed-loop and in real-time. This is because, fundamentally, MCTS requires a large number of roll-outs up to the terminal state for its estimates to become reliable.

A more viable alternative is the use of Model Predictive Control (MPC) and Receding Horizon Planning (RHP) like approaches to perform look-ahead planning in real-time. If the goal state is not within the horizon reach, a learned value function is used as a terminal cost at the horizon state in order to solve a finite-horizon approximation to a long or infinite horizon task. Kartal et al. [57] extend an Actor-Critic RL algorithm to predict the temporal closeness to terminal states. The temporal closeness is then used to enable limited-depth MCTS roll-outs with around a 100 roll-outs per action selection. Despite significant improvement over previous works, the number of performed roll-outs remains far above real-time performance for physics simulation. In a physics simulator, Tong et al. [110] propose training a policy network to generate a sequence of actions up to a certain horizon. Then using another value function trained network and an evolutionary algorithm, the sequence of actions is optimized and the first action of the optimized sequence is executed. Thananjeyan et al. [108] implements deep MPC over learned dynamics in a constrained environment. To ensure exploration without violating the constraints, generated trajectories are conditioned to where a plan exists for navigating back to a safe set. Although the approaches stated so far are showing promising results and have a reminiscence to our work, some of their underlying assumptions, such as having access to a dense reward function or that the initial and goal state distributions form a tight subset of the state space, are not applicable to physics-based manipulation in clutter tasks.

3.2 Related Work in Environment with Occlusions

3.2.1 POMDP Planners

In the presence of occlusions, manipulation in clutter is often associated with interactive search, that is leveraging manipulation actions to simultaneously gain visibility and accessibility [17]. Thanks to recent advances in model-based online planners under uncertainty, like DESPOT [104] and PA-POMCP [118], this field is gaining momentum towards achieving everyday real-world manipulation tasks. Li et al. [73] address uncertainty on the type of a detected object caused by partial occlusion. They propose exploiting spatial constraints to reduce the number of feasible actions to be explored for information gain. Using a fixed-length horizon planner, decision making takes place online to select which obstructing object to be moved to minimize occlusion. In addition to spatial constraints, Wong et al. [115] use object semantics to focus the search in containers where observed objects are most similar to the desired object. Pajarinen et al. [88] manage to solve long-horizon manipulation by combining particle filtering and value estimates in an online POMDP solver. The environment setting includes multiple objects spaced apart such that multi-object contacts are avoided. The approaches stated so far have largely overcome the computational complexity associated with large state space and observation history where explicit modelling of uncertainty is tractable.

In environments with multi-object contacts, however, the uncertainty can quickly degenerate to multi-modal and non-smooth distributions [3]. Scaling the belief update over occluded spaces and the belief planner to long action sequences become impractical. Hence, model-based online POMDP planners tend to avoid multi-object contacts by planning over collision-free single object actions, such as $Lift(i)$, $Move\_Out(i)$, $Grasp(i)$, etc. where i is the object ID. Reasoning over actions on this higher level of abstraction can have its benefits on reducing planning time. But potentially, it can also limit the robot from exploiting the physics-based interactions for a more efficient behaviour.

3.2.2 Model-Free Policies with Recurrent Units

The emergence of model-free approaches with function approximation as viable methods for a reactive and generalizable behaviour have fostered interests in their deployment in partially observable environments [38, 46]. Model-free approaches are based on a direct mapping from observation history to manipulation actions. NN function approximators, with appropriate inductive biases in their structures, bypass the need for a closed-form representation of the belief update and environment dynamics. Heess et al. [41] show that by using Long Short-Term Memory (LSTM) cells as a tool to summarize a history of partial observations, it is possible to train a policy for pushing an object to an initially observed pose. Karkus et al. [55] propose a model-free approach that trains a NN on expert demonstrations to approximate a Bayesian filter and a POMDP planner. Garg et al. [34] trains a policy to grasp an object under uncertainty. They show that by representing the policy with a NN with recurrent units, the policy can integrate noisy observations to generate grasp actions. These approaches demonstrate the potential of model-free policies to scale to arbitrary large state spaces and with long observation history. Nevertheless, they remain focused on single object manipulation and do not ensure long-term reasoning over the physics.

3.2.3 Searching in Clutter Through Manipulation

The goal of our work is most aligned with the objective of Danielczuk et al. [25]. They define it as "Mechanical Search", a long sequence of actions for retrieving a target object from a cluttered environment within a fixed task horizon while minimizing time. They propose a data-driven framework for detecting then performing either push, suction, or grasp actions until the target object is found. They tackle top-down bin decluttering by removing obstructing objects until the target is reachable. Such an approach requires a separate storage space to hold obstructing objects. To address environments where a separate storage space is not available, Gupta et al. [36] interleaves look-ahead planning with object manipulation on a shelf. They propose moving objects to unoccupied spaces within the same shelf to increase scene visibility from a fixed camera view angle.

In a similar shelf setup, Dogar et al. [28] use a heuristic to plan pushing actions until the target object is found. The approaches stated so far, perform the search by manipulating one object at a time, avoiding sequential reasoning over multi-contact physics. Avoiding all obstacles remains, however, impossible (and often undesirable) in many partially observable and cluttered environments.

Most recently, Novkovic et al. [86] propose a closed-loop decision making scheme for generating push actions in a multi-contact physics environment with a top-mounted camera. Their approach relies on encoding the observation history in a discretized representation of the environment. The encoding is used by an RL trained policy to generate the next push action for revealing hidden spaces. We adopt a similar decision making scheme, but we avoid the limitations of encoding the observation history in a discretized representation. Instead, we rely on the NN's recurrent units to capture the observation history.

3.3 Related Work Summary

The related work on manipulation in clutter reveals the advantages and drawbacks of state-of-the-art approaches and the challenges that are yet to be solved. Model-based approaches leverage available models to solve problems that require long-term sequential decision making. To deal with the physics uncertainty that comes with the sim-to-real transfer, they avoid multi-contact physics whenever possible while also exploiting uncertainty reducing actions. They are, however, computationally expensive as simulating the physics remains the main computational bottleneck for real-time applications.

Model-free approaches are gaining an increasing momentum in manipulation applications. They offload many of the hard design decisions that the algorithm designer has to make to the learning process. They offer a reactive and generalizable behaviour best suited for time-critical applications. The training process benefits from environments where actions have an immediate measurable effect manifested by a dense reward function. They have been best deployed for single object manipulations.

More recently, hybrid approaches that combine model-free approaches with look-ahead planning are overcoming many of the challenges associated with long-term sequential reasoning. Their deployment to manipulation in clutter, however, remains limited. Overcoming the computational cost of simulating the physics remains an open challenge. Further, the stability of the learning algorithms that enables these hybrid approaches are also challenged by balancing exploration and exploitation to avoid premature convergence to a sub-optimal behaviour.

Extending existing works to also account for occlusions adds to the complexity of the problem. Approaches that rely on neural networks (NN) with recurrent units are leading the way in applications that require acting under uncertainty. We build on many recent breakthroughs to address the specificity of acting under physics and occlusion uncertainty.

Chapter 4

Learning to Act with Receding Horizon Planning

4.1 Introduction

Many real-world manipulation tasks would benefit from robots that can operate in unstructured and cluttered spaces with uncertain dynamics. Examples include retrieving an object from a warehouse shelf or rearranging an item in a food display rack. To operate in such environments, a robot is expected to react, and even leverage, multi-object physics interactions in real-time. By way of illustration, consider the example shown in Fig. 4.1, where a robot is tasked with manipulating, on a planar space, the orange fruit to the target region. Accomplishing this task involves interpreting the task goal, representing the environment, reasoning over the environment's physics, and executing in real-time a long sequence of prehensile and non-prehensile actions while satisfying the constraint of not dropping any of the other objects. In this chapter, we focus on building the theoretical foundation and experimental evaluation for tackling physics-based manipulation in environments with no occlusions.

4.1.1 Planning and Closed-Loop Execution

The multi-step, sequential, and high dimensional nature of such tasks makes planning-based approaches an attractive option for solving the problem. There has been significant recent interest in sampling-based planning for manipulation tasks in clutter, and impressive planners have been proposed [39, 62, 60]. Plans are generated off-line in a physics simulator and then executed in open-loop. Real-world execution, however, still poses great challenges. The real motion of the objects can differ significantly from the motion predicted by the planners. The main difficulty is due to the inevitable inaccuracy in the physics model used by the planners. This inaccuracy is exacerbated particularly when multiple

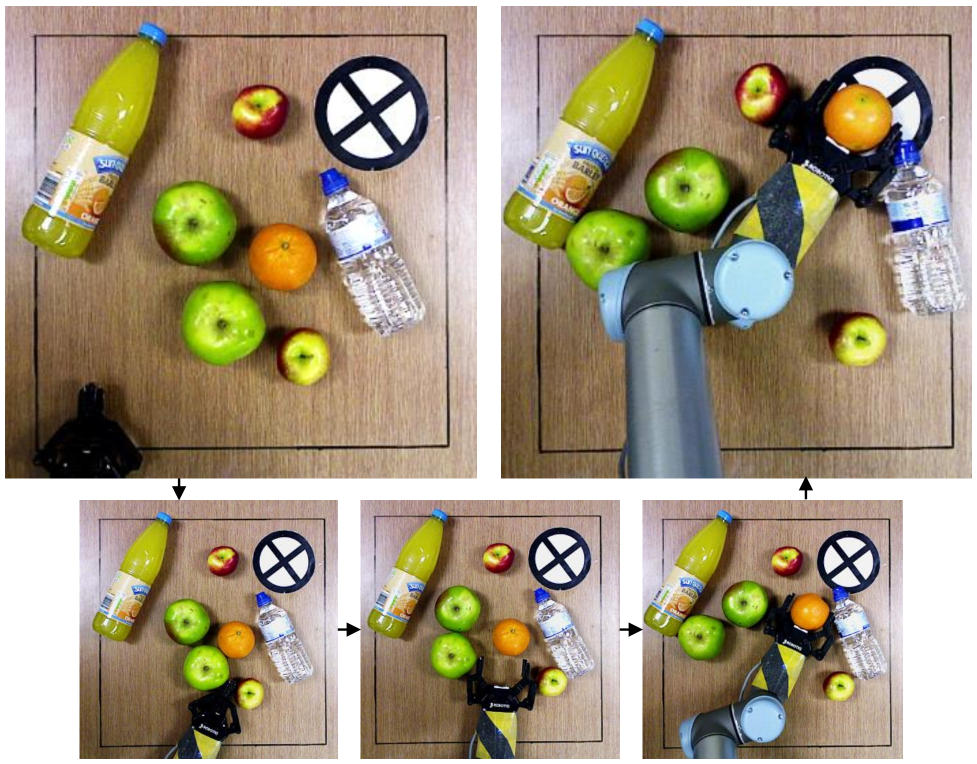


Figure 4.1: The robot is tasked with moving, whithin the surface edges, the orange fruit to the target region using prehensile and non-prehensile planar actions.

objects are in contact, which is common in the application domains mentioned above.

The inaccuracies in modelling object-to-object, object-to-surface, and object-to-robot interactions can be overcome with closed-loop decision making [69]. By interleaving planning and execution, a plan is continuously updated to address real-world observations. In this approach, a sequence of actions is planned, but only the first action in this sequence is executed. Then, the current state is updated by observing the environment, after which another sequence of actions is planned, and the routine is repeated. This idea is commonly used in domains that involve uncertainty and underlies many similar methods with different names, among which: rolling horizon planning, receding horizon control, and model predictive control.

Re-planning makes the computation cost of sampling-based planners impractical for real-time applications as they treat every new planning instance independently of previous experiences and they also have to solve the problem all the way to the goal. Instead, shortening the planning horizon can reduce the computation cost and mitigate the uncertainty associated with manipulation in clutter.

Planning up to a short horizon requires (*i*) an efficient local search strategy, (*ii*) and a cost-to-go function expressing the expected cost for the rest of the plan beyond the horizon. Receding Horizon Planning (RHP) is a planning framework aimed at continuous planning with a limited look-ahead [76]. We adopt this paradigm, and propose a learning-based approach for a heuristic-guide RHP to run in near real-time in a physics environment. RHP relies on its heuristic to perform the local search using the forward model of a physics simulator, and to evaluate the potential consequences of finite sequences of actions, before executing the first action of a chosen sequence [57]. In physics-rich domains, defining such a heuristic is a challenge on its own. In addition to enabling informed decision making by RHP, the heuristic must ensure generalization and transferability to different task and environment settings.

4.1.2 End-to-End Closed-Loop Execution

Another framework that is attracting momentous interest in real-world manipulation is the use of model-free end-to-end systems [121, 109, 95, 49]. The problem is formulated as learning a direct mapping function from the current real-world sensory data, typically RGB images, to control actions. By using images for state representation, this framework enables greater *generalization* over the geometric features of the environment. It also relieves the algorithm designer from manually having to define what features are relevant for the task which might hinder the robot from leveraging the full dynamics of the environment.

The full potential of end-to-end systems is made possible by the use of Neural Networks (NN) as function approximators. Deep Reinforcement Learning (RL) has been successfully used for end-to-end skill learning in large and continuous state spaces. Training end-to-end systems requires a substantial amount of exploration to cover the large variations observed in an unstructured image-based representation of the state. Exploration in end-to-end systems particularly benefits from an articulated description of the solution, also known as reward shaping. For example, by receiving intermediate rewards for aligning the gripper with the desired object and target destination, or for reducing the distance between the desired object and target destination. Inevitably, defining a dense reward function for a manipulation in clutter task introduces human biases on how to solve the problem, potentially limiting performance. Solutions to the task are highly sensitive to the exact configuration of the clutter. In some configurations, the best strategy might be to directly push the desired object through the clutter, whereas in a different configuration, it might be better to clear the robot path from obstacles before reaching for the desired object.

Avoiding reward shaping, and instead opting for sparse and delayed rewards, places a higher burden on the NN to reason over the sequential nature of the problem. It is significantly more challenging, however, for the NN to capture the complexity introduced by non-linear and non-continuous dynamics of the physics environment whilst proposing actions with long-term consequences.

When a model is available, solving problems with sparse reward functions can benefit enormously from incorporating look-ahead planning in the learning process and at execution time [7, 72, 69]. This has shown to compensate for inaccuracies in the learned utility of state-action pairs [76]. At a slightly higher computation cost, problems that require *sequential* decision making in a relatively large and continuous space, can be approached in near real-time.

4.1.3 VisualRHP

In this chapter, we introduce **VisualRHP**, a novel approach that combines the generalization advantages of image-based learning with the sequential reasoning of RHP. We design a framework around VisualRHP to interleave real-world execution with abstract image-based look-ahead planning in a physics simulator.

At execution time, the real-world state is abstracted to a compact colour-labelled image representation rendered from the simulator state. In the simulator, VisualRHP uses a learned image-based heuristic that acts on the abstract state representation to efficiently solve a short horizon approximation to a multi-step sequential decision making problem. The generated action is resolved to the robot joint space and executed in the real world.

Two key features stand at the core of VisualRHP. First, is an abstract image-based representation of the state. It uses colour labelling to specify the functionality of the different items in the scene. Second, is an image-based heuristic learned in simulation prior to the execution phase. We present a discrete and a continuous action space version of the heuristic together with their corresponding learning algorithms. We highlight in this chapter the novel formulation we introduced to existing IL and RL algorithms to improve on the stability of learning algorithms in an image space with sparse rewards.

The contributions of this chapter are (*i*) a near real-time framework which integrates real-world execution with physics-based look-ahead planning in simulation (Fig. 4.2 and Sec. 4.2), (*ii*) an abstract image-based representation which uses colour-labelling to represent the state of the manipulation task (Sec. 4.4.1), and (*iii*) the VisualRHP algorithm which uses an image-based heuristic to run

RHP in discrete and continuous action spaces (Sec. 4.4.2). While our overall framework is agnostic to the particular way the heuristic is learned, (*iv*) we introduce a stable heuristic learning approach in Sec. 4.5.

These contributions culminate in VisualRHP acquiring prehensile and non-prehensile manipulation skills that generalize to novel environments, for instance with a different number of objects and shapes, and transfer to tasks with a different desired object to manipulate in the real world. VisualRHP is also capable of reasoning over complex physics interactions, such as avoiding objects falling off the edge even in multi-contact environments.

4.2 Framework

The framework presented in this chapter is composed of two phases: first, the heuristic learning phase which takes place in simulation, then the execution phase in which VisualRHP interleaves the real-world actions with the physics simulator at run time. We will begin by describing the execution phase (Sec. 4.4) assuming learning has already taken place, and then characterize the learning phase that makes it possible (Sec. 4.5).

The execution phase consists of a closed-loop control scheme (Fig. 4.2-Bottom). It dynamically maps the state of the real world to the simulator, where an action is selected and then executed by the real robot. The control scheme cycle starts by processing sensory data from the real world to produce a corresponding state in the physics simulator (Sec. 4.4.1). Then, in the simulator, VisualRHP performs a local look-ahead search by simulating multiple physics-based roll-outs up to a certain horizon. Each roll-out is evaluated by computing its expected *return*. In this process, a heuristic is required for (*i*) guiding the local search towards parts of the state space with high expected *returns*, (*ii*) and for estimating the expected *return* beyond the horizon state (Sec. 4.4.2). VisualRHP returns the *first* action of the best roll-out. Lastly, the selected action is resolved to the joint motion of the real robot (Sec. 4.4.3).

The learning phase consists of training a NN to be used as the Visual-RHP heuristic (Fig. 4.2-Top). The NN in the discrete action space is trained to approximate the optimal action-value function, whereas the NN in the continuous action space is trained to approximate the optimal policy as well as the value function of the learned policy. For both the discrete and continuous action space implementations, we use deep IL (Sec. 4.5.2) followed by deep RL (Sec. 4.5.3) to train the NN.

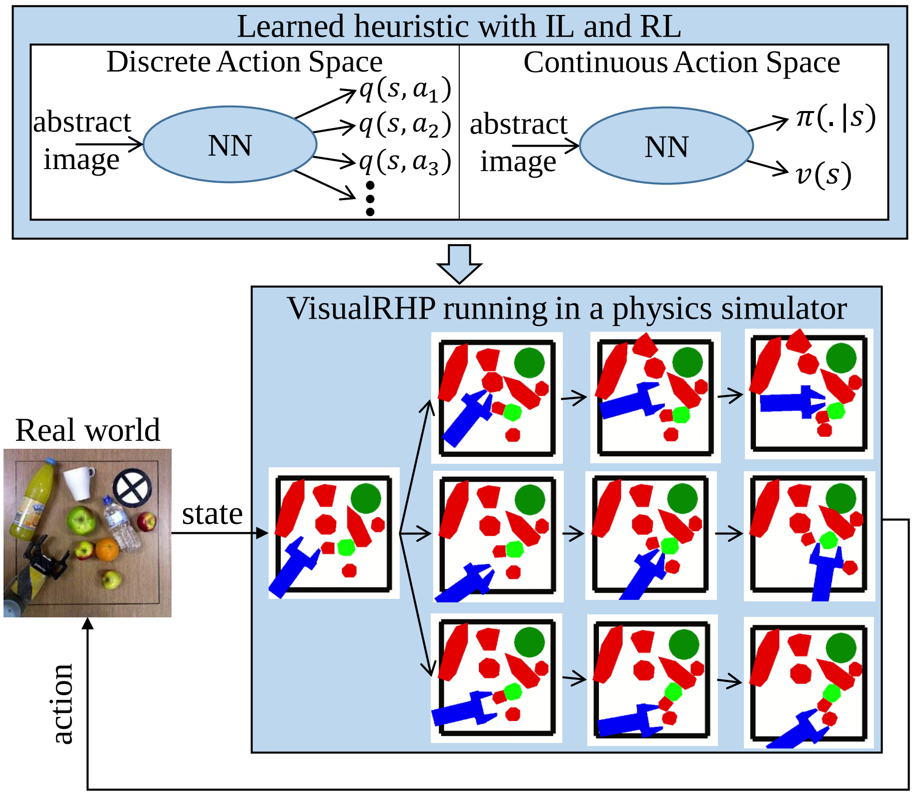


Figure 4.2: Framework overview. Example with the orange fruit as the desired object.

4.3 Problem Formulation

We formalize the problem as an MDP, represented as a tuple $M = \langle S, A, T, r, \gamma \rangle$ where S is the set of the environment variables such that a state $s \in S$ at time t is given by $s_t = [Rob, Obj^{des}, \ldots, Obj^{m}, edges, tarReg]$, where:

- Rob is the Cartesian pose, gripper state (open or closed), and shape of the robot end-effector.
- Obj^{i}: is the Cartesian pose, shape, and type (e. g., apple, cup, etc.) of object i. We refer to the desired object by Obj^{des}, and the environment can include up to a total of m objects.
- $edges$ represents the set of coordinates defining the edges of the manipulation surface.
- $tarReg$ is the location $(x, y)^{\text{tarReg}}$ and radius rad^{tarReg} of the circular target region with $(x, y)^{\text{tarReg}}$ being within the surface $edges$.

Further, A is the set of actions that the robot can execute for moving over a planar surface and for closing and opening the gripper. A can be either defined over a discrete or a continuous space; $T : S \times A \times S \to [0, 1]$ is the transition probability function (Eq. 2.2), $r : S \times A \times S \to \mathbb{R}$ is the reward function (Eq. 2.4); $\gamma \in [0, 1)$ is the discount factor.

We denote as S_{val}, the set of *valid* states where all the of objects lie within the surface edges. The set of *invalid* states, S_{inval}, consists of the states where any of the objects are located outside of the surface edges. The goal states, $S_g \subset S_{val}$, is identified by the arrangement where the desired object is in the target region, satisfying:

$$||(x, y)^{\text{des}} - (x, y)^{\text{tarReg}}|| \leqslant rad^{\text{tarReg}}.$$

Intuitively, it is expected from the robot to manipulate the desired object to the target region with the least number of actions without violating the surface edge constraints. We avoid shaping the reward function in order not to skew the robot's behaviour towards any preconceived human intuition which might artificially limit the *return*. Instead, we opt for a constant negative reward per action casting the problem as a shortest path. When an object trespasses a surface edge, the task is terminated and an additional large negative reward is received. The robot's goal is to maximise the expected *return* by manipulating the objects in the environment along a sequence of states $\langle s_t \rangle_{t=0}^{L}\, s.t.\, s_t \in S_{\text{val}}$ for t in $[0, L]$, where $L + 1$ is the number of traversed states, from $s_0 \in S_{\text{val}}$ to $s_L \in S_g$.

As presented in Chap. 2, the optimal policy can be approximated either indirectly with value iteration methods or directly with policy iteration methods. **In the discrete action space**, we use value iteration to approximate the optimal action-value function. A NN parametrized by a vector ψ, $q_\psi(s, a)$, is trained with off-policy value iteration to approximate the $q^*(s, a)$ over a continuous state space:

$$q^*(s, a) = \int_{s' \in S} T(s, a, s')[r(s, a, s') + \gamma \max_{a'} q^*(s', a')]ds', \qquad (4.1)$$

In the continuous action space, the parameters of the policy can be directly learned with an Actor-Critic method. It uses value and policy iterations until it converges to the optimal policy $\pi^*(a|s)$. The performance of a θ-parametrized stochastic policy $\pi_\theta(a|s)$, i. e., the Actor, is iteratively improved by evaluating its performance with respect to a ϕ-parametrized value function $v_\phi(s)$, i. e., the Critic (see Eq. 2.24 and Eq. 2.28). $\pi_\theta(a|s)$ models a multi-dimensional Gaussian distribution $\mathcal{N}(\mu_\theta, \sigma_\theta)$ from which actions can be sampled.

It is common, in applications where the visual input is significant, for the value function and the policy to share some of their parameters, i. e., ϕ and θ, mostly the convolutional part of the NN. The motivation behind shared parameters is that the low level features useful for estimating the value function could also be useful for modelling the policy and vice versa. Furthermore, optimizing the parameters of the value function and policy together acts as a regularizing element which leads to greater stability in the learning process [79, 101]. In the rest of this chapter, we will refer with θ to the parameters of the Actor-Critic NN.

Instead of acting greedily on the learned action-value function and policy, using them as heuristics for RHP mitigates inaccuracies in their learned approximations of the optimal behaviour. The horizon can mitigate these inaccuracies, by ranging from infinity, with the robot planning all the way to the goal, to zero, with the robot acting greedily on the learned behaviour. With an infinite horizon, the action-value and value function are ignored in the discrete and continuous actions spaces, respectively. While with zero depth horizon, the behaviour depends entirely on the learned approximations. In practice, a short but non-zero horizon takes advantage of both the planner and the learned approximations without relying on either one entirely.

In this chapter, we assume (*i*) a quasi-static physics model with limits on the velocity of the robot motion. (*ii*) Further, since this chapter does not consider occlusions, we assume that all objects in the scene to be visible at any point in time. Although some information on the underlying state is lost in the image-based representation, we treat the environment in this chapter as fully observable as the information contained in the image-based representation is sufficient to solve the manipulation task. We also assume (*iii*) discrete time steps and (*iv*) that the actions are parallel to the manipulation surface in the planar Cartesian space of the gripper. (*v*) We do not consider access to a separate storage space.

4.4 Execution Phase

In this section, we present a closed-loop control scheme (Fig. 4.2-Bottom) that balances real-time execution with long-term goal-oriented actions. First, we explain the mapping from the real world to the simulator and the colour-labelled abstract image used for state representation (Sec. 4.4.1), and then we explain how VisualRHP uses a heuristic acting on the abstract state representation to

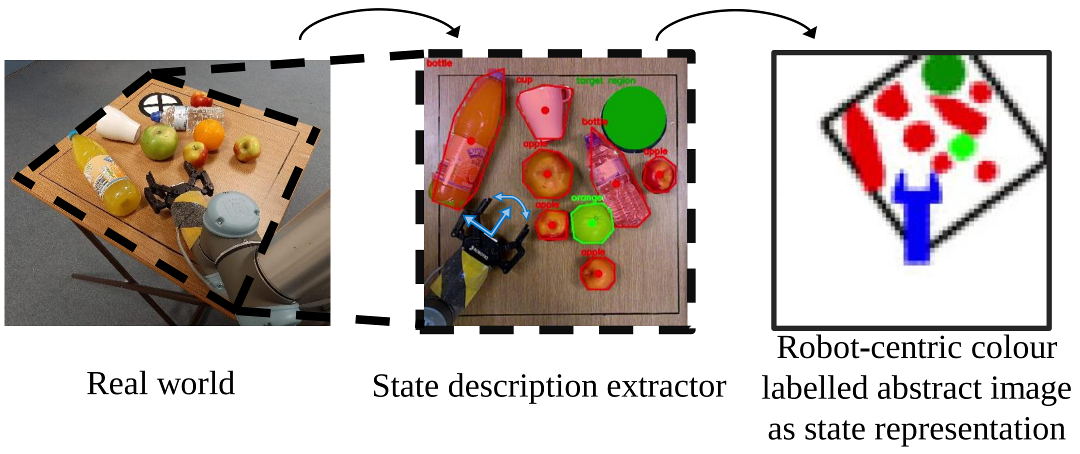

Real world
State description extractor
Robot-centric colour labelled abstract image as state representation

Figure 4.3: Mapping the real-world state to a colour-labelled abstract image-based state representation for a task where the orange fruit is the desired object.

suggest actions (Sec. 4.4.2). Once an action is chosen, it is resolved back to the robot joint space and executed by the real robot (Sec. 4.4.3).

4.4.1 Mapping the Real World to an Abstract Representation

Our mapping captures the state of the real world while leaving out information that is not relevant for the task, such as object texture, background colour, lighting sources, etc. As illustrated in Fig. 4.3-Middle, we apply *instance segmentation* on real-world images to detect the real-world state. This operation is performed using *Mask R-CNN* [40] which also identifies the type of each detected object. The object type is used to identify the desired object and the obstacle objects. We detect the target region *tarReg* using simple template matching. We localize with forwards kinematics the end-effector pose over the planar Cartesian space and the gripper state. The simulator uses this information to create objects with the same contour shapes as in the real world. In this thesis, the shape of the end-effector and the surface edges are pre-loaded into the physics model as they are kept unchanged from one task to another[1].

The input to the NN is in the form of an image rendered from the state of the physics simulator. The objects in the simulator are colour labelled based on their functionality. As in the example of Fig. 4.3-Right, the desired object is always of the same colour (light green), all other objects are of another common colour (red). The same applies to the end-effector (blue), the surface edges (black), the

[1]If required, the detection of the shape of the end-effector and the surface edges can also be automated.

target region (dark green), and the scene background colour (white) across all task instances. The advantage of using colour labelling is that it allows for the seamless transfer of manipulation skills to different desired objects. Different task settings can be mapped to the same representation space where VisualRHP is performed. For example, any real-world object can be assigned the colour of the desired object and the NN will treat it as such.

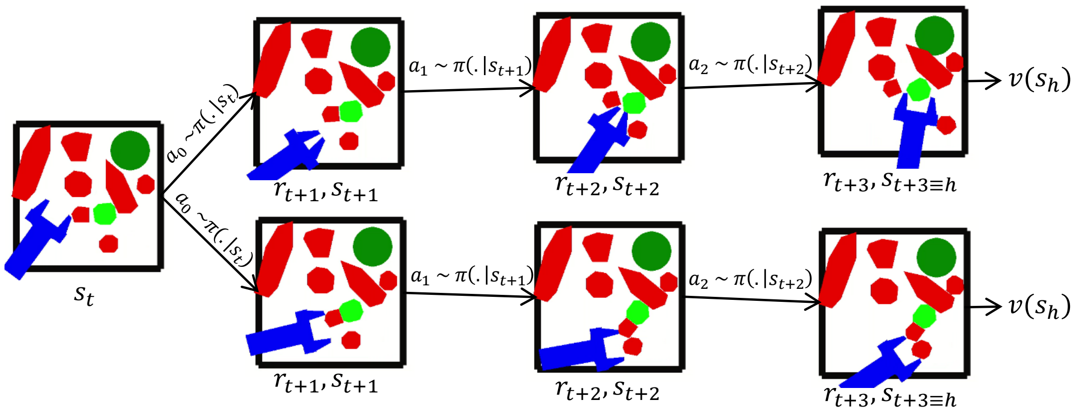
Furthermore, the abstract images are robot centric, i. e., the image tracks the robot from a top-view perspective (Fig. 4.3-right). The robot centric view reduces the amount of data required by the learning algorithm due to the symmetry of the scene when compared to a fixed view.

An equally important advantage of using this abstract representation is in reference to the size of the NN architecture need to operate on it. For real-time operation, it is essential to have a small network to ensure fast inference time. The abstract representation allows for a much smaller convolutional part of the NN to capture relevant features when compared to the convolutional architectures designed to handle real-world or realistically rendered images [16]. Consider, for example, a heuristic-based planner that queries the NN 10 times per action selection. The inference time for a NN designed to operate on the compact representation is around 0.003 *seconds* (subject to the available CPU and GPU power), whereas it is around 1 *second* for a NN operating on high fidelity images [23, 16]. This entails that, per action selection, the planner using a small architecture would spend 0.03 *seconds* in neural computation time compared to 10 *seconds* for using a large architecture.

4.4.2 VisualRHP

VisualRHP can be seen as a combination of two processes. The first process consists of performing a local search starting from the current state of the simulator up to a certain horizon depth. However, an exhaustive search would scale badly with a horizon depth h and the size of the action set A, $\mathcal{O}(|A|^h)$ in the discrete case, and it would be inapplicable in the continuous case. Instead, as illustrated in an example in Fig. 4.4, we run a small number of n roll-outs up to a short horizon of depth h while using the heuristic as a stochastic policy to orient the roll-outs' expansion towards directions with high *return*. The stochastic policy for the **discrete action space** is the soft-max of the action-value function:

$$\pi(a|s) = \frac{exp(q_\psi(s,a)/\tau)}{\Sigma_{a_i \in A} exp(q_\psi(s,a_i)/\tau)}, \tag{4.2}$$



s_t
$a_0 \sim \pi(.|s_t)$
$a_0 \sim \pi(.|s_t)$
r_{t+1}, s_{t+1}
$a_1 \sim \pi(.|s_{t+1})$
r_{t+2}, s_{t+2}
$a_2 \sim \pi(.|s_{t+2})$
$r_{t+3}, s_{t+3 \equiv h}$
$v(s_h)$
r_{t+1}, s_{t+1}
$a_1 \sim \pi(.|s_{t+1})$
r_{t+2}, s_{t+2}
$a_2 \sim \pi(.|s_{t+2})$
$r_{t+3}, s_{t+3 \equiv h}$
$v(s_h)$

Figure 4.4: VisualRHP example of $n = 2$ roll-outs with $h = 3$ horizon depth in continuous action space.

where τ is the temperature parameter, while it is represented explicitly in the **continuous action space** with $a \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$. The stochastic policy would favour exploring actions that are more likely to lead to higher *return*.

The second process consists of computing the expected h-step *return* of a roll-out, using the first h rewards generated by the model and the expected *return* beyond the horizon state s_h. In discrete action space, the value of a horizon state s_h is computed as $v(s_h) = \max_a q_\psi(s_h, a)$, whereas in the continuous action space it is the output of the value head of the NN: $v(s_h) = v_\theta(s_h)$. The *return* of a roll-out is therefore computed as:

$$R_{t:h} = r_1 + \gamma r_2 + \ldots + \gamma^{h-1} r_h + \gamma^h v(s_h). \tag{4.3}$$

The VisualRHP algorithm, detailed in Alg. 1, returns the *first* action of the roll-out that obtained the highest *return* $R_{t:h}$. In this algorithm, the heuristic plays two roles: it informs the search through sampling, and as an approximation of the rewards that are not sampled from the model.

4.4.3 From the Simulator to the Real World

Although VisualRHP can be extended to arbitrary dimensions of the action space, we limit our implementation to actions parallel to the manipulation surface in the planar Cartesian space of the gripper. It is safe to assume that a sequence of actions performed over a small planar Cartesian space can be resolved to the joint motion of a redundant manipulator. Kinematic singularities can be avoided using an inverse kinematics solver based on non-linear optimization [65]. Therefore, the action returned by VisualRHP is resolved to the robot

Algorithm 1: VisualRHP(s_{cur}, n, h)

```
Input: Current state $s_{cur}$, number of roll-outs $n$, horizon depth $h$
Output: action $a$
$RolloutsReturn \leftarrow [\,]$; $FirstAction \leftarrow [\,]$
for $i$ = 1,2, ..., $n$ do
    $s \leftarrow$ setSimulatorTo($s_{cur}$)
    $R \leftarrow 0$
    for $j$ = 1,2, ..., $h$ do
        $a \sim \pi(.|s)$
        if $j$ is 1 then
            $FirstAction$.append($a$)
        $s, r \leftarrow$ simulatePhysics($s, a$)
        $R \leftarrow R + \gamma^{j-1} r$
        if isTerminal(s) then
            break
    if not isTerminal(s) then
        $R \leftarrow R + \gamma^{h} v(s)$
    $RolloutsReturn$.append($R$)
return $FirstAction$[argmax($RolloutsReturn$)]
```

joint motion and executed in the real world. In case a pose is not reachable, the action is not executed and VisualRHP is queried for another action.

4.5 Learning Phase

As described in Sec. 4.4, the performance of VisualRHP is dictated by the quality of its heuristic. We are interested in approximating the optimal heuristic for VisualRHP, whether in the form of the optimal action-value function or in the form of the optimal policy and value function for the discrete and continuous actions space, respectively. We formulate the heuristic learning as an RL problem where the robot is trained in simulation to maximize the *return* (Eq. 2.5).

Training a randomly seeded RL algorithm in a cluttered environment under edge constraints is unlikely to converge to a good solution as transition samples leading to the goal will not be observed enough times. For this reason, we use a probabilistically complete sampling-based planner as a starting point for the search. The NN is jump-started with IL from demonstrations generated by this planner. With enough knowledge captured from demonstration, RL would have better chances at convergence and it would require a relatively small number of transition samples to refine the robot's behaviour. Therefore, we divided the heuristic learning process into three sequential steps: (i) generating demonstrations, (*ii*) IL, and (*iii*) RL. We detail each of these steps in discrete and continuous action spaces.

4.5.1 Generating Demonstrations

Sampling-based planners provide a probabilistically complete tool to solve complex planning problems in high dimensional state and action spaces without requiring a hand-crafted or domain-dependent heuristic. In particular, Kino-dynamic planners are one family of the sampling-based Rapidly exploring Random Trees planners (Kino-dynamic RRT), specific for solving planning problems that involve dynamic interactions. We implement a discrete and a continuous version of the state-of-the-art Kino-dynamic planner [39] used for solving planning problems on physics-based manipulation in clutter. In the discrete action space, RRT can expand a node in the tree along a one of the discrete actions. A node is considered fully explored once it has been expanded along all the discrete actions. In continuous action space, a node is expanded along a sampled action from the continuous action space. We set a limit to how many branches can be expanded from a node before considering that node fully explored.

We generate P task instances. Each task instance is initialized with random environment setups. This includes the location of the target region, the initial robot pose and objects' arrangement, and the shape and number of objects. Then, for each task instance p, we run the Kino-dynamic planner to generate a demonstration of the form $\langle a_0^p, \ldots, a_{L-1}^p \rangle$ with state sequence $\langle s_0^p, \ldots, s_L^p \rangle$.

4.5.2 Imitation Learning

In this section, we show how to use the generated demonstrations to train the NN to reproduce the behaviour of the Kino-dynamic RRT.

In the discrete action space, the goal is to learn the action-value function from the transition samples observed in the demonstrations. We train the NN to predict the value of the actions selected by the Kino-dynamic planner at the visited states by minimizing the mean squared error w. r. t. the Monte-Carlo target:

$$q^{tar}(s_l^p, a_l^p) = \sum_{k=0}^{L-l-1} \gamma^k r_{l+k+1}, \tag{4.4}$$

where p stands for the index of the demonstration and l for the index of the state-action pair in that demonstration.

While the NN, trained as above, learns to predict the action-values for the actions selected along the visited states, the values predicted by the NN for actions that have not been selected by the Kino-dynamic planner along the visited states can be arbitrary. As a result of function approximation, these actions must have a value even though they do not appear in the training set. The value can converge to an arbitrary number determined by the effect of the target action value of the traversed state-action transitions. A possible undesirable effect is that the values of the actions not selected by the Kino-dynamic planner can be higher than the selected one. This can later cause an action that was not favoured by the Kino-dynamic planner to look more favourable to VisualRHP that uses the action-value function as a heuristic (Eq. 4.2).

To counteract this phenomenon, we use for the unselected actions a target value that is lower than the value of the selected actions. The minimum allowed difference between the value of the selected action and the other actions is referred to as the *value margin* function (λ) [43, 93]. We propose a definition of λ driven by the observation that, in the domain of planar manipulation tasks, a mistake is in most cases not irreparable, but can be overcome through a number of η additional actions of fixed cost $r_{cost} > 0$, such that,

$$\lambda = \sum_{k=1}^{\eta} \gamma^{L-l-1+k} r_{cost}. \tag{4.5}$$

This definition scales λ down the further away s_l is from the final state in a demonstration.

Hence, we also minimize the mean squared error between the predicted action-value of the unselected actions and the λ penalized action-value target at visited state:

$$q^{tar}(s_l^p, a_u^p) = \begin{cases} q^{tar}(s_l^p, a_l^p) - \lambda, & \text{if } q_\psi(s_l^p, a_u^p) \geq q^{tar}(s_l^p, a_l^p) - \lambda \\ q_\psi(s_l^p, a_u^p), & \text{otherwise,} \end{cases} \tag{4.6}$$

where $a_u \in A \setminus \{a_l\}$ is an unselected action. If the value that the NN converges to does not favour an unselected action then we leave it unchanged. Lastly, we add an L_2 regularization term to avoid over-fitting on the demonstrations.

In continuous Action space, the value head of the θ-parameterized NN is trained to estimate the future sum of discounted rewards that are expected to be collected if RRT were to be engaged at the current state. The update target is similar to Eq. 4.4 but it is computed over a state instead of a state-action pair:

$$v^{tar}(s_l^p) = \sum_{k=0}^{L-l-1} \gamma^k r_{l+k+1}. \tag{4.7}$$

Furthermore, we train the policy head of the θ-parameterized NN to estimate the action distribution over states visited in the demonstrations while penalizing high entropy distributions. One way this can be achieved is by minimizing the loss function that combines the policy and the value function:

$$\begin{aligned}\mathcal{L}_{\text{il}}(\theta) = \mathbb{E}_{s_l^p, a_l^p}[-\ & \Psi log \pi_\theta(a_l^p|s_l^p) \\ +\ & c_1\ (v^{tar}(s_l^p) - v_\theta(s_l^p))^2 \\ +\ & c_2\ H(\pi_\theta(.|s_l^p))], \end{aligned} \tag{4.8}$$

where c_1 and c_2 are hyper-parameters. Ψ is a positive constant indicating a positive advantage of action a_l^p at state s_l^p. The first term on the right of Eq. 4.8 increases the likelihood of the actions selected by the planner at the visited states. The second term updates the value estimate w. r. t. the Monte-Carlo target, and the last term is an entropy penalty added to reduce the probability of unselected actions at visited states[2].

We experimented with NNs of different sizes and expressive power in both the discrete and continuous action cases, but none could reliably represent the behaviour of the planner over a large number of task instances. As a consequence, we show in the next section how the information compiled in the NN can be further optimized to play a valuable role when used as the VisualRHP heuristic.

4.5.3 Reinforcement Learning

So far, the knowledge encapsulated in the NN has two shortcomings: first, the plans generated by the Kino-dynamic planner are, in general, sub-optimal; and second, some information is lost in the approximation by the NN, with consequent performance degradation with respect to the Kino-dynamic planner. To overcome these problems, we use RL to (i) improve the NN to better estimate the *return* of the optimal policy and/or to better estimate the optimal policy and to (ii) learn the value of the not experienced state-action transitions.

ϵ-VisualRHP as the RL Policy in Discrete Action Space: Operating in the discrete action space allows for a straightforward implementation of an off-policy RL algorithm. Off-policy makes it possible to leverage RHP in the exploration policy of the RL algorithm to exploit actions that are more likely to

[2]It is also possible to penalize low entropy instead. A resulting policy with high entropy from IL will be less consistent in replicating the behaviour observed in the demonstrations, but it will ensure that RL starts with higher exploration.

lead to the goal. We implement the DQN with VisualRHP-based exploration policy.

We initialize the NN with the IL trained ψ parameters. We also initialize a large buffer D^{replay} with the demonstration transition samples. Further, we formulate a novel exploration policy, that we call ϵ-VisualRHP, which selects a random action with probability ϵ and with probability $1-\epsilon$ the policy queries VisualRHP for an action. Compared to ϵ-Greedy where with probability $1-\epsilon$ a greedy action is selected based on the action-values of the current state, i. e., $a = \arg\max_{a \in A} q(s,a)$, we found that focusing the search towards the goal by augmenting the RL policy with VisualRHP reduces the chances of the action-value function from diverging which is a common problem in RL when used in conjunction with NNs as function approximators. The robot uses ϵ-VisualRHP to collect transition samples over task instances initialized with random environment setups. Throughout the data collection process, the robot stores the newly collected transition samples in the buffer D^{replay}. The old samples in the buffer get gradually replaced by new ones that are collected with ϵ-VisualRHP. When enough new transition samples are collected, the NN is updated by running a batch optimization over randomly sampled transitions from D^{replay}. The loss function over a batch $B = \{\langle s_i, a_i, r_i, s'_i \rangle_{i=1}^{M}\}$ of M transition samples is defined as:

$$\mathcal{L}_q(\psi) = \frac{1}{M}\sum_{i=1}^{M}(r_i + \gamma \max_{a'} q_\psi(s'_i, a') - q_\psi(s_i, a_i))^2. \tag{4.9}$$

An L_2 regularization loss is also added on the network parameters.

As mentioned in Sec. 2.4, D^{replay} helps in counteracting the high correlation in the transition samples and in using the transition samples more efficiently. Additionally, in transitioning from IL to RL, using the D^{replay} leads to a smooth change in the action-value function, and consequently in the robot behaviour, as it shifts from estimating the *return* based on the average behaviour observed in the Kino-dynamic demonstrations to estimating the optimal action-value function.

Critic Correction Conditioned Policy Optimization (C3PO) extension for A2C in Continuous Action Space: In the continuous action space, we propose reformulating the loss function for A2C style on-policy algorithms. The motivation is to improve on the stability of existing algorithms that use NNs with shared parameters and are highly sensitive to changes in the policy, more specifically to help avoid *catastrophic forgetting.*

One way of implementing A2C with shared parameters is by (i) running multiple simulation environments in parallel, each with random task parameterization and with the same copy of the NN. In each environment, the robot is controlled by π_θ. (*ii*) Once all the transition samples in D^{replay} are replaced with the ones collected with π_θ, the parameters θ of the NN are stored as θ_{old}. (*iii*) Then, the policy and the value function are updated together by minimizing in batches $B = \{\langle s_i, a_i, r_i, s'_i \rangle_{i=1}^{M}\}$ the loss function w. r. t. θ:

$$\begin{aligned}\mathcal{L}_{\text{actor-critic}}(\theta) = \frac{1}{M}\sum_{i=1}^{M} &- Adv(s_i, r_i, s'_i)\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)} \\ &+ c_3\ (r_i + \gamma v_{\theta_{old}}(s'_i) - v_\theta(s_i))^2 \\ &- c_4\ H(\pi_\theta(.|s_i)), \end{aligned} \tag{4.10}$$

where c_3 and c_4 are hyper-parameters. We recall the *Adv* as the advantage function estimate:

$$Adv(s_i, r_i, s'_i) = r_i + \gamma v_{\theta_{old}}(s'_i) - v_{\theta_{old}}(s_i), \tag{4.11}$$

computed w. r. t. the learned baseline, namely the value function $v_{\theta_{old}}$ (see Eq. 2.28). H is the entropy term added to encourage exploration by limiting the premature convergence to a sub-optimal policy [79].

The problem with this formulation of the loss function is that, in the policy update component of Eq. 4.10, *Adv* is using a baseline $v_{\theta_{old}}$ that has not yet been updated to capture the value of the policy $\pi_{\theta_{old}}$ used to collect the samples in D^{replay}. This means that the baseline for updating the policy is always one step behind the policy used to collect the data. Put differently, whilst the critic has not yet been updated over the latest round of collected data, it is being used as a baseline for updating the actor that generated the data. This often goes unnoticed as the policy updates are usually bounded to small changes in policy gradient based algorithms. For example, PPO uses the *clip* function on the ratio $\frac{\pi_\theta}{\pi_{\theta_{old}}}$ (see Eq. 2.30) [101], whereas TRPO imposes a constraint on the KL-divergence between the new policy and the old policy [99]. However, in environments with non-linear and non-continuous dynamics, such as the case in cluttered environments, even a very small change in the policy can possibly entail a drastic change in the value function cascading into *catastrophic forgetting.*

To overcome this problem, we propose updating $v_{\theta_{old}}$ prior to the optimization step of Eq. 4.10, i. e., before using $v_{\theta_{old}}$ as a critic in *Adv*. The baseline is updated to improve the estimate of the value of the policy used to collect the

Algorithm 2: Condition Critic Correction Policy Optimization (C3PO) extended A2C

```
$D^{replay} \leftarrow [\ ]$
for $iteration$ = 1,2, ... do
    while $|D^{replay}| < M$ do
        for $actor$ = 1,2, ..., $N$ do
            Generate random task instance
            Run episode with policy $\pi_\theta$
            $D^{replay}$.append( $\langle s_i, a_i, r_i, s'_i \rangle_{i=0}^{L-1}$ )
    $\theta_{old} \leftarrow \theta$
    Optimize $\mathcal{L}_{\text{baseline}}$ w.r.t. $\theta$ (Eq. 4.12)
    $\theta_{old} \leftarrow \theta$
    Optimize $\mathcal{L}_{\text{actor-critic}}$ w.r.t. $\theta$ (Eq. 4.10)
    $D^{replay} \leftarrow [\ ]$
```

latest round of data, while also refraining from causing a change to the action distribution of this policy. This is achieved by first doing an update of the value function w. r. t. θ:

$$\mathcal{L}_{\text{baseline}}(\theta) = \frac{1}{M} \sum_{i=1}^{M} c_5 \left(r_i + \gamma v_{\theta_{old}}(s'_i) - v_\theta(s_i)\right)^2 + D_{KL}(\ \pi_{\theta_{old}}(.|s_i)\ ||\ \pi_\theta(.|s_i)\ ), \quad (4.12)$$

where c_5 is a hyper-parameter. The first term on the right updates the value function of the policy used to collect the transition samples. Since the policy and the value function share the same body of the NN and updating one perturbs the other, the second term on the right penalizes the KL-divergence between the action distribution of the policy used to collect the data $\pi_{\theta_{old}}$ and any resulting change in the action distribution of π_θ that might be induced by the update of the value function v_θ. This procedure, which we call Critic Correction Conditioned Policy Optimization (C3PO), is outlined in Algorithm 2. This algorithm follows the same structure of A2C algorithms with the addition of the $\mathcal{L}_{\text{baseline}}$ optimization step. Hence, C3PO can be used as an extension to state-of-the-art A2C algorithms with shared neural network parameters.

Learning a policy and value function to act as a heuristic for VisualRHP is also possible with off-policy RL algorithms, such as TD3 [33] and SAC [37]. However, since sample efficiency is not a key factor, as the NN is first optimized with IL, we use C3PO in combination with PPO as PPO is a relatively stable algorithm and does not require extensive hyper-parameter tuning.

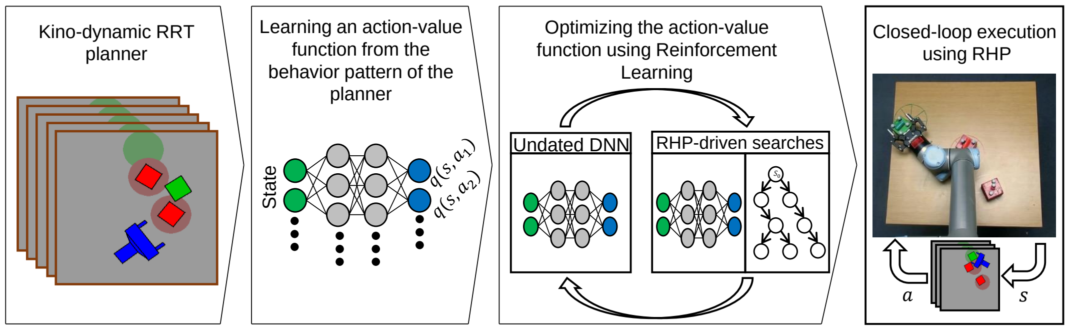

Kino-dynamic RRT planner
Learning an action-value function from the behavior pattern of the planner
State
q(s, a1)
q(s, a2)
Optimizing the action-value function using Reinforcement Learning
Undated DNN
RHP-driven searches
Closed-loop execution using RHP
a
s

Figure 4.5: $C.RHP_{das}$ training process and execution phase.

4.6 Preliminary Experiments and Evaluation

We conducted a preliminary round of experiments to evaluate the validity of combining RHP with learned heuristics in simulation and for real-world applications. The initial experiments were performed over a more basic and slightly different version of the problem that was presented in Sec. 4.3. (*i*) We removed the robot's access to prehensile actions (gripper cannot be closed) and limited the action space to discrete actions. (*ii*) The number of objects was fixed to $m = 3$ (one desired and two obstacles) of 6×6 cm square shape. (*iii*) We used a feature vector instead of images to represent the state. The feature vector was defined by the Cartesian poses of the objects in the robot frame. (*iv*) We dropped the edge constraint and replaced it with a condition that the obstacle objects must not be moved far away from their initial pose.

We call the approach used to solve this problem $CartesianRHP_{das}$, or $C.RHP_{das}$ for short. It was trained following a similar learning procedure to the one presented in Sec. 4.5 for the discrete action space. The learning procedure is illustrated in Fig. 4.5. The number and shapes of the objects was fixed throughout the learning process.

The manipulation scenario is inspired by manipulation tasks where it is desired not to significantly reshuffle obstacle objects. An illustrative example is shown in Fig. 4.6. The robot, shown in blue, has to push the green square into the target region of radius 6 cm, while having the rest of the objects placed by the end of the task as close as possible to their initial pose. The target regions are depicted by light colored circles corresponding to their designated objects.

The goal of the experiments is to evaluate the following:

- First, we measure *the effect of the number of demonstrations* (that is, demonstrations generated by the Kino-dynamic planner) on the quality of the action-value function. We show that after a certain number of

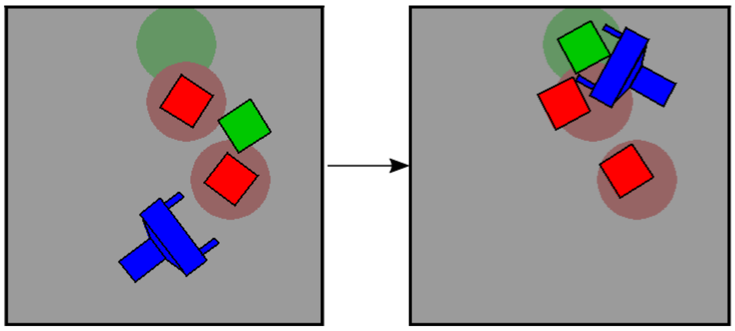


Figure 4.6: The initial configuration (left) and the final configuration (right) of an example scenario.

demonstrations from the planner, the performance of the induced policy hits a plateau.

- Second, we compare the performance of $C.RHP_{das}$ to different baseline approaches.
- Lastly, we demonstrate running $C.RHP_{das}$ in real-world scenarios.

In our simulation experiments, we modeled the world in the Box2D physics simulator [19]. The robot motion is generated by applying momentary forces to its end-effector, and waiting until the robot and objects came to a stop due to frictional damping forces. We tuned the amount of time the force is applied to the robot such that each translational action moves the gripper by a distance of around 5 cm, and each rotational action moves the gripper for around 30^o. The robot has access to 6 actions, 4 to apply a force in each of the cardinal directions, and 2 to rotate the gripper clockwise and counterclockwise in the task space.

We used the TensorFlow [2] library to build and train a feed-forward NN model consisting of 5 fully connected layers. The first 4 layers have 330, 180, 80, and 64 neurons, respectively, with *ReLU* activation function. The output layer consists of 6 neurons, one per action, with linear activation functions. The training and experiments were conducted on an Intel Xeon E5-2665 computer equipped with NVIDIA Quadro K4000 GPU card.

The Kino-dynamic planner typically needs 20 to 30 actions to solve a task. Therefore, we set 50 actions as the limit for the number of actions per task. At the end of a run, we call it a failure if the desired object is not in the target region or if any of the other objects was moved outside of its corresponding region. Otherwise, we consider it a success.

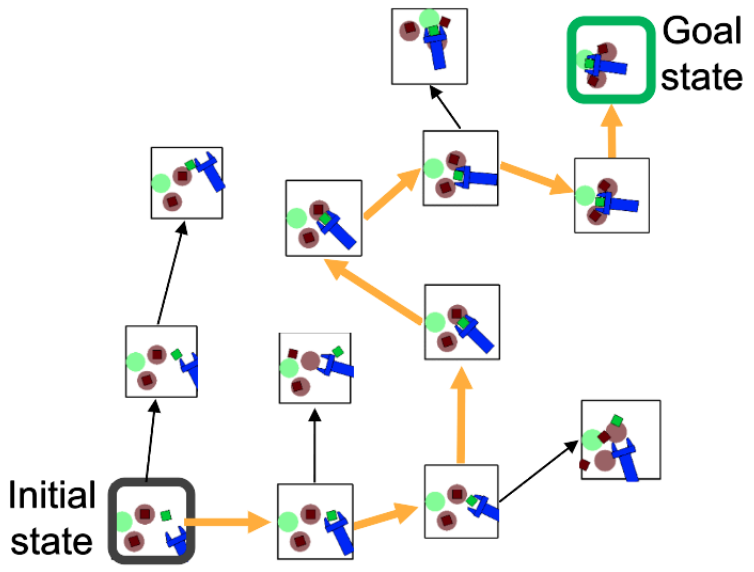


Figure 4.7: Example solution of a demonstration generated by the Kino-dynamic planner.

4.6.1 The Effect of the Number of Demonstrations on the Performance

We generated multiple task instances by randomly sampling non-colliding initial object poses and also randomly sampling a target region for the desired object. We then ran the Kino-dynamic planner to generate a demonstration for each of the task instances. An example of such solution is shown in Fig. 4.7. We parametrized the IL procedure by a *value margin* λ with $\eta = 3$ and $r_{cost} = 1$. We use a learning rate of 0.0001 for the training process, a discount factor of $\gamma = 0.995$, and a batch size of $M = 2000$. We use a replay buffer D^{replay} that can fit 500000 transition samples. The reward function is set to $r = -1$ per action.

To test the action-value function encoded by the network, we generated 300 random task instances, and ran a greedy policy on them, that is executing the action with the highest value estimate. The horizontal axis in Fig. 4.8 shows the number of demonstrations P generated by the Kino-dynamic planner, and the vertical axis shows the average success rate of the policy trained with that many demonstrations.

As expected, the graph shows an increasing trend w.r.t. the number of available demonstrations. After reaching a P of 8000 demonstrations, we see that it starts to plateau before it hits 50% success rate. This result demonstrates that the NN alone could not encode a behavior as good as the planner which we show in the next section achieves a success rate of 98%.

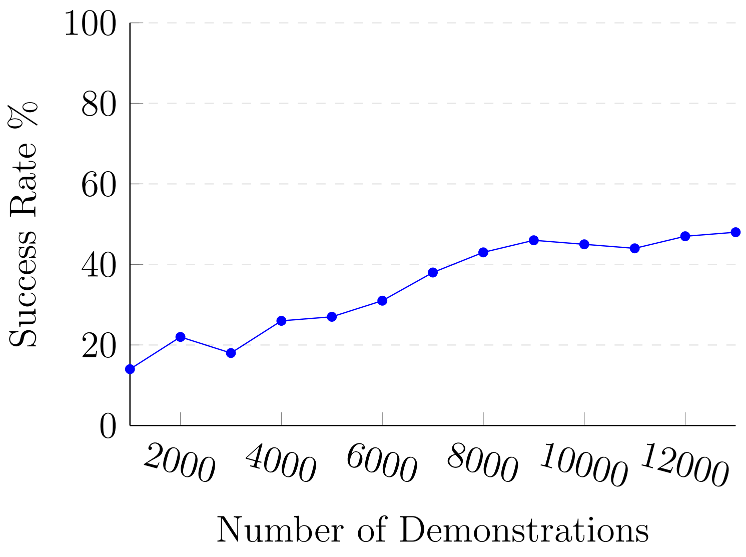


Figure 4.8: Performance of the greedy policy induced by the action-value function trained over a different number of demonstrations.

4.6.2 Performance Evaluation

Next, the network that encoded the best action-value function, as measured by the performance in the number of demonstrations experiment, was further trained with RL where the exploration policy is ϵ-RHP[3]. ϵ is set to 0.2. Each RHP query runs $n = 6$ roll-outs of $h = 6$ horizon depth each. We decreased the learning rate to 0.00001.

To evaluate the effectiveness of every step in the learning process, we compared two groups of RHP based policies:

- In the first group, the action-value function is solely learned from the demonstrations of the Kino-dynamic planner. We categorize them under *IL* in Table 4.1.
- In the second group, the action-value function is further updated with the ϵ-RHP guided RL. We categorize them under *IL + RL* in Table 4.1.

We evaluated each of these groups by using the trained action-value function in three different ways: *Greedy* policy, RHP with $n = 3,\ h = 3$ ($C.RHP^{3\times3}_{das}$), and RHP with $n = 6,\ h = 6$ ($C.RHP^{6\times6}_{das}$). We also include the open-loop execution based on the Kino-dynamic planner ($k.RRT$) as a baseline.

When we evaluated a certain policy, we injected different levels of noise on the physics model as a way of gauging how a policy copes with dynamics

[3]ϵ-RHP is similar to ϵ-VisualRHP but with the NN acting on the Cartesian state representation instead of abstract images

Table 4.1: The performance results of the different policies under different noise levels

		Planner	**IL**			**IL + RL**		
noise		***k.RRT***	***Greedy***	$C.RHP_{das}^{3\times3}$	$C.RHP_{das}^{6\times6}$	***Greedy***	$C.RHP_{das}^{3\times3}$	$C.RHP_{das}^{6\times6}$
No	suc. rate [%]	98.0 ± 2	48.0 ± 0	78.0 ± 1	88.0 ± 1	51.5 ± 5	88.8 ± 1	94.4 ± 2
	avg. time [s]	49.4 ± 14	0.7 ± 0	5.8 ± 1	17.7 ± 2	0.6 ± 0	7.9 ± 0	21.2 ± 2
Low	suc. rate [%]	24.5 ± 17	48.2 ± 7	77.2 ± 4	86.2 ± 3	48.6 ± 6	88.2 ± 2	94.0 ± 2
	avg. time [s]	41.1 ± 11	0.7 ± 0	6.4 ± 0	18.6 ± 1	0.6 ± 0	8.0 ± 1	22.7 ± 5
Med.	suc. rate[%]	28.5 ± 25	42.2 ± 12	73.4 ± 4	85.8 ± 8	47.3 ± 9	88.0 ± 2	91.2 ± 4
	avg. time [s]	42.5 ± 9	0.6 ± 0	7.3 ± 1	19.7 ± 3	0.6 ± 0	8.1 ± 1	18.4 ± 5
High	suc. rate[%]	15.7 ± 15	44.7 ± 29	71.5 ± 7	82.8 ± 8	45.6 ± 10	87.3 ± 5	90.1 ± 2
	avg. time [s]	37.6 ± 8	0.7 ± 0	7.1 ± 1	14.5 ± 2	0.7 ± 0	8.5 ± 2	17.3 ± 2

and object geometries that are different then the one it was trained on. The performance under such artificial noise is a way of estimating the robustness of each policy, and approximating how a policy would perform under real-world uncertainty. The rows in Table 4.1 correspond to these noise levels.

To inject noise into the model, we considered physics parameters: shape, friction, and density of the objects. During evaluation, the noise is sampled from a Gaussian distribution centered around the value of the parameters used in the training (and planning in the Kino-dynamic planner case)[4].

In each cell, Table 4.1 shows the success rate and the average computation time per successful run. The latter includes planning time, whether Kino-dynamic planning or RHP, and the time required to compute the physics interactions in Box2D. Also, a computation time limit of 3 minutes was imposed on all trials. The results presented are averaged over 300 random task instances with 95% confidence interval.

The Kino-dynamic planner with no noise shows a high success rate. The few cases where it failed are due to the imposed time limit. Nevertheless, the decreasing performance with noise and the relatively high computation time confirms the limitation of using open-loop planning in execution. The left image in Fig. 4.9 shows how a pre-computed plan can fail during execution when the geometry of an object is slightly different than the one used at planning time. At planning time, only square objects were used. When the plan is executed in open-loop but with slightly different object geometry, i. e., a rectangular shape for the green object instead of the square shape used during planning, the green object slides outside of the robot trajectory. In general, this indicates that this

[4]*Mean* values of the objects' physics parameters are as follows: shape: 6×6 *cm*, density: $1\ kg/m^2$, friction coefficient: 0.3.
Standard deviation on the objects' physics parameters: Low = $0.05 \times mean$, Medium = $0.15 \times mean$, High = $0.25 \times mean$.

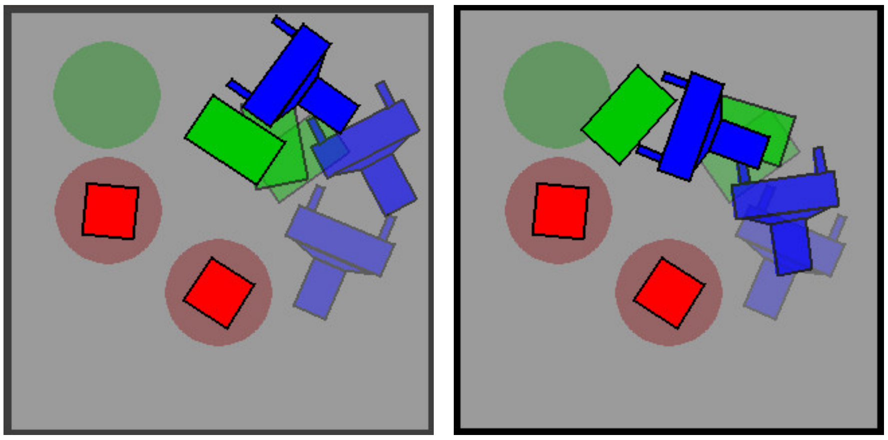


Figure 4.9: Evaluation at execution time with different shapes.

type of planning is favorable when a high-fidelity model and high-processing power for re-planning are available.

We also notice that when RHP is engaged there is a notable increase in performance. The longer the horizon and number of roll-outs the higher is the success rate and the more robust it is against noise. The performance increase comes at a cost of an increased computation time. However, it is still within reasonable limits for real-world applications. In contrast to using an open-loop control scheme, the right image in Fig. 4.9 illustrates how the robot can adapt to unexpected physics interactions.

Looking at the overall performance between the two groups of policies, we see that further optimizing the action-value function with RL contributed to a higher success rate and robustness to uncertainty. A common trend in the results also shows that the average computation time per successful run increases slightly when RL is used. This increase is justified by the fact that with higher success rate the RL-optimized policies are successfully solving more challenging settings, which would require more actions to solve.

Further, $C.RHP_{das}^{6\times6}$ outperformed all of the others w. r. t. the success rate while still offering a reasonable planning and execution time for real-time applications. We used this policy successfully to command a robot in the real world.

4.6.3 Real-robot Execution

We performed experiments on a UR5 robot[5]. We created the three task instances shown in Fig. 4.10, Fig. 4.11, and Fig. 4.12. In each task, we tested the

[5]https://www.universal-robots.com/products/ur5-robot/

trained $C.RHP^{6\times6}_{das}$ policy (bottom row in figures) and compared it to the open-loop execution of the Kino-dynamic planner (top row). During the execution, closed-loop feedback on object poses was supplied using an OptiTrack system for RHP to run the roll-outs on the model. As expected, the reactive capability of RHP made its reaction robust to the dynamics of the real world and succeed in these three tasks. In two out of three tasks, the open-loop execution failed. A video of these experiments is available on https://youtu.be/xwa0fTTuQ1g.

The preliminary experiments on this basic version of the problem show promising results to combining real-world execution with physics-based look-ahead planning.

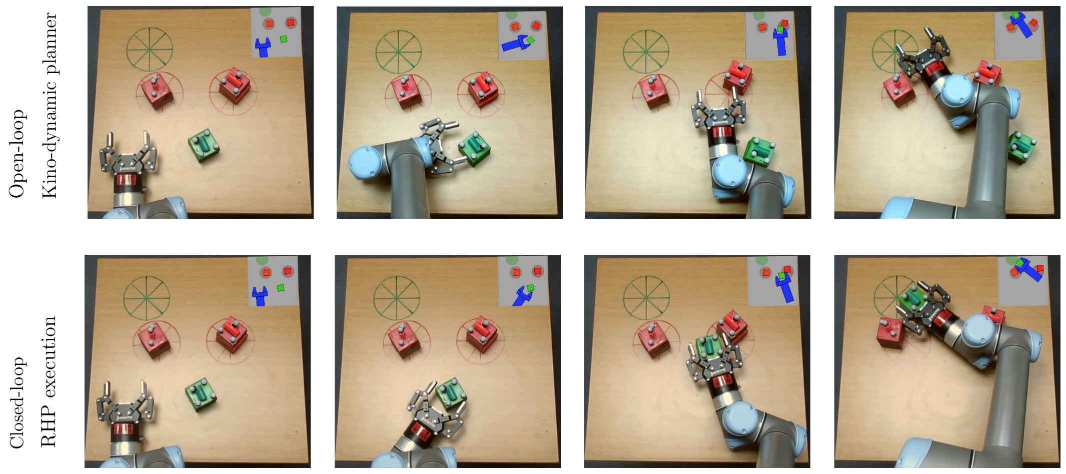


Figure 4.10: Top: robot failing to push the green object to the target region by following a pre-computed plan using Kino-dynamic planning. Bottom: robot successfully executing the task using closed-loop $C.RHP^{6\times6}_{das}$ execution.

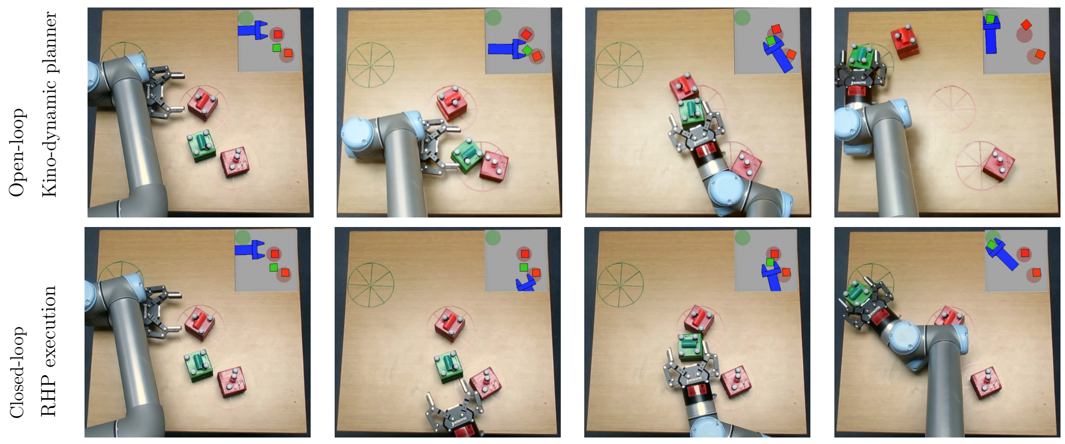


Figure 4.11: Top: robot failing to keep the red object close to its initial position by following a pre-computed plan using Kino-dynamic planning. Bottom: robot successfully executing the task goal using $C.RHP^{6\times6}_{das}$.

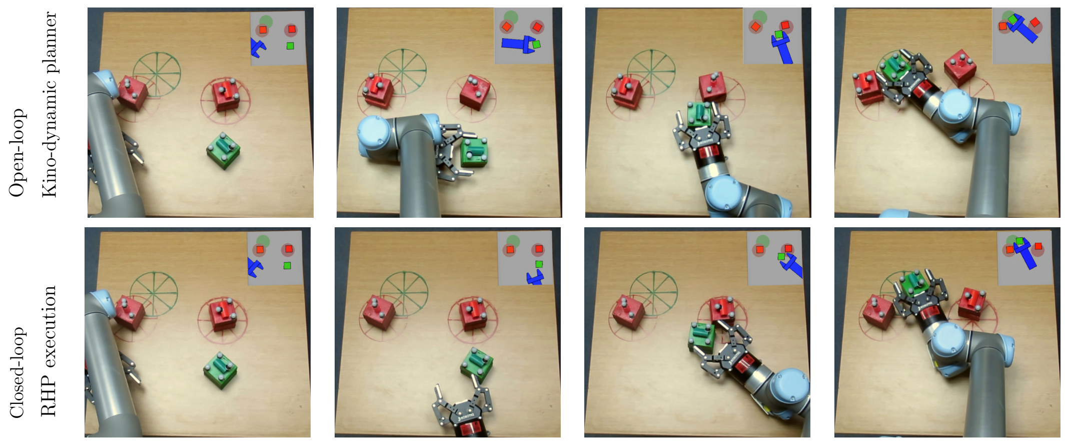


Figure 4.12: Top: robot successfully executing the task goal by following pre-computed plan using Kino-dynamic planning. Bottom: robot successfully executing the task goal using $C.RHP^{6\times6}_{das}$.

4.7 Experimental Setup and Implementations

In the preliminary experiments, we only used discrete and non-prehensile actions in a Cartesian state space with a fixed and small number of objects. In the next round of experiments, we evaluated the performance of VisualRHP over a variable number of object shapes in an image-based state representation and using prehensile and non-prehensile actions.

We ran a series of experiments conducted in simulation and on the real robot to evaluate and validate VisualRHP. The focus of the experiments are: (*i*) to evaluate the performance contribution of each of the main elements of VisualRHP in discrete and continuous action spaces with respect to state-of-the-art alternatives (Sec. 4.8.1), (*ii*) to assess the algorithms' robustness to un-modelled dynamics for real-world applications compared to open-loop execution (Sec. 4.8.2), (*iii*) to evaluate if the acquired behaviour learned transferable skills to different real-world manipulation environments (Sec. 4.9).

During the experiments, we varied the environment parameters introduced in Sec. 4.3. The environment consists of the target region, the end-effector of the robot arm, one desired object, and $m-1$ obstacles[6]. We trained the NN on task instances with different clutter densities, ranging from $m=1$ object, i. e., only the desired object, to $m=7$ objects. The shape of an object is randomly selected from a pool of polygons with a random number of vertices centred around the polygon centre of mass. Some of the objects are too large to fit within the gripper fingers to be grasped, whereas others are small enough to be grasped with force closure from any approach angle, and some others are directionally graspable. The location of the target region is sampled from a uniform distribution over the manipulation surface. With the aim of only describing the task objective, the reward function is set to $r=-1$ per action and $r=-50$ if an object is dropped outside of the surface edges.

4.7.1 Action Spaces

In our simulation environment, we modelled the world in the Box2D physics simulator. The robot motion is driven by a proportional controller, where an action represents a target velocity vector to be maintained for a fixed amount of time.

[6]The surface edge dimensions are 50×50 cm, the target region has a radius of 7 cm, and the objects and robot density is 1 kg/m^2 with 0.3 as the friction coefficient. The robot dimensions are modeled after the *Robotiq 2F-85* gripper.

Discrete action space (*das*): We define 8 actions: four along cardinal directions that achieve a 5 *cm* translation in any of these directions, two 30^o rotational actions (CW and CCW), and the last two are for closing and opening the gripper.
Continuous action space (*cas*): The action distribution is modelled by a 4-dimensional Gaussian distribution with a diagonal covariance matrix. The first three correspond to the longitudinal, lateral, and rotational directions along the robot end-effector. A hyperbolic tangent activation function at the output of the NN is used to bound the means of the Gaussian distribution such that the induced translation and rotational step falls within -10 *cm* and 10 *cm*, and -50^o and 50^o, respectively. The fourth dimension corresponds to closing and opening the gripper.

4.7.2 Network Architecture

It is important to design a NN architecture with an inference time small enough to be queried multiple times per action selection and still return an action in near real-time. We built similar NN architectures for the discrete and continuous action spaces, using the TensorFlow library, with the main difference being the output heads.

Discrete action space: The NN modelling the action-value function is composed of a CNN part connected to dense layers. The input to the CNN is a $64 \times 64 \times 3$ image. The CNN consists of 3 sequences of: 2 coordConv appended channels[7], 2D convolution, normalization, and max-pooling layers. We use 3×3 kernel sizes and 8, 8, 16 filters for the convolution layers, respectively. The output of the CNN is flattened into a vector of 256 features. The input to the dense layers is a feature vector that concatenates the flattened output of the CNN and a binary value corresponding to the gripper state. The dense layers are composed of 3 fully connected layers. The first 2 layers have 128 neurons each. We use leaky *ReLU* as activation function all throughout the network. The output layer consists of 8 neurons, one per action, with linear activation functions.

Continuous action space: The architecture of the CNN and the input vector to the dense layers are the same as in the network for the discrete action space. The CNN is followed by two shared fully connected dense layers of 256

[7]The coordConv helps in capturing geometric information that are translation dependent [75].

and 128 neurons. The value head has a fully connected layer of 64 neurons with leaky *ReLU* activation function. It is followed by a single neuron output layer for the value function with a linear activation function. The Policy head has a fully connected layer of 128 neurons with leaky *ReLU* activation function. The output layer is a fully connected layer with 8 neurons modelling the mean and standard deviation parameters of a 4-dimensional Gaussian that is representing the policy. The 4 outputs corresponding to the means have a hyperbolic tangent (*tanh*) as activation function, and the other 4 outputs corresponding to standard deviations have a sigmoid activation function[8].

The training and experiments were conducted on an Intel Xeon E5-26650 computer equipped with an NVIDIA Quadro P6000 GPU card.

4.7.3 Evaluation Metrics

Data for each experiment is collected over 300 runs. Unless otherwise specified, the performance is evaluated in simulation with respect to three metrics:

- **Success rate** represents the percentage of the successfully completed tasks. We consider a task to be successfully completed when the desired object is moved to the target region in under 50 actions without having any of the objects falling off an edge.
- **Action efficiency** is a relative measure in view of the scene complexity represented by the clutter density. It is calculated as:

  $$\frac{number\ of\ objects\ in\ the\ scene}{number\ of\ actions\ until\ completion}.$$

  The higher the ratio of an approach, the more efficient it is.
- **Average execution time per run** is computed as the number of actions times the average time required for selecting an action, which can change for different planners, and the time required to roll the physics in the simulator.

[8]It is common in the continuous action space literature to use a linear activation function to model the standard deviations, however, we found that a sigmoid function makes it more likely for the RL algorithm to converge without having any noticeable downside on the exploration.

4.7.4 Training Procedure

We collected, for each of the discrete and continuous action space approaches, up to $P = 14000$ demonstrations over random task instances using the Kinodynamic RRT planner. We trained the NNs over these demonstrations as detailed in Sec. 4.5.2 on IL. For the **discrete action space**, the *value margin* parameters are $\eta = 3$ and $r_{cost} = 1$. For the **continuous action space**, the hyper-parameters are set to $c_1 = 0.7$, $c_2 = 0.01$, and $\Psi = 0.5$. We use a learning rate of 0.0001 for the training process in both action spaces, a discount factor of $\gamma = 0.995$, and a batch size of $M = 2000$.

To ensure that enough demonstrations were collected for IL, we show in Fig. 4.13 how the success rate changes when increasing the number of available demonstrations. The plots show the average success rate. As expected, the graph shows an increasing trend w. r. t. the number of available demonstrations. After reaching a P of around 8000, the success rate starts to plateau at around 60% and 70% for the discrete and continuous action spaces, respectively. We also report a remarkably low action efficiency of 0.15 and 0.20 for the discrete and continuous action spaces, respectively.

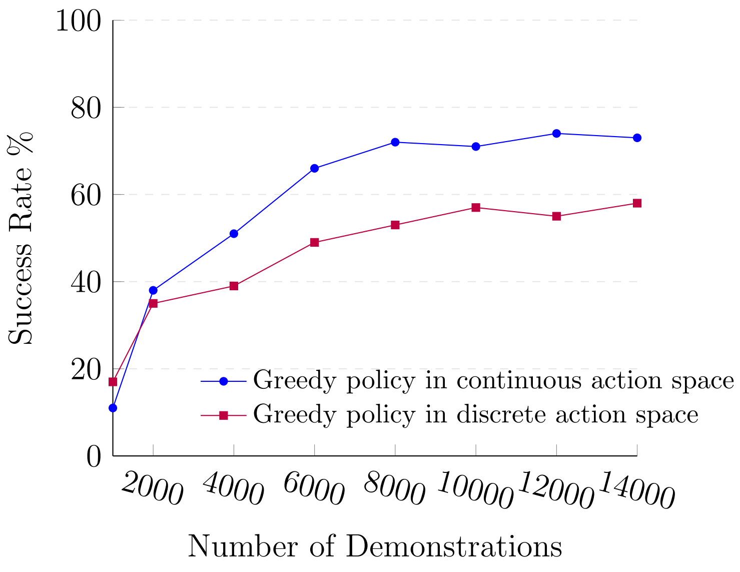


Figure 4.13: The effect of the available number of demonstrations on the performance of the learned behaviour.

Next, the NNs of both action spaces were further trained with RL. We kept the same batch size of $M = 2000$ and the discount factor of $\gamma = 0.995$. We decreased the learning rate to 0.00001. In the **discrete action space**, we use ϵ-VisualRHP with $\epsilon = 0.2$, $n = 3$, and $h = 3$. We use a replay buffer D^{replay} that can fit 500000 transition samples. The transition samples are collected by 10 agents running in parallel. We run 3 optimization epochs after every 2000 newly collected transition samples. In the **continuous action space**, we use C3PO in conjunction with PPO. We set the PPO *clip* value to 0.075 and the hyper-parameters to $c_3 = 0.7$, $c_4 = 0.1$, and $c_5 = 0.35$. We set the size of the replay buffer D^{replay} to 10000 transition samples. We also run 10 environments in parallel. For the optimization step, we run 20 epochs over $\mathcal{L}^{baseline}$ and 15 epochs over $\mathcal{L}^{actor-critic}$.

4.7.5 VisualRHP and Baselines

We conducted an ablation study to assess how each element in our proposed approach affects the final performance. We looked at the effect of the image-based abstract representation, the use of a learned heuristic in image space, and the integration of the physics-based look-ahead planning in the control strategy. Accordingly, in addition to VisualRHP, we composed four corresponding baseline methods. We use $n \times h$ in the superscript of a method's name to denote the VisualRHP parameters (e. g., *VisualRHP*$^{3\times3}$). All baseline methods were trained with the same procedure as ours unless otherwise specified:

VisualRHP: We experimented with four VisualRHP formats. Two in the discrete action space and two in the continuous action space. For each of the action spaces, there is one version that uses $n = 3$ roll-outs of $h = 3$ horizon depth, and another one that uses $n = 6$ and $h = 6$.

Cartesian Pose Baseline (CartesianRHP): Instead of using abstract images for the state representation, *CartesianRHP* uses the relative Cartesian poses of the objects and the target region with respect to the end-effector, and the absolute Cartesian pose of the end-effector and a binary gripper state. The discrete and continuous action space versions of *CartesianRHP* use the same NN architectures. We ran *CartesianRHP* in conjunction with RHP parameters of $n = 3$ and $h = 3$. The NN architecture used by *CartesianRHP* has an inherent limitation. It can only be trained on a specific number of objects and can not generalize to arbitrary clutter densities. Adding or removing objects, i. e., changing the size of the input layer, requires the use of a different NN architecture. Hence, *CartesianRHP* requires the training of multiple NNs, each designed to operate on a specific number of objects. This baseline is inspired by the one used in the preliminary experiments.

Handcrafted Heuristic Baseline (CraftedHeuristicRHP): We ask the question of whether the problem can still be solved in closed-loop with a handcrafted heuristic to estimate the cost-to-go from a horizon state to the goal, rather than the learned one. Hence, *CraftedHeuristicRHP* implements RHP with a handcrafted heuristic. We found that *CraftedHeuristicRHP* performs best with $n = 8$ random roll-outs of depth $h = 4$ by sampling random actions from the discrete or continuous action spaces. The cost-to-go function, presented in Eq. 4.13, is a weighted sum of the Euclidean distance (d_1) and the angular displacement (d_2) between the robot and the desired object, the Euclidean distance between the target region and the desired object (d_3). It also includes a term that encourages the alignment between robot, desired object, and target region, with increasing emphasis on the robot facing the target region once it is positioned behind the desired object (d_5). A penalty term is added to dropping any of the objects outside the surface edges (d_{out}). The weights, balancing these different components, were empirically optimized to favour a behaviour where the robot would first approach the desired object from the back, then pushes it towards the target region.

$$\begin{aligned}
cost\,to\,go =\; & 0.4\,d_1 + 0.8\,d_3 + 0.7\,d_2 + 1.4\,d_4 + d_{out} \;\ni \\
& d_1 = dis(Rob, Obj^{des}), \\
& d_2 = ang(Rob), \\
& d_3 = dis(tarReg, Obj^{des}), \\
& d_4 = \begin{cases} 0.6(d_1 - dis(Rob, tarReg)) + 2\; ang(tarReg, Rob, Obj^{des}) \\ \qquad\qquad \text{if } ang(Rob, Obj^{des}, tarReg) > 0.5\pi \\ 0.8\,(d_1 - dis(Rob, tarReg)) + \pi, \qquad \text{otherwise} \end{cases} \\
& d_{out} = \begin{cases} 100, & \text{if any object is outside the surface } edges \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{4.13}$$

where

$$\begin{aligned}
& dis(a,b) : \text{Euclidean distance between } a \text{ and } b \\
& ang(a,b) : \text{Angular displacement of } b \text{ in the frame of } a \\
& ang(a,b,c) : \text{Angular displacement formed by } \widehat{abc}
\end{aligned} \tag{4.14}$$

The use of a handcrafted cost function for manipulation in clutter is reminiscent to trajectory optimization-based approaches such as in [5, 90].

Greedy Baseline (Greedy): Almost ubiquitously, an RL trained robot for manipulation tasks would act greedily at execution time on the learned policy without running look-ahead planning [121, 95, 122]. In this thesis, we argued

that it is hard for a greedy policy to accurately anticipate how the environment will unfold under complex interaction dynamics, especially in an environment rich with physics contacts. *Greedy* challenges this claim by running a greedy policy on the trained NN in the image space. Action selection is based solely on the current state as observed in the abstract image representation. In the discrete action space, the action with the highest value estimate is selected. In the continuous actions space, the action vector is set to the mean vector of the policy distribution as outputted by the policy head of the NN. Therefore, the simulator at execution time is only used to render the abstract images on which the greedy policy acts. The most similar state-of-the-art approach to this baseline is [120].
Kino-dynamic RRT Baseline (K.RRT): All the previous baseline control strategies run in closed-loop. As an alternative, we used an open-loop sampling-based planner, namely the Kino-dynamic RRT introduced in Sec. 4.5.1. A computation time limit of 3 minutes was imposed on *K.RRT* before declaring a failure. In the discrete action space, the planner has access to the 8 discrete actions per state. In the continuous actions space, the planner can sample up to 8 random actions per state.

We report that we omitted the following two baselines from the results, as even after extensive systematic hyper-parameter tuning, we did not succeed in getting them to converge to a satisfactory behaviour:

- NN parameters were randomly initialized and then solely trained with RL (i. e., without IL). This was repeated for several initialization trials. The policy often converged to a behaviour that drives the robot to shove the objects by the side of the gripper towards the target region without much consideration to the surface edges constraints, often causing objects to drop outside of it.

- A baseline wherein the RL part of our training procedure, in the continuous action space, uses PPO without the additional C3PO optimization step of $\mathcal{L}^{baseline}$. The original version of PPO resulted in the NN diverging causing the loss of the acquired knowledge from the IL part. Fig. 4.14 shows a comparison of the success rate over the RL training process between PPO and C3PO with PPO. After every 10000 newly collected transition samples, 20 optimization epochs are performed. Each data point is averaged over 5 RL trials with 95% confidence interval. Both learning algorithms are evaluated with the greedy policy.

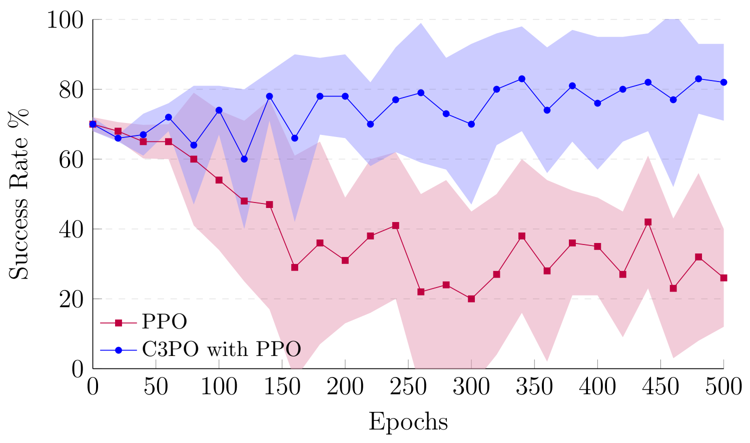


Figure 4.14: The effect of the C3PO optimization step on the learning stability.

4.8 Simulation Results and Discussion

In this section we present the results and discuss the implications of the simulation experiments.

4.8.1 Performance in Different Environment Setups

The first group of simulated experiments looks at the success rate, the action efficiency, and the average execution time per run in environments of different levels of difficulty. We consider the effect of changing clutter densities, ranging from 1 object, i. e., only the desired object without clutter, to 7 objects on the surface. We also examined the performance in environments with random number of objects (up to 7) but of different average sizes (small, mixed, and large) relative to the dimensions of the gripper. It is expected that the shape of small objects is less significant to the manipulation task compared to large objects.

The success rate results are reported in Fig. 4.15 and in Fig. 4.16. The plots show that *VisualRHP* and *CartesianRHP* outperform the other two baselines, with a slight advantage for performing a higher number of deeper roll-outs. For a low clutter density, all approaches show high success rate. Not surprisingly, increasing the clutter density causes a drop in the success rate across all baselines as it becomes much more likely for objects to fall off the edges or for the robot not to find its way through the clutter. *Greedy* suffers from the sharpest drop with respect to the number of objects. We also observe in Fig. 4.16 that large objects seem to be slightly more difficult to manipulate as reflected in a decrease in the success rate. *VisualRHP*, on the other hand, is more robust to the increase in object sizes. This result can be attributed to the fact that

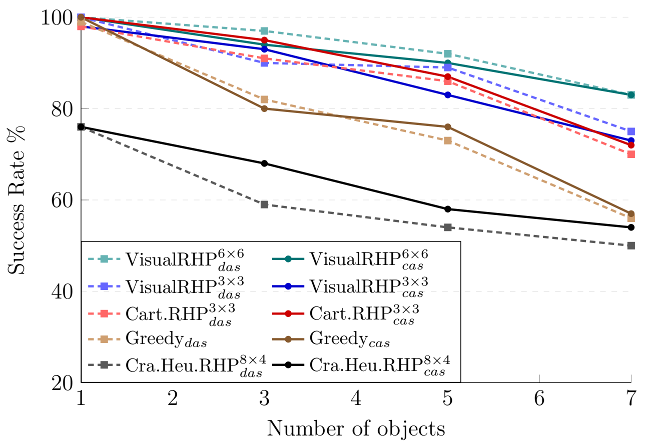


Figure 4.15: Success rate w. r. t. clutter density.

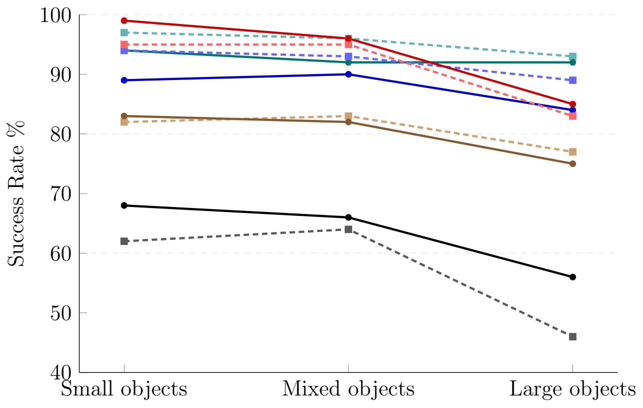


Figure 4.16: Success rate w. r. t. to objects sizes.

with higher clutter density and/or objects sizes, more physical interactions are involved. The decision making process must account for these interactions over the short and the long term to avoid irreparable arrangements. In this sense, *VisualRHP* compensates for the NN deficiency to anticipate how the environment will unfold under a sequence of actions. Operating in continuous versus discrete action space has no significant effect on the success rate.

Looking at the action efficiency results in Fig. 4.17 and in Fig. 4.18 reveals more insight on the difference between operating in continuous and discrete actions spaces. There is a clear advantage of operating in the continuous action space. It consistently scores higher across all baselines. This is because the policy has finer control over the positioning of the robot.

We note that RL caused a significant increase in the action efficiency compared to the IL action efficiency. In the mixed object experiment, the action efficiency of the greedy policies increased from 0.15 to 0.22 and from 0.20 to 0.35 in the discrete and continuous action spaces, respectively.

Furthermore, using higher number and deeper roll-outs (*VisualRHP*$^{6\times6}$) does not have a noticeable action efficiency advantage over using less (*VisualRHP*$^{3\times3}$)

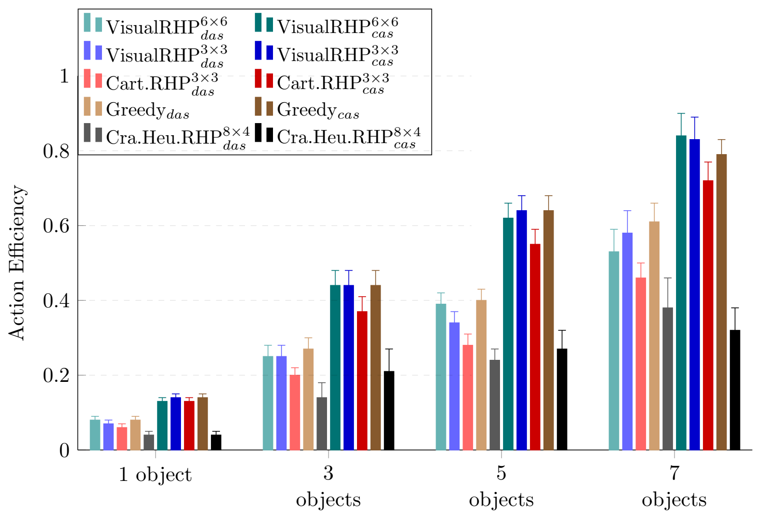


Figure 4.17: Action efficiency w. r. t. clutter density.

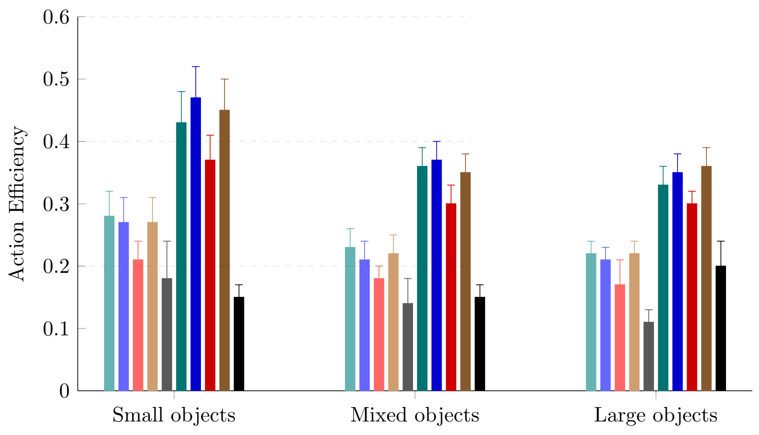


Figure 4.18: Action efficiency w. r. t. object sizes.

or no roll-outs (*Greedy*). The action efficiency results also show that although *CartesianRHP*$^{3\times3}$'s success rate is on par with *VisualRHP*$^{3\times3}$, *CartesianRHP*$^{3\times3}$ slightly but consistently scores lower on the action efficiency compared to all the approaches that rely on the abstract image-based representation. This results confirms our intuition on the use of engineered features for the state representation. *CartesianRHP*$^{3\times3}$ always converged to a behaviour where the robot would approach the desired object from the back and pushes it towards the target region. Although robust to the variation in shape of the objects, each NN of *CartesianRHP*$^{3\times3}$, trained over a specific number of objects, resulted in a behaviour that requires more actions for solving the task when compared to a behaviour where the robot can actually leverage the shape of the objects in order, for instance, to grasp the desired object and move it to the target region.

The importance of the action efficiency metric is reflected in the average execution time per task as presented in Fig. 4.19. We see that the average execution time per task is always lower in the continuous action space. This

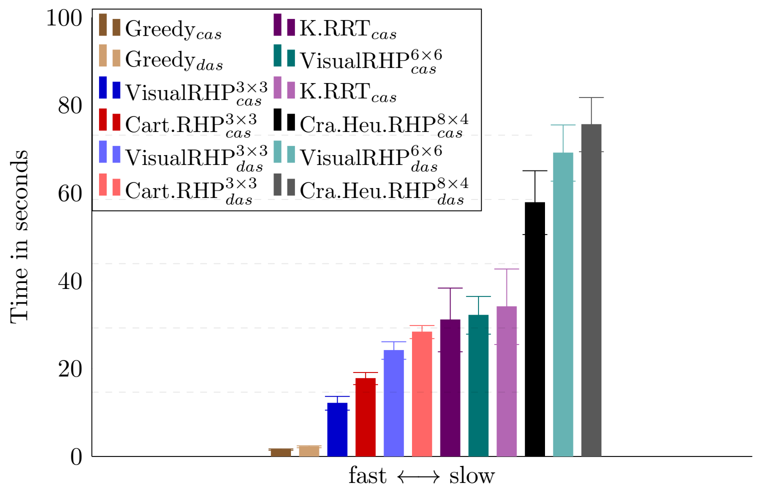


Figure 4.19: Average planning and execution time.

can be explained as a direct consequence of the action efficiency[9]. In addition, the average execution time is also affected by the number and depth of the roll-outs. For instance, *VisualRHP*$^{6\times6}$ and *CraftedHeuristicRHP*$^{8\times4}$ simulate 6×6 and 8×4 actions before returning an answer. This places them at the slowest end of the spectrum. In contrast, *Greedy* does not need to perform any roll-out and exhibit the fastest execution time, albeit with a trade-off on the success rate. We also included in this figure the results for Kino-dynamic RRT. It stands in the middle of the rank, but as we will see in the next section, it might not be the best suited for these application domains.

[9]The difference in the inference time between different NN architectures is minimal and has no measurable effect on the average execution time.

4.8.2 Robustness to Un-modelled Dynamics

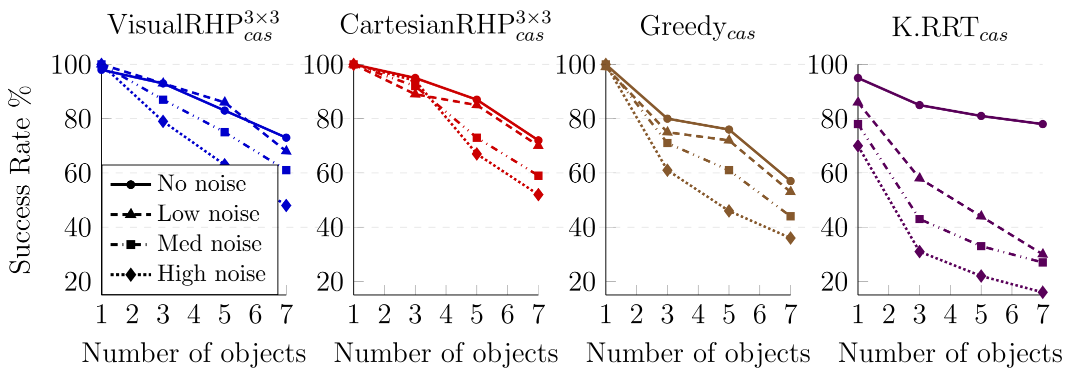


Figure 4.20: Performance w. r. t. clutter density in *cas* for different noise level on the physics and geometry parameters.

Following the results of the first group of simulated experiments, we identify that *VisualRHP*$^{3\times3}_{cas}$, *CartesianRHP*$^{3\times3}_{cas}$, *Greedy*$_{cas}$, and *K.RRT*$_{cas}$ are potentially suitable for near real-time applications in the real world. They offer a reasonable balance between computation time and success rate.

As a way of gauging how these approaches cope with dynamics that are different than the one they were trained for, we ran a second group of simulated experiments where we compared them against different levels of artificially injected noise on the physics and geometric parameters at evaluation time (similarly to the preliminary experiments in Sec. 4.6). The results are presented in Fig. 4.20. The different line styles correspond to the different noise levels. Each data point corresponds to 300 runs initialized with random target location, arrangement, and objects shapes.

The *K.RRT* results reiterates our finding in the preliminary experiments. Open-loop execution performs well under low noise conditions and suffers from a sharp drop in in the success rate with increased noise. *Greedy*$_{cas}$ preforms remarkably well even with high noise but only for when there is one object in the scene. This can explain the wide spread use in literature of the greedy policies for manipulating a single object in the real world. It has real-time reactive behaviour but it fails to cope with high clutter environment with unknown dynamics.

When looking at *CartesianRHP*$^{3\times3}_{cas}$ in high noise environments, we see that its success rate surpasses the *VisualRHP*$^{3\times3}_{cas}$. A possible explanation is the conservative policy learned by *CartesianRHP*$^{3\times3}_{cas}$ makes it more robust against noise on the shape of the objects. Because of the engineered feature vector of the state representation, the shape of the objects are unknowable to the policy. Hence, each of the NNs in *CartesianRHP*$^{3\times3}_{cas}$, one for every specific number of objects, has converged to a behaviour that increases the chances of success irrespective of the shapes but at the expense of a lower action efficiency. In contrast, $\textit{VisualRHP}_{cas}^{3\times3}$ tailors the behavior to the exact shape of the objects making it more action efficient and generalizable, but also slightly more susceptible to high noise. We expect that a decently parameterized physics simulator would be good enough for $\textit{VisualRHP}_{cas}^{3\times3}$ to achieve high success rate.

4.9 Real-World Experiments and Discussion

We built on the simulation results to compare the approaches evaluated in the previous section in real-world experiments that require transferable manipulation skills over different task setups. We used a *Robotiq 2F-85* two finger gripper[10] mounted on a 6-DOF *UR5* robot. The robot operates over a 50×50 *cm* surface and target region of 7 *cm* in radius. The manipulation objects include bottles, apples, oranges, and cups. Using a top mounted RGB camera, instance segmentation is performed using a *Mask R-CNN* [40] vision system trained on the *COCO Dataset* [74]. A video of these experiments is available on https://youtu.be/raKHTnJLikQ.

Starting from a similar initial environment setup, Fig. 4.21 and Fig. 4.22 shows *K.RRT* (top), $\textit{Greedy}_{cas}$ (middle), and $\textit{VisualRHP}_{cas}^{3\times3}$ (bottom) tasked with manipulating the orange fruit to the target region in low and high clutter environments, respectively. *K.RRT* succeeds in solving the task in a low clutter environment. This is because problems associated with the physics discrepancy between the simulator model and the real world are mitigated by minimal physical interactions in a low clutter environment. However, in a high clutter environment, small discrepancies between the physics model of the simulator and the real world, e. g., gripper-orange-bottle interactions, are compounded causing task failure as shown by the red apple falling outside the surface edge. Further, $\textit{Greedy}_{cas}$ also performs well in a low clutter environment. Although it does not foresee how the environment will unfold under a certain action, it can react fast to a dynamic environment. We see the robot chasing the orange after it was unintentionally knocked to the side of the table. The high clutter environment shows that $\textit{Greedy}_{cas}$ is prone to getting trapped in some parts of the state space (cyclic or oscillatory behaviour [112]), only escaping it after executing several ineffective actions. Also, by acting solely based on the current observation the robot fails to anticipate the upcoming states as exemplified by the red apple being pushed, via the bottle, outside the surface edge. On the

[10]https://robotiq.com/products/2f85-140-adaptive-robot-gripper

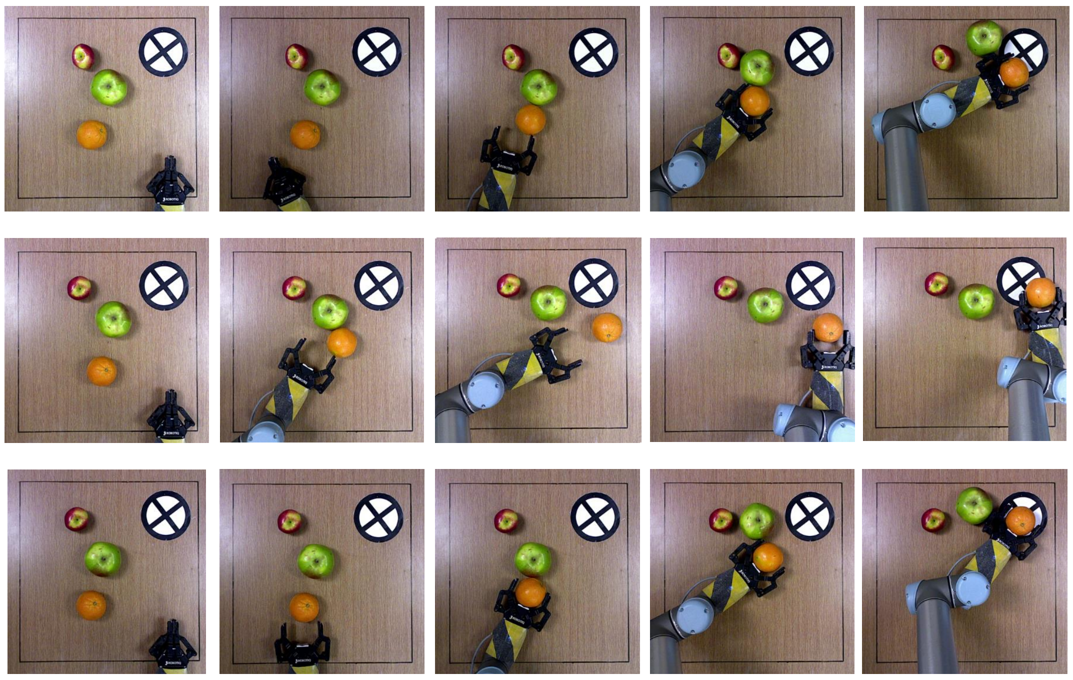


Figure 4.21: *Comparing Kino-dynamic RRT* (top), *Greedy* (middle), and *VisualRHP* (bottom) policies running in continuous action space in **low** clutter environment. The "orange" is the desired object.

other hand, *VisualRHP*$^{3\times3}_{cas}$ evaluates the potential consequences of an action before being executed in the real world. At a slightly higher computation cost, the policy selects informed actions based on the predicted environment dynamics.

In Fig. 4.23 and with the bottle being the desired object, we observe two distinct strategies that *CartesianRHP*$^{3\times3}_{cas}$ and *VisualRHP*$^{3\times3}_{cas}$ converges to. In the top row, *CartesianRHP*$^{3\times3}_{cas}$ drives the robot around the apple obstacle. Unaware of the actual geometry of the bottle, the robot barely manages to push the bottle to the target region. In the bottom row, the robot behaviour, controlled by *VisualRHP*$^{3\times3}_{cas}$, exhibits awareness of the geometries of the objects by approaching the bottle from a graspable angle and manoeuvring it to the target region.

Additionally, Fig. 4.24 shows the robot being tasked with manipulating a small apple in the top row and a large apple in the bottom row using *Visual-RHP*$^{3\times3}_{cas}$. When the geometry of the desired object is relatively small, i. e., it fits within the fingers of the gripper, the robot manoeuvres its way through the clutter, grasps of the apple, and pulls it to the target region. Whereas, when the geometry of the desired object is relatively large, the robot resorts to pushing it towards the target region.

Fig. 4.25 compares the robot behavior in discrete (top) and continuous (bottom) action spaces. In this experiment the robot is tasked with manipulating

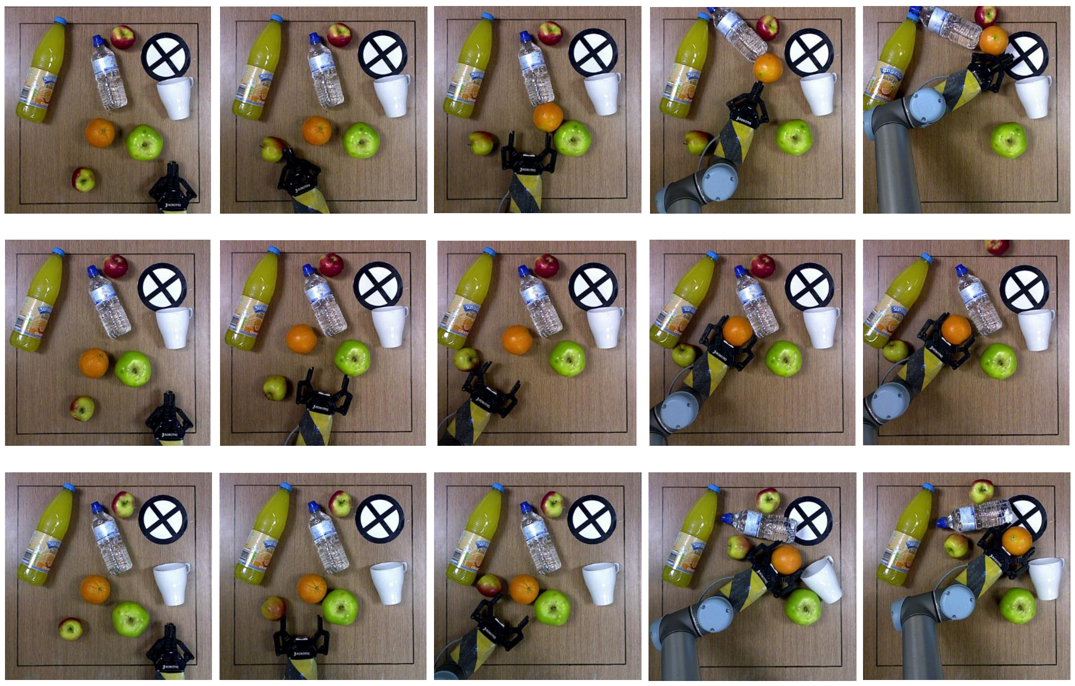


Figure 4.22: Comparing *Kino-dynamic RRT* (top), *Greedy* (middle), and *VisualRHP* (bottom) policies running in continuous action space in **high** clutter environment. The "orange" is the desired object.

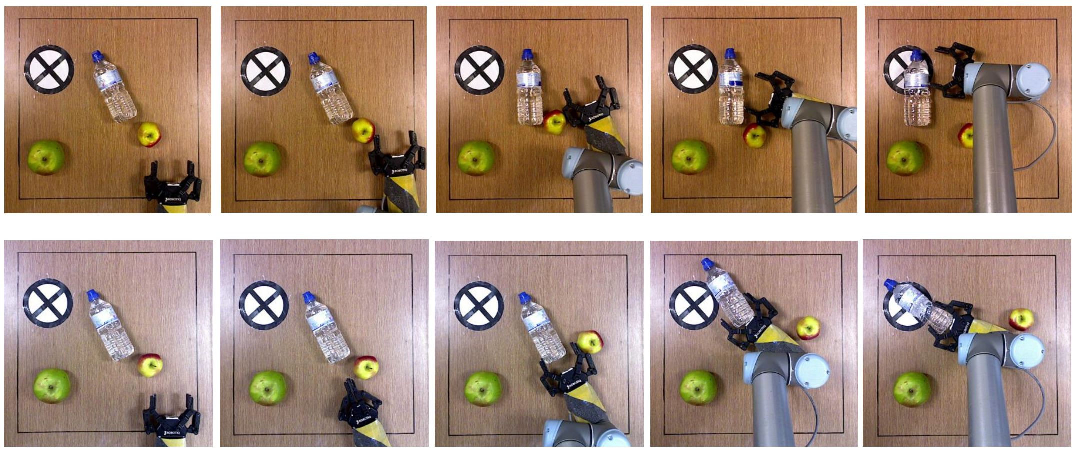


Figure 4.23: Comparing *CartesianRHP* (top) and *VisualRHP* (bottom) policies running in continuous action space. The "bottle" is the desired object.

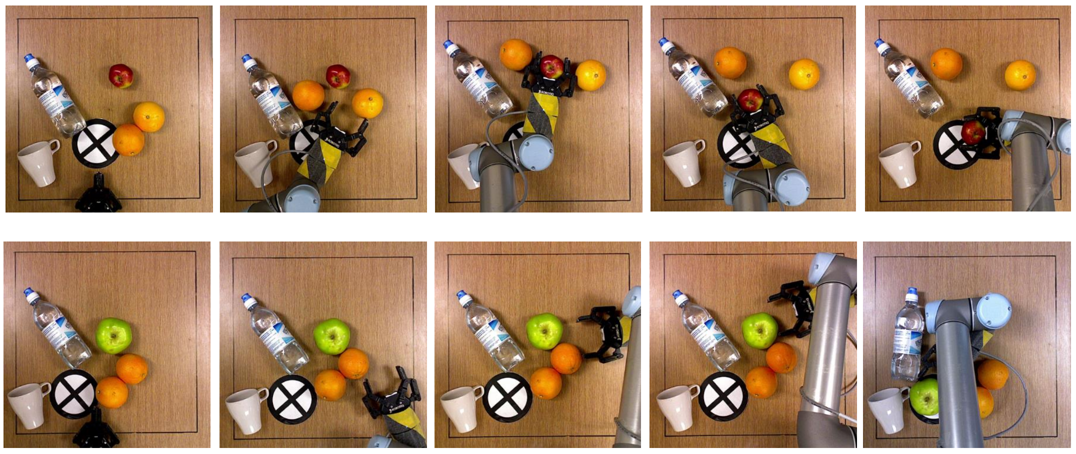


Figure 4.24: Comparing *VisualRHP*, running in continuous action space, in manipulating **small** (top) and **large** (bottom) desired object. The "apple" is the desired object.

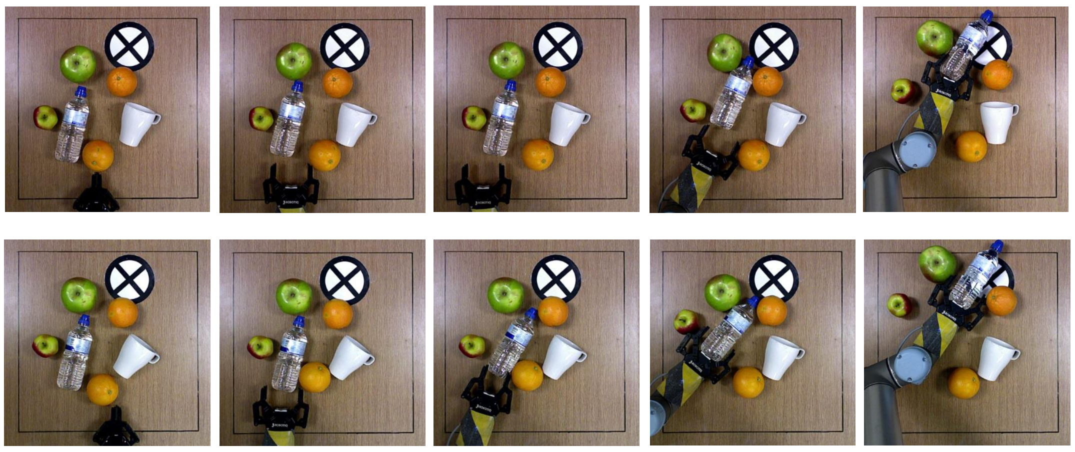


Figure 4.25: Comparing *VisualRHP* in **discrete** (top) and **continuous** (bottom) action spaces. The "bottle" is the desired object.

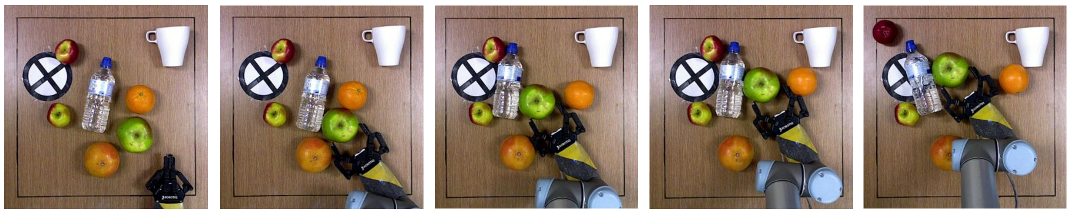


Figure 4.26: Manipulation examples with *VisualRHP* in **challenging environment setups**. The "bottle" is the desired object.

the bottle to the target region. Having only access to discrete actions, the robot performs several actions, particularly when the robot is positioned under the bottle, before the robot is able to get the bottle within the gripper fingers and then pushes it to the target region. In the continuous action space, the robot performs fewer actions to position itself directly under the bottle such that it can be grasped, rotated, then pushed towards the target region.

Lastly, we show $\mathit{VisualRHP}^{3\times3}_{cas}$ operating in challenging environment setups where the desired object is surrounded by obstacles with little room to manoeuvre. In Fig. 4.26, with the bottle being the desired object, the robot leverages the obstacles by pushing the larger apple which in turn is pushing against the desired object and driving it to the target region. In Fig. 4.1, with the orange being the desired object, $\mathit{VisualRHP}^{3\times3}_{cas}$ does not only use its control over the opening and closing of the gripper for prehensile actions, but also as a tool for the robot to squeeze its way through the clutter and then opening the gripper for clearing the robot's path to the desired object.

4.10 Conclusions

The conducted ablation study provides strong evidence for the necessity of the different components of VisualRHP. (*i*) The abstract image-based representation provides the basis for transferable and generalizable manipulation skills. The robot performed in environments with different clutter densities and object shapes while also handling a variety of desired objects. (*ii*) The two learning algorithms were capable of learning robust heuristics to un-modelled dynamics. A learned heuristic with IL and RL drives VisualRHP to achieve high success rate and action efficiency compared to a handcrafted heuristic or open-loop execution. When using the continuous action space heuristic, the robot benefits from finer control over its actions resulting in higher action efficiency relative to operating in a discrete action space. (*iii*) The closed-loop control scheme that alternates between real-world execution and VisualRHP in a physics simulator ensures a balance between real-time execution and informed action generation for solving sequential decision making problems.

The success of the experiments relies on the full observability assumption of the environment. However, the full observability assumption may not always hold true when operating in a tight work-space. In the next chapter, we explore the adaptation of VisualRHP to partially observable environments.

Chapter 5

Occlusion-Aware Manipulation under State and Physics Uncertainties

"Yet while the darkness might be frightening, it is also an invitation to those bearing torches"
- Invicta YouTube Channel:
How Carthage Explored the World in Antiquity

5.1 Introduction

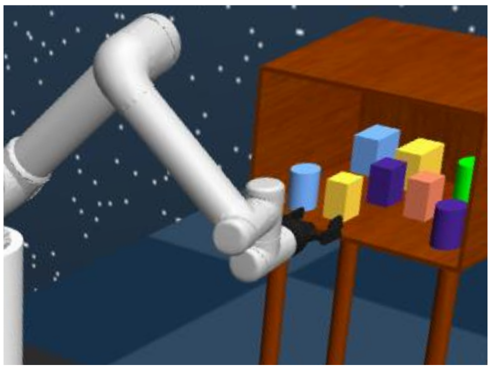
Robots are envisioned to seamlessly adapt to human-centric environments with minimal to no changes to the environment to match the robots' capabilities [71]. The first step, however, is to find the item the robot is looking for [44]. Whether for retrieving an oil bottle from a cluttered pantry or a specific item from a supermarket shelf, installing overhead cameras over every manipulation space defeats the concept of robots adapting to human-centric environments. In this chapter, we propose a manipulation approach for robots with a hand-mounted camera to search and retrieve a desired object from a cluttered environment with occlusions.

A sequence of prehensile and non-prehensile actions in a partially observable and contact-rich environment requires reasoning over occlusions and physics-based uncertainty. Even when high-accuracy object detection systems are available, occlusion remains an inherent source of uncertainty challenging the search for a desired object [53]. The robot has to reason over a history of partial observations to efficiently explore where the desired object might be. In addition, it is notoriously hard to predict the outcome of an action in multi-contact physics environments [97, 21, 69]. Modelling error on the physics parameters such as friction, inertia, and objects shapes impede open-loop execution of long action sequences.

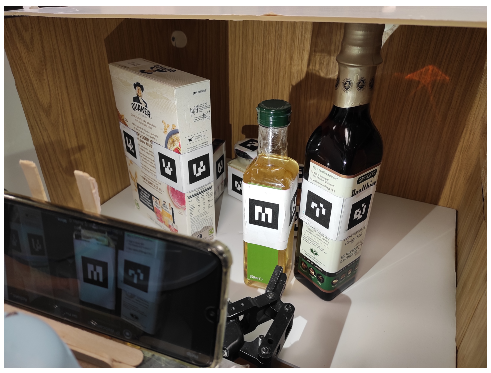
Under the assumption of a fully observable environment, we have shown in the previous chapter how VisualRHP can be used with a heuristic to guide physics-based roll-outs and to estimate the cost-to-go from the horizon to the goal. VisualRHP provides a reliable solution to balance the advantages of model-based sequential reasoning with a model-free scalable heuristic. However, in a partially observable environment, the desired object is not always detected and hence cannot be simulated by VisualRHP. In this chapter, we explore learning to predict the location of the desired object.

We propose (i) a data-driven approach for maintaining a distribution over the desired object pose from a stream of partial observations (ii) and an occlusion-aware heuristic to run VisualRHP under partial observability. These two key ideas form what we call the hybrid planner. The hybrid planner uses the distribution to suggest potential poses of the desired object for VisualRHP to explore. We also present the learning algorithm for simultaneously learning a generative model for the pose distribution of the desired object and an occlusion-aware heuristic in a continuous action space. We evaluate the proposed approach in different simulation environments with varying clutter densities and artificially injected noise. We also validate the proposed approach in a real-world environment.

5.2 Problem Definition



(a) 3D realistic world: the robot perceives the world from a simulated camera attached to its end-effector (see Fig. 5.7 for snapshots from the camera).



(b) Real-world: the robot is shown using a phone camera mounted on its end-effector (see Fig. 5.8 for snapshots from the camera).

Figure 5.1: Execution environment setup.

The robot's task is to retrieve a desired object from a shelf following a sequence of prehensile and non-prehensile actions without dropping any of the other objects off the shelf. The robot carries a hand-mounted camera. A typical execution environment setup is shown in Fig. 5.1. We treat the search, reach, grasp, and pull-out of the desired object as a single optimization problem with the objective of minimizing the total number of actions for retrieving the desired object.

In contrast to the previous chapter, the goal configuration in this chapter consists of having the desired object grasped and removed from the manipulation surface while all the other objects remain on the shelf.

5.2.1 Formalism

We model the problem as a POMDP $\langle S, A, O, T, \Omega, r, \gamma \rangle$, where S is the set of states where a state s at time t is given by $s_t = [Rob, Obj^{desObj}, \ldots, Obj^m, shelf]$, with $shelf$ representing the set of coordinate defining the shelf walls and its edges. A, T, r, and γ are the set of actions, transition probability function, reward function, and discount factor, respectively (similar to Sec. 4.3)[1]. $\Omega : S \times O \to [0,1]$ the observation model. O the set of possible observations, such that an observation $o \in O$ contains a subset of the state variables (e. g., the visible objects), and the geometry of occluded spaces: the shadowed areas behind objects and areas outside the camera's FOV.

Since the state is not always accessible because of occlusions, decision making relies on maintaining a belief $b : S \to [0,1]$ as a distribution over possible states. The belief is continuously updated from the stream of observations with $b_{t+1} = b(s_{t+1}|b_t, a_t, o_t)$ (Eq. 2.31). The POMDP policy π maps a belief b_t to an action a_t. The value of that policy is also computed based on the belief b_t: $v_\pi = \mathbb{E}_{a \sim \pi, s_t \sim b_t}[\sum_{k=t} \gamma^{k-t} r_{k+1}]$. In a model-free approach, it is possible to bypass the explicit modelling of the belief by computing the policy and value function directly from the observation history $\bar{o}$, such that $a \sim \pi(.|\bar{o})$ and $v_\pi = v(\bar{o})$.

5.2.2 Chapter Overview

We build on the framework laid out in the previous chapter and extend it to partially observable environments. The updated framework of the closed-loop control scheme is shown in Fig. 5.2. It is interpreted as follows:

[1]We do not consider discrete actions in this chapter as the experiments from Sec. 4.7 have proven that operating in a continuous action space converges to a more efficient behaviour.

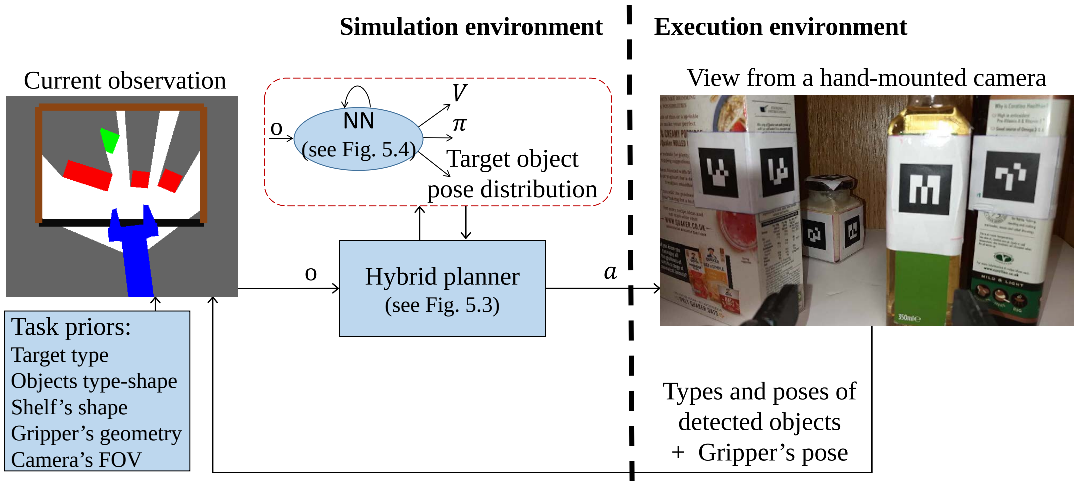


Figure 5.2: Approach overview.

- ***Observe***: The poses and types of visible objects in the execution environment, as detected by the hand-mounted camera, and task priors are used to recreate, in the simulation environment, a state with only the currently detected objects. The current observation, a top-down view of the scene, is rendered from the simulation environment (Sec. 5.3.1). But since the location of the desired object is not always known, it cannot be placed in the observation.

- ***Plan***: The hybrid planner uses the observation history, including the current observation, to update a distribution over the likely poses of the desired object. The estimated desired object poses are used to hypothesize root states, each with a desired object (if the predicted desired object pose is in an occluded area, it would still be hidden in the observation). VisualRHP uses an occlusion-aware heuristic to explore and evaluate physics roll-outs from each of the root states. VisualRHP returns the best action to execute at each root state and its corresponding estimated *return* (Sec. 5.3.2).

- ***Execute***: The *returns* are weighted by the likelihood of their root states, and the action with the highest weighted *return* is executed in the execution environment (Sec. 5.3.2). After a single step of execution, the system goes back to the observation step, for a closed-loop execution.

At the core of the hybrid planner is a Neural Network (NN) with recurrent units that maps an observation history into: (i) a distribution over the pose of the desired object $\hat{y}(\bar{o})$ with $\bar{o}$ being the observation history, (ii) a stochastic policy $\pi(.|\bar{o})$, (iii) and its corresponding value function $v_\pi(\bar{o})$, (Sec. 5.4). The

NN is trained in the physics simulation environment with Imitation Learning (IL) and curriculum-based Reinforcement Learning (RL) (Sec. 5.4).

5.2.3 Assumptions

In this chapter, we adopt the same assumptions as in the previous chapter, but we drop the assumption on full observability. We also add the following assumptions. (*i*) A library of object type-shape pairs is given. (*ii*) Objects are small enough to be graspable from at least one approach angle.

5.3 Decision Making Under Occlusion

5.3.1 Observation Space

It is essential to have an expressive representation of the observation yet compact enough to keep the NN size relatively small as it will be queried multiple times per action selection. Even though in the execution environment the camera is hand-mounted, before we feed the observation into the NN, we render it in a top-down view, as shown in the top-left of Fig. 5.2, making the spatial relationships between objects and the geometry of occluded and observable areas more explicit.

We built on the abstract image-based representation of a fully observable environment. In addition to colour labelling objects based on their functionality (e. g., desired object in green and clutter objects in red), we represent occluded and observable spaces by white and grey coloured areas, respectively. The geometry of the occluded areas is computed by illuminating the scene from the robot's camera perspective. We use a black line to represent the shelf edge and brown for the shelf walls. The top-down view enables data from the execution environment and task priors to be combined.

- Object detection on the execution environment identifies the poses and types of visible objects in the camera FOV. The objects' poses and types allow the simulation environment to place the correct object shape and colour in the abstract image-based representation of the observation.

- The task priors consist of observation-invariant information: the type of the desired object, the shape corresponding to every object type, the shape of the shelf (walls and edge), the geometry of the gripper, and the camera FOV. By including task priors in the representation, the learner does not need to remember them from the observation stream.

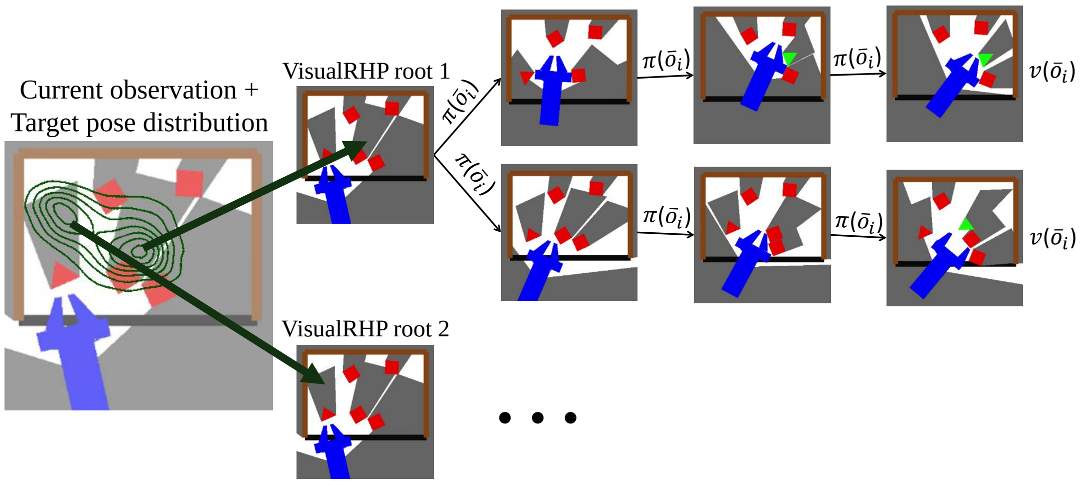


Figure 5.3: Hybrid planner running 2 VisualRHP queries, one for each peak represented by the contour lines (left). VisualRHP is shown executing 2 roll-outs of depth 3 for each root state.

All abstract images are made robot centric before being fed into the NN to reduce the observation space by exploiting symmetries.

5.3.2 Hybrid Planner

The hybrid planner algorithm, presented in Alg. 3 and illustrated in Fig. 5.3, is detailed as follows:

State Generation (Alg. 3, line 2): Having information from previous observations captured in the NN recurrent units, the NN uses the current observation to generate a distribution over the pose of the desired object. For each peak in the distribution, the hybrid planner creates a state with the desired object at the peak location, while the poses of the obstacles remain the same as in the current observation. The weight of a root state is computed as the relative likelihood of its corresponding peak. It measures how likely it is for the desired object to be found at the predicted location compared to the other potential sites. VisualRHP is then called over each of the root states (Alg. 3, line 4)

Occlusion-aware VisualRHP (Alg. 4): VisualRHP performs n stochastic roll-outs from root state s_r up to a fixed horizon depth h in the physics simulator. Each roll-out is executed by following the stochastic policy $\pi(\bar{o})$ acting on the observation history. The *return* $R_{r:h}$ of a roll-out is computed as the sum of the discounted rewards generated by the model and the expected *return* beyond the horizon estimated by the value function $v(\bar{o}_h)$:

$$R_{r:h} = r_1 + \gamma r_2 + \ldots + \gamma^{h-1} r_h + \gamma^h v(\bar{o}_h). \tag{5.1}$$

VisualRHP returns the *first* action a_r and $R_{r:h}$ of the roll-out that obtained the highest *return*.

Action Selection (Alg. 3, line 7): The *return* of a VisualRHP query is scaled by the weight of its root state (Alg. 3, line 6). Therefore, the hybrid planner picks the action that maximizes the *return* with respect to both the probability of the roll-out, and the probability of the location of the desired object.

```
Algorithm 3: Hybrid planner (NN, $\bar{o}$, $n$, $h$)
  Input: observation history $\bar{o}$, number of roll-outs $n$, horizon depth $h$
  Output: action $a_r$
1 rootActions ← [ ],  weightedReturns ← [ ]
2 rootStates, rootWeights ← generateStates(NN, $\bar{o}$)
3 for $s_o, w$ in [rootStates, rootWeights] do
4     $a_r, R_{r:h}$ ← VisualRHP(NN, $s_r$, $\bar{o}$, $n$, $h$)
5     rootActions.append($a_r$)
6     weightedReturns.append($w \times R_{r:h}$)
7 return rootActions[argmax(weightedReturns)]
```

```
Algorithm 4: VisualRHP (NN, $s_r$, $\bar{o}$, $n$, $h$) with an occlusion-aware heuristic
  Input: root state $s_r$, obs. history $\bar{o}$, number of roll-outs $n$, depth $h$
  Output: action $a_r$, return $R$
  RolloutsReturn ← [ ],  FirstAction ← [ ]
  for $i$ = 1,2, ..., $n$ do
      $R \leftarrow 0$,  $\bar{o}_i \leftarrow \bar{o}$
      $s$, $o$ ← setSimulatorTo($s_r$)
      $\bar{o}_i$.append($o$)
      for $j$ = 1,2, ..., $h$ do
          $a \sim \pi(.|\bar{o}_i)$
          if $j$ is 1 then
              FirstAction.append($a$)
          $s$, $o$, $r$ ← simulatePhysics($s, a$)
          $R \leftarrow R + \gamma^{j-1} r$
          $\bar{o}_i$.append($o$)
          if isTerminal(s) then
              break
      if not isTerminal(s) then
          $R \leftarrow R + \gamma^h v(\bar{o}_i)$
      RolloutsReturn.append($R$)
  return FirstAction[argmax(RolloutsReturn)],
         max(RolloutsReturn)
```

5.4 Model-Free Learning

Prior to using the NN in the closed-loop control scheme, the NN is trained in a physics simulation environment (the same environment that will be used by the hybrid planner). The NN must (i) generalize over a variable number of objects and shapes in the observations, (ii) and maintain a belief from the observation stream in order to predict the distribution over the desired object pose and to generate an informed search and retrieve policy and value function for VisualRHP to use them as a heuristic. The NN architecture that satisfies these conditions is illustrated in Fig. 5.4. The first two components are a Convolutional Neural Network (CNN) connected to Long Short-Term Memory (LSTM) units. The CNN takes advantage of having an abstract image-based representation of the observation to ensure generalization over object shapes and numbers. The output of the LSTM layer, $\hat{b}$, summarizes the stream of CNN embeddings into a latent belief vector. $\hat{b}$ is then passed through a feed-forward Deep Neural Network (DNN) that models the policy, another DNN for the value function, and a generative head for estimating the pose distribution of the desired object. The generative head outputs a heat-map, $\hat{y}$, of size equal to the input image, where higher pixel values indicate higher chances that the desired object is at that location. While the policy and value function share some of the NN parameters, we also found that having the generative head sharing the CNN and LSTM components of the NN with the policy and value function acts as a regularizing element.

Training a randomly seeded θ-parametrized NN with recurrent units over images in a partially observable environment with complex physics and in a continuous actions space is particularly challenging [81]. To increase the likelihood of convergence, we follow a similar training procedure to the one introduced in Sec. 4.5 with an additional curriculum for the RL training [85].

Imitation Learning: In the first learning phase, we collect P demonstrations, in the form of state-action sequences $\langle s_0, y_0, a_0, \ldots, s_{L-1}, y_{L-1}, a_{L-1}\rangle_p$. The demonstrations are generated by solving random task instances using the

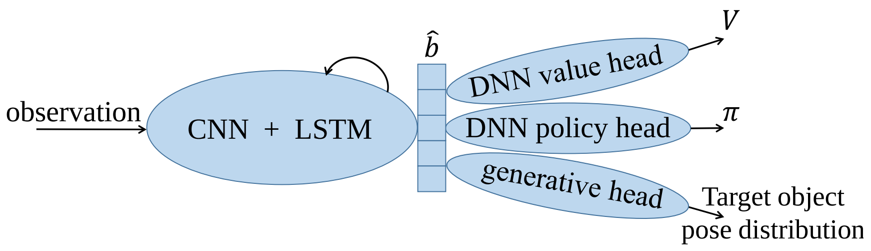


Figure 5.4: NN architecture.

Kino-dynamic RRT planner over a fully observable version of the problem, that is, with $o \equiv s$. y is the heat-map showing the ground truth pose of the desired object as given by the simulator. The NN is trained over the fully observable and sub-optimal demonstrations. We formulate a similar loss function to Eq. 4.8, but the policy and value function are computed over the observation history and not just the current state. An additional term is also added for training the generative head to estimate y:

$$\begin{aligned}\mathcal{L}_{\text{il}}(\theta) = \mathbb{E}_{\bar{o}_l^p, y_l^p, a_l^p}[-\ & \Psi log\pi_\theta(a_l^p|\bar{o}_l^p) \\ +\ & c_1\ (v^{tar}(\bar{o}_l^p) - v_\theta(\bar{o}_l^p))^2 \\ +\ & c_2\ H(\pi_\theta(.|\bar{o}_l^p)) \\ -\ & c_3\ \frac{1}{jk}\sum_{j,k}(y_l^{p,jk}\text{log}\hat{y}_\theta^{jk}(\bar{o}_l^p) + (1 - y_l^{p,jk})\text{log}(1 - \hat{y}_\theta^{jk}(\bar{o}_l^p))],\end{aligned} \tag{5.2}$$

where j and k are the heat-map pixel indices. This learning phase is not enough to solve the problem particularly under partial observability. Nevertheless, it offers RL a better starting point than a randomly seeded NN.

Curriculum-based Reinforcement Learning: Next, the NN is trained under partial observability with curriculum-based RL. The curriculum is constructed over three task parameterizations to gradually increase the clutter density and, by consequence, the occlusion in the environment. The first parameterization consists of environments with a random number of objects between 1 and 4. The initial poses of the desired and clutter objects are sampled from a uniform distribution over the shelf. The next task parameterization uses between 5 and 10 objects. The final task parameterization limits the minimum number of objects to 7 and the pose of the desired object is sampled from a uniform distribution covering only the back half of the shelf. Throughout the training, we use random polygon-shaped objects for the NN to learn generalizable features.

The policy and the value function are trained with an A2C algorithm. The generative head is trained in the same supervised fashion as in the previous learning phase. The combined loss function over batches $B = \{\langle s_i, y_i, a_i, r_i, s_i' \rangle_{i=1}^M\}$

is therefore:

$$\begin{aligned}\mathcal{L}(\theta) = \frac{1}{M}\sum_{i=1}^{M} - \ & Adv(\bar{o}_i, r_i, \bar{o}_i')\frac{\pi_\theta(a_i|\bar{o}_i)}{\pi_{\theta_{old}}(a_i|\bar{o})} \\ & + c_1\ (r_i + \gamma v_{\theta_{old}}(\bar{o}_i') - v_\theta(\bar{o}_i))^2 \\ & - c_2\ H(\pi_\theta(.|\bar{o}_i)) \\ & - c_3\ \frac{1}{jk}\sum_{j,k}(y_i^{jk}\log\hat{y}_\theta^{jk}(\bar{o}_i) + (1 - y_i^{jk})\log(1 - \hat{y}_\theta^{jk}(\bar{o}_i)), \qquad (5.3)\end{aligned}$$

where the advantage function is computed over the observation history:

$$Adv(\bar{o}_i, r_i, \bar{o}_i') = r_i + \gamma v_{\theta_{old}}(\bar{o}_i') - v_{\theta_{old}}(\bar{o}_i).$$

5.5 Simulation Experiments

We ran a number of experiments in different physics environments. The goals of the experiments are two-fold: (*i*) to evaluate the performance of the proposed approach in dealing with occlusion and physics uncertainties, (*ii*) to verify the approach's transferability to environments with different physics parameters. A video of these experiments is available on https://youtu.be/khweZ4FXWfo.

5.5.1 Evaluation Metrics

We select evaluation metrics that allow us to quantitatively measure the aforementioned goals.

- The first metric is **success rate**. A task is considered successful if the desired object is retrieved in under 50 actions, the total task planning and execution time is under 2 minutes, and none of the objects are dropped off the shelf.
- As we also target real-time applications, the second metric is the **average planning and execution time per task**.
- The **average number of actions per task** is the third metric as the learning objective is to solve the problem with the minimum number of actions.

Each data point in the experiment results is averaged over 300 task instances. The experiments were conducted on an Intel Xeon E5-26650 computer equipped with an NVIDIA Quadro P6000 GPU.

5.5.2 The hybrid Planner and Baseline Methods

In addition to the hybrid planner, we present three reference baseline approaches.

Hybrid planner: The NN is trained as in Sec. 5.4. It takes a $64 \times 64 \times 3$ input image. The CNN is composed of three consecutive layers of convolution, batch normalization, and maxpooling. We use 8, 8, 16 filters of size 3×3 and strides 2×2. The CNN is followed by a single LSTM layer of 128 units. The policy head is composed of two dense layers with 128 neurons each. The policy output layer has 8 neurons corresponding to the means and standard deviations of the horizontal, lateral, rotational, and gripper actions. We use *tanh* activation function for the means and *sigmoid* for the standard deviations. The value head has two dense layers with 128 and 64 neurons respectively, and a single neuron for the output with linear activation function. The generative head follows a sequence of three upsampling and convolution layers. The filter sizes are 8, 8, 16 and 3×3. The final layer is a $64 \times 64 \times 1$ convolution layer with linear activation function followed by a *sigmoid* function to decode the heat-map. Except for the output layers, we use a leaky *ReLU* activation throughout the network. The NN is updated using the RMSProp optimizer in TensorFlow [2]. We use the PPO formulation to *clip* the policy loss. We use the following learning parameters: $P = 10000$, $\Psi = 0.5$, an IL learning rate of 0.0001, an RL learning rate of 0.00005 and a D^{replay} size of 10000 transition samples. We set $c_1 = 0.5$, $c_2 = 0.01$, c_3=1.0, γ=0.995, and $M = 1500$. We use 10 parallel environments each with its own agent. We run 15 epochs at the optimization step. We compare three versions of the hybrid planner with n and h VisualRHP parameters of 2×2, 4×4, and 6×6.

Hybrid planner limited: Instead of performing weighted evaluations of multiple VisualRHP queries, this baseline only evaluates the most likely desired object pose and executes the predicted root action for that pose. We implement it with $n = 4$ and $h = 4$.

Greedy: This policy presents a deterministic model-free approach. The NN is trained similarly to our approach excluding the generative head from the architecture. The robot is directly controlled by the policy head of the NN (without VisualRHP). Actions are defined by the mean of the action distribution outputted by the policy head over the continuous actions space. It is inspired by [86].

Stochastic: This policy is a stochastic version of the greedy policy. Actions are sampled from the policy output. As shown in [47], RL trained stochastic policies provide higher *return* than deterministic ones in a POMDP.

Hierarchical planner: This approach offers a model-based baseline. The low level plans are generated either with the Kino-dynamic RRT planner or following a hand-crafted heuristic. The low level plans are executed in open-loop. The high level planner has access to the following actions:

- Search( ): positioned outside the shelf, the robot moves from the far left to the far right of the shelf while pointing the camera inwards. Throughout this motion, information is collected on the pose and type of detected objects.
- Rearrange(Obj^i): move a certain object to a free-space in the back of the shelf by planning with Kinodynamic RRT on collected information from the previous Search action.
- Move_out( ): rotates the robot to face the inside of the shelf, then moves the robot out following a straight line heuristic.
- Retrieve(Obj^{des}): plan with Kinodynamic RRT on available information to reach, grasp, and pull-out the desired object.

The high level planner is outlined in Alg. 5. This baseline is an adaptation of [28].

Algorithm 5: Hierarchical planner

```
while desired object not retrieved do
    Search( )
    if desired object not located then
        Rearrange(closest object to robot)
        Move_out( )
    else Retrieve($Obj^{des}$) ;
```

5.5.3 2D Simulation Experiments' Setup

Setup: We use two Box2D physics simulators, one acting as the execution environment and the other as the simulation environment where planning is performed. The reward function for this task is $r = -1$ per action and $r = -50$ for dropping an object of the shelf. The experiments evaluate the performance w. r. t. increased clutter density and increased noise level on the shape and physics parameters in the execution environment. The increase in clutter density is aimed at challenging the robot with higher occlusion ratios and more complex multi-object interactions. The increase in the noise level addresses

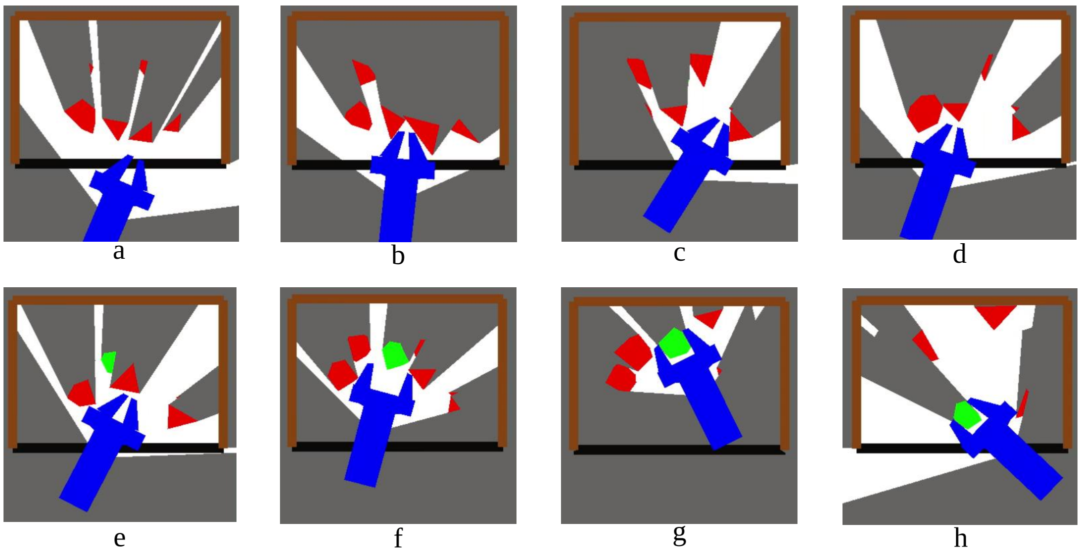


Figure 5.5: Snippets of the current observation with noise level=0.15. Task solved with Hybrid$^{4\times4}$.

modelling errors between the execution environment and the simulation environment. The noise is added on the shape and physics parameters of an object before the execution of an action as in Sec. 4.8.2. We experimented with noise levels ranging from 0.0 to 0.25 standard deviation around the mean, and with a random number of obstacles up to 10. An experiment with noise level $= 0.15$ using Hybrid$^{4\times4}$ is shown in Fig. 5.5. The width and depth of the shelf are W:50×D:35 *cm*. The dimensions of the gripper are modelled after a *Robotiq 2F-85* gripper mounted on a *UR5* robot.

5.5.4 Simulation Results and Analysis

The results are shown in Fig. 5.6. In terms of success rate (Fig. 5.6-left), we observe a decreasing trend w. r. t. clutter density and higher noise levels. This is expected as the task becomes more challenging with higher occlusion ratio and changing dynamics. The hybrid planner outperforms the other baselines. Its success rate improves with higher number of roll-outs and horizon depth. Performing a weighted evaluation over the predicted poses achieves a slightly higher success rate than just evaluating the most likely one. Furthermore, the stochastic policy outperforms the greedy policy. This improvement may be the result of the additional information gained from a stochastic motion. The stochastic and greedy policies exhibit similar success rates with higher noise levels. This is because the changes in physics and object shapes introduce enough randomness in the system for the greedy policy to act in a similar fashion to the stochastic policy. The hierarchicalh planner suffers from the

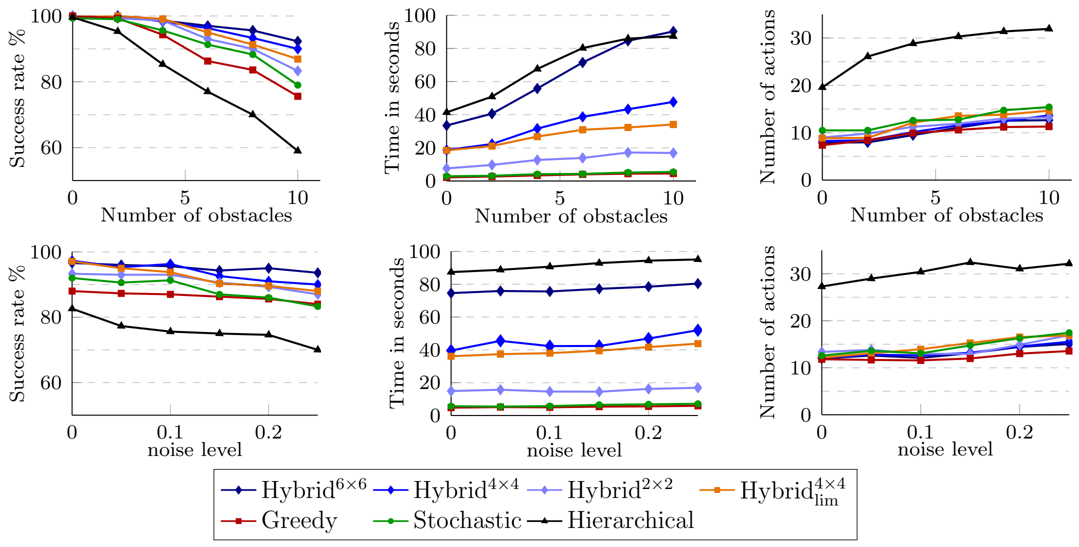


Figure 5.6: Performance w. r. t. different clutter densities and noise levels.

sharpest drop in success rate in both experiments. The open-loop execution often fails to produce the intended results.

The average time per task results (Fig. 5.6-middle) show a clear advantage for the model-free approaches. Actions are generated almost instantaneously. The Hybrid planner time degrades with more exhaustive VisualRHP searches. The difference between $Hybrid^{4\times4}$ and $Hybrid^{4\times4}_{\text{lim}}$ is not significant despite the latter achieving lower time per task. This result indicates that the hybrid planner does not often generate a large number of potential positions for the desired object which would have otherwise resulted in a bigger time difference. The hierarchical planner average time is on par with the Hybrid$^{6\times6}$ planner. These results indicate that simulating the physics during planning is the computation bottleneck in a contact-rich environment.

Except for the hierarchical planner, all of the approaches perform a similar number of actions per task (Fig. 5.6-right). Evidently, the stochastic policy performs slightly worse than the hybrid planner, while the greedy policy is the most efficient. The hybrid planner, despite relying on stochastic roll-outs, executes fewer actions than the stochastic policy as decision making is better informed with VisualRHP. The scale of the number of actions for the hierarchical planer is highly dependent on the parameters of the underlying low level planners. Nevertheless, with a high noise level and clutter density, the high level planner increasingly calls the low level planner for re-planning.

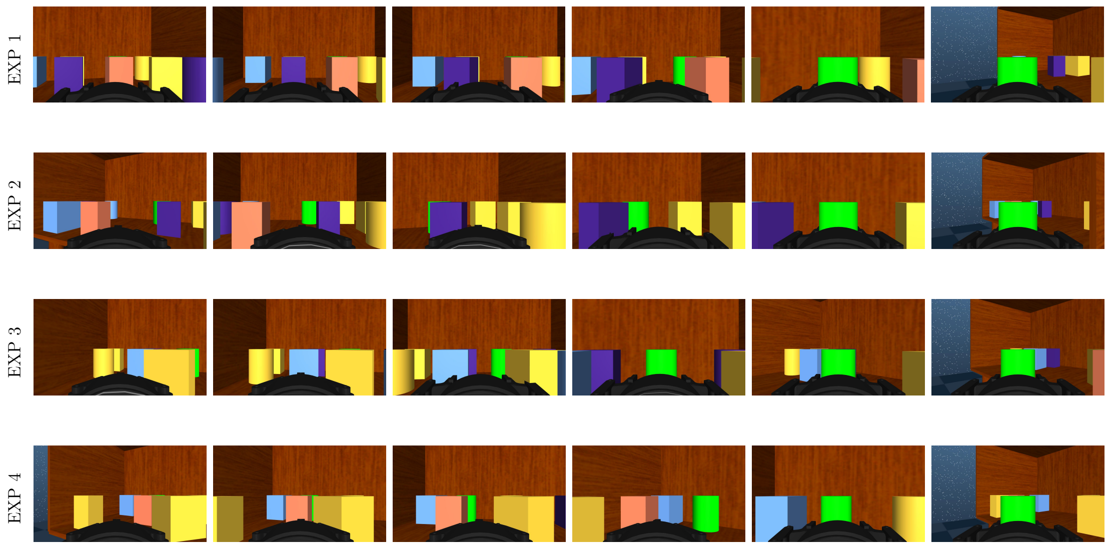


Figure 5.7: Snapshots of using $Hybrid^{4\times4}$ to retrieve the green object.

5.5.5 Realistic Experiments

The simulation results showed that the hybrid planner can be reliably used in environments with different physics parameters. To further validate this finding, we tested our approach in a realistic setup. We use the 3D MuJoCo physics engine with the Deepmind Control Suite [107] as the execution environment, and Box2D as the simulation environment for the hybrid planner.

To replicate a conservative performance of real-world object detection tools from a single image in clutter (e. g., AR-markers, PoseCNN [117], DenseFusion [113], etc.), the execution environment (having access to the ground truth) would only report to the simulation environment the poses and types of objects whose more than 50% of their body is visible within the current camera view.

We conducted 30 trials with random number of obstacles, up to 10. The VisualRHP parameters are set to $n = 4$ and $h = 4$ as they offer a reasonable balance between success rate and execution time. The shelf dimensions are W:50×D:35×H:30 cm. We also experimented with the stochastic policy as it showed the second best success rate in the previous experiments.

The hybrid planner and the stochastic policy achieved a success rate of 88% and 79%, respectively. These results are similar to the previous experiment with high noise levels. Examples of tasks solved with the hybrid planner are shown in Fig. 5.7, and in the experiments' video. The hybrid planner demonstrates that when the target object is not visible, the robot performs information-gathering actions by advancing into the shelf and manipulating obstacles to

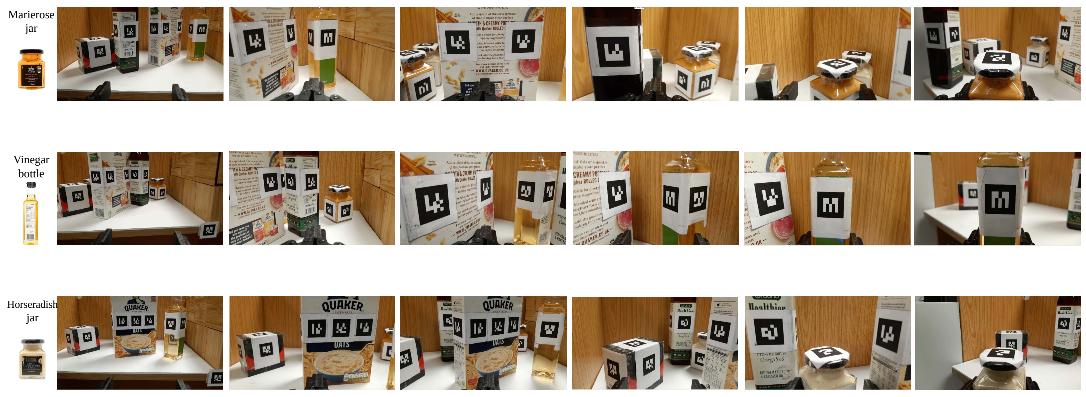


Figure 5.8: Snapshots of using $Hybrid^{4\times4}$ to retrieve the different desired objects in the real-world.

increase visibility. When the robot loses sight of a previously detected target object, due for example to an obstacle blocking the camera view, the robot focuses its search on the area where the target object was last seen.

5.6 Real-World Experiments

To validate the results of the previous experiments, we conducted a number of real-world experiments with $Hybrid^{4\times4}$ for retrieving a variety of everyday objects from a cluttered shelf.

We mounted an RGB camera on the end-effector of a *UR5* with a *Robotiq 2F-85* gripper. We used the Alvar AR tag tracking library for object poses and types detection [98][2]. We used Box2D as the simulation environment to run the hybrid planner. The shelf dimensions are W:50×D:35×H:40 *cm*.

Snapshots from these expreiments are shown in Fig. 1.1 and Fig. 5.8. A video of these experiments is also available on https://youtu.be/4iSsogfCkMc. The robot behaviour in the real world is reminiscent to what we observed in the realistic experiment.

In the experiment where the robot is tasked with retrieving the oil bottle, the robot first approaches the middle of the shelf and searches the area behind the clutter. Once the oil bottle is spotted, the robot goes around the cereal box, losing sight of the oil bottle, then reaches again for the oil bottle from a less cluttered direction and retrieves it from the shelf. In the second experiment where the robot is tasked with retrieving the marierose jar, the robot pushes

[2]A more natural setting would avoid fiducial markers, similar to our Chapter 4 implementation. Due to the additional depth dimension and partial occlusions in this chapter, another set of computer vision tools would be required. We leave a fiducial-free implementation for future work.

obstructing objects to clear an approach for grasping and retrieving the jar. In the experiment where the robot is tasked with retrieving the vinegar bottle, we observe the importance of a reactive behaviour when the bottle slips from the robot grasp. The robot is able to recover from this situation by reopening the gripper, approaching the bottle, and then re-grasping it and pulling it out of the shelf.

5.7 Preliminary Work on Adaptive VisualRHP

In this work, VisualRHP uses fixed values for the number of roll-outs and the horizon depth parameters. To save on planning and execution time, VisualRHP would benefit from dynamically adjusting its parameters after every execution step. Two concepts have to be considered in setting the values of these parameters.

The first concept is related to the *difficulty* of solving the task from the current state. States from which it is easier to reach the goal require fewer and shallower roll-outs, and vice versa. Quantifying the *difficulty* in view of the robot skill level is not straight forward. Some of the factors that we suspect affect the *difficulty* include: how close objects are to the surface edges, how spread out or clustered the objects are, how many obstacles are in between the robot and the desired object, etc. One potential approach is to handcraft a *difficulty* function. However, such a function doesn't take into account the performance of the current policy. For instance, a certain policy might be good at navigating tightly packed objects while another might be better at handling situations where objects are close to the surface edges. Alternatively, it might be possible to train a model-free oracle to predict the probability of success from the current state. The oracle would be trained on labelled states or observation histories generated by the current policy. A label would simply be $\{0, 1\}$ identifying if a state or an observation history belongs to an episode that resulted in success or failure. The idea of training an oracle is left for future work.

The second concept is related to the *confidence* of the policy in generating the best action for the current state. Sates with high confidence score would requires fewer roll-outs and shorter depth, and vice versa. We hypothesise that the policy's entropy can be interpreted as a proxy for the inverse of the *confidence*. In other words, a high entropy at a certain state reflects that the policy is not committed to a specific action and hence must further leverage VisualRHP. We ran an experiment to test this hypothesis. We discretized the entropy into

Table 5.1: Comparing the hybrid planner with fixed and adaptive VisualRHP parameters

		Hybrid$^{4\times4}$	Hybrid$^{\text{Adaptive}}$
With occlusions	suc. rate [%]	93	89
	avg. time [s]	23	14
Without occlusions	suc. rate [%]	94	92
	avg. time [s]	31	19

6 levels ranging from H_{min} to H_{max} which are the minimum and maximum entropy values observed in our previous experiments. We set $n = h = H_i$ where H_i is the corresponding entropy level of $H(\pi(.|s))$ and $H(\pi(.|\bar{o}))$ in environments with and without occlusions, respectively. We compared in simulation two hybrid planners, one running with adaptive VisualRHP parameters and the other using VisualRHP with $n =$ and $h = 4$. We ran 300 random task instances with up to 7 objects. The results are shown in Table 5.1. The adaptive parameters show promising results. The average time per task drops by around 40% when using the adaptive parameters in both experiments. The difference in success rate is not significant despite a small advantage to the fixed parameters.

However, using the entropy as a proxy for the inverse of the *confidence* is not always a valid assumption. In some states, a wide range of actions could have similar expected *return*, and thus they are equally good (or bad). The approximation of the optimal policy at these states would have high entropy but it would be wrong to interpret it as low *confidence*. A better definition and quantification of *confidence* would further reduce the average time per task and possibly improve the success rate by performing more and deeper roll-outs where needed.

5.8 Conclusion

The experiments have shown the robustness and transferability of the hybrid planner in challenging environments. They provide evidence for the effectiveness of combining a model-free pose estimator with model-based look-ahead planning. The robot's behaviour validates that the NN stores relevant information from past observation to guide future actions. The robot exhibited an informed search behaviour by executing actions that increase visibility of previously unobserved spaces and by manipulating objects to reveal occluded areas behind them. In comparison to the other baseline approaches, the hybrid planner can react in near real-time while also accounting for complex physics interactions under occlusion.

Chapter 6

Future Work and Conclusion

"Inventions reached their limit long ago, and I see no hope for further development."
- Julius Frontinus (Highly respected engineer in Rome, 1^{st} century A.D.)

6.1 Results Summary

We have shown the strength and weaknesses of different manipulation in clutter approaches. Choosing which approach to use is highly application dependent. All approaches we experimented with presented a reliable behaviour in environments with low clutter density. In environments with higher clutter densities, model-free approaches proved to be the best option for time-critical applications, but less so if the cost of failure is high. The model-based approaches can be reliably used where it is more likely to find a collision-free trajectory, e. g., decluttering objects from the front of the shelf. They are best suited for applications where high-fidelity physics-simulator and object models are available.

Occlusions and physics uncertainty, however, makes it hard to determine the environment's characteristics a priori. Under different environment uncertainties, the hybrid planner provided the most robust behaviour for near real-time applications. Nevertheless, the hybrid planner still has room for improvements.

6.2 Optimizing the Learning Process

In this work, we focused on the convergence of the learning algorithm to a good heuristic irrespective of the computation cost. The learning time was treated as a sunken cost since it does not affect the performance at execution time. However, for various reasons (energy cost, limited computation resources, etc.) it is also desirable to minimize the computation cost of the learning process. In what was presented, we generated a large number of demonstrations for the Imitation Learning (IL) algorithm to extract as much knowledge as possible from the transition samples. Reinforcement Learning (RL) was then used to further refine the knowledge capture by IL. Alternatively, it is possible to reduce the amount of time spent on data collection for IL in favour of RL . A fair study of the proposed learning process would require investigating the trade-off between data collection for IL and data collection for RL. Such study would shed light on how the computation cost and the performance of the heuristic are affected by the source of the collected data.

6.3 From 2D to 3D Environments

Despite the many applications that acting in a 2D plane can tackle, they remain a small subset of the overall manipulation tasks that a robot is expected solve. We advocate extending the hybrid planner to reasoning over 3D geometries. Leveraging the framework presented in this thesis, we propose investigating the use of an abstract colour-labelled 3D voxelized representation of the space. The voxelized representation would carry information on the current observation. Similarly to our abstract image representation, voxels that are not within the camera FOV would be coloured in a certain colour while free-space would have another colour. The same colour-labelling concept would also apply to the desired object, obstacles, robot geometry, shelf walls, and surface edges. Such extension would have to overcome 3 main challenges:

- The first is to accurately estimate the 6D objects poses from a single image in clutter. For everyday applications, the use of AR markers would not be practical. Despite the continuous improvement in state-of-art 6D pose estimators from RGB and RGB-D images, having a camera, and possibly a depth sensor, close to the objects still poses a challenge to the detection algorithms.

- 3D physics simulators are continuously being improved for accuracy and computation efficiency. Albeit, simulating 3D contacts remains a challenge. Compared to the 2D physics simulator used in this work, 3D simulators are substantially slower and less stable. Our experience with MuJoCo proved that much work is still needed to accurately model multi-object contacts. In the video showing the realistic experiments, one can notice how the body of the gripper sometimes penetrates the inside an object's geometry. We have also faced stability problems when using the 3D simulator OpenRave with the *Open Dynamics Engine* physics engine

[26]. Further, simulating contacts between 3D meshes (compared to primitive object shapes such as boxes and cylinders) proved to be even more of a challenge. Nevertheless, promising advances in this field such as the physics engine *PhysX* 4.0 by NVIDIA might mitigate some of the stability, accuracy, and computation efficiency challenges. A quantitative study comparing the available 3D simulators and physics engines would surely accelerate the researcher in this field.

- In a 3D environment, the stability of the learning process would be challenged by a larger parameter, state, and action spaces compared to the 2D environment used in this work. The NN architecture could have a similar structure to the one we used, with the difference being the use of 3D CNN instead of the currently used 2D CNN. The increase in the dimensions of the input space would necessitate more exploration before convergence. The increase in the action space would mean that the policy will have to learn how to achieve stable grasps in 3D and implicitly learn how 3D objects interact, although the latter can be mitigated to some extent by a look-ahead planner. We suggest investigating the use of state-of-art off-policy RL algorithms in continuous action space such as SAC [37]. An off-policy algorithm would allow the acting RL policy to exploit VisualRHP for exploration. Such an approach would be particularly useful if implemented with adaptive VisualRHP: at the early stages of the training, the acting policy would rely extensively on long roll-outs to collect goal leading samples. Then, as the policy starts to improve, VisualRHP parameters will be reduced, decreasing the dependency on the model in favour of the learned policy. In other words, slow model-based planning will gradually shift to fast model-free decision making. This training strategy is inspired to us by the book "*Thinking, Fast and Slow*" [54].

6.4 Conclusions

This thesis provided a stepping stone towards applications such as object retrieval from fridges and supermarket shelves with limited height. The bulk of this work focused on developing stable learning algorithms and real-time planners that guide a robot in complex physics environments. We conclude with the following:

- Simplified representations that carry enough geometric and task relevant information can be tremendously leveraged to ensure generalization, transferability, and real-time decision making.
- Simulation-based physics predictions, when interleaved with real-world execution, play a major role in reducing occlusions and physics uncertainty.
- Coupling model-free and model-based approaches opens-up opportunities to scalable solutions for simultaneously reasoning over the observation history and future actions.

We believe the findings of this thesis form a solid foundation for future research to achieve the ultimate goal of ubiquitous manipulation skills.

Bibliography

[1] "A roadmap for us robotics from internet to robotics, 2020". In: *Computing Community Consortium & University of California, San Diego* (2020).

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015. URL: `https://www.tensorflow.org/`.

[3] Agboh, W., Grainger, O., Ruprecht, D., and Dogar, M. "Parareal with a Learned Coarse Model for Robotic Manipulation". In: *arXiv preprint arXiv:1912.05958* (2019).

[4] Agboh, W. C. and Dogar, M. R. "Pushing fast and slow: Task-adaptive planning for non-prehensile manipulation under uncertainty". In: *International Workshop on the Algorithmic Foundations of Robotics.* Springer. 2018, pp. 160–176.

[5] Agboh, W. C. and Dogar, M. R. "Real-Time Online Re-Planning for Grasping Under Clutter and Uncertainty". In: *IEEE-RAS 18th International Conference on Humanoid Robots.* IEEE. 2018.

[6] AISBL, E. R. "Robotics 2020 multi-annual roadmap for robotics in Europe, call 1 ict23–horizon 2020". In: *Initial Release B* 15.01 (2014), p. 2014.

[7] Anthony, T., Tian, Z., and Barber, D. "Thinking fast and slow with deep learning and tree search". In: *Adv Neural Inf Process Syst.* 2017.

[8] *Aussies Win Amazon Robotics Challenge.* `https://spectrum.ieee.org/automaton/robotics/industrial-robots/aussies-win-amazon-robotics-challenge`. Accessed: 2020-03-10.

[9] Bejjani, W., Agboh, W. C., Dogar, M. R., and Leonetti, M. "Occlusion-Aware Search for Object Retrieval in Clutter". In: *arXiv preprint arXiv: 2011.03334* (2020).

[10] Bejjani, W., Dogar, M. R., and Leonetti, M. “Learning Physics-Based Manipulation in Clutter: Combining Image-Based Generalization and Look-Ahead Planning”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 6562–6569.

[11] Bejjani, W., Leonetti, M., and Dogar, M. R. “Learning Image-Based Receding Horizon Planning for Manipulation in Clutter”. In: *Robotics and Autonomous Systems* (2021).

[12] Bejjani, W. “Automated Planning of Whole-body Motions for Everyday Household Chores with a Humanoid Service Robot”. MA thesis. Technische Universität Dortmund, 2015.

[13] Bejjani, W., Dogar, M. R., and Leonetti, M. “Heuristic Learning for Look-Ahead Manipulation Planning in Continuous Action Space”. In: *IROS workshop: Different Approaches, the Same Goal: Autonomous Object Manipulation* (2019).

[14] Bejjani, W., Papallas, R., Leonetti, M., and Dogar, M. R. “Learning a Value Function Based Heuristic for Physics Based Manipulation Planning in Clutter”. In: *IROS workshop: Machine Learning in Robot Motion Planning* (2018).

[15] Bejjani, W., Papallas, R., Leonetti, M., and Dogar, M. R. “Planning with a Receding Horizon for Manipulation in Clutter using a Learned Value Function”. In: *IEEE-RAS 18th International Conference on Humanoid Robots*. IEEE. 2018.

[16] Bianco, S., Cadene, R., Celona, L., and Napoletano, P. “Benchmark analysis of representative deep neural network architectures”. In: *IEEE Access* 6 (2018), pp. 64270–64277.

[17] Bohg, J., Hausman, K., Sankaran, B., Brock, O., Kragic, D., Schaal, S., and Sukhatme, G. S. “Interactive perception: Leveraging action in perception and perception in action”. In: *IEEE Transactions on Robotics* 33.6 (2017), pp. 1273–1291.

[18] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012).

[19] Catto, E. *Box2D*. `http://box2d.org/`. 2015.

[20] Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., and Mouret, J.-B. “A survey on policy search algorithms for learning robot controllers in a handful of trials”. In: *IEEE Transactions on Robotics* (2019).

[21] Chung, S.-J. and Pollard, N. “Predictable behavior during contact simulation: a comparison of selected physics engines”. In: *Computer Animation and Virtual Worlds* 27.3-4 (2016), pp. 262–270.

[22] Clavera, D. I. and Abbeel, P. “Policy transfer via modularity”. In: *IROS. IEEE* (2017).

[23] Coleman, C., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., and Zaharia, M. “Dawnbench: An end-to-end deep learning benchmark and competition”. In: *Training* 100.101 (2017), p. 102.

[24] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. “Analysis and observations from the first amazon picking challenge”. In: *IEEE Transactions on Automation Science and Engineering* 15.1 (2016), pp. 172–188.

[25] Danielczuk, M., Kurenkov, A., Balakrishna, A., Matl, M., Wang, D., Martín-Martín, R., Garg, A., Savarese, S., and Goldberg, K. “Mechanical search: Multi-step retrieval of a target object occluded by clutter”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 1614–1621.

[26] Diankov, R. “Automated Construction of Robotic Manipulation Programs”. PhD thesis. Carnegie Mellon University, Robotics Institute, 2010. URL: `http://www.programmingvision.com/rosen_diankov_thesis.pdf`.

[27] Dogar, M. and Srinivasa, S. “A framework for push-grasping in clutter”. In: *Robotics: Science and systems VII* 1 (2011).

[28] Dogar, M. R., Koval, M. C., Tallavajhula, A., and Srinivasa, S. S. “Object search by manipulation”. In: *Autonomous Robots* 36.1-2 (2014), pp. 153–167.

[29] Dogar, M. R. and Srinivasa, S. S. “A planning framework for non-prehensile manipulation under clutter and uncertainty”. In: *Autonomous Robots* 33.3 (2012).

[30] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. “Benchmarking deep reinforcement learning for continuous control”. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[31] Eppner, C., Höfer, S., Jonschkowski, R., Martín-Martín, R., Sieverling, A., Wall, V., and Brock, O. “Lessons from the amazon picking challenge: Four aspects of building robotic systems.” In: *Robotics: science and systems*. 2016.

[32] Fang, K., Bai, Y., Hinterstoisser, S., Savarese, S., and Kalakrishnan, M. “Multi-task domain adaptation for deep learning of instance grasping from simulation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 3516–3523.

[33] Fujimoto, S., Van Hoof, H., and Meger, D. “Addressing function approximation error in actor-critic methods”. In: *arXiv preprint arXiv: 1802.09477* (2018).

[34] Garg, N. P., Hsu, D., and Lee, W. S. “Learning To Grasp Under Uncertainty Using POMDPs”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2751–2757.

[35] Ghallab, M., Nau, D., and Traverso, P. *Automated Planning: theory and practice*. Elsevier, 2004.

[36] Gupta, M., Rühr, T., Beetz, M., and Sukhatme, G. S. “Interactive environment exploration in clutter”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 5265–5272.

[37] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *arXiv preprint arXiv:1801.01290* (2018).

[38] Hausknecht, M. and Stone, P. “Deep recurrent q-learning for partially observable mdps”. In: *2015 AAAI Fall Symposium Series*. 2015.

[39] Haustein, J. A., King, J., Srinivasa, S. S., and Asfour, T. “Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states”. In: *ICRA*. IEEE. 2015.

[40] He, K., Gkioxari, G., Dollár, P., and Girshick, R. “Mask r-cnn”. In: *ECCV*. 2017.

[41] Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. “Memory-based control with recurrent neural networks”. In: *arXiv preprint arXiv:1512.04455* (2015).

[42] Hernandez, C., Bharatheesha, M., Ko, W., Gaiser, H., Tan, J., Deurzen, K. van, Vries, M. de, Van Mil, B., Egmond, J. van, Burger, R., et al. "Team Delft's robot winner of the Amazon Picking Challenge 2016". In: *Robot World Cup*. Springer. 2016.

[43] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., et al. "Learning from Demonstrations for Real World Reinforcement Learning". In: *arXiv preprint arXiv:1704.03732* (2017).

[44] Hodson, R. "How robots are grasping the art of gripping". In: *Nature* 557.7706 (2018), S23–S23.

[45] Hogan, F. R. and Rodriguez, A. "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics". In: *arXiv preprint arXiv:1611.08268* (2016).

[46] Igl, M., Zintgraf, L., Le, T. A., Wood, F., and Whiteson, S. "Deep variational reinforcement learning for POMDPs". In: *arXiv preprint arXiv: 1806.02426* (2018).

[47] Jaakkola, T., Singh, S. P., and Jordan, M. I. "Reinforcement learning algorithm for partially observable Markov decision problems". In: *Advances in neural information processing systems*. 1995, pp. 345–352.

[48] James, S., Davison, A. J., and Johns, E. "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task". In: (2017).

[49] James, S. and Johns, E. "3d simulation for robot arm control with deep q-learning". In: *arXiv preprint arXiv:1609.03759* (2016).

[50] Jeong, R., Aytar, Y., Khosid, D., Zhou, Y., Kay, J., Lampe, T., Bousmalis, K., and Nori, F. "Self-Supervised Sim-to-Real Adaptation for Visual Robotic Manipulation". In: *arXiv preprint arXiv:1910.09470* (2019).

[51] Johns, E., Leutenegger, S., and Davison, A. J. "Deep learning a grasp function for grasping under gripper pose uncertainty". In: *IROS*. IEEE. 2016.

[52] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[53] Kaelbling, L. P. and Lozano-Pérez, T. "Integrated task and motion planning in belief space". In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227.

[54] Kahneman, D. *Thinking, fast and slow.* Macmillan, 2011.

[55] Karkus, P., Hsu, D., and Lee, W. S. "Qmdp-net: Deep learning for planning under partial observability". In: *Advances in Neural Information Processing Systems.* 2017, pp. 4694–4704.

[56] Kartal, B., Hernandez-Leal, P., and Taylor, M. E. "Action Guidance with MCTS for Deep Reinforcement Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.* Vol. 15. 1. 2019, pp. 153–159.

[57] Kartal, B., Hernandez-Leal, P., and Taylor, M. E. "Using Monte Carlo tree search as a demonstrator within asynchronous deep RL". In: *arXiv preprint arXiv:1812.00045* (2018).

[58] Kearns, M. and Singh, S. "Near-optimal reinforcement learning in polynomial time". In: *Machine learning* 49.2-3 (2002), pp. 209–232.

[59] Kimmel, A., Shome, R., and Bekris, K. "Anytime motion planning for prehensile manipulation in dense clutter". In: *Advanced Robotics* 33.22 (2019), pp. 1175–1193.

[60] King, J. E., Haustein, J. A., Srinivasa, S. S., and Asfour, T. "Nonprehensile whole arm rearrangement planning on physics manifolds". In: *ICRA.* IEEE. 2015.

[61] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[62] Kitaev, N., Mordatch, I., Patil, S., and Abbeel, P. "Physics-based trajectory optimization for grasping in cluttered environments". In: *2015 IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2015, pp. 3102–3109.

[63] Kloss, A., Schaal, S., and Bohg, J. "Combining learned and analytical models for predicting action effects". In: *arXiv preprint arXiv:1710.04102* (2017).

[64] Kocsis, L. and Szepesvári, C. "Bandit based monte-carlo planning". In: *European conference on machine learning.* Springer. 2006.

[65] Konietschke, R. and Hirzinger, G. “Inverse kinematics with closed form solutions for highly redundant robotic systems”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 2945–2950.

[66] Koval, M. C., King, J. E., Pollard, N. S., and Srinivasa, S. S. “Robust trajectory selection for rearrangement planning as a multi-armed bandit problem”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015.

[67] Laskey, M., Lee, J., Chuck, C., Gealy, D., Hsieh, W., Pokorny, F. T., Dragan, A. D., and Goldberg, K. “Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations”. In: *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE. 2016.

[68] Lee, J., Cho, Y., Nam, C., Park, J., and Kim, C. “Efficient obstacle rearrangement for object manipulation tasks in cluttered environments”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 183–189.

[69] Leidner, D., Bartels, G., Bejjani, W., Albu-Schäffer, A., and Beetz, M. “Cognition-enabled robotic wiping: Representation, planning, execution, and interpretation”. In: *Robotics and Autonomous Systems* (2018).

[70] Leidner, D., Bejjani, W., Albu-Schäffer, A., and Beetz, M. “Robotic agents representing, reasoning, and executing wiping tasks for daily household chores”. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2016.

[71] Leidner, D., Borst, C., and Hirzinger, G. “Things are made for what they are: Solving manipulation tasks by using functional object classes”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 429–435.

[72] Leonetti, M., Iocchi, L., and Stone, P. “A synthesis of automated planning and reinforcement learning for efficient, robust decision-making”. In: *Artificial Intelligence* 241 (2016).

[73] Li, J. K., Hsu, D., and Lee, W. S. “Act to see and see to act: POMDP planning for objects search in clutter”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 5701–5707.

[74] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. “Microsoft coco: Common objects in context”. In: *ECCV*. Springer. 2014.

[75] Liu, R., Lehman, J., Molino, P., Such, F. P., Frank, E., Sergeev, A., and Yosinski, J. “An intriguing failing of convolutional neural networks and the coordconv solution”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9605–9616.

[76] Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., and Mordatch, I. “Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control”. In: *arXiv preprint arXiv:1811.01848* (2018).

[77] Lynch, K. M. and Mason, M. T. “Stable pushing: Mechanics, controllability, and planning”. In: *The International Journal of Robotics Research* 15.6 (1996), pp. 533–556.

[78] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. *PDDL-the planning domain definition language*. 1998.

[79] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. 2016.

[80] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[81] Mohammed, M. Q., Chung, K. L., and Chyi, C. S. “Review of Deep Reinforcement Learning-based Object Grasping: Techniques, Open Challenges and Recommendations”. In: *IEEE Access* (2020).

[82] Morrison, D., Tow, A. W., Mctaggart, M., Smith, R, Kelly-Boxall, N., Wade-Mccue, S., Erskine, J., Grinover, R., Gurman, A., Hunn, T, et al. “Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7757–7764.

[83] Muhayyuddin, Moll, M., Kavraki, L., and Rosell, J. “Randomized Physics-based Motion Planning for Grasping in Cluttered and Uncertain Environments”. In: *ArXiv e-prints* (Nov. 2017).

[84] Murphy, K. P. “A survey of POMDP solution techniques”. In: *environment* 2 (2000), p. X3.

[85] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. *Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey.* arXiv preprint arXiv:2003.04960. 2020.

[86] Novkovic, T., Pautrat, R., Furrer, F., Breyer, M., Siegwart, R., and Nieto, J. "Object finding in cluttered scenes using interactive perception". In: *2020 IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2020, pp. 8338–8344.

[87] *OpenAI Baselines:ACKTR and A2C.* `https://openai.com/blog/baselines-acktr-a2c/`. Accessed: 2020-10-26.

[88] Pajarinen, J. and Kyrki, V. "Robotic manipulation of multiple objects as a POMDP". In: *Artificial Intelligence* 247 (2017), pp. 213–228.

[89] Papallas, R., Cohn, A. G., and Dogar, M. R. "Online Replanning with Human-in-The-Loop for Non-Prehensile Manipulation in Clutter—A Trajectory Optimization based Approach". In: *IEEE Robotics and Automation Letters* (2020).

[90] Papallas, R. and Dogar, M. R. "Non-prehensile manipulation in clutter with human-in-the-loop". In: *2020 IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2020, pp. 6723–6729.

[91] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. "Sim-to-real transfer of robotic control with dynamics randomization". In: *arXiv preprint arXiv:1710.06537* (2017).

[92] Pinto, L., Mandalika, A., Hou, B., and Srinivasa, S. "Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions". In: *arXiv preprint arXiv:1810.10654* (2018).

[93] Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. "Maximum margin planning". In: *Proceedings of the 23rd international conference on Machine learning.* 2006, pp. 729–736.

[94] Riccio, F., Capobianco, R., and Nardi, D. "DOP: Deep Optimistic Planning with Approximate Value Function Evaluation". In: *arXiv preprint arXiv:1803.08501* (2018).

[95] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T. Van de, Mnih, V., Heess, N., and Springenberg, J. T. "Learning by Playing-Solving Sparse Reward Tasks from Scratch". In: *arXiv preprint arXiv:1802.10567* (2018).

[96] Rimon, E. and Burdick, J. *The Mechanics of Robot Grasping.* Cambridge University Press, 2019.

[97] Rönnau, A., Sutter, F, Heppner, G., Oberländer, J., and Dillmann, R. "Evaluation of physics engines for robotic simulations with a special focus on the dynamics of walking robots". In: *2013 16th International Conference on Advanced Robotics (ICAR)*. IEEE. 2013, pp. 1–7.

[98] *ROS wrapper for Alvar, an open source AR tag tracking library*. `http://wiki.ros.org/ar_track_alvar`. Accessed: 2020-11-02.

[99] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. "Trust region policy optimization". In: *International conference on machine learning*. 2015, pp. 1889–1897.

[100] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[101] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv: 1707.06347* (2017).

[102] Shome, R., Tang, W. N., Song, C., Mitash, C., Kourtev, H., Yu, J., Boularias, A., and Bekris, K. E. "Towards robust product packing with a minimalistic end-effector". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9007–9013.

[103] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017).

[104] Somani, A., Ye, N., Hsu, D., and Lee, W. S. "DESPOT: Online POMDP planning with regularization". In: *Advances in neural information processing systems*. 2013, pp. 1772–1780.

[105] Song, H., Haustein, J. A., Yuan, W., Hang, K., Wang, M. Y., Kragic, D., and Stork, J. A. "Multi-Object Rearrangement with Monte Carlo Tree Search: A Case Study on Planar Nonprehensile Sorting". In: *arXiv preprint arXiv:1912.07024* (2019).

[106] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[107] Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., and Heess, N. *dm_control: Software and Tasks for Continuous Control*. 2020. arXiv: `2006.12983 [cs.RO]`.

[108] Thananjeyan, B., Balakrishna, A., Rosolia, U., Li, F., McAllister, R., Gonzalez, J. E., Levine, S., Borrelli, F., and Goldberg, K. "Safety Augmented Value Estimation from Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks". In: *IEEE Robotics and Automation Letters* (2020).

[109] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IROS*. IEEE. 2017.

[110] Tong, X., Liu, W., and Li, B. "Enhancing Rolling Horizon Evolution with Policy and Value Networks". In: *2019 IEEE Conference on Games (CoG)*. IEEE. 2019, pp. 1–8.

[111] Viereck, U., Pas, A. t., Saenko, K., and Platt, R. "Learning a visuomotor controller for real world robotic grasping using simulated depth images". In: *arXiv preprint arXiv:1706.04652* (2017).

[112] Wagner, P. "A reinterpretation of the policy oscillation phenomenon in approximate policy iteration". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2573–2581.

[113] Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., and Savarese, S. "Densefusion: 6d object pose estimation by iterative dense fusion". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3343–3352.

[114] Watkins, C. J. and Dayan, P. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[115] Wong, L. L., Kaelbling, L. P., and Lozano-Pérez, T. "Manipulation-based active search for occluded objects". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 2814–2819.

[116] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation". In: *Advances in neural information processing systems*. 2017, pp. 5279–5288.

[117] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes". In: *arXiv preprint arXiv:1711.00199* (2017).

[118] Xiao, Y., Katt, S., Pas, A. ten, Chen, S., and Amato, C. “Online planning for target object search in clutter under partial observability”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8241–8247.

[119] Yu, K.-T., Bauza, M., Fazeli, N., and Rodriguez, A. “More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing”. In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 30–37.

[120] Yuan, W., Hang, K., Kragic, D., Wang, M. Y., and Stork, J. A. “End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer”. In: *Robotics and Autonomous Systems* (2019).

[121] Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. “Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1803.09956* (2018).

[122] Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., et al. “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.