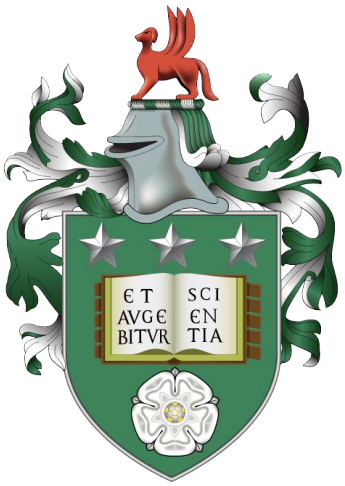
Human-In-The-Loop Planning and Control for Non-Prehensile Manipulation in Clutter

Rafael Papallas

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



The University of Leeds
School of Computing
July 2021

Dedicated to the memory of my father. Also dedicated to my mother and grandfather; you always believed in me.

Declaration

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Some of the results and work presented in this thesis have been published in the following papers:

> Papallas, Rafael, and Mehmet R. Dogar. "Non-prehensile manipulation in clutter with human-in-the-loop." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.
>
> Papallas, Rafael, Anthony G. Cohn, and Mehmet R. Dogar. "Online Replanning with Human-in-The-Loop for Non-Prehensile Manipulation in Clutter—A Trajectory Optimization based Approach." IEEE Robotics and Automation Letters 5.4 (2020): 5377-5384.

The above publications are primarily the work of the candidate.

Acknowledgements

I would like to thank my supervisor, Dr. Mehmet Dogar, for his exceptional support, guidance, and friendship. I am grateful to have worked together; I learnt a lot, and you helped me develop as a researcher. Furthermore, I would also like to thank Professor Anthony G. Cohn for his valuable feedback throughout my PhD. A big thank you to the Head of School, Professor Andy Bulpitt, for his support during the national lockdown.

The School of Computing was a great place to be. I had the chance to meet and work with great people and participate in teaching activities with great advisors including Dr. Nick Efford, Dr. Brandon Bennett, Dr. Samuel Wilson and Dr. Mehmet Dogar. I am also grateful for being a recipient of a UKRI/EPSRC Doctoral Training Partnership (DTP) scholarship; without this scholarship, it would have been really difficult to obtain this PhD.

A big thank you to all the people in the robotics lab for the great discussions, coffee breaks and the fun we had together all these years; Wissam Bejjani, Wisdom Agboh, Mohamed Hasan, Logan Dunbar, Alexia Toumpa, Luis Figueredo, Francesco Foglino, Ricardo Luna Gutierrez and Simon Obute. You made this journey easier and more enjoyable!

Thanks to my family for their constant support; Kyproula, Andreas, Michaela, Harrys and Sotiria. This would not have been possible without your support. Ioanna, thank you for your support and patient while we were going through this together.

Finally, I would like to truly thank my examiners Dr. Mohan Sridharan and Dr. Jordan Boyle for their time and valuable feedback which significantly improved my thesis.

Abstract

This thesis presents motion planning and control algorithms to tackle the problem of Reaching Through Clutter (RTC). I consider problems where a robot needs to reach in cluttered environments, like a fridge or a shelf, to retrieve a goal object. The robot considers non-prehensile manipulation and is, therefore, required to interact with objects and push them out of the way to create space. Reaching Through Clutter is in the NP-Hard class [1] and is, therefore, a computationally challenging problem. This is due to several problems; the state space is of high dimensionality, the system is under-actuated and such manipulation requires physics-reasoning through a physics simulator, which is computationally expensive to run. All of these are motion planning challenges, but beyond motion planning, physics-uncertainty poses challenges when executing a valid solution in the real-world. That is, a valid trajectory in simulation could be invalid during execution due to physics prediction errors.

I focus on all of these challenges and propose algorithms with a human-in-the-loop. The systems I propose in this thesis focus on minimal human input that results in significantly higher success rates and faster planning times. Initially, I employed kinodynamic sampling-based planners, like kinodynamic RRT and KPIECE, and propose a framework with a human-in-the-loop. The results suggest that human input is effective and the framework outperforms the baselines both in success rate and planning times. However, the solutions suffer from lengthy and noisy trajectories, which causes trajectories to fail in the real-world due to the physics uncertainty problem. To address these shortcomings, I proposed a stochastic trajectory optimization-based planner with online-replanning that significantly improves the quality of the trajectories, the success rate in the real-world, and the planning times. Finally, throughout this thesis, I argue that human time is valuable, and propose approaches that allow a single human operator to guide up to twenty robots simultaneously using a predictive approach. This approach predicts future optimization costs and employs human help earlier for robots that deal with hard instances of the problem.

Abbreviations

DOF Degrees of Freedom. 75

GRTC Guided Reaching Through Clutter. 43

HITL Human-In-The-Loop. 32

KPIECE Kinodynamic Planning by Interior-Exterior Cell Exploration. 16, 39, 49

MPC Model Predictive Control. 25

NAMO Manipulation/Navigation Among Movable Obstacles. 23, 39, 46, 47

OL Open-Loop. 78, 79, 80

OR Online Replanning. 25

PGF Predictive Guided Framework. xii, 86, 94, 96, 97, 98, 101

PRM Probabilistic Roadmap. 13, 14

RRT Rapidly-exploring Random Tree. 14, 15, 39, 49

RTC Reaching Through Clutter. 4, 29, 35, 41

STOMP Stochastic Trajectory Optimisation for Motion Planning. 19

TAMP Task and Motion Planning. 22

Contents

1 Introduction **1**

1.1 Thesis Scope and Terminology 3
1.2 Main Themes and Challenges 4
1.3 Aim 6
1.4 Contributions 7
1.5 Structure 7
1.6 Publication Note 7

2 Background & Literature Review **9**

2.1 Motion Planning and Control 9
2.1.1 Concepts 10
2.1.2 Sampling-based planning 13
2.1.3 Trajectory optimisation-based planning 17
2.1.4 Learning-based motion planning 20
2.1.5 Task and motion planning 22
2.1.6 Model predictive control 24
2.2 Manipulation Planning 26
2.2.1 Prehensile manipulation 27
2.2.2 Non-prehensile manipulation 29
2.3 Shared-Autonomy 32
2.4 Remarks 36

3 Integrating Human Input in Sampling-based Kinodynamic Planning **37**

3.1 Introduction 37
3.2 Assumptions, Objectives and Contributions 40
3.3 Problem Formulation 41
3.4 Guided-RTC Framework 42

3.4.1 A generic approach for Guided-RTC planning 43
3.4.2 Guided-RTC with Human-In-The-Loop (GRTC-HITL) 45
3.4.3 Guided-RTC with NAMO 46
3.4.4 Guided-RTC with straight line heuristic 47
3.5 Experiments & Results 49
3.5.1 Hypotheses 50
3.5.2 Experimental setup 51
3.5.3 Simulation results 51
3.5.4 Parallel guidance 53
3.5.5 Real-robot results 55
3.6 Conclusions 56

4 Integrating Human Input in Trajectory Optimisation-based Planning with Online-Replanning 59
4.1 Introduction 59
4.2 Assumptions, Objectives and Contributions 63
4.3 Problem Formulation 64
4.4 Online Replanning with Human-In-The-Loop 65
4.4.1 Framework overview 65
4.4.2 Stochastic optimisation 67
4.4.3 User input 68
4.4.4 Cost function 69
4.4.5 Initial trajectories 70
4.4.6 Asking for human help with a fixed timeout 71
4.4.7 Adaptively asking for human-help 72
4.5 Experiments & Results 72
4.5.1 Hypotheses 73
4.5.2 Experimental setup 74
4.5.3 Framework evaluation 75
4.5.4 Handling uncertainty 77
4.5.5 Warehouse problem 80

4.6 Conclusions 81

5 Learning When to Ask for Human Help 83
5.1 Introduction 83
5.2 Assumptions, Objectives and Contributions 86
5.3 Problem Formulation 87
5.4 Predictive Guided Framework (PGF) 87
5.4.1 Predictive solver 88
5.4.2 Allocator 90
5.5 Predictions 91
5.5.1 The prediction algorithm 91
5.5.2 Structure of the datasets 92
5.5.3 Network architecture 93
5.6 Experiments & Results 93
5.6.1 Hypotheses 94
5.6.2 Experimental setup 94
5.6.3 Model performance 94
5.6.4 Coping with a fleet of demanding robots 96
5.7 Conclusions 98

6 Conclusions & Future Work 99
6.1 Conclusions 99
6.2 Future Work 102
6.2.1 Computer vision and partially-observable environment 102
6.2.2 Handling deformable objects and non-standard geometry objects 103
6.2.3 Handling more complex non-prehensile interactions 103
6.2.4 Moving from a 2-dimensional space to a 3-dimensional 105
6.2.5 Different types of human input 105
6.2.6 Learning from human input 106
6.3 Final Remarks 106

References 107

List of Figures

1.1 Robot assisting a motor-impaired person with daily tasks. 1
1.2 Human assisting the robot through a graphical user interface. Points an object and a region to be pushed (red arrow and region). 2
1.3 An example of a Reaching Through Clutter problem. Robot initially fully autonomously tries to reach the green can but fails to find a solution. It then queries a human, the human suggests a high level input (arrow), robot continues and succeeds. 5

2.1 Two popular sampling-based planners. Parts adapted from [2]. 14
2.2 Trajectory Optimisation: solid blue line initial trajectory, dotted green best candidate trajectories of their iteration, solid green final optimised trajectory. Triangles are the controls applied at each step of the trajectory. 18
2.3 (a) prehensile manipulation; the object to be manipulated (wine glass) is firmly grasped by the hand (b) non-prehensile manipulation; wine glass is placed on a tray. 27

3.1 A human-operator guiding a robot to reach for the green goal object, o_g. Arrows indicate human interaction with the robot. In (a) the operator indicates o_2 to be pushed to the blue target region. From (a) to (c) the robot plans to perform this push. In (d) the operator indicates to the robot to reach for the goal object. From (d) to (f) the robot plans to reach the goal object. 38
3.2 Human-operator guiding a robot in the real-world. 38

3.3 Approaching states: The blue circle is the target region, the red rectangle the object to manipulate. We compute two approaching states, x_{a1} and x_{a2}. The two approaching states encourage side-ways and forward pushing actions respectively, and they increase the chance of a successful pushing. 44
3.4 GRTC-Heuristic: (a) Initial state. (b) The robot moves on a straight line to the goal object, o_g, to obtain the first blocking obstacle (o_7) and the swept volume (yellow area). (c) The heuristic produces a high-level action for o_7 indicated by the arrow and the target region (blue). This process is repeated until V_{swept} contains no blocking obstacle. 48
3.5 Initial states of different problems in simulation (S1-S4) and real world (R1-R4). Goal object is in green. 49
3.6 Simulation results, for each scene (S1-S10): (Top) Success rate. (Bottom) Mean planning time. The error bars indicate the 95% CI. For GRTC-HITL and GRTC-Heuristic, the dark shade indicates the planning time where the light shade indicates the time it took to produce the high-level actions (for GRTC-HITL this is a fraction of the time). The results suggest that the proposed approach (GRTC-HITL) performs better than the baselines both in success rate and planning times, and that in some scenes human input is especially effective (e.g., S2, S3, S5, S9). 52
3.7 Parallel Guidance: Robots guided in parallel in group of fours. The first robot is planning to reach for the goal object, the second one executes a solution, the third robot successfully reached the goal object, the fourth robot is waiting for human input (operator's main focus). . 54
3.8 A guided planning demonstration in the real world. The robot tries to reach and grasp the green object. The human input is indicated by the black arrow which points an object (orange) and a location to be pushed (blue region). The robot executes the high-level action provided by the human and then reaches successfully for the goal object. 56

4.1 Proposed System. Each row shows a different robot working in parallel. Human input is requested only when needed (*blue* colour). Human high-level input is shown with a white arrow. Planning is shown with *green* and execution with *red* colour. 60
4.2 Robot tries to reach for the goal object (green). Arrow indicates human input (2nd picture). The robot executes the human provided high-level action successfully (3rd picture) and successfully reaches the goal object (4th picture). 62
4.3 Initial trajectories. The arrow illustrates the trajectory. In (b) the object to be pushed is the box the arrow penetrates. 71
4.4 Heuristic: automatic approach to obtain high-level action. The robot first moves straight to the goal to find the first blocking obstacle. It captures the swept volume of the robot (yellow area). It then finds a collision-free position for the blocking obstacle outside the swept volume. White arrow is the suggested high-level plan obtained. 75
4.5 Real-world scenes 79

5.1 Two different instances of an RTC problem. The problem on the left is challenging, and the robot is very likely to ask for help, while the problem on the right is less challenging and the robot might be able to solve it fully autonomously. 84

5.2 The proposed framework. There is a human operator available to guide n robots. Each of the robots, using trajectory optimisation, tries to optimise and find a feasible trajectory fully autonomously. At each iteration, each robot predicts a human gain (c_{gain}) for some l iterations in the future. This human gain indicates how impactful human input will be in the current scene. High gain means that the robot will be able to solve the problem only if human intervenes, while low gain means that the robot might benefit from human input but might also be able to solve the problem autonomously. The "Allocator", observes the predicted human gains from all the robots and assigns the human to the robot with the highest expected gain. 85

5.3 $\phi_{autonomous}$ (left) and ϕ_{human} (right). Blue line is the loss function (Root Mean Squared Error) on the training set and orange line is the loss function on the validation set. 95

5.4 Example scenes from the experiments in a bidding window. There are twenty potential robots. Six of them join the bidding window (a-f) at the same time. c_{gain} is their predicted gain. 96

5.5 Planning times (left) and success rate (right) for Predictive Guided Framework (PGF) (blue) and OR-HITL (orange) as the number of robots under the supervision of a single human-operator increases. Error shadow indicates the 95% Confidence Interval. The results suggest that a single human operator can guide efficiently up to 20 robots using the proposed approach (PGF). As the number of robots under the human supervision increases from 20 to 30, the performance starts to become equal to the baseline (OR-HITL). 97

6.1 Which are some of the remaining challenges with respect to this task? 99

6.2 Grasping the goal book requires different non-prehensile actions. A similar example and figure was first introduced in [3] which inspired the illustration of this figure. 104

List of Tables

3.1 Guidance comparison between GRTC-HITL (human guidance) and GRTC-Heuristic (automatic guidance). The results show that the proposed actions of human are in general successful and hence the guidance time (time spent guiding) is low compared to the automatic approach where it spends more time proposing high-level actions unsuccessfully. 53

3.2 Guidance performance when a robot is guided individually (no distractions) compared to when it is guided in a group of other robots (human distracted guiding other robots simultaneously). The results show that planner waits for some time and that overall planning time is slightly higher. 54

3.3 Results when solutions are executed in the real world. GRTC-HITL is in general more successful than the baselines and baselines suffer from planning and execution failures. 55

4.1 Framework evaluation. Comparing the performance of the proposed approach against two fully autonomous baselines ("Autonomous" and "Adaptive Heuristic") and two variants of the proposed approach ("Fixed 5" and "Fixed 20"). The results present performance of an experience user (the author of this thesis) and of a novice user (no prior experience). The results show that the proposed approach is, in general, better both in success rate and total time (planning, guidance and execution time). Error indicates the 95% CI. 76

4.2 Simulation results with physics uncertainty. Experiments conducted with artificial noise added to the position of objects to simulate uncertainty and test the robustness of the online-replanning strategy proposed. "Adaptive" is the proposed approach with a human-in-the-loop and online-replanning, "Autonomous" is a baseline with online-replanning but without human input, "OL Autonomous" is a baseline *without* online-replanning and without human input. The results suggest that the online-replanning strategy is robust to physics-uncertainty. Errors indicate 95% CI. 78

4.3 Real-world results: success rate of proposed approach (Adaptive) compared to open-loop baseline (OL Adaptive). Adaptive uses online-replanning while OL Adaptive does not use online-replanning. "Optimisation Failures" indicate the planning failures, while "Execution Failures" indicate failures during execution. The results suggest that Adaptive is robust to real-world physics uncertainty, with one execution failure compared to six execution failures of the open loop planner. . 80

4.4 Warehouse problem: Results for the warehouse problem. Six robots operating in parallel in a virtual warehouse picking objects with a single human operator. The proposed approach (Adaptive) performs better than the baseline (Autonomous) both in success rate and total time. Errors indicate 95% CI. 81

Chapter 1

Introduction

> Robots succeeded in environments the average human has never been, yet fail miserably in environments every human has been, our houses.

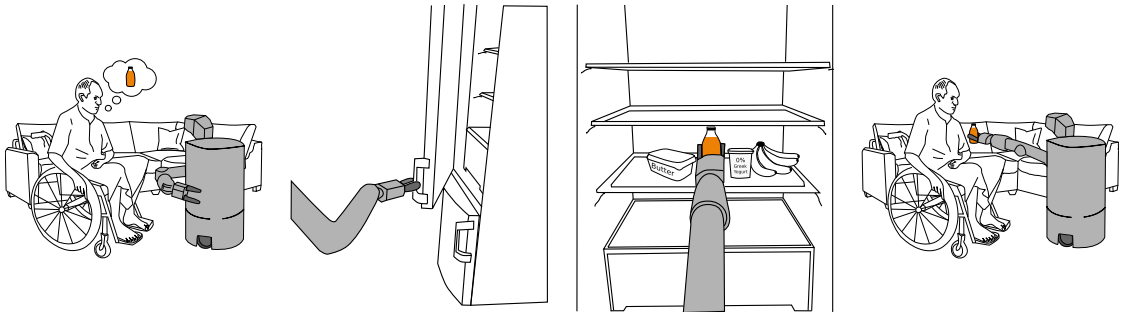


Figure 1.1: Robot assisting a motor-impaired person with daily tasks.

Consider Figure 1.1 where a robot is assisting a motor-impaired person with daily tasks. The robot needs to retrieve the orange juice bottle from the fridge. The robot autonomously navigates to the kitchen, opens the fridge, identifies the desired object (which is at the back of the fridge) and other obstacles blocking it. It finds a plan to manipulate the environment and safely grasps and retrieves the object. It closes the fridge and returns to the living room.

If in the above description the actor was a human, one will say that the task is trivial; humans solve such tasks every day, sometimes "without" even thinking. However, this is not the case for robots. We do not have fully autonomous robots, today, that can accomplish such complex manipulation actions (like retrieving a bottle from a cluttered fridge) in dynamic, cluttered and unstructured environments like humans do [4, 5]. This is because robotic manipulation is a *hard* problem due to numerous challenges that need to be addressed at the same time; perception, motion planning and control, modelling and uncertainty, and in some cases even

hardware/mechanical limitations [5, 6]. For realistic instances of these problems, robots' success rate is low [7]. Additionally, even when the robot finds a solution, executing it successfully in the real-world poses challenges due to the problem of physics-uncertainty [8, 9].

Now picture the same robot assisting the same person with the same task. The robot autonomously navigates to the kitchen, opens the fridge, identifies the desired object and other obstacles. While the robot is planning for a solution to reach and grasp the orange juice, it fails and queries the human for help. The human, through a user interface (like the one in Figure 1.2), remotely inspects the scene and provides an effective high-level input to the robot within seconds. The robot leverages the input, finds and executes a plan successfully and returns to the living room with the orange juice bottle.

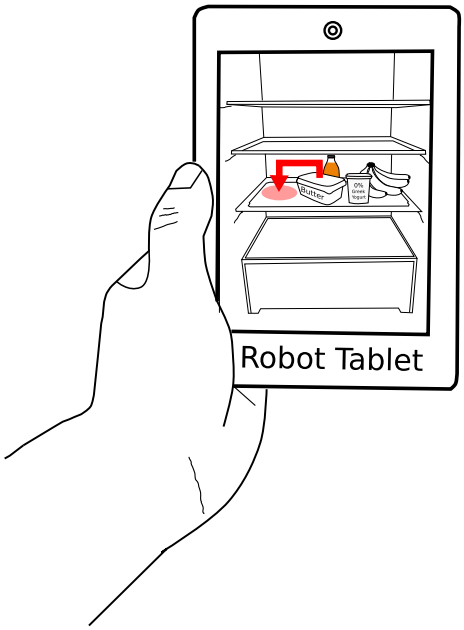


Figure 1.2: Human assisting the robot through a graphical user interface. Points an object and a region to be pushed (red arrow and region).

Such *semi-autonomous robots* can overcome many challenges that autonomous robots experience, with minimal input [10, 11]. This could permit deploying robots to the real-world faster [12]. I am interested in such semi-autonomous robots, and my thesis focuses on the *motion planning and control* aspect of the problem.

The research community in robotic manipulation has mainly focused on pick-and-place manipulation, or more formally prehensile manipulation [13, 14]. They developed effective algorithms and systems to solve pick-and-place problems. These systems, however, require structured environments and objects to be directly accessible by a collision-free robot configuration. In other words, robots avoid any interaction with surrounding obstacles. In a way, they treat the manipulation problem as a geometric one where the robot needs to move from an initial configuration to a

goal-configuration without any collisions. This can be effective in certain cases but very limited or inappropriate in other cases.

In problems like the one I described earlier, it rarely happens that there is a collision-free trajectory to grasp a desired object; interaction with other objects is, therefore, inevitable [15]. Humans manipulate objects in very cluttered environments with great dexterity. In many cases, they move objects as a preparation to grasping [16]. Effective manipulation, therefore, especially in cluttered environments, can be achieved through contact-based manipulation or more formally, *non-prehensile manipulation* [17, 18]. In a world "governed" by physics, robots can use this kind of manipulation to interact with objects beyond just grasping; through *physics-based manipulation* [7, 19, 20]. In the earlier scenario, for example, the robot might choose to push the butter to create space before reaching for the orange juice bottle. In this thesis I am particularly interested in such manipulation actions.

1.1 Thesis Scope and Terminology

The example I introduced in Figure 1.1 requires a number of challenges to be solved, but some of these challenges are beyond the scope of this thesis:

1. **Navigation:** Navigation is a research area by itself, and in this thesis I assume that the robot does not need to navigate in space.

2. **Perception:** A great challenge of this problem is perception. In this thesis, I do not address this challenge and instead I use a Motion Capture System with markers to capture the state of the objects to be manipulated.

3. **Partial-observability/incomplete knowledge:** Another challenge, related to the perception problem, is partial observability and the fact that objects are occluded. In this thesis, I assume a fully observable environment and that the robot has complete knowledge of the task.

4. **Grasping:** Grasping objects is also an area of research by itself, my contributions are not in robotic grasping and I assume that objects are easily graspable

when the robot closes its fingers.

5. **Natural Language Processing:** Finally, since there is interaction with a novice human, I do not consider the problem of language understanding, but instead I use straight-forward user interfaces where the human communicates their input directly.

My PhD focuses on the problem of motion planning and control for manipulation tasks. In the example in Figure 1.1, my research focuses with the motion planning and control problem of reaching and grasping the orange juice (i.e., the robot working in the fridge).

I use the terms "fully autonomous" and "semi-autonomous" (or "shared autonomy"). These terms mean different things to different people. The use of "fully autonomous" robot or planner in this thesis means a robot that plans for a solution without human intervention at a motion planning level[1]. On the contrary, "semi-autonomous" robot or planner means a robot that plans for a solution while considering human input at some point during the motion planning process.

1.2 Main Themes and Challenges

So far, I introduced some main terms, (1) motion planning, (2) motion control, (3) physics-based non-prehensile manipulation and (4) shared-autonomy. These are the main themes of this thesis and I motivate them as a way to tackle the challenging problem of *Reaching Through Clutter (RTC)*. That is, the problem where a robot needs to reach and grasp a goal object in cluttered and dynamic environments, like a fridge or a shelf. Problems like the Reaching Through Clutter are in the NP-Hard class [1].

Challenges

Figure 1.3 depicts a real example of the tasks I am tackling in this thesis. The robot needs to reach and grasp the green object, while it is allowed to interact and

[1] Although it requires a human to provide a goal and run the robot.

Figure 1.3: An example of a Reaching Through Clutter problem. Robot initially fully autonomously tries to reach the green can but fails to find a solution. It then queries a human, the human suggests a high level input (arrow), robot continues and succeeds.

push other objects. In this specific example, the robot tries initially to plan fully autonomously, but fails during planning. It then queries a human for help, which helps the robot to find a solution. Some reasons why the reaching through clutter problem is challenging for robots today:

1. **High-dimensional space:** First, the number of objects makes the state space of high-dimensionality because the planner needs to reason about the robot state and all the movable objects.

2. **Under-actuated system:** This is an under-actuated problem, since the objects cannot be controlled by the robot directly, but only indirectly through contact.

3. **Physics-reasoning is computationally expensive:** Predicting the evolution of the system state requires running computationally expensive physics simulators, to predict how objects would move as a result of a robot pushing.

4. **Physics-uncertainty:** Even with the use of the best physics simulator, the world cannot be modelled perfectly. As a result, the trajectory is only an approximate solution to what the dynamics will unfold in the real-world, and executing it blindly without tracking might yield to an unsuccessful state or even damage the robot or the environment.

5. **Human-input integration:** Integrating human-input in such setting is challenging. It is not clear what the input of the human should be, when the robot should ask for help and how the input will be integrated in a motion planning problem to improve it significantly. More importantly, the human-input should be captured quickly and within a reasonable amount of time that will not downgrade the performance of the system.

I focus on these challenges and propose algorithms to tackle them.

1.3 Aim

I aim to investigate how human-input can be integrated in motion planning to effectively solve the problem of reaching through clutter and tackle the challenges introduced in Section 1.2. I ask whether human-operators can be used to provide a minimal amount of input that results in a significantly higher success rate and faster planning times for non-prehensile physics-based manipulation in clutter. The approaches developed consider *minimal* interaction between a robot and a human and, therefore, I look into ways to enable a *single* human operator to guide *multiple* robots simultaneously.

Further study of the reaching through clutter problem is important to develop approaches to solve the problem more successfully and faster. It is a problem that has a potential for major near-term impact in warehouse robotics (reaching for objects on shelves) and personal home robots (reaching for object in a fridge). The algorithms that we currently have, however, cannot solve reaching through clutter problems in the real world in a fast and consistent way. The Amazon Picking Challenge, a competition that encouraged fully autonomous solutions for the reaching through clutter problem, demonstrated that even with *relaxed assumptions* the task is very challenging [21, 22]. To the best of my knowledge, semi-autonomous systems for non-prehensile physics-based manipulation has never been studied before.

1.4 Contributions

The contributions of this thesis are thus as follows:

1. The integration of human-input in sampling-based kinodynamic planning for non-prehensile manipulation (Chapter 3).

2. The integration of human-input in a trajectory optimisation-based planning for non-prehensile manipulation (Chapter 4).

3. An online-replanning strategy for motion control that integrates with the human input and the trajectory optimiser to tackle the problem of physics-uncertainty (Chapter 4).

4. A predictive system that allows a fleet of robots to learn when to ask for help, thereby allowing a single human operator to more effectively guide multiple robots (Chapter 5).

The findings of this thesis suggest that humans are capable to effectively and rapidly identify good high-level inputs (like which obstacles need to be manipulated and where) and as a result, they can be used to guide a fleet of robots simultaneously.

1.5 Structure

In this chapter, I motivated the work, introduced the main themes this PhD builds on and noted the contributions of the thesis. Next, in Chapter 2, I introduce the main ideas in more detail and review related-work. The technical work starts from Chapter 3 to Chapter 5. Finally, in Chapter 6, I conclude and suggest possible future work.

1.6 Publication Note

This thesis expands on the findings of published work, including published source code:

1. Chapter 3 content appears in:
   (a) **Published conference paper:** International Conference on Robotics and Automation (ICRA) 2020 [23].
   (b) **Published workshop paper:** IROS2019 [24].
   (c) **Source code:** `https://github.com/rpapallas/hitl-clutter` under the GPL-3.0 license.
   (d) **Demonstration video:** `https://youtube.com/watch?v=nfr1Fdketrc`.
   (e) **Website:** `https://pubs.rpapallas.com/icra2020`.
2. Chapter 4 content appears in:
   (a) **Published journal paper:** Robotics and Automation Letters (RA-L) 2020 [25] (and presented at IROS 2020).
   (b) **Published workshop paper:** ICRA2020 [26].
   (c) **Source code:** `https://github.com/rpapallas/hitl-trajopt` under the GPL-3.0 license.
   (d) **Demonstration video:** `https://youtu.be/t3yrx-J8IRw`.
   (e) **Website:** `https://pubs.rpapallas.com/ral2020`.
3. Chapter 5 content appears in:
   (a) **Source code:** `https://github.com/rpapallas/predictive-guided-framework` under the GPL-3.0 license.

Chapter 2

Background & Literature Review

This chapter serves two purposes. First, as a background research introducing core ideas and concepts this thesis builds on and second as a literature review.

The structure of this chapter is as follows. First, in Section 2.1, I introduce the area of "Motion Planning and Control". I introduce some important concepts and I discuss important and related work. In Section 2.2, I discuss important work in "Manipulation Planning". Finally, in Section 2.3, I discuss important work in the shared-autonomy literature.

2.1 Motion Planning and Control

The traditional goal of motion planning is to find collision-free motions for a robotic system, whether this is an autonomous vehicle or a robot manipulator [27]. The term "collision-free" motions, however, can be misleading in the context of this work, since I am looking at physics-based motion planning where the robot in fact *can* generate non-collision-free motions. Therefore, in the context of this work, the motion planning goal is to generate *safe motions* for the robot to reach its goal while respecting some hard constraints. In the example task in Figure 1.1, where the robot assists a motor-impaired user with retrieving the orange juice, the robot needs to come in contact with other movable obstacles in the fridge to push them out of the way to reach for the orange juice, but without violating hard constraints like damaging the movable obstacles, the environment or itself.

Another distinction is between path planning and motion planning. Although some people use the two interchangeably, here when I say *path planning* I refer to algorithms that generate a path *without* consideration of the controls that move the robot from one state to another. I use the term *motion planning* to refer to algorithms that generate a full trajectory (consisting of states and controls): the

robot can directly execute the solution.

Motion planning is considered a computationally hard problem and has seen success in various complex problems beyond robotics [27]. Next, in Section 2.1.1, I introduce important motion planning and control concepts. In Section 2.1.2, I introduce sampling-based planning, two important algorithms (RRT and PRMs), and discuss related work in this area. In Section 2.1.3, I introduce and discuss related work in the trajectory optimisation planning literature. In Section 2.1.4, I discuss learning-based motion planning approaches and in Section 2.1.5, I discuss task and motion planning approaches. Finally, in Section 2.1.6, I discuss Model Predictive Control approaches.

2.1.1 Concepts

Configuration and state space

An important concept in motion planning is the configuration space. It is usually denoted with C-space. C-space represents all the possible configurations of the system in the environment given its kinematic description. This abstract representation allows a complex configuration of the system to be mapped to a single point in C-space. The dimensionality of this space is the minimum number of parameters needed to specify a configuration, or in other words is equal to the number of degrees of freedom of the robot [27].

We also have the notion of a state and a state space. In this thesis, I define the state space as the space that captures all possible cases that the system can represent. This includes the configuration and velocity of the robot, and the pose and velocities of the movable objects. Given a state, we can describe the entire system at any point exactly. We therefore have an initial state denoted with x_0, and a goal state denoted with x_n.

Free space

Another important concept is the *C-space obstacle region*, denoted with C_{obs}. This is the set of configurations that the robot collides with obstacles. $C_{\text{free}} = C \setminus C_{\text{obs}}$ defines the set of configuration that are collision-free. Usually, the aim of many motion planners is to traverse a path in C_{free} from x_0 to x_n [27].

Control space

Beyond the configuration and the state space, we also have the notion of the control space. This is the space of all possible actions that the system can take to move from one state to another. Given a state, say x_0, and a control, say u_0, the result is a new state, say x_1, after applying the control for some duration from x_0. Motion planning algorithms try to find the right controls to move the robot from an initial state to a goal state.

Task space

Task space is usually defined by the position and orientation of the robot's end-effector and represents all possible poses of the robot's end-effector. The task space is also known as the operation space. The space dimensionality of a robot with one end-effector is traditionally $\mathcal{R}^3 \times SO(3)$, that is, $\mathcal{R}^3$ is the Cartesian coordinates for the position of the end-effector and $SO(3)$ is its orientation [28]. However, as the name implies, the task space depends on the task. In the case of this thesis where I am interested in reaching in clutter for objects, I assume that the robot's end-effector operates on the plane and hence the dimensionality of the task space in this thesis is $SE(2)$.

Kinematics

An important concept in motion planning is the robot kinematics. Kinematics describe the pose and all higher-order derivatives of the pose of the robot [29]. Forward kinematics is the problem of determining the pose of the end-effector given

the configuration of the robot (joint values). Inverse kinematics is the problem of determining the configuration of the robot (joint values) given an end-effector pose. Inverse kinematics could be useful in the context where planning happens in the task space (i.e., planning in the end-effector space) and the robot needs to map the solution from the task space to the configuration space.

Planners

A planner usually refers to an algorithm that generates a solution to the motion planning problem. The traditional input to a planner is a description of the configuration, state and control space, an initial state and a set of goal states as well as some constraints that the robot needs to respect. The planner simply needs to find a solution to move the system from the initial state to a goal state [30–33].

Geometric and physics-based motion planning

The motion planning problems can generally be divided into two categories: geometric [30, 31, 34] and physics-based [14, 20, 35, 36] motion planning. The former focuses on the problem of finding a collision-free motion to move a robot from an initial configuration to a goal configuration while considering the geometry of the robot and of the obstacles (finding a solution in C_{free}). It is the most widely studied approach to motion planning. The latter considers physical interaction of the robot with the environment and the objects. It usually requires a physics model to evaluate the robot's actions in the environment while planning. The attention of this thesis is on physics-based motion planning.

These are some important concepts of motion planning and control. Sampling-based planning is a major category of motion planning algorithms which are known to work well in high-dimensional spaces like the ones I am interested in this thesis. In the next section, I introduce sampling-based planning and discuss important and related-work.

2.1.2 Sampling-based planning

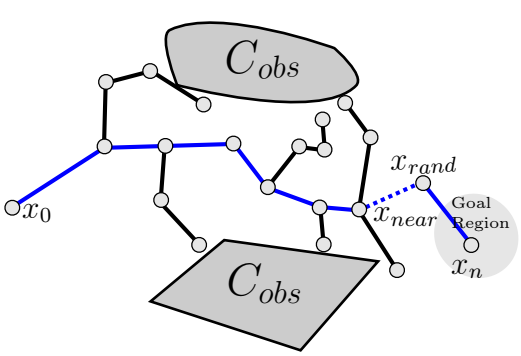
Sampling-based planners rose in popularity in the recent years due to their effectiveness in planning in high-dimensional spaces [37]. They have been applied successfully in a variety of complex systems, including robotics. The general idea is to exploit fast collision-detection and state validity algorithms to decide if a sampled configuration is valid. These are usually treated by the motion planning algorithm as "black box" functions. This simple idea allows these planners to not explicitly require definition of the C_{obs} during planning, since states passing through the C_{obs} will be discarded by the collision-detection algorithm [2]. They provide probabilistically complete guarantees: given enough time, and if a solution exists, they will eventually find it. This guarantee is weaker than complete algorithms, but this property allows them to effectively plan in high-dimensional spaces (greater than three degrees of freedom) unlike complete algorithms [27]. The solutions of sampling-based planners are usually global; they search for a solution in the whole space.

Sampling-based planners can be divided into two major categories, single-query and multi-query planners [27]. Single-query planners are conceptually easier and more popular and hence more widely studied. Therefore, here I will describe briefly about multi-query planners and expand further on single-query planners since I am also using single-query sampling-based planners in Chapter 3.

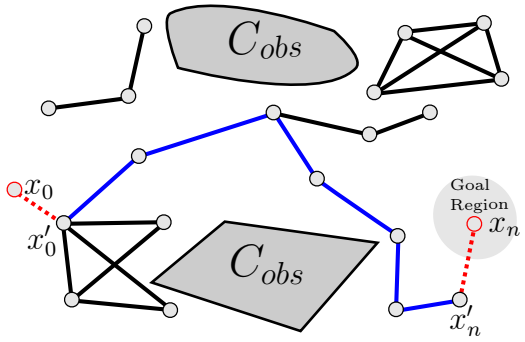
Multi-query sampling-based planners

Multi-query sampling-based planners build a graph (usually called a roadmap) of the C_{free} initially, and subsequent queries to the planner extend the graph to cover more parts of C_{free} [27]. We define an undirected graph $G(V, E)$ where V are the vertices of the graph representing collision-free configurations and E edges to collision-free paths connecting the vertices [27]. Multi-query sampling-based planners construct and maintain such a graph and then uses graph search algorithms like A* to find a solution.

A well known multi-query planner is PRM by Kavraki et al. [30]. It is illustrated in Figure 2.1(b). It consists of two phases. First, in the learning-phase, the planner



(a) Rapidly-exploring Random Tree (RRT): blue path indicates the solution, dotted line shows the expansion x_{rand} from neighbour x_{near}. x_n is a goal state within the goal region.



(b) Probabilistic Roadmap (PRM): black nodes and edges indicated initial roadmap, red nodes and edges the query tuple (initial state and goal state) connected to nearby vertices, blue is a solution path.

Figure 2.1: Two popular sampling-based planners. Parts adapted from [2].

constructs a graph-based roadmap by generating free configurations of the system and attempts to connect them using a local planner. The output of the first phase is an undirected graph with nodes representing configurations and edges paths connecting the configurations. In the second phase (the query phase), given an initial state, x_0, and a goal state x_n, the planner will try to connect these nodes with the nearest nodes in the graph x'_0 and x'_n respectively. If successful, the problem becomes a path finding one. Assuming that the environment (obstacles and robot) are kept fixed, multiple such queries can be made to the same roadmap, without the need of re-generating it from scratch.

Single-query sampling-based planners

Single-query sampling-based planners construct a tree on-the-fly for every query. We also have a graph $G(V, E)$ where vertices V represent collision-free configurations and edges E represent paths between those configurations. Unlike multi-query sampling-based planners, single-query sampling-based planners choose a vertex $x_{explore} \in V$ for expansion and attempts to sample a new collision-free configuration that connects

them. This way, the graph grows with new configurations and the algorithm checks if a solution path exists between the start and the goal configuration [27].

The most well-known single-query sampling-based algorithm is RRT by LaValle [31] illustrated in Figure 2.1(a). RRT builds a tree from the initial state, x_0, and then samples a random state and tries to extend the tree from the nearest neighbour to that state. It uses a distance metric to find the nearest neighbour, and the nodes in the tree represent configurations of the system. Usually, the planner, to bias the tree towards the goal, samples, in frequent intervals, a goal state and attempts to connect it to the tree. Once the planner samples a goal state and successfully connects it to the nearest node in the tree, the algorithm returns a path from the initial state to that goal state.

Both RRT and PRM have been used in the same context. For example, Denny et al. [38] propose a general framework for collaborative planning that allows a human to guide a sampling-based planner to solve path planning problems. The human provides regions to either attract or repel the planner. The framework works with RRT-based, PRM-based or combination of the two.

RRT has seen a great success in numerous tasks and several variants of the RRT algorithm were proposed as a result improving on several aspects of the original algorithm [32, 33, 39–44]. Kuffner and LaValle [32], for example, propose RRT-Connect which builds two trees (instead of the traditional one) from both ends (start and goal configuration). At each iteration, in an alternating fashion, the planner grows one tree and attempts to connect, using a greedy heuristic called "Connect", with the nearest node in the other tree (then the roles alternate; the second tree grows and tries to connect to the first one). RRTs, traditionally, consider the geometric problem.

A notable variant of the original RRT algorithm is Kinodynamic RRT [45] which is especially important to this thesis as it can be used for physics-based manipulation. Kinodynamic RRT considers kinematic, velocity, and acceleration constraints. It samples *controls* to bring the nearest neighbour near to the random state. It was employed successfully by several people to solve kinodynamic problems [9, 46–50],

for several tasks including manipulation. For example, Haustein et al. [46, 50] and King et al. [47], employ kinodynamic planning for rearrangement planning, while [9] use kinodynamic planning for reaching through clutter problems. Kinodynamic planning is relevant to my work and I review in more detail some of these works in the manipulation section of this chapter.

Despite the great success of RRT-based algorithms, the planner suffers from unguided search in a large space which could cause the planner to generate unnecessarily lengthy solutions. Several methods were proposed to tackle this challenge including a post-processing technique [51] (where the algorithm smooths out long RRT trajectories) and RRT variants with guided methods [52, 53]. Additionally, Şucan and Kavraki [54], propose a new sampling-based single-query kinodynamic planner called *Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE)*. KPIECE builds a tree from the state and control space until a goal is reached. KPIECE uses the notion of space coverage to guide its exploration in the state space by constructing and updating a discretization of the state space. KPIECE has been used in challenging motion planning problems. For example, Rusu et al. [55] use KPIECE along with a 3D scene perception system for reaching and grasping objects while avoiding collisions using replanning with a Dynamic Obstacle Map for collision avoidance. Muhayy ud din et al. [9] propose a variant of KPIECE called p-KPIECE to handle physics-uncertainty for grasping tasks.

The Open Motion Planning Library (OMPL) [56] is an open-source and free library that implements many state-of-the-art sampling-based motion planning algorithms including PRMs, RRT, KPIECE and other well-known motion planning algorithms. In Chapter 3, I use OMPL to build on Kinodynamic RRT and KPIECE for the problem of manipulation in clutter but considering human input.

This review of related-work highlights that sampling-based planning is effective in solving challenging problems like the ones I am looking in this thesis. What I propose instead is the use of sampling-based kinodynamic planning for physics-based manipulation with a human-in-the-loop. To the best of my knowledge, this intersection was not studied before and I believe that human input can be beneficial to this motion planning problem.

Going back to the probabilistically-complete guarantee of sampling-based planners, although this guarantee makes them effective in finding a global solution in high-dimensional spaces, if a solution does *not* exist, they will never be able to report it. They, therefore, rely on a time limit to decide when to stop. This time-out can be challenging to adjust from problem to problem and can slow down the system searching unnecessarily for a solution for an unsolvable problem. Trajectory optimisation-based planning searches for local solutions, and they can help to address this problem more elegantly, in most cases.

2.1.3 Trajectory optimisation-based planning

In the literature, "trajectory planning" usually refers to the problem of "translating" a motion planning solution to a valid trajectory (which respects the robot's mechanical limitations which might not be considered originally by the motion planner) [2]. Here, when I use the term trajectory optimisation planning, I refer to motion planning formulated as an optimisation problem, where a cost function is defined over the trajectory and the planning is framed as the optimisation of its cost.

More formally, we define a trajectory $\tau \in \mathcal{T}$ that maps time to C-space configurations and controls. The controls tells the robot how to move over time to reach certain C-space configurations. We also define a cost function (also known as the objective function), $\mathcal{C}$, that maps a trajectory to a positive real value, $\mathcal{C} : \mathcal{T} \rightarrow \mathbb{R}^+$. The cost function encodes factors that define the optimality of the trajectory including but not limited to the length or efficiency of the trajectory, obstacle avoidance, motion smoothness, and/or for reaching a specific goal etc. We are looking to find an optimal trajectory $\tau^* \in \mathcal{T}$:

$$
\begin{aligned}
\tau^* &= \arg\min_{\tau \in \mathcal{T}} \mathcal{C}[\tau] \\
\text{s.t.} \quad & \tau_{[0]} = x_0 \\
& \tau_{[n]} = x_{goal}
\end{aligned}
$$

Where $\tau_{[0]} = x_0$ is the initial state of the system at the first waypoint of the trajectory, and $\tau_{[n]} = x_{goal}$ is the goal state at the last waypoint of the trajectory.

As noted already, traditionally, motion planning focuses on the problem of collision-free motions. Certain motion planning problems might require constraints beyond just collision-free motions. For example, imagine a robot moving a glass of water from an initial configuration to a final configuration. The glass should stay upside-up throughout the trajectory execution otherwise the trajectory, even if mechanically feasible, will be considered invalid [57]. Trajectory optimisation lends itself better to such problems because it tries to minimise a cost with any number of such objectives, and hard constraints, like the one mentioned, can also be considered during optimisation.

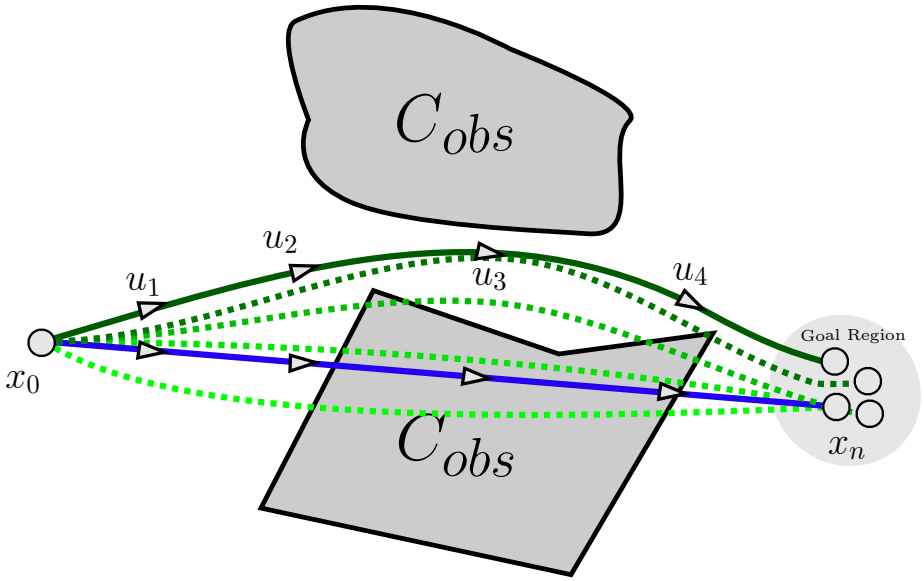


Figure 2.2: Trajectory Optimisation: solid blue line initial trajectory, dotted green best candidate trajectories of their iteration, solid green final optimised trajectory. Triangles are the controls applied at each step of the trajectory.

A simple trajectory optimisation planner is illustrated in Figure 2.2 concerned with collision-free trajectory to move from an initial state to a goal state. It starts with an initial trajectory (for example a straight-line trajectory) and using a cost function at each iteration it tries to optimise around it.

An algorithm similar to what I described above is proposed by Kalakrishnan et al. [57] who describe a stochastic trajectory optimisation approach called Stochastic Trajectory Optimisation for Motion Planning (STOMP). The algorithm starts with an initial trajectory which is assumed to be the best trajectory so far. It then generates noisy trajectories around the best trajectory to explore the surrounding space, and updates it to combine the best parts from every other noisy trajectory at a step-level. STOMP assumes that the start and goal states of the trajectory are given, and that they are kept fixed during the optimisation. STOMP uses derivative-free stochastic optimisation method which allows it to optimise arbitrary costs for which their derivatives might not be available, non-differentiable or non-smooth.

Ratliff et al. [58] introduce CHOMP (Covariant Hamiltonian Optimisation for Motion Planning). CHOMP uses functional gradient techniques to optimise an initial trajectory. CHOMP is designed to generate smooth and collision-free trajectories for robots with large number of degrees of freedom. It can generate trajectories that satisfy hard constraints and minimize soft costs. STOMP and CHOMP are trying to solve similar problems, however, STOMP requires minimal parameter tuning, and it does not require cost function gradients. Inspired by these approaches, I propose a similar approach in Chapter 4 but my work differs in that I focus on the problem of manipulation in clutter, and I consider human input in my optimisation formulation. This essentially allows a human operator to inject high-level hints into the optimisation.

Other notable optimisation-based motion planners include ITOMP [59] that considers online replanning in dynamic environments and a Trajopt [60] which is closely related to CHOMP. Ichnowski et al. [61] propose Grasp-Optimised Motion Planning (GOMP), a trajectory optimisation motion planning approach for bin-picking tasks. The approach is concerned with motion execution optimality to improve the Pick Per Hour (PPH) rate of the robot.

Trajectory optimisation was employed to solve various manipulation problems including manipulation of deformable objects [62], grasping [63], pushing [64, 65] and reaching for objects [7, 66]. Agboh and Dogar [64] formulate the problem of pushing an object as an MDP with action-dependent stochasticity and use a trajectory optimiser

to solve it. The proposed algorithm considers different tasks and adapt its actions based on the accuracy requirement of the given task; pushing the object slowly for tasks requiring high-precision and pushing faster for tasks that high-precision is not required. In [66], Agboh and Dogar use trajectory optimisation to solve the problem of reaching through clutter similar to what I implement in Chapter 4. However, I consider a human-in-the-loop and I consider static obstacles (boundaries) which makes the problem more challenging since objects can be jammed.

2.1.4 Learning-based motion planning

Another approach to motion planning is to leverage advances in Machine Learning. These approaches usually try to address shortcomings of the existing motion planning algorithms by learning an aspect of the problem to guide the planners [35, 67–75]. These learning approaches require a vast amount of training data during the *learning or training phase*, where the system tries to extract useful patterns from the data and generalize. In this thesis, I am not using learning-based methods for motion planning, but I employ machine learning in Chapter 5 to learn when to ask for human help. Here I review briefly important and related work because they rose in popularity in the last years.

Learning-based methods have been used to learn approximate physics models [76–79]. Agrawal et al. [76], for example, present an approach where a robot learns the effect of poking objects by executing hundred of thousands pokes on objects. The paper presents a novel approach based on DNN (Deep Neural Networks) that learns the "intuitive" physics model directly from image observations. Similarly, Zeng et al. [19], propose a model-free unsupervised deep reinforcement learning method that learns to combine both prehensile and non-prehensile actions to achieve a manipulation task. They argue that certain problems require first the robot to "declutter" the scene using non-prehensile manipulation to enable the robot fingers to pick up a desired object.

Laskey et al. [80] propose a learning-from-demonstration approach to solve RTC problems with a 2-DOF robot. The demonstrations are provided by a hierarchy of

supervisors. At the bottom of the hierarchy is a simplified motion planner, followed by crowd-sourced demonstrators from Amazon Mechanical Turk, and finally, at the top of the hierarchy expert human demonstrators. This hierarchy aims to expedite the learning process and reduce the burden from the expert human demonstrators.

The work of Bejjani et al. addresses problems similar to the ones I consider in this thesis. In [71], Bejjani et al. tries to solve the problem of pushing an object from an initial configuration to a goal configuration while considering minimum interactions with other objects. They try to learn a model to predict future cost-to-go from the horizon to the goal through a Receding Horizon Planner by exploiting previous experiences with a kinodynamic planner. They use Reinforcement Learning to improve the learned function for optimality. The learned model is limited to certain objects with predefined features, and the work considers a discrete action space. They address the former issue in [35] by using image-based state representation and the latter in [72] by learning manipulation actions in a continuous action space. In [73], Bejjani et al. address the problem of occlusion in partially-observable environments for the problem of reaching through clutter. They propose a data-driven approach that learns a distribution over the pose of the target object, which is hidden behind the clutter. The distribution is updated in a closed loop manner to permit the planner to generate occlusion-aware actions.

Learning-based planning has also been used to address the problem of non-prehensile rearrangement [81]. This approach trains policies end-to-end using deep Q-learning with CNN in simulation, and then uses a transfer approach to transfer the policy to the real-world using few real-world supervised examples. The work considers rearrangement of a single object each time.

More related to my work is the work by Hasan et al. [69] who propose an approach that learns high-level human-like actions for reaching through clutter by observing humans manipulating objects in a virtual environment (using a VR headset). This approach, however, is restricted to structured (objects aligned in predefined rows) environments without static boundaries.

Although learning-based approaches demonstrate good results for manipulation problems, they usually require a large amount of training data and, in the context

of non-prehensile manipulation, most approaches consider simplified versions of the problem. The approaches we discussed so far focused purely on motion planning generation. Another approach is to plan both in task and motion space. Next, I discuss this line of work known as Task and Motion Planning since is relevant to this thesis.

2.1.5 Task and motion planning

A final approach to motion planning is Task and Motion Planning (TAMP). TAMP combines two planning methods, task high-level planning (or symbolic planning) and motion planning [82–85]. Traditionally, task planning and motion planning were studied independently. Recent works, however, combine the two with the observation that such planners, where a planner first plans in the task space, could help guide the motion planning part to relevant spaces constrained by the task [86]. In some problems, TAMP makes sense. A robot that navigates to the kitchen to open the fridge and retrieve the orange juice bottle is basically tackling four problems at once. Each task requires a different motion planning approach (navigate to the kitchen, open the fridge, grasp orange juice bottle, return to living room). Akbari et al. [87] present a TAMP framework for physics-based manipulation for tasks where the robot needs to navigate in the environment while it is allowed to push movable obstacles. The framework consists of an ontology representing a knowledge base about the world, a task planner that uses physics-based reasoning, a cost function to identify a task plan with the least task-feasibility cost, and a sampling-based motion planner to plan motions.

In a high-dimensional task space, TAMP can be suboptimal since the task planner could propose a task plan that might be infeasible to plan for the motion planner dropping the performance of the system by interleaving task and motion planning unnecessarily. Wells et al. [70], use SVM (Support Vector Machine) in a Task-and-Motion-Planning setting to learn feasibility of actions to solve this problem. This allows a task planner to find actions that are more likely to be feasible, thereby reducing the total number of calls to a motion planner. In [88], the authors consider

a mobile manipulator, and propose an approach to minimize the number of obstacles to be relocated. The robot tries to find the right base position to reach for a desired object in clutter such that the number of obstacles that need to be relocated (at a task-level) is minimized.

TAMP can also be useful in the problem where the robot is trying to reach for a goal object in clutter alone. A TAMP approach could identify first which blocking obstacles to manipulate and where to place them before it plans to reach for the goal object. These sub-tasks (obstacles and where to place them) can then become motion planning queries. For example, Stilman et al. [89] formulates the problem of *Manipulation/Navigation Among Movable Obstacles (NAMO)* as a high-level search over the orderings of objects to be moved, combined with a low-level motion planner that pick objects up and move in that order. Lee et al. [90] and Nam et al. [91] propose similar polynomial algorithms for task and motion planning by aiming to minimize the number of object relocations. The algorithms find a sequence of obstacles to rearrange (using prehensile manipulation) until a collision-free path exists to grasp the desired object. I use a similar high-level plan structure, i.e., an ordering of objects, but I focus on *non-prehensile manipulation* of objects, rather than pick-and-place actions.

Finally, other works tackle the problem of partial observability. Zhang et al. [92], for example, present a mixed logical inference and probabilistic planning for target object localization tasks in indoor domains using primarily visual data. The authors use a declarative language to represent and perform the inference assuming incomplete domain knowledge. It is formulated as a Partially Observable Markov Decision Process (POMDP). The proposed architecture allows a robot with incomplete domain knowledge to infer and localise target objects that appear in different rooms. The architecture allows for incremental knowledge revision as the robot moves and senses the environment. Similarly, in [93], Zhang et al., propose a hierarchical decomposition of POMDP for vision-based sensing, information processing and collaboration for a group of robots. This hierarchical decomposition aims to deal with the problem of high dimensionality in complex and large state spaces of POMDPs. Such systems and formulations are relevant in the general problem of manipulation in clutter where

a robot might not have complete knowledge of the task or full observability of the environment. Although this is a challenging and open problem in manipulation planning, in this thesis my focus is mainly on the challenges of motion planning and control and, therefore, I assume a fully observable environment and that the robot has complete knowledge about the task.

So far, we discussed motion planning; sampling-based planning, trajectory-optimisation-based planning, learning-based motion planning, and task and motion planning. Next, we review related work in the *control* literature and specifically focussing on model predictive control.

2.1.6 Model predictive control

The problem of physics-based manipulation is challenging in many ways, but one particular challenge is the physics uncertainty when executing a trajectory in the real-world [94]. With physics uncertainty, in this thesis, I refer to the problem when a feasible trajectory in simulation can be infeasible when executed in the real-world. There are multiple reasons why uncertainty arises. Uncertainty can occur from model inaccuracies, control errors during execution, and from sensor errors [95]. This is an inevitable problem, and this thesis proposes ways to reduce uncertainty (for example in Chapter 3 where human input forces the robot to push obstacles away from the goal object and, therefore, the uncertainty around the goal object is reduced) or *plan with uncertainty* (for example in Chapter 4 where human input, planning, and execution are tied together in an online replanning framework that handles uncertainty by correcting the trajectory in real-time).

When it comes to execute a trajectory in the real-world, open-loop execution refers to execution with no feedback from the environment and assumes that execution of commands will unfold without unexpected consequences. It is not particularly effective against physics uncertainty, but it is faster than closed-loop controllers and effective in scenes where dynamic interactions are non-existent or at a minimum (for example pick-and-place tasks or pushing an object in free space).

Several works tackle the physics-uncertainty problem at a motion planning level

but the motion is executed in an open-loop manner [9, 96–99]. For example, Melchior and Simmons [96] propose Particle RRT that uses a stochastic process to simulate multiple times an extension node using different variations which are then grouped in similar clusters forming a single node in the tree. This way, the planner can find more robust paths for a given problem. Muhayy ud din et al. [9] propose p-KPIECE that extends KPIECE to handle physics uncertainty. It uses a custom motion sampler that samples multiple candidate motions and computes the belief about their robustness and returns the one with the highest belief to be robust, at every propagation step. Furthermore, they enhance the planner's exploration strategy to bias the tree towards robust regions based on the computed motion beliefs. Dogar and Srinivasa [97] propose an approach to grasping that allows a robot to plan a trajectory to grasp an object in clutter. It reduces the uncertainty about an object's pose and tackle physics-uncertainty by "decluttering" the target object by pushing it away from obstacles before it is grasped. In Chapter 3, I look at how the human input can achieve the same; the high-level input of the human could help declutter the space around the goal object, thereby reducing the physics uncertainty (at a planning level) when reaching for an object in subsequent planning, however, the solution is executed open-loop.

On the other hand, closed-loop execution considers feedback from the environment while executing the motion through a controlled variable. In this thesis, I focus specifically on Model Predictive Control (MPC) or Online Replanning (OR) approaches. In MPC we have a model of the system that we try to control and the model is used to predict the effects of robot's actions (as opposed to "feedback-control" methods such as PID control [100, 101], which do not require a model of the system). MPC approaches are especially appealing to the problem of physics uncertainty. Trajectory optimisation-based planning approaches like the ones I already described in Section 2.1.3, are particularly effective for online replanning because a new optimisation cycle can be warm-started with the previous solution, and convergence can be achieved in few optimisation iterations. I leverage this observation later in Chapter 4 and propose a novel online-replanning strategy with a human-in-the-loop.

A Model Predictive Path Integral (MPPI) approach is proposed by Williams et al.

[102] to follow an already optimised trajectory. They run *one* iteration of optimisation after *each* step execution in the real-world. A similar approach is proposed by Arruda et al. [103], Hogan and Rodriguez [104] for a pusher-slider system, and by Agboh and Dogar [66] for manipulation in clutter. In Chapter 4, I propose a similar approach to [102] and [66] but in a framework for non-prehensile manipulation with a human-in-the-loop. Moreover, each of the optimisation steps have the constraint to reach a goal state (e.g., having the goal object in the robot's hand), instead of simply optimising a soft cost like [102] and [66], and the problem I consider impose static boundaries and jamming unlike [66] which makes the problem more challenging due to the extra constraints. Recently, Agboh and Dogar [8] proposed an approach to segment a trajectory into robust and non-robust segments using a robustness metric. The system then executes robust segments in an open-loop manner, while non-robust segments are executed using a closed-loop controller.

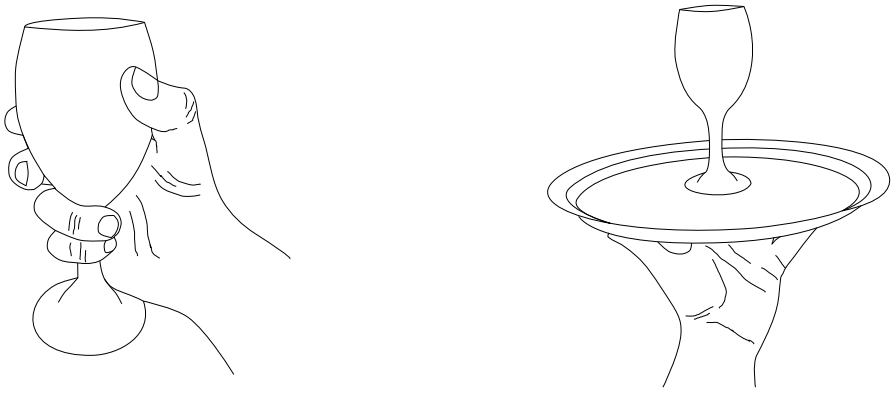
In this section, we discussed in detail about motion planning and control. I introduced important concepts, algorithms and related-work. Next, I introduce manipulation planning.

2.2 Manipulation Planning

Robotic manipulation is a broad topic in robotics, and Mason defines manipulation as "an agent's control of its environment through selective contact" [5]. In this work, I look at the robotic manipulation problem from a motion planning and control perspective and specifically robotic manipulation in cluttered environments. The problem that considers object-object interaction but the objects' final configurations are not part of the goal state, is an NP-Hard problem [1]. The problem of Reaching Through Clutter (RTC) is, therefore, in this time complexity class.

In the robotic manipulation literature, the term "clutter" is used to describe scenes where multiple objects are in a close proximity to each other and usually blocking the robot's reach. As a result, the robot poses motion planning challenges as it is required to come in contact with multiple movable obstacles while solving another problem. The term "clutter" is not defined in the literature and people might use the

term to describe scenes with 5 or 10 objects [71, 105] while others might consider 15 or more objects [9, 66, 91]. Even the number of objects alone is not defining the difficulty of the task. Some problems consider large open space and as a result, a robot can more easily push objects out of the way [9, 66], while others consider more dense spaces with little free space to manoeuvre [106, 107].



(a) (b)

Figure 2.3: (a) prehensile manipulation; the object to be manipulated (wine glass) is firmly grasped by the hand (b) non-prehensile manipulation; wine glass is placed on a tray.

Manipulation planning can generally be divided in two categories: prehensile and non-prehensile manipulation. First, in the next section, I discuss prehensile manipulation and important work in this space. In Section 2.2.2, I discuss non-prehensile manipulation and review related work.

2.2.1 Prehensile manipulation

Prehensile manipulation refers to the manipulation that requires the robot to *firmly* grasp an object (see an example in Figure 2.3(a)). That is, the grasp is *force-closed* and compensates external wrenches, thereby allowing the robot to control the motion of the object to follow the motion of its end-effector [18]. A traditional task of prehensile manipulation is Pick & Place. The robot needs to find a grasp pose, a motion to go and grasp the object and a motion to place it elsewhere. This works well

in structured and non-cluttered scenes and with objects that can be grasped by the robot's end-effector. Kimmel et al. [13] proposes the Jacobian Informed Search Tree (JIST) method for manipulation planning in cluttered scenes. It is a heuristic-guided sampling-based search algorithm that first plans in the task space and then in the robot's configuration space to generate collision-free motions for object retrieval. JIST, using Jacobian-based steering, biases the tree's expansion towards promising end-effector poses guided by the heuristic.

Prehensile manipulation can be effective in assembly operations. For example, Dogar et al. [108] propose a planner for collaborative assembly operations involving sequential grasps of parts between three robots to assemble complex structures like a chair. They formulate the problem as a Constraint Satisfaction Problem (CSP) and they propose an any-time planner to solve it. Initially, the planner tries to quickly find a solution without regrasp constraints. The planner then improves on the initial solution by gradually imposing more transfer constraints to find a solution with minimal number of re-grasps, if possible. The system assumes perfect observability, however. In [109] the planner is extended by Rosman et al. to also solve this problem by allowing the robots to choose poses that will also increase the observability of the system, thereby reducing uncertainty in the objects' poses.

For the problem of reaching through clutter, Nam et al. [91] propose an approach for object retrieval from cluttered environments. The algorithm generates a plan to relocate objects blocking the goal object to grasp the goal object with a collision-free trajectory. They generate a graph representing movable paths of objects. Using this graph, the algorithm finds a path to relocate objects to free up the space to the goal object. The approach does not consider non-prehensile actions like pushing, however, but only pick-and-place actions.

For prehensile manipulation is not always necessary that the robot is only producing collision-free motion, and people realize that even for pick-and-place tasks the robot might still need to come in contact with other objects, and they consider physics-based motion planning for pick-and-place tasks [14]. This increases the success rate of the planner since the robot is not any more strictly constraint from contacting other objects. In this specific work, Saleem and Likhachev [14], realize that motion

planning can be speeded up if the planner evaluates with physics simulation actions which involve interactions only with relevant objects (instead of avoiding interaction at all) and the rest actions are evaluated with a simple collision-checker. They propose a recursive approach to identify these relevant objects but they only consider prehensile manipulation.

Although such prehensile manipulation approaches are effective in solving various problems, for problems like reaching through clutter, it requires that the robot picks and places numerous blocking obstacles, slowing down the overall planning time of the system. As we will see in the next section, people turned to non-prehensile manipulation to manipulate very cluttered environments and use actions like pushing to deal with blocking obstacles.

2.2.2 Non-prehensile manipulation

Non-Prehensile manipulation refers to manipulation without grasping [5]. Consider a waiter who is carrying a tray with a wine glass on top like the one illustrated in Figure 2.3(b). The object is controlled, indirectly, by the waiter while moving the tray, however, it is not possible to compensate all external wrenches directly [18]. In non-prehensile manipulation, the robot use other manipulation primitives like pushing, pulling or sweeping objects. Such manipulation primitives can be useful in cluttered scenes where free-space is limited, and, therefore, a robot cannot find a collision-free motion to pick an object. They can also be more effective since the robot will be able to manipulate multiple objects simultaneously, as opposed to picking and placing multiple obstacles. It is also effective for manipulating objects that are too large or too heavy to manipulate [94] and as a preparation step to grasp an object [20].

The RTC problem is the main problem I address in this thesis. RTC is concerned with the problem where a robot needs to reach in clutter to grasp a goal object. The robot might need to push objects out of the way to create space for reaching the goal object and we, therefore, consider RTC to be a problem that requires non-prehensile manipulation.

Non-prehensile manipulation was effectively used for different challenging manipulation tasks [19, 110–114]. Lee et al. [110] describe a hierarchical planning approach for planning a sequence of prehensile and non-prehensile actions to accomplish a certain manipulation task. They propose the decomposition of the problem into three search problems, each represented as a graph, to more efficiently search for a solution. This will yield to a sequence of contact states, object poses and end-effector finger positions to solve the given problem. Their approach, however, is limited to a 2-dimensional space, with two robot contacts only. Moreover, they do not consider the robot kinematics during planning which simplifies the problem significantly.

Muhayy ud din and Rosell [15] present a knowledge-oriented physics-based motion planning for grasping in clutter. The approach uses ontology classes to capture semantic knowledge about the problem. Using this semantic knowledge, a reasoning process is used to obtain target regions, where the motion planning should be focused, and manipulation regions around obstacles. These regions are then used by kinodynamic RRT to bias the state sampling and guide the motion planner.

Numerous works focus on the problem of rearrangement planning [34, 47, 115]. This class of problems considers the problem where several objects need to be rearranged to specific locations. Although the problem I tackle in this thesis is not rearrangement planning, there are similarities and common challenges. For example, King et al. [47] focus on rearrangement planning with non-prehensile manipulation (using pushing primitives). They propose an approach that combines both object-centric and robot-centric actions. Object-centric actions refer to actions where the purpose is to manipulate a single object, whereas robot-centric actions refer to actions that consider directly the robot motion and indirectly the interaction with other obstacles. Robot-centric actions allow the robot to manipulate multiple objects simultaneously. They employ a kinodynamic RRT algorithm to solve this problem and show that a combination of the two action types improves the success rate and planning times. In this thesis, when a robot plans fully autonomously, we consider robot-centric actions. When the human provides high-level inputs (in terms of an object and where to be pushed) we use object-centric actions. In a way, my work also combines object-centric and robot-centric actions, but I use a human to provide

object-centric planning goals.

Dogar et al. [20] present an approach to grasping through clutter, where the robot is allowed to contact obstacles during a grasp attempt to clear away obstacles in the desired path. The planner pre-computes and caches robot-object (involving a single object and the robot) interactions. According to the findings, such an approach, that permits object interactions, increases the number of possible grasps and the success rate in most scenes compared to baselines which do not permit interaction with surrounding obstacles.

Effective algorithms have also been developed for Reaching Through Clutter problems [9, 20, 46, 66, 88]. However, the problem remains a challenging one, where the planning times are still in the order of tens of seconds or minutes, and the success rates are low for difficult problems. Some of these works employ the power of randomized *kinodynamic planning* as already discussed in Section 2.1.2. For example, Haustein et al. [46] use a kinodynamic RRT planner to sample and generate a sequence of robot pushes on objects to reach a goal state. Muhayy ud din et al. [9] use the KPIECE algorithm to solve this problem while others employed trajectory optimisation [7, 66]. These planners report some of the best performance (in terms of planning times and success rates) in this domain so far. I build on these kinodynamic planning approaches, but I investigate using them with human input.

Similar to the trajectory-optimisation approach I propose in Chapter 4, is the work by Kitaev et al. [7]. They are looking at the problem of reaching through clutter and physics-based manipulation. Their approach works by using the algorithm in [20] to sample initial straight-line trajectories at different approach angles, which are then pre-processed to minimize the number of obstacle contacts the robot makes. It then uses an optimisation method to optimise around it. However, the average running time of the proposed approach, especially for 20 or more objects in a shelf setting, like the ones I consider in this work, is in the order of minutes. The authors attribute this to the computational bottleneck of the gradient computation of the objective and the dynamics. My approach is based on STOMP [57] which does not require cost function gradients, and I consider human input in the optimisation which demonstrates significant improvement in the overall running time and success rate.

Moreover, my approach, unlike [7], considers the problem of physics-uncertainty as well.

Beyond prehensile and non-prehensile manipulation, another factor that could distinguish related-work in this thesis is shared-autonomous approaches. Unlike most approaches I discussed so far, semi-autonomous approaches consider human input to improve the system's performance. In the next section, I introduce important and related-work in the shared-autonomy literature.

2.3 Shared-Autonomy

Robotic teleoperation is concerned with operation of robotic systems from distance. There is usually no "intelligence" (or at least artificial) in these systems, and instead a human operator controls the robot directly. Such systems are dated back in the 1940s and 1950s to nuclear research [5, 116].

Teleoperation is still a preferred way to accomplish certain manipulation tasks (for example, robot-assisted surgery [116]). Over the years, with the advancements of Artificial Intelligence and Machine Learning, for certain tasks, the community turned to "fully-autonomous" systems[1] Section 1.1. This paradigm motivates that robots are capable to sense, reason and act on the environment autonomously, without a human controlling them directly. Certain tasks have been extremely successful with fully autonomous systems, like Simultaneous Localization and Mapping (SLAM) for static and structured environments [117]. There are tasks, however, that are extremely challenging for autonomous robots today.

People started to realize that teleoperation and full autonomy are not mutually exclusive. Shared-Autonomy, Human-In-The-Loop (HITL) Systems, Supervisory Control or Semi-Autonomy are all terms to describe systems that use autonomy for certain parts of the system, yet allow a human to control and help the robot. Without loss of generality, these systems, usually, delegate high-level reasoning to a human (since searching for one autonomously in a high-dimensional space is time-consuming

[1]The system requires no human intervention in its decision making but a human might still be needed to run the robot or provide it with a goal. Please see a detailed discussion in

and challenging) and they thrive with low-level tasks (like motion planning). In this thesis, I focus exactly on this aspect of shared-autonomy, where the human supports high-level reasoning and the robot supports the low-level work to accomplish the task.

Shared-autonomy has seen success for many robotic tasks, including path planning [38, 118–120], perception [121, 122], navigation [123], and grasping [11, 124].

Human input has been used to provide contextual information for a given task. In specific, Witzig et al. [124], highlight the importance of context information when performing a grasp. For example, if the task is to pour the contents of a bottle into a bowl, grasping the bottle from the top will be a poor choice. The human provides high-level contextual information that are then encoded in a probability density function using a Bayesian Network. This contextual knowledge is used to sort grasp hypotheses. The user might also add training data to the network by selecting one or more grasps hypotheses visualized to them through a Graphical User Interface.

Studies of human-in-the-loop systems suggest that systems that involve more robot autonomy and less human-control yield in general to better overall performance. In specific, Leeper et al. [11], present a "point-and-click" interface for grasping objects in clutter with different levels of autonomy. The results suggest that more robot autonomy leads to better performance, with fewer collisions and more successful grasps. The system, however, requires human input at all times and human input is either a low-level input (where the human controls almost directly the robot) or it is a filtering task (select a grasp from the set). In addition, Bringes et al. [125] looked into different ways human input can be integrated in motion planning for pick-and-place tasks and concluded to the same finding. That is, human-input increases success rate and improves planning times in general, however, the system benefits the most when human input is restricted to non-precise work and the robot autonomously helps on the precise work. The input of human in this work was captured using a haptic device and the human was mainly teleoperating the robot to some new configuration followed by autonomous planning. My work also builds on this belief that human input should be focused on the high-level reasoning of the task and the robot should work on the low-level part of the work. However, I focus on physics-based non-prehensile manipulation, and I am interested in integrating high-level input in

the motion planning framework directly, instead of using a human to teleoperate the robot to a new initial position before autonomous planning.

A challenge when it comes to communicate the course of motion planning to the user is how to represent a high-dimensional space to the user. Bayazit et al. [126] use haptic and visual interfaces to communicate with the human for solving path finding problems. One of the contributions of the work is the projection of the configuration space into the workspace that allows the planner to visualize its progress to a human more intuitively. The task of the human is to capture useful configurations, ideally in narrows passages, for the planner. They show that the system can even benefit from approximate paths (that are not collision-free) provided by the human by transforming them into collision-free paths with some processing.

Other works look into failure recovery systems. Sankaran et al. [127] present a state machine-based failure recovery system with shared autonomy that can query a user when other autonomous solutions are exhausted. Every state in the state machine is bidirectionally linked to a meta state called the oracle. The oracle is simply there to transmit from a failed state to any other state and to act as the communication link between the system and the user. The oracle classifies the failure into different classes, low, moderate and high failure. The user is queried according to these classifications and based on a pre-defined logic. Islam et al. [128] take a similar approach where a Multi-Heuristic A* motion-planning algorithm is used to autonomously solve a motion planning problem and they propose the use of a heuristic-based approach to estimate when a robot is in a "stagnation" region so they can query a human for guidance.

This line of work, where a system can incorporate human intervention when needed and adjust its level of autonomy, is called *sliding-autonomy* and other people have proposed similar systems [129–132]. The recent work by Swamy et al. [129], for example, propose a shared-autonomy approach that tries to learn to automatically assign to the human the robot that he would have chosen to operate. The approach works by observing the human operating a small number of robots. Then, it tries to fit a model to explain the user's choices based on an internal scoring function that allows the system to map a state to a real value. Using this learned model the

system is capable, in a larger group of robots, to predict which robot the human would choose to operate and assign it to him automatically. This approach addresses simple toy tasks. Learning a scoring model for real-world problems like RTC might be challenging. I take a similar approach in Chapter 5 but trying to learn to predict the optimisation cost of the problem in the future before and after human intervention to allow the system to calculate a predicted gain that can then be used to decide which robot the human should supervise.

The idea that a system starts fully autonomous and falls-back to a human when necessary is of interest to this thesis. In Chapter 4, I investigate how this can be achieved in a motion planning problem for non-prehensile manipulation, but I leverage trajectory optimisation techniques to detect when an autonomous motion planner is expected to fail (i.e., local minima) before falling-back to a human operator, and I extend it to a predictive system in Chapter 5.

Other approaches to shared-autonomy are used to allow a human operator to teleoperate a robot [10, 12] or to command the robot to do some task, essentially making the human input acting as a transition from one task to another [133]. These systems employ autonomous functionalities, but the human input is usually used to teleoperate or command the robot to do certain tasks. As an example, Ciocarlie et al. [12] developed a mobile manipulation platform with a human-in-the-loop to assist a disabled user. They present a "point-and-click" user interface that allows a user to control the robot to perform complex manipulation tasks in complex and dynamic environments. Certain functionalities permit the robot to act autonomously (like navigation), while for other tasks (manipulation), the user can directly teleoperate the robot or provide a high-level input. The work demonstrates non-prehensile manipulation through *direct teleoperation*, not as a result of a semi-autonomous motion planner. Muszynski et al. [10] propose a similar approach, but focusing on the user interfaces for mobile devices while considering adjustable autonomy. The user might choose to adjust the level of autonomy of the robot depending on the task or if the robot fails to act autonomously. Both of these systems use the human input to move the system from one state to another or to replace an autonomous subsystem entirely. I am interested in a similar setting, but focusing especially on

non-prehensile manipulation semi-autonomous planning and how human input can be integrated in the motion planner.

2.4 Remarks

This chapter introduced in more depth the four main pillars this thesis builds on; (1) motion planning, (2) motion control, (3) physics-based non-prehensile manipulation, and (4) shared-autonomy. Through in-depth review of related-work, I introduced important works in the field and how they differ from this work. I also emphasized on the difficulties of robotic manipulation, especially in dynamic, unstructured and cluttered environments.

Up to this date, physics-based non-prehensile manipulation in clutter is still an open problem, and it takes several seconds or minutes for existing solutions to solve hard instances of the problem consistently. The study of physics-based non-prehensile manipulation in clutter with a human-in-the-loop, however, to the best of my knowledge, was not studied in detail before. This thesis focuses on this specific intersection with the aim to understand how a human-in-the-loop system can be used in such problems and to develop approaches for effective, fast and robust motion planning and control for non-prehensile manipulation in clutter. In specific, I propose a framework for sampling-based planning that extends existing sampling-based approaches (like kinodynamic RRT and KPIECE) in Chapter 3, and in Chapter 4, I introduce a trajectory-optimization-based framework with human-in-the-loop. Finally, in Chapter 5, I present a predictive guided framework that learns when to ask for human help. My findings are encouraging and show that human-input in such problems is effective and can accelerate the robots' performance significantly.

In the next chapter, I start exploring how human input can be integrated in sampling-based motion planners.

Chapter 3

Integrating Human Input in Sampling-based Kinodynamic Planning

Chapter Deliverables

Demonstration video: `https://youtube.com/watch?v=nfr1Fdketrc`

Source code: `https://github.com/rpapallas/hitl-clutter`

3.1 Introduction

In this chapter, I begin by proposing a framework for non-prehensile manipulation and a human-in-the-loop by employing sampling-based kinodynamic planning (Section 2.1.2). In my framework, the human operator supplies a *high-level plan* to make the underlying planners solve the problem faster and with higher success rates.

Example problems are depicted in Figures 3.1 and 3.2. The target of the robot is to reach and grasp the green object. To do this, however, the robot first has to push other objects out of the way (Figure 3.1(b) to Figure 3.1(e)). This requires the robot to plan which objects to contact and where to push those objects so that it can reach the goal object. I present an approach to this problem where a human-in-the-loop provides a *high-level plan*, which is used by a low-level planner to solve the problem.

For example, in Figure 3.1(a), the human operator supplies the high-level action of first pushing the object o_2 to the blue region. A key point here is that pushing o_2 in Figure 3.1(a) to its target region is itself a problem which requires kinodynamic planning through clutter, since the object and the robot may need to contact and push other objects during the motion.

Therefore, the action in Figure 3.1(a) requires the use of a planner that can potentially solve the original larger problem of reaching the green object through

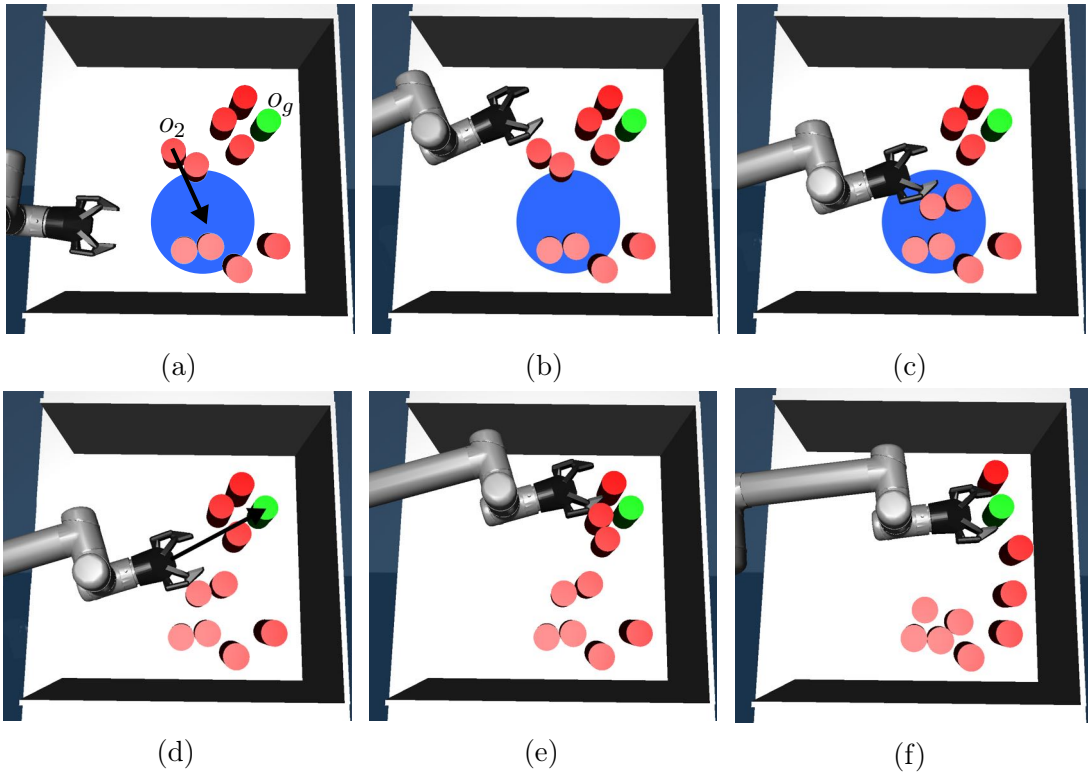


Figure 3.1: A human-operator guiding a robot to reach for the green goal object, o_g. Arrows indicate human interaction with the robot. In (a) the operator indicates o_2 to be pushed to the blue target region. From (a) to (c) the robot plans to perform this push. In (d) the operator indicates to the robot to reach for the goal object. From (d) to (f) the robot plans to reach the goal object.

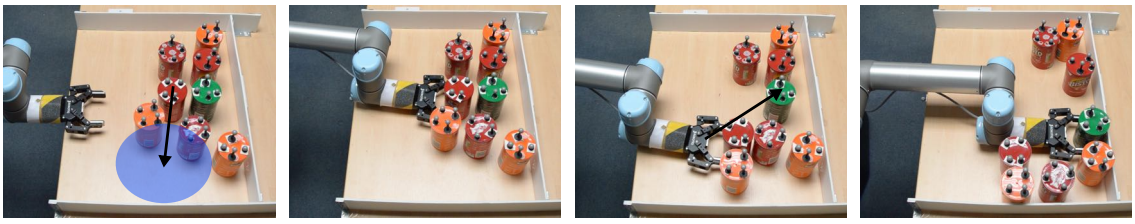


Figure 3.2: Human-operator guiding a robot in the real-world.

clutter. However, solving the original problem in highly cluttered settings can take a long and infeasible amount of time. The human can suggest high-level actions that decompose the original problem into a sequence of easier problems of the same type. The robot uses a kinodynamic planner to approach o_2 in Figure 3.1(b) and to push it to the target region in Figure 3.1(c). The human operator's input here is limited to selection of the object and an approximate goal location for that object. The actual pushing actions are planned by the autonomous low-level planner. In Figure 3.1(d) the operator points directly the actual goal object (green). The kinodynamic planner in Figure 3.1(e) finds a way to push other objects out of the way and in Figure 3.1(f) successfully reaches the goal object. The human-operator's role in the system is not to guide the robot all the way to the goal, but to provide key high-level actions to help the robot. At any point during the interaction, even at the very beginning, the operator can decide not to provide any further high-level actions (either because the scene is easy enough for the low-level planner or because the operator is busy) and she can command the system to plan directly for the actual goal object. The system degrades nicely to state-of-the-art kinodynamic planning if no high-level actions are provided.

I compare this framework to using kinodynamic methods without any high-level plans, e.g. KPIECE and RRT and to hierarchical methods which generate high-level plans autonomously. For the latter, I implemented a non-prehensile variation of the NAMO planner [89] as well as an approach which uses a straight-line motion heuristic to generate candidate objects for the high-level plan. I performed experiments in simulation and on a real robot, which show that the human-in-the-loop approach produces more successful plans and faster planning times. This gain, of course, comes at the expense of a human operator's time. The results show that this time is minimal and to evaluate this further, I experiment with a single human operator providing high-level plans in-parallel to multiple robots, and present an analysis. I discuss whether such an approach may be feasible in a warehouse automation setting.

While a good high-level plan can make the low-level planning problem easier to solve, the autonomous generation of a good high-level plan is itself a computationally expensive problem. For example, for the problem of reaching through clutter, a

high-level planner would need to search in a space of all possible permutations of objects, combined with all possible goal locations for these objects. Furthermore, the high-level planner must be able to choose high-level actions that are feasible for the low-level planner. This feasibility would mean checking or predicting whether the robot would be able to push a certain object to a certain location, which either requires the use of computationally expensive physics simulations, or a heuristic to estimate the probability of successful push. While these decisions are computationally expensive for an autonomous planner, they can be easy for a human.

The structure of this chapter is as follows. In the next section, Section 3.2, I outline some assumptions, the objectives of this chapter and the contributions. In Section 3.3, I formulate the problem and introduce important notations for this chapter. I then introduce the proposed approach, the Guided-RTC Framework, as a general framework for solving RTC problems in Section 3.4.1. Then, I provide concrete implementations of the general framework in Sections 3.4.2 to 3.4.4. Finally, I evaluate the proposed algorithms in simulation, on a real robot and in a parallel setting in Section 3.5 and provide an analysis.

3.2 Assumptions, Objectives and Contributions

Assumptions

In this chapter, I make the following assumptions:

- The robot is constrained on the plane and that the objects cannot be grasped from the top. Therefore, the robot motion is constrained in $SE(2)$.
- The goal object is easily graspable by closing the robot fingers.
- There is a perception system that detects object poses with good accuracy.

Objectives

The objective of this chapter is to explore and extend sampling-based planners (e.g., RRT and KPIECE) to consider human-input for the problem of physics-based

non-prehensile manipulation in clutter. This is a challenging problem for existing sampling-based kinodynamic methods, and employing human input in this problem might be beneficial. The aim of this chapter is to propose such approaches and to evaluate and compare them against existing state-of-the-art methods.

Contribution

The contribution of this chapter is therefore a sampling-based framework for non-prehensile physics-based manipulation in clutter with a human-in-the-loop.

3.3 Problem Formulation

The environment comprises a robot r, a set of movable obstacles $\mathcal{O}$, and other static obstacles. The robot is allowed to interact with the movable obstacles, but not with the static ones. There is also a *goal* object $o_g \in \mathcal{O}$ to reach.

I am interested in problems where the robot needs to reach for an object in a cluttered shelf that is constrained from the top, and therefore the robot motion is constraint to the plane and its configuration space, $\mathcal{X}^r$, to $SE(2)$. The configuration of a movable object $i \in \{1, \ldots, |\mathcal{O}|\}$, x^i, is its pose on the plane (x, y, θ). I denote its configuration space as $\mathcal{X}^i$. The configuration space of the complete system is the Cartesian product $\mathcal{X} = \mathcal{X}^r \times \mathcal{X}^g \times \mathcal{X}^1 \times \cdots \times \mathcal{X}^{|\mathcal{O}|-1}$.

Let $x_0 \in \mathcal{X}$ be the initial configuration of the system, and $\mathcal{X}_{goals} \subset \mathcal{X}$ a set of possible goal configurations. A goal configuration, $x_n \in \mathcal{X}_{goals}$, is defined as a configuration where o_g is within the robot's end-effector (see Figure 3.1(f)).

Let U be the control space comprised of the robot velocities. Let the system dynamics be defined as $f : \mathcal{X} \times U \to \mathcal{X}$ that propagates the system from $x_t \in \mathcal{X}$ with a control $u_t \in U$.

I define the RTC problem as the tuple $(\mathcal{X}, U, x_0, \mathcal{X}_{goals}, f)$. The solution to the problem is a sequence of controls from U that move the robot from x_0 to a $x_n \in \mathcal{X}_{goals}$.

3.4 Guided-RTC Framework

In this section, I describe a *guided* system to solve RTC problems. A Guided-RTC system accepts high-level actions. A high-level action can suggest to push a particular obstacle object into a certain region, or it may suggest to reach for the goal object. The high-level action is formally defined with the triple (o_i, x_i, y_i), where $o_i \in \mathcal{O}$ is an object, and (x_i, y_i) is the centroid of a *target region* that o_i needs to be pushed into. The target region has a constant diameter d. When $o_i = o_g$, this is interpreted as the high-level action to reach for the goal object, and the centroid can be ignored. The high-level actions may be suggested by an automated high-level planner or by a human-operator.

Consider Figure 3.1 as an example. A human-operator suggests a high-level action $(o_2, 1.15, 0.4)$ (Figure 3.1(a)), where $(1.15, 0.4)$ is the centroid of the blue target region. The Guided-RTC system finds the controls to push o_2 into the target region (Figure 3.1(c)). When the human-operator suggests reaching for the goal object o_g (Figures 3.1(d) to 3.1(f)), the system finds the controls to perform this action.

I investigate how a Guided-RTC system with a human-in-the-loop performs when compared with (a) solving the original RTC problem directly using kinodynamic approaches, and (b) using Guided-RTC systems with automated ways of generating the high-level actions. In this chapter, when we plan with a kinodynamic planner (either RRT or KPIECE) we will use the notation KINODYNAMICPLANNING(x_{start}, *goal*) with a start configuration of the system, x_{start}, and some *goal* input.

In Section 3.4.1, I present a generic algorithm to implement the Guided-RTC framework which is agnostic to how the high-level actions are generated. Then I present different approaches to generate the high-level actions, including a human-in-the-loop approach in Section 3.4.2, as well as two other automated approaches in Sections 3.4.3 and 3.4.4.

Algorithm 1 Guided Reaching Through Clutter (GRTC)

Input: x_0: The initial state of the system
$\mathcal{X}_{goals}$: A set of goal states

1: **procedure** GRTC
2: $x_{current} \leftarrow x_0$
3: **do**
4: $o_i, x_i, y_i \leftarrow$ getNextHighLevelAction($x_{current}$)
5: **if** $o_i \neq o_g$ **then**
6: $x_{a1}, x_{a2} \leftarrow$ compute approaching states to o_i
7: kinodynamicPlanning($x_{current}, \{x_{a1}, x_{a2}\}$)
8: **if** planning in line 7 fails **then go to line 4**
9: kinodynamicPlanning(x_{a1} or $x_{a2}, (o_i, x_i, y_i)$)
10: **if** planning in line 9 fails **then go to line 4**
11: $x_{current} \leftarrow$ execute solutions from lines 7 and 9
12: **while** $o_i \neq o_g$
13: kinodynamicPlanning($x_{current}, \mathcal{X}_{goals}$)
14: **if** planning succeeds **then**
15: $x_{current} \leftarrow$ execute solution from line 13

3.4.1 A generic approach for Guided-RTC planning

The generic algorithm, Guided-RTC, is presented in Algorithm 1. The initial configuration of the problem is assumed to be the current configuration, $x_{current}$, of the system (line 2). The next high-level action is decided based on the current configuration (line 4). In the concrete implementation of this framework, the high-level action can be provided either by a human operator or an automatic process. Here we are not concerned with where the high-level input comes from, but that is informed somehow based on the current state of the system. If the object in the high-level action is not the goal object (line 5), then it is pushed to the target region between line 6 and line 11, and a new high-level action is requested. If it is the goal object, the robot tries to reach it between line 13 and line 15 and the system terminates.

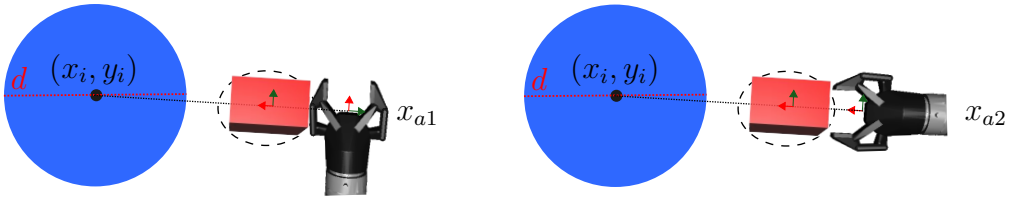


Figure 3.3: Approaching states: The blue circle is the target region, the red rectangle the object to manipulate. We compute two approaching states, x_{a1} and x_{a2}. The two approaching states encourage side-ways and forward pushing actions respectively, and they increase the chance of a successful pushing.

Pushing an object

The plan to push an object to its target region is done in two steps. In line 7 we plan to an intermediate *approaching state* near the object, and then in line 9 we plan from this approaching state to push the object to its target region. Specifically, given an object to push, o_i, we compute two approaching states x_{a1} and x_{a2} (line 6). Figure 3.3 shows how these approaching states are computed, based on the object's current position, the centroid (x_i, y_i) and the minimum enclosing circle of the object. The approaching state x_{a1} encourages side-ways pushing, where x_{a2} encourages forward pushing. I also experimented planning without first approaching the object but I found that approaching the object from a good pose yields to faster pushing solutions. Using both approaching states as the goal we plan to move to one of them (multi-goal planning) in line 7. Then, from the approaching state reached (either x_{a1} or x_{a2}) we push o_i to its target region (line 9). If any of the two planning calls in lines 7 and 9 fails, then the algorithm proceeds to the next high-level action (line 4). It is assumed that the next high-level action will be different since if the initial high-level action fails, this is communicated to the human and is assumed that they will choose a different high-level action, and in the case of automatic approaches, they will provide a new random centroid which will result to a new high-level action as well. Otherwise, we execute the solutions sequentially in line 11, which changes the current system configuration $x_{current}$.

Reaching for the goal object

Finally, in line 13, we plan to reach and grasp the goal object and in line 15 we execute the solution if it is successful. We use kinodynamic planners (e.g. kinodynamic RRT or KPIECE) to support the planning in lines 7, 9 and 13.

Algorithm 1, runs up to an overall time limit, $T_{overall}$, or until a goal is reached. The pushing planning calls in lines 7 and 9 have their own shorter time limit, $T_{pushing}$ and they should find a valid solution within that time limit. The planning call in line 13 is allowed to run until the overall time limit is over.

Now that the Guided-RTC is defined, in the next section I define an implementation of the framework with a human-in-the-loop.

3.4.2 Guided-RTC with Human-In-The-Loop (GRTC-HITL)

Algorithm 2 GRTC-HITL

Input: $x_{current}$: The current state of the system
Output: A high-level action

```
1: function getNextHighLevelAction
2:   o_i ← get object selection from human operator
3:   if o_i ≠ o_g then
4:     x_i, y_i ← get region centroid from human operator
5:     return o_i, x_i, y_i
6:   return o_g
```

Guided-RTC with Human-In-The-Loop (GRTC-HITL) is an instantiation of the GRTC Framework. A human-operator, through a graphical user interface, provides the high-level actions. In Algorithm 2, I present GRTC-HITL *getNextHighLevelAction* function (referenced in Algorithm 1, line 4).

The human provides high-level actions, until she selects the goal object, o_g. The GRTC framework (Algorithm 1) plans and executes them. The state of the system changes after each high-level action and the human operator is presented with the resulting state each time ($x_{current}$). Note here that *the operator can decide not to provide any guidance* (by selecting the goal object straightaway), which would be

equivalent to running a state-of-the-art kinodynamic planning on the original RTC problem.

I developed a simple user interface to communicate with the human-operator. The operator at every step is presented with a window showing the environment and the robot. The operator, using a mouse pointer, provides the input (an object and a point on the plane) by first clicking on the desired object and then a point on the plane (Figure 3.1(a)) that becomes the centroid of the target region.

When planning fails (for example in Algorithm 1, in lines 8 and 10) a message is shown on the window to the user to indicate that the planning failed and to inform the user that a new input is required (i.e., line 4).

The approach I propose here uses a human-operator to decide on the high-level plan. One question is whether one can use automatic approaches, and how they would perform compared to the human suggested actions. To make such a comparison, I implemented two automated approaches.

3.4.3 Guided-RTC with NAMO

NAMO by Stilman et al. [89] is a well established algorithm for the problem of Navigation Among Movable Obstacles and can, therefore, be used as an automated approach to finding a high-level rearrangement plan, similarly to what a human suggests in GRTC-HITL. NAMO has originally been used for pick-and-place manipulation. Here I adapted NAMO to work for non-prehensile tasks by using a kinodynamic planner as the low-level planner, instead of collision-free motion planners as in the original work.

To determine the ordering of objects to manipulate and where to place them, i.e. the high-level plan, NAMO uses backward planning. Considering the initial problem of reaching in the fridge to grasp the orange juice (Figure 1.1), NAMO starts by running the low-level planner to reach for the orange juice, assuming the robot can travel through other movable objects. The resulting volume of space swept by the robot to reach the orange juice is then checked to see which movable objects intersect with it (for example identifies the butter and the yogurt). These objects are then added to a queue to be moved out of this swept volume. The algorithm then pops

out an object from this queue, and makes a recursive call to reach and move that object. This process continues until the queue is empty, meaning that (1) there is a plan to reach and move every object out of the way, and (2) there is a position to place every object out of the accumulated robot swept volume. The last object planned for is the first one to be moved during execution.

Since NAMO plans backwards, to decide on the first object to be moved, it needs to determine all the objects to be moved and their target positions. While this means NAMO can offer theoretical guarantees when a plan exists, it also means that in highly cluttered environments like ours (consisting of multiple obstacles and little free space), NAMO can quickly run out of space to place objects, before it resolves all the constraints. In my experimental setting, which includes ten objects in a restricted shelf space, NAMO failed in all cases by filling up the space with the robot swept volume before a plan for all objects in the queue have been found.

This motivated me to design a heuristic approach similar to NAMO, but one that plans forward, by directly identifying the first object to move out of the way.

3.4.4 Guided-RTC with straight line heuristic

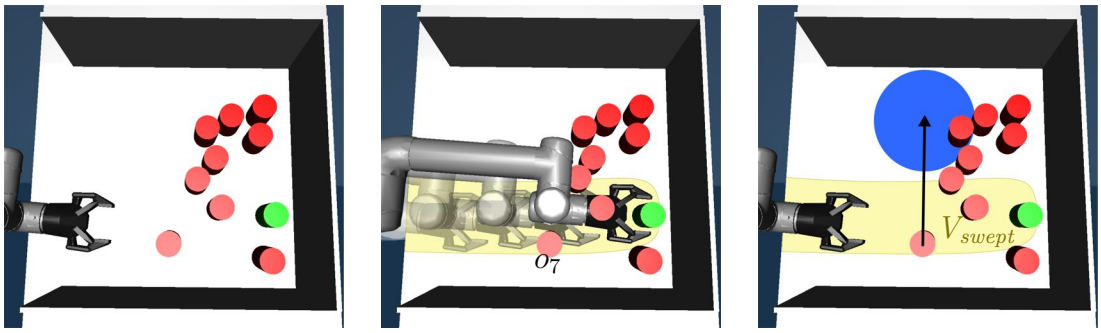
Algorithm 3 GRTC-Heuristic Planner

Input: $x_{current}$: The current state of the system
Output: A high-level action

```
1: function getNextHighLevelAction
2:   o_b ← find the first blocking obstacle to o_g
3:   if there exists a blocking obstacle o_b then
4:     x_b, y_b ← find collision-free placement of o_b
5:     return o_b, x_b, y_b
6:   return o_g                          ▷ No blocking obstacle, reach the goal
```

I present this approach (GRTC-Heuristic) in Algorithm 3 and illustrate it in Figure 3.4. This heuristic assumes the robot moves on a straight line from its current position towards the goal object (Figure 3.4(b)). The first blocking object, o_b in line 2, is identified as the next object to be moved. During the straight line motion we

capture robot's swept volume, V_{swept} (Figure 3.4(b)). We randomly sample a collision-free target region centroid outside V_{swept} (Algorithm 3 line 4 and Figure 3.4(c)). The object and the centroid are then returned as the next high-level action (Algorithm 3 line 5). When the heuristic finds the centroid of the high-level action (i.e., the new location of where the blocking obstacle should be pushed), it initially tries to find a valid pushing centroid 30cm around the obstacle's initial position to maximize the chance of a successful pushing. That is, pushing the object anywhere on the shelf might be a difficult or infeasible motion planning problem, while pushing the object near its initial position will more likely be a feasible motion planning problem. If there is no collision-free space around o_b, then we sample from the entire space.



(a) (b) (c)

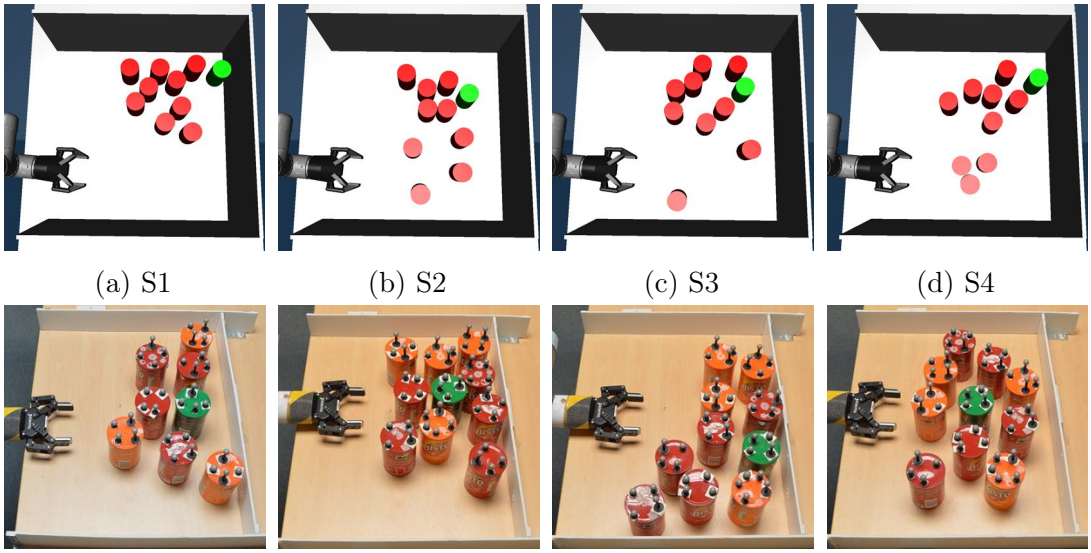
Figure 3.4: GRTC-Heuristic: (a) Initial state. (b) The robot moves on a straight line to the goal object, o_g, to obtain the first blocking obstacle (o_7) and the swept volume (yellow area). (c) The heuristic produces a high-level action for o_7 indicated by the arrow and the target region (blue). This process is repeated until V_{swept} contains no blocking obstacle.

After every high-level action suggested by the heuristic, the Guided-RTC framework (Algorithm 1) plans and executes it and the state of the system is updated ($x_{current}$). The heuristic then suggests a new high-level action from $x_{current}$ until there is no blocking obstacle (Algorithm 3 line 6).

This heuristic, although simple and effective, is a greedy algorithm and assumes that one object lookahead (i.e., finding the next object to manipulate at a time) will suffice to find a solution to the problem. This might not always be the case and it

could even get into a case where there is no feasible high-level action. In this case, the planner will keep searching for one until a time limit is reached. For the purposes of these experiments, and as a baseline, one object lookahead was in general sufficient and found solutions to problems where the unguided planners (i.e., KPIECE and RRT) failed.

3.5 Experiments & Results



(a) S1 (b) S2 (c) S3 (d) S4

(e) R1 (f) R2 (g) R3 (h) R4

Figure 3.5: Initial states of different problems in simulation (S1-S4) and real world (R1-R4). Goal object is in green.

The algorithms I evaluate are: (1) GRTC-HITL which is the main algorithm I propose and uses a human operator to obtain the high-level actions (Section 3.4.2), (2) GRTC-Heuristic which uses a straight line heuristic to automatically determine the high-level actions (Section 3.4.4) and (3) Kinodynamic RRT and KPIECE (Section 2.1.2) which plan to reach for the goal object without a high-level plan. As explained in Section 3.4.3, NAMO failed to find solutions in our problems and therefore I did not include results for it here.

3.5.1 Hypotheses

The hypotheses I am evaluating in this section are:

1. Human-input is effective in the problem of physics-based non-prehensile manipulation.
2. The time the human operator spends on supervising the robot is minimal.
3. A single human operator could effectively guide more than one robot simultaneously.
4. Human-input is effective in the real-world as well and should minimize the problem of physics-uncertainty.

These are the working hypotheses of this chapter. The first hypothesis states that the proposed approach should be effective for solving RTC problems. It can be measured based on the success rate of solving various RTC problems and the planning time (how fast is the proposed approach). This is evaluated in Section 3.5.3.

The second hypothesis suggests that human time with the system is minimal. This can be measured with experiments in simulation and by recording the actual time spent for the human to suggest high-level actions. This is also evaluated in Section 3.5.3.

The third hypothesis states that a human operator can supervise a number of robots at the same time. This can be evaluated by running experiments with more than one robot under human supervision and comparing the performance of the average robot when guided in parallel to one when guided individually. This hypothesis is evaluated in Section 3.5.4.

Finally, the fourth hypothesis states that human input should minimize the problem of physics-uncertainty. The intuition here is that the high-level action(s) of the human should result in a simpler reaching through clutter problem with fewer interactions with other obstacles. This can be evaluated by conducting experiments in the real-world and measured by the success rate of the proposed system compared to state-of-the-art baselines. This hypothesis is evaluated in Section 3.5.5.

3.5.2 Experimental setup

For all experiments, I use the Open Motion Planning Library (OMPL) [56] implementation of RRT and KPIECE. I use MuJoCo[1] [134] to implement the system dynamics, f. For all planners, the overall planning time limit, $T_{overall}$, is 300 seconds, after which it was considered a failure. For GRTC-HITL and GRTC-Heuristic, $T_{pushing}$ is 10 seconds. For GRTC-HITL, the human-interaction time was included in the overall time limit. All the experiments between methods had the same initial conditions, with the only exception of the real-world experiments, where the initial conditions (positions of objects and of the robot) were approximately the same between trials. I conducted all the experiments in this chapter. Since I am interested in an industrial/warehouse scenario where human-operators would be trained to use the system, I am mainly interested in the performance of trained operators, rather than novices. For real-world experiments, I performed experiments using a UR5 manipulator on a Ridgeback omnidirectional base. I used the OptiTrack motion capture system to detect initial object/robot poses and to update the state in the human interface after every high-level action.

3.5.3 Simulation results

I evaluated each approach 100 times by running them 10 times in 10 different, randomly-generated, scenes. I use the names S1-S10 to refer to these ten randomly-generated scenes and each approach was evaluated in the *same* 10 scenes. I use a randomizer that places the goal object at the back of the shelf and then incrementally places the remaining (nine) objects in the shelf such that no object collides with each other. Some scenes are presented in Figures 3.5(a) to 3.5(d).

For GRTC-HITL, the human-operator interacted with each scene *once* and from the last state left by the human-operator I ran the planner (Algorithm 1 line 13) to reach for the goal object *10 times*. For GRTC-Heuristic I ran all 100 experiments with both RRT and KPIECE as the low-level planners and I present the better performing

[1]On a computer with Intel Core i7-4790 CPU @ 3.60GHz, 16GB RAM.

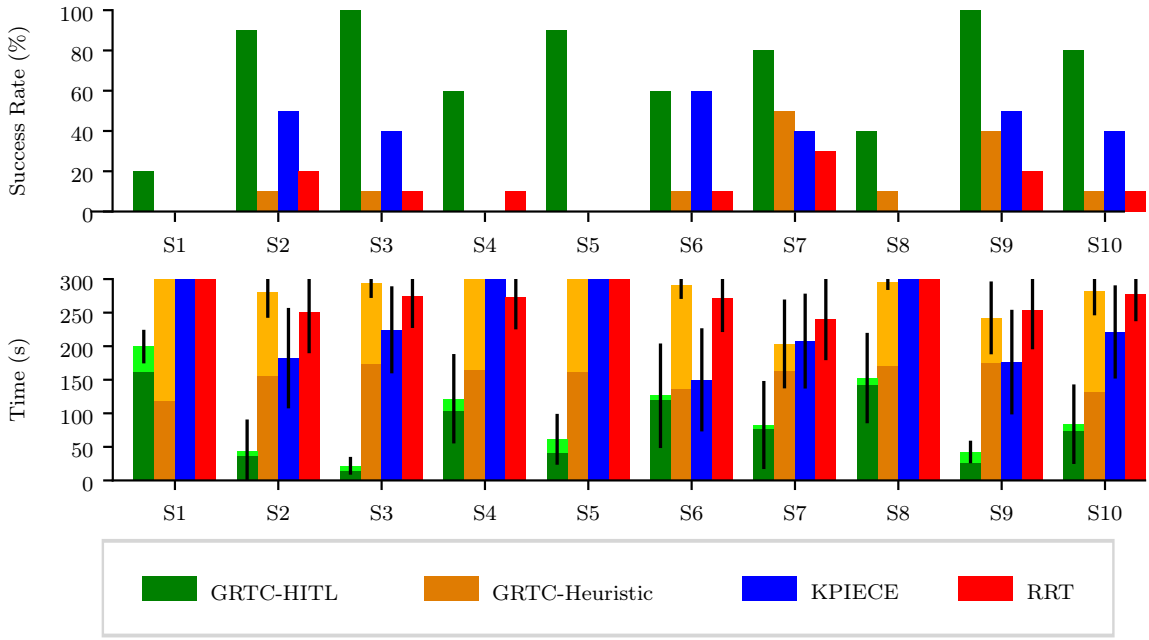


Figure 3.6: Simulation results, for each scene (S1-S10): (Top) Success rate. (Bottom) Mean planning time. The error bars indicate the 95% CI. For GRTC-HITL and GRTC-Heuristic, the dark shade indicates the planning time where the light shade indicates the time it took to produce the high-level actions (for GRTC-HITL this is a fraction of the time). The results suggest that the proposed approach (GRTC-HITL) performs better than the baselines both in success rate and planning times, and that in some scenes human input is especially effective (e.g., S2, S3, S5, S9).

one. For GRTC-HITL and GRTC-Heuristic the low-level planner is RRT.

Figure 3.6 summarizes the results of the experiments for each of the random scenes (S1-S10). Figure 3.6-Top shows that GRTC-HITL is more successful per scene than any other approach except scene S6 in which it was as successful as KPIECE. The overall success rate for each approach is 72% for GRTC-HITL, 11% for RRT, 28% for KPIECE and 14% for GRTC-Heuristic. Figure 3.6-Bottom shows that GRTC-HITL improved the planning time in *all* scenes. This suggests that the first working hypothesis holds true, the proposed approach with the human-in-the-loop is effective for the problem of physics-based non-prehensile manipulation.

Table 3.1 summarizes the guidance performance for GRTC-HITL and GRTC-

Table 3.1: Guidance comparison between GRTC-HITL (human guidance) and GRTC-Heuristic (automatic guidance). The results show that the proposed actions of human are in general successful and hence the guidance time (time spent guiding) is low compared to the automatic approach where it spends more time proposing high-level actions unsuccessfully.

	GRTC-HITL		**GRTC-Heuristic**	
	μ	σ	μ	σ
Proposed Actions	4.9	3.3	88.4	58.2
Successful Actions	3.1	1.0	3.0	1.4
Guidance Time (s)	13.6	10.0	124.3	81.7

Heuristic for all ten scenes. Proposed Actions indicates the total number of high-level actions proposed. This number includes the successful actions (actions that the planner managed to satisfy) and failed actions (actions that the planner could not find a solution for). Guidance Time indicates the time spent on generating the high-level actions in seconds (in case of GRTC-HITL the time the human-operator was interacting with the system and for GRTC-Heuristic the time took for the heuristic to generate the high-level actions). On average, the human proposed around 5 actions, of which around 3 were successful. On the other side, GRTC-Heuristic proposed on average around 88 actions, of which only 3 were successful. The human operator spent on average 14 seconds interacting with the system while GRTC-Heuristic spent on average 124 seconds proposing high-level actions. These results suggest that the second working hypothesis holds true, the time the human spends supervising the robot is a fraction of the planning time.

3.5.4 Parallel guidance

Since the human spends only a small amount of time guiding the robot using GRTC-HITL, a single human-operator can be used to guide multiple robots simultaneously, e.g. in a warehouse where a small number of operators remotely guide a large number of robots. I present an example in Figure 3.7. This experiment aims to investigate if guiding multiple robots simultaneously affects the quality of the guidance and if,

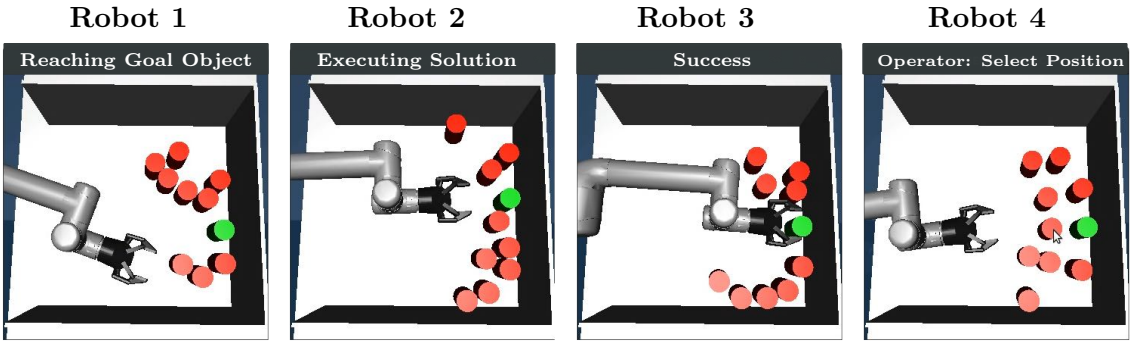


Figure 3.7: Parallel Guidance: Robots guided in parallel in group of fours. The first robot is planning to reach for the goal object, the second one executes a solution, the third robot successfully reached the goal object, the fourth robot is waiting for human input (operator's main focus).

therefore, a single human can guide multiple robots at the same time. I compare a single human operator guiding a single robot at a time compared to guiding four robots in parallel. I generated twenty randomly-generated scenes and compared twenty robots in groups of four (Parallel) with twenty robots guided individually (Individual).

Table 3.2: Guidance performance when a robot is guided individually (no distractions) compared to when it is guided in a group of other robots (human distracted guiding other robots simultaneously). The results show that planner waits for some time and that overall planning time is slightly higher.

	Parallel		**Individual**	
	μ	σ	μ	σ
Guidance Time (s)	21.1	28.0	13.0	14.4
Overall Planning Time (s)	139.7	117.9	122.8	120.0
Planner Idle Time (s)	14.8	9.4	0.0	0.0

Table 3.2 summarizes these results. The success rate of parallel guidance is 60% and for individual guidance 70%. This efficient use of the human-operator's time comes with the cost of slightly increased planning time and lower success rate. I also measured the time the system was waiting for human input which was on average 15 seconds. This suggests that, although there is a decrease in success rate and slight

increase in planning times, the benefits of the proposed system is still effective when guiding multiple robots.

3.5.5 Real-robot results

Simulation is an approximation of the real-world and is not an exact representation of the real-world. I am particularly interested in solutions that are robust in the real-world, not in simulation. In these experiments, I, therefore, evaluate the proposed approach in a realistic environment in presence of physics uncertainty and compare it against state-of-the-art baselines. With physics uncertainty, I describe the problem when the robot's actions in the real-world brings the robot in different states than the ones predicted by the simulator and therefore the solution, although valid in simulation, is invalid in the real-world. A more elaborated discussion on the problem of physics uncertainty is discussed in Section 2.1.6.

I evaluated RRT, KPIECE and GRTC-HITL performance in *ten* different problems in the real world. I show some of the scenes in Figures 3.5(e) to 3.5(h). The video in the deliverables, at the beginning of the chapter, shows an example of these results.

Table 3.3: Results when solutions are executed in the real world. GRTC-HITL is in general more successful than the baselines and baselines suffer from planning and execution failures.

	GRTC-HITL	**KPIECE**	**RRT**
Successes	7	1	2
Planning Failures	2	4	8
Execution Failures	1	5	0

Table 3.3 summarizes the success rate of each approach in the real world. When I say that the robot failed during execution, I mean that although the planner found a solution, when the robot executed the solution in the real-world, it either failed to reach the goal object, or it violated some constraint (hit the shelf or dropped an object to the floor). These execution failures were due to the physics uncertainty in the real world.

The success rate for GRTC-HITL, RRT and KPIECE is 70%, 20%, and 10% respectively. GRTC-HITL failed 20% during planning and 10% during execution. KPIECE was more successful during planning than RRT but failed most of the time during execution. RRT on the other hand accounts for more failures during planning than any other approach. This suggests that the last hypothesis also holds true and human-input is effective in the real-world compared to state-of-the-art baselines.

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

Figure 3.8: A guided planning demonstration in the real world. The robot tries to reach and grasp the green object. The human input is indicated by the black arrow which points an object (orange) and a location to be pushed (blue region). The robot executes the high-level action provided by the human and then reaches successfully for the goal object.

In Figures 3.2 and 3.8 I show two examples. In the first example, the human operator provides the first high-level action in Figure 3.2(a) and then indicates the goal object in Figure 3.2(c) which is reached in Figure 3.2(d). In the second example, the human-operator provides initially two high-level actions (Figure 3.8(a) and Figure 3.8(b)). The operator in Figure 3.8(c) indicates the goal object and the robot reached the goal object in Figure 3.8(d).

3.6 Conclusions

In this chapter, I introduced a new human-in-the-loop framework for physics-based non-prehensile manipulation in clutter (GRTC-HITL). The proposed approach leverages high-level human inputs in a kinodynamic motion planning framework. The input of the user is an object and an approximate location of where it should be pushed and is

captured quickly through a mouse-based graphical user interface. The planner then uses this input to find a kinodynamic plan to push the object before it reaches for the goal object. In this chapter, I consider sampling-based kinodynamic planners like KPIECE and kinodynamic RRT as the low-level planning algorithms. I compare the proposed approach against three baselines and showed through simulation and real-world experiments that GRTC-HITL is more successful and faster in finding solutions than these baselines. One of the baselines uses a heuristic to automatically find a high-level plan (similar to the one provided by the human) and the results suggest that human intuition is still useful in hard instances of the problem. Additionally, the human engagement with the system is minimal (less than 14 seconds) yet it significantly improves the performance of the robot.

A noteworthy question is whether a single human operator can guide effectively a number of robots in parallel. I presented experiments where a single human-operator guides four robots in parallel, to make best use of the operator's time. The results suggest that a single human operator can guide a number of robots in parallel to the cost of slightly increased planning times and lower success rates. This is the first step towards the goal of this thesis to investigate how human input can be used in physics-based non-prehensile manipulation algorithms.

Although this approach shows increase in success rate and improvements to the planning times, there are certain shortcomings:

1. Since sampling-based planners search for a global solution and they are sampling-based, the resulting trajectories tend to be long and noisy. That is, the robot moves unnecessarily in space and interacts with non-relevant objects. This introduces unnecessary interactions with movable obstacles that can increase the real-world failure rate of the robot due to the physics-uncertainty problem.

2. The current implementation of the system requires human input *before* planning. Although this can be beneficial in hard instances of the problem, there were many cases where the problem is easy enough to be solved by the robot *without* human guidance.

3. As a result of the previous point, guiding multiple robots in parallel results

in slightly increased planning times and lower success rates as the robots are waiting for the human more than when they are guided individually.

4. The real-world experiments demonstrated that the robot can generate a successful trajectory yet fail when executed in the real-world due to the physics-uncertainty problem. The current implementation of the system with single-query sampling-based planners requires that the robot replans from scratch, which could increase their planning times.

I address these shortcomings in the next chapter, where I look into integrating human input in a trajectory optimisation-based planning framework.

Chapter 4

Integrating Human Input in Trajectory Optimisation-based Planning with Online-Replanning

Chapter Deliverables

Demonstration video: `https://youtu.be/t3yrx-J8IRw`

Source code: `https://github.com/rpapallas/hitl-trajopt`

4.1 Introduction

In the previous chapter I proposed a framework for non-prehensile manipulation with a human-in-the-loop with sampling-based kinodynamic planning. Although this approach was effective for planning solutions in clutter there are certain drawbacks:

1. The solutions generated by the planner are sometimes long and noisy (the robot unnecessarily moves in space and interacts with non-relevant obstacles). This yields to unnecessary interactions with surrounding obstacles which could cause the trajectories to fail due to physics uncertainty.

2. Human input is required *before* planning. This requires the human to provide input for all robots, regardless if the problem is trivial or not.

3. Replanning for a new solution after execution failure requires replanning from scratch.

In this chapter, I alleviate all the above limitations by implementing a trajectory optimisation-based motion planner (like the ones I described in Section 2.1.3). I

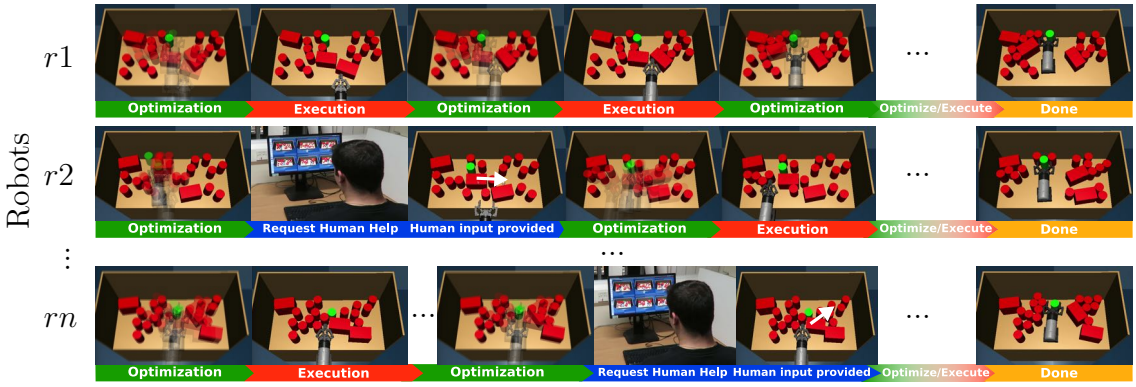


Figure 4.1: Proposed System. Each row shows a different robot working in parallel. Human input is requested only when needed (*blue* colour). Human high-level input is shown with a white arrow. Planning is shown with *green* and execution with *red* colour.

propose a trajectory optimisation-based planner with human-in-the-loop that queries the human for help only when the robot fails to plan a solution autonomously. The local search nature of trajectory optimisation is more appealing to generate smooth trajectories. Finally, trajectory optimisation lends itself more easily to online replanning, by warm-starting the optimisation with the trajectory from the previous iteration.

To illustrate the enhancements introduced by this new approach based on trajectory optimisation, consider the example in Figure 4.1, where different robots are illustrated in each row. The horizontal axis illustrates time. Each robot has the task of reaching onto a shelf to grasp an object (in green), by moving obstacles out of the way. There is one human available for guidance. Initially, in frame 1 of each row, all robots try to generate a plan, using trajectory optimisation *fully autonomously*. The robots create autonomously a straight-line trajectory towards the goal object and try to optimise over it. Unlike GRTC-HITL in Chapter 3, human engagement is not needed at this point. Some of the robots, e.g. robots $r1$ and rn, quickly generate a feasible trajectory and start autonomous execution without requiring any human help. Since there is uncertainty in how objects move, the robots perform online replanning (similar to model predictive control), where they re-optimise and execute

the trajectory at each time step. Robot $r2$, however, decides to ask for human help in frame 1, and prompts the user. In frame 2, the human engages with robot $r2$, quickly inspects the scene, uses an interface to provide high-level input (white arrow in the figure), and disengages. In frame 4, robot $r2$ tries to generate a trajectory again, this time making use of the human provided input, and then proceeds with autonomous execution using online replanning. Meanwhile, after a duration of autonomous execution, the objects in rn's environment move very differently from the planner's expectations, resulting in rn requiring human help. The human is prompted for input, and execution proceeds.

This new approach, attempts to minimise the human effort required while the planner receives high-level inputs that help it significantly. Another advantage of the system described above is the use of *online replanning*. When a robot executes a non-prehensile plan, objects in the real-world move differently to the model's predictions, which makes it necessary to update the plan. Trajectory optimisation based planning approaches are particularly effective in such settings, because a new optimisation cycle can be warm-started with the previous solution, and convergence can be achieved in few optimisation iterations.

Therefore, a key novel feature of the proposed system is the use of trajectory optimisation and performing trajectory optimisation with human input. To achieve this, I propose to make the human input part of the objective/cost function, minimized by the optimiser. This enables the human input to be easily integrated into the optimisation performed at each step of the online-replanning process.

A final novel feature of the proposed system is the *efficient use of human time*. The previous system, in Chapter 3, requires human help irrespective of whether an autonomous planner can solve the problem efficiently or not. I propose that the human should be recruited for help only if and when an autonomous planner is expected to fail or when human help is expected to speed up planning significantly. Such a system should be at least as good as the state-of-the-art fully autonomous system with the addition that, when needed, a human is in the loop to help. I propose two different approaches to realise this. The first approach will ask for human help if the planning fails to generate a plan within a fixed amount of time. The second

approach, better integrated with trajectory optimisation, will ask for human help if the optimisation gets stuck at a local minimum.

To evaluate the proposed system, I ran a number of different experiments. First, in Section 4.5.3, I evaluated two approaches of asking for human help and how they compare with two autonomous approaches. I conducted experiments with both an experienced and a novice user. In Section 4.5.4, I conducted experiments in simulation with artificial uncertainty, and on a real-robot (Figure 4.2) to check the robustness of the online replanning execution mechanism. Finally, in Section 4.5.5, I test the entire system, in simulation, with a fleet of six robots trying to solve a number of planning problems simultaneously with a single human-operator and I compare this with parallel fully autonomous planners.

The organisation of this chapter is as follows. First, in Section 4.2, I outline the assumptions, objectives and the contributions of this chapter. Then, in Section 4.3, I define the problem formulation and important notations for this chapter. In Section 4.4, I provide the implementation details of the proposed approach and in Section 4.5, I evaluate and compare this approach with a number of baselines including experiments in simulation with a novice user, experiments on a real robot and experiments in a virtual warehouse with a fleet of robots.

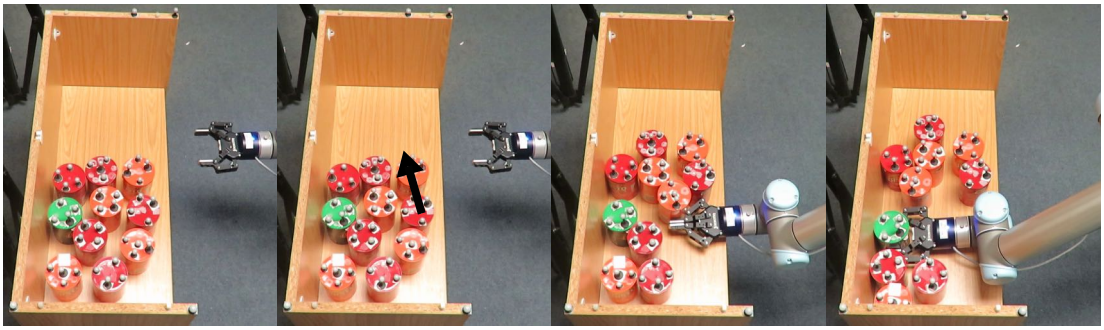


Figure 4.2: Robot tries to reach for the goal object (green). Arrow indicates human input (2nd picture). The robot executes the human provided high-level action successfully (3rd picture) and successfully reaches the goal object (4th picture).

4.2 Assumptions, Objectives and Contributions

Assumptions

In this chapter, I assume the following:

1. The robot is constrained on the plane and that the objects cannot be grasped from the top. Therefore, the robot motion is constrained in $SE(2)$.
2. The goal object is easily graspable by closing the robot fingers.
3. There is a perception system that detects object poses with good accuracy.

Objectives

The objectives of this chapter are: (1) to explore how human input can be integrated in a trajectory-optimization-based planner, (2) how human input can be better utilised to allow a human operator to guide more robots simultaneously, and (3) how the robot can handle the problem of physics uncertainty.

Contributions

The novel contributions of this chapter are thus as follows:

1. Integration of human input into an online-replanning trajectory optimisation-based motion planning for non-prehensile manipulation. This enables the system to use human help at *any point* in time during execution, not only as an initial input to the planning problem.
2. The use of an adaptive approach to decide when to ask for human input based on the problem difficulty. This allows a human operator to manage a fleet of robots at the same time for non-prehensile manipulation more effectively, offloading the human from guiding robots tackling trivial problems.
3. A robust execution method to address real-world physics uncertainty.

4.3 Problem Formulation

The environment comprises a robot r, $\mathcal{O}$ movable obstacles that the robot can interact with and other static obstacles that the robot should not interact with. We also have the goal object to reach, $o_g \in \mathcal{O}$.

The state of the robot is denoted by $x^r = (q^r, \dot{q}^r) \in \mathcal{X}^r$. $q^r \subset SE(2)$ is the robot's configuration and $\dot{q}^r$ is the robot's velocities. Similarly, I denote the state of a movable obstacle $i \in \{1, \ldots, |\mathcal{O}|\}$ with $x^i = (q^i, \dot{q}^i, \ddot{q}^i) \in \mathcal{X}^i$ where $q_i \subset SE(2)$ is the object's configuration and $\dot{q}^i$ and $\ddot{q}^i$ the object's velocities and accelerations respectively. The state space of the entire system, $\mathcal{X}^E$, is given by the Cartesian product: $\mathcal{X}^E = \mathcal{X}^r \times \mathcal{X}^1 \times \mathcal{X}^2 \times \cdots \times \mathcal{X}^{|\mathcal{O}|}$.

The robot's control space is denoted by U and is comprised of the linear and angular robot velocities denoted by $u_t \in U$ applied at time t for a fixed duration Δt. We also have a trajectory $\tau = \langle u_0, u_1, \ldots, u_{n-1} \rangle$ of n steps. I use $\tau_{[0,n-1]}$ to denote a subsequence of controls where $[0, n-1]$ is a closed interval indicating the start and end of the subsequence. For example, for the trajectory $\tau = \langle u_0, u_1, u_2, u_3 \rangle$, $\tau_{[0]}$ refers to the first element of the trajectory (u_0), while $\tau_{[1,3]}$ refers to the subsequence $\langle u_1, u_2, u_3 \rangle$.

The state of the environment at time t is given by $x_t = \{x^r, x^1, \ldots, x^{|\mathcal{O}|}\} \in \mathcal{X}^E$. The discrete time dynamics of the system are given by $x_{t+1} = f(x_t, u_t) + \zeta$ where ζ is the system stochasticity. I do not explicitly represent the system stochasticity ζ. Instead, I take an online-replanning approach, which replans a new trajectory at every step during execution using the deterministic model f.

We say that we *rollout* a trajectory τ using f from an initial state $x_0 \in \mathcal{X}^E$ using a rollout function $R(x_0, \tau)$ to obtain a sequence of states $S = \langle x_0, \ldots, x_n \rangle$. We also have a cost function $\mathcal{C}(S)$ to compute the cost of the state sequence.

Given an initial state $x_0 \in \mathcal{X}^E$ and a goal object o_g, the goal is for the robot to execute a sequence of controls to move the robot from x_0 to a state in which is grasping o_g while handling real-world physics uncertainty.

4.4 Online Replanning with Human-In-The-Loop

I use an optimisation-based approach that integrates human input to solve the problem of reaching through clutter. The system starts tackling the problems fully autonomously and decides to ask for human help only when needed. In this way, the system is capable of solving trivial problems fully autonomously without any human intervention, where possible. The proposed system integrates optimisation and execution in a unified online-replanning framework that constantly optimises and executes the solution in the real-world robustly.

In Section 4.4.1, I describe the proposed framework, OR-HITL. In Section 4.4.2, I describe a stochastic optimiser that supports the optimisation part of the OR-HITL framework. In Section 4.4.3, I define the user-input and how I capture this input. In Sections 4.4.4 and 4.4.5, I define the cost function and how I compute the initial trajectories used in the optimisation. Finally, in Sections 4.4.6 and 4.4.7, I describe two approaches to *decide* when to ask for human help.

4.4.1 Framework overview

Algorithm 4 describes OR-HITL that unifies optimisation and execution in one framework and alternates between the two to increase real-world execution robustness.

The algorithm starts with an initial trajectory τ for reaching the goal object (see Figure 4.3(a)) and a cost function $\mathcal{C}$ with the optimisation objectives (Section 4.4.4). In line 2 we observe the real-world state and if this state is a goal state, we stop and declare success (line 3). If not, then we proceed with the *optimisation* part of the framework.

Optimisation: In line 5 using the SOLVE function (Section 4.4.2) we pass to the solver the initial state x^{world}, the current trajectory τ and the current cost function, $\mathcal{C}$. The solver will optimise for some duration and then return a *result*. The result can either be "human input required" or "success". If the solver returns "success", it also updates τ with the new trajectory. Note here that the trajectory returned is a *feasible* trajectory and not necessarily the optimal one. If the solver decides that human

Algorithm 4 OR-HITL Framework

Input: τ: A trajectory
$\mathcal{C}$: The cost function

1: **function** OR-HITL
2: $x^{world} \leftarrow$ observe current real-world state
3: **if** x^{world} reached the goal **then** ***stop***
4: **do**
5: *result* $\leftarrow$ SOLVE(x^{world}, τ, $\mathcal{C}$)
6: **if** *result* is "human input required" **then**
7: *input* $\leftarrow$ obtain input from human
8: update cost function $\mathcal{C}$ based on *input*
9: update τ based on *input*
10: **while** *result* is not "success"
11: execute $\tau_{[0]}$ in real-world
12: $\tau \leftarrow \tau_{[1,n-1]}$ and expanded with u_{togoal}
13: **return** OR-HITL (τ, $\mathcal{C}$)

input is required (I describe how this decision is taken in Sections 4.4.6 and 4.4.7), then in line 7 we obtain a high-level input from a human operator (Section 4.4.3). This high-level input includes information to update the cost function (line 8) and to instantiate a new initial trajectory τ (line 9). We repeat these steps until *result* is "success" (line 10). Success here means that the trajectory is a feasible one, (i.e., its cost is below a threshold). Once the optimisation is successful, we proceed with the *execution* part of the framework.

Execution: To cope with physics uncertainty when executing a trajectory in the real-world, I propose an Online-Replanning approach. In line 11 we execute the first control of the trajectory in the real-world. We then update our trajectory τ in line 12 to be the remaining τ trajectory, expanded with a control towards the goal (u_{togoal}). This padded control at the end of the trajectory provides freedom to the trajectory to be further optimised in the future. Once we update the trajectory, we recurse in line 13 and we get the real-world state in line 2 that might be different to the one predicted by the simulator (physics uncertainty). The optimisation in line 5 *warm-starts* with the trajectory from the previous iteration, and therefore is likely to

be successful in the current iteration, requiring little or no additional work from the solver. This closed-loop execution and alternation from optimisation to execution allows the system to cope with execution uncertainty and correct the trajectory early on.

4.4.2 Stochastic optimisation

Algorithm 5 Trajectory Optimisation-based Solver

Input: x_0: The initial state of the system
τ: A trajectory
$\mathcal{C}$: The cost function
Output: "Success" with the feasible trajectory otherwise "human input required"

```
1:  function SOLVE
2:    S ← rollout τ from x_0 using R(x_0, τ)
3:    obtain the cost of rollout using C(S)
4:    while not successful do
5:      if humanHelpRequired() then
6:        return "human input required"
7:      sample k noisy trajectories from τ
8:      rollout each of the k trajectories from x_0 using R
9:      obtain cost for each rollout using C
10:     τ ← trajectory with the lowest cost
11:   return "success"
```

In Algorithm 4, in line 5, we make a call to a solver. Algorithm 5 describes this solver. The solver accepts an initial state, x_0, an initial trajectory to optimise, τ, and a cost function $\mathcal{C}$ for computing the cost of the trajectories.

We begin by rolling out the trajectory τ from the initial state x_0 (line 2) and then using the provided cost function, $\mathcal{C}$, we compute the cost of the trajectory (line 3). If the solver is successful (line 4), then we return "success" straight away (line 11). To decide if the solver is successful, we check if the robot has reached the goal and that the cost is minimized below some threshold. If the solver is not successful, we continue with the optimisation.

First we check if human help is required in line 5. We describe ways to make this decision in Sections 4.4.6 and 4.4.7. If human help is required, we return a signal that human help is required (line 6).

If human help is not required, we proceed with the optimisation. The optimisation of the initial trajectory happens between lines 7 and 10 and iterates until either a feasible trajectory is found (line 11) or if human help is required (line 5). The optimisation happens in four steps. First, we sample k noisy trajectories from τ (line 7). To create these k trajectories, we add Gaussian noise to the controls of τ using $\mathcal{N}(0, v)$ where $\mathcal{N}$ is the Gaussian distribution and v is the variance for a degree-of-freedom of the robot. We then rollout each of the k trajectories from x_0 using the rollout function R (line 8) and then obtain a cost for each of the trajectories (line 9). In line 10, we update τ to be the trajectory with the lowest cost. Therefore, in each iteration τ is updated from the k sampled trajectories. It is possible that all k trajectories have higher cost than the current τ, in this case τ does not change. We repeat these steps until the solver is successful (line 4).

4.4.3 User input

A user's high-level action suggests a particular object o_i to be pushed to particular point on the plane. I formalize this high-level action with the triple (o_i, x_i, y_i), where $o_i \in \mathcal{O}$ is an object, and (x_i, y_i) is a point of on the plane that o_i needs to be pushed to.

To capture the user's high-level action, I developed a simple user interface. The operator is presented with a window showing the environment and the robot. The operator, using a mouse pointer, provides the input by first clicking on the desired object and then a point on the plane (Algorithm 4 line 7).

Using the human input we can now define the cost function and the initial trajectory that makes use of the human input in the next sections.

4.4.4 Cost function

The cost function, $\mathcal{C}$, is used in Algorithm 5 (lines 3 and 9) but is also updated in Algorithm 4 (line 8) to integrate the human input.

No human-input provided: If no human-input is provided, then the cost function for a state sequence S is defined as $\mathcal{C}(S) = \mathcal{C}_1 + \mathcal{C}_2$. Where $\mathcal{C}_1$ is the cost for reaching the goal object:

$$\mathcal{C}_1 = w_g \cdot \mathrm{d}(\mathbf{q^{ee}}, \mathbf{q^g}) \tag{4.1}$$

$\mathcal{C}_1$ is the weighted Euclidean distance from the robot's end-effector to the goal object, o_g. $\mathcal{C}_2$ defines three cost terms with their corresponding weights:

$$\mathcal{C}_2 = \sum_{i=1}^{n} c_e(x_i) + c_f(x_i) + c_s(x_i) \tag{4.2}$$

- $c_f(x_i) = \sum_{j=1}^{|\mathcal{O}|} w_f \cdot F(x_i^j)$: For a state x_i I penalize any movable object that applies high forces to any other movable or static obstacle. F is a function that given a state of an object x_i^j returns the contact forces of that object.

- $c_e(x_i) = \sum_{j=1}^{|\mathcal{O}|} w_e \cdot \mathbb{1}_e(q_i^j)$: For a state x_i I penalize any movable object $o_j \in \mathcal{O}$ that is geometrically outside the configuration space. $\mathbb{1}_e$ is an indicator function that returns 0 if the objects is geometrically inside the configuration space, 1 otherwise.

- $c_s(x_i) = w_s \cdot \mathbb{1}_s(x^r)$: For a state x_i I penalize the robot for colliding with a static obstacle. $\mathbb{1}_s$ is an indicator function that returns 1 if the robot, r, collides with any of the static obstacles or 0 otherwise.

Human-input provided: If human-input is provided (Algorithm 4, line 7) the cost function is updated (Algorithm 4, line 8). This update is two-fold, we first push that object to the human indicated position using $\mathcal{C}(S) = \mathcal{C}_3 + \mathcal{C}_2$, and then we reach for the goal object using $\mathcal{C}(S) = \mathcal{C}_1 + \mathcal{C}_2$.

$$\mathcal{C}_3 = w_p \cdot \mathrm{d}(\mathbf{q_n^i}, \mathbf{q_{desired}^i}) \tag{4.3}$$

For a high-level input (o_i, x_i, y_i), $\mathcal{C}_3$ is the weighted Euclidean distance of o_i position at the final state, q_n^i, with the user's provided position, $q_{desired}^i$, of that object. This cost term in the optimisation will encourage the solver to explore solutions where the object indicated by the human is pushed towards the desired position.

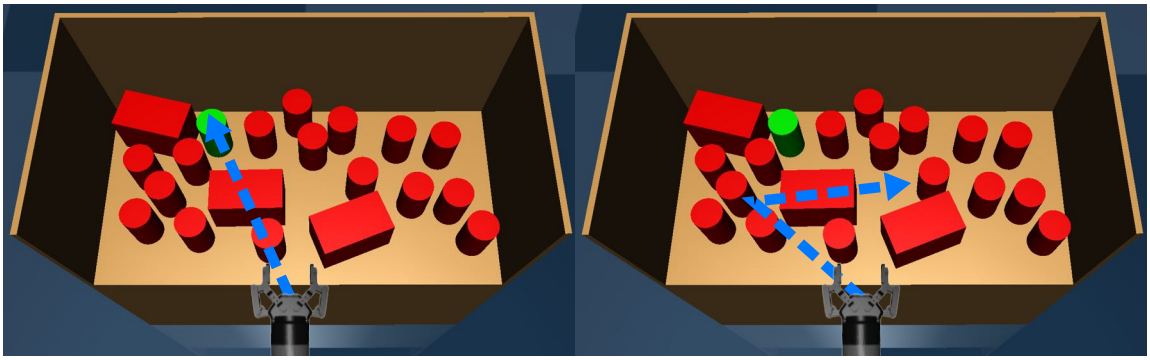
$\mathcal{C}_1$ essentially encourages trajectories to reach the desired object while $\mathcal{C}_2$ encourages trajectories to avoid pushing forcefully objects, throwing objects on the floor, and the robot from colliding with static obstacles. When it comes to pushing objects, $\mathcal{C}_3$ ensures that the high-level action is respected (object is pushed to the desired location). These are some of the problems that could occur when a robot is moving in such environments and the cost functions ensure that a feasible trajectory respects these constraints. The weights require tuning and I present their exact values I used in the experiments in the results section.

4.4.5 Initial trajectories

When we start the optimisation, we need to provide the solver with an initial trajectory. I use straight-line trajectories because they are cheap to compute (no physics simulation).

Two such initial trajectories are depicted in Figure 4.3. The first trajectory, Figure 4.3(a), is the initial trajectory for reaching the goal object and is a straight line trajectory from the robot's current position, q^r, to the position of the goal object, q^g. The second trajectory, Figure 4.3(b), is the initial trajectory for pushing an object to its desired position. This trajectory is following a straight line passing from the object's current position and ending at the object's desired position.

Next, in Sections 4.4.6 and 4.4.7 I describe two ways to decide when to ask for human input. This decision is taken in Algorithm 5 line 5 and I describe two different approaches to take this decision.



(a) Reaching goal object (b) Pushing an object

Figure 4.3: Initial trajectories. The arrow illustrates the trajectory. In (b) the object to be pushed is the box the arrow penetrates.

4.4.6 Asking for human help with a fixed timeout

A straightforward way to decide if human input is required, is based on a fixed timeout limit. The solver tries to find a solution for some fixed time limit and if a solution is not found it will time-out and request human input.

I denote this time-limit with "Fixed z" where $z \in \mathbb{Z}^+$. Therefore, in Algorithm 5 in line 5 when using Fixed OR-HITL, `humanHelpRequired` will return true if z seconds are reached. For example, for Fixed 20, `humanHelpRequired` will return true every 20 seconds if the solver cannot find a solution.

Although this is a simple and straightforward approach, it can be problematic, in some cases. For example, if the solver is able to solve a problem in 25 seconds but the Fixed Time limit is set to 20 seconds, then the solver will ask for human help. Similarly, for a hard problem, giving more time to time-out could make the system waste time unnecessarily before asking for human help.

This shortcoming of the Fixed Timeout inspired an adaptive approach that decides when to ask for human help depending on the problem at hand.

4.4.7 Adaptively asking for human-help

One property of trajectory optimisation is that the convergence rate of the cost of a problem from iteration to iteration can indicate whether the solver can explore new solutions (i.e., more time is needed) or if the solver is stuck at a local minimum (i.e., immediate human input could be beneficial). I leverage this property to adaptively decide when to ask for human help based on the problem at hand. If at some point during the optimisation we find that the solver hits a local minimum, then we send a signal that human input is required.

To decide if the solver is stuck at a local minimum, we look at the absolute difference between the previous iteration's cost, $c_{previous}$, and at the current iteration's cost, $c_{current}$. If this difference is lower than a threshold for a number of consecutive iterations, then we say that we are stuck at a local minimum[1]. Since this is a stochastic optimisation, we need to check this for some consecutive iterations to conclude that we are stuck at a local minimum because it is likely that in an iteration the cost does not improve, but this is not an indication that we are stuck at a local minimum.

Therefore, in Algorithm 5 in line 5 when using Adaptive OR-HITL, `humanHelpRequired` will return true if we hit a local minimum for some consecutive iterations. After experimentation with different values, in my implementation, we stop if local minimum is found for two consecutive iterations.

4.5 Experiments & Results

In this section I evaluate the proposed algorithm against several baselines both in simulation and in the real-world. Some of these experiments are shown in the accompanying video[2]. I employed two operators in these experiments. First, I conducted all the human experiments throughout this section (indicated with "Experienced User") except the experiments marked with "Novice" where a novice operator was employed instead. The novice had no prior experience with robotic systems or robotics research.

[1] This threshold is related to the cost function definition and was experimentally chosen. Changing the cost function definition might require adjustment of this threshold as well.

[2] Also available at `https://youtu.be/t3yrx-J8IRw`.

Next, I highlight the working hypotheses.

4.5.1 Hypotheses

The hypotheses I am evaluating in this section are:

1. The proposed approach is effective for the problem of physics-based non-prehensile manipulation compared to state-of-the-art fully autonomous trajectory-optimization-based motion planners.

2. The adaptiveness of the proposed algorithm is effective compared to a fixed timeout approach.

3. The online-replanning strategy is robust to physics-uncertainty compared to an open-loop execution strategy.

4. The adaptiveness allows more effectively a human to guide a number of robots simultaneously.

The first hypothesis states that the proposed system based on trajectory-optimization is more effective than the baselines. The measurement here is the success rate and planning times of the proposed approach and the baselines in various random RTC problems. The second hypothesis focuses on the adaptiveness of OR-HITL. This adaptiveness allows the robot to query human input when it is stuck in local minima. The working hypothesis is that this will be, in general, better than having a fixed timeout parameter to decide when to ask for human help. This hypothesis can be evaluated by running experiments with the proposed approach and with various fixed-timeout variants. The measurement is the success rate, planning times and human time in various RTC problems. Both hypotheses are evaluated in Section 4.5.3.

The third hypothesis focuses on the problem of physics-uncertainty. The proposed algorithm employs a novel Model Predictive Control strategy to tackle this problem, and the hypothesis is that this control strategy will be more successful when executing the solution in the real-world. This can be evaluated by running the algorithm against an open-loop variant in the presence of physics-uncertainty (running experiments in

the real-world or in an uncertain physics simulator). The measurement is the success rate in various RTC problems. This hypothesis is evaluated in Section 4.5.4.

Finally, the fourth hypothesis suggests that the adaptiveness of the OR-HITL, since it will allow the robot to solve trivial problems autonomously, will allow a single human operator to guide multiple robots simultaneously. The evaluation can be done by running a number of robots in parallel solving random RTC problems while a single-human is supervising them, compared to running those robots alone fully autonomously. The measurement will be the success rate and planning times. The experiments for this hypothesis are shown in Section 4.5.5.

4.5.2 Experimental setup

For all the experiments I used MuJoCo[3] [134] to implement the system dynamics. I used a randomizer to generate random simulation scenes. The randomizer placed the goal object first at the back of the shelf and then the remaining objects in collision-free positions within a radius of 30cm around the goal object. The total time limit for every experiment was 180 seconds, after which the robot was stopped and the run was marked a failure. This time-limit includes combined optimisation times as well as human interaction time.

Optimisation Parameters: The optimisation parameters were: $k = 15$ noisy trajectories at each iteration, variance of 0.04 m/s for the robot's linear velocities and $0.04 rad/\text{s}$ for the robot's angular velocity. The trajectories are 3-seconds long with 8 steps each. Note that each action will result to a new trajectory; therefore each trajectory is 3-seconds long, but solving a problem with 3 high-level actions will result to 3 consecutive trajectories. To rollout a 3-second trajectory with an integration step size of 0.0015[4] it took on average 1 second on my computer. To execute a 3-second trajectory in simulation took around 3 seconds. The cost function's parameters are: $w_g = 2000$, $w_f = 50$, $w_e = 300$ and $w_s = 300$. Finally, the success threshold is 70.0

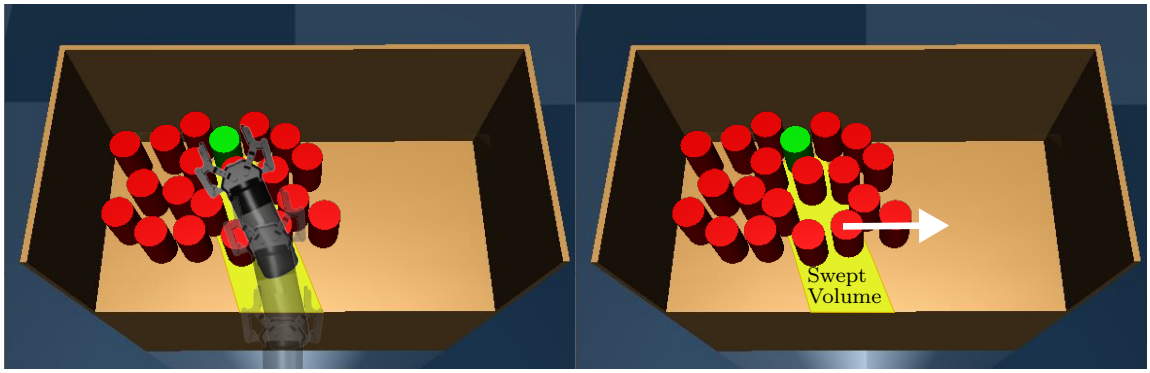
[3]On a computer with Intel Core i7-4790 CPU @ 3.60GHz, 16GB RAM.

[4]The integration step size was chosen experimentally and is the largest step size that does not cause unstable simulations.

(Algorithm 5, line 4)[5].

Robot: I use a 2-finger Robotiq gripper on a UR5 robot mounted on a Clearpath Ridgeback. We controlled the hand in simulation (3 Degrees of Freedom (DOF): 2 linear and 1 angular). The gripper has 1 DOF but we do not consider it in the optimisation, instead we close the gripper at the beginning of the optimisation and we open it before the last action of the trajectory. When executing the solutions in the real-world we mapped the gripper velocities to Ridgeback velocities.

4.5.3 Framework evaluation



(a) (b)

Figure 4.4: Heuristic: automatic approach to obtain high-level action. The robot first moves straight to the goal to find the first blocking obstacle. It captures the swept volume of the robot (yellow area). It then finds a collision-free position for the blocking obstacle outside the swept volume. White arrow is the suggested high-level plan obtained.

I compare Fixed 5, Fixed 20 (introduced inSection 4.4.6) and Adaptive (introduced in Section 4.4.7) with an experienced and a novice user with two autonomous planners. The novice user had no prior knowledge of the system or the problem. The novice user was trained for around 45 minutes.

[5]The weights and the threshold were chosen experimentally.

I generated 30 scenes for *each* planner. I generated a different set of problems for each planner where a human was involved to avoid the chance where a pattern of the problem is memorized by the users. For the Autonomous and the Heuristic planners, since there is no learning, I evaluate them over all the scenes.

Table 4.1: Framework evaluation. Comparing the performance of the proposed approach against two fully autonomous baselines ("Autonomous" and "Adaptive Heuristic") and two variants of the proposed approach ("Fixed 5" and "Fixed 20"). The results present performance of an experience user (the author of this thesis) and of a novice user (no prior experience). The results show that the proposed approach is, in general, better both in success rate and total time (planning, guidance and execution time). Error indicates the 95% CI.

	Success (%)	Planning Time (s)	Human Time (s)	Total Time (s)
Experienced User				
Fixed 5	90.0	38.1 ± 15.6	9.6 ± 4.1	68.8
Fixed 20	93.3	44.2 ± 13.8	7.0 ± 1.8	63.2
Adaptive	**96.6**	31.0 ± 12.8	$\mathbf{2.5 \pm 0.9}$	**42.5**
Novice User				
Fixed 5	86.6	$\mathbf{27.8 \pm 11.7}$	19.8 ± 8.8	62.6
Adaptive	90.0	33.6 ± 16.5	5.5 ± 1.0	45.1
No Human				
Autonomous	74.6	79.8 ± 11.2	–	82.8
Adaptive Heuristic	82.5	86.4 ± 13.5	–	98.4

I compare two autonomous planners here. First, "Autonomous" is a planner that uses the solver highlighted in Algorithm 5 but aims to reach for the goal object without considering any high-level plan. I also implemented a straight-line heuristic to replace the human from the OR-HITL framework. The robot moves on a straight line to the goal object to find the first blocking obstacle and to capture the robot's swept volume. It then finds a collision-free position outside the swept volume for this obstacle. The object and the new position is then returned to the framework (i.e., substituting the human input entirely) to plan and push the object to the new

position. I illustrate this straight-line heuristic in Figure 4.4. I call this planner "Adaptive Heuristic".

Table 4.1 presents the simulation results. Planning time is the average planning time per problem. Human time is the average time spent by a user providing guidance. Total time includes the planning time, human time (if applicable) and execution time, providing an overall average time to solve a problem. The results show that the planners with a human-in-the-loop were more successful than the autonomous planners and they had dramatically improved the planning times. Adaptive requested less human intervention with an average of 2.5 seconds of human time per problem. Fixed 5 requested human intervention more frequently and shows that human engagement with this approach is considerably high. The novice user confirmed that the interaction with the Fixed 5 was more tedious as he was prompted too often and he found it challenging to provide high-level input using the Fixed 5 because the 5 seconds timeout forced him to provide high-level actions that are easy optimisation problems. The novice participant found Adaptive more intriguing and comfortable to use and he enjoyed the fact that the robot managed to solve more problems on its own.

4.5.4 Handling uncertainty

The framework is designed to handle physics uncertainty during execution. To evaluate OR-HITL and its Online-Replanning control strategy, I run two experiments in two uncertain environments. First, I evaluated the system in *simulation* with artificial uncertainty added to the objects' motion, during execution. This creates an uncertain environment, similar to the real-world where execution of trajectories could fail due to physics-uncertainty. Second, to test the system in a realistically uncertain environment, I also evaluated the system with a *real robot* manipulating objects on a real shelf. The physics uncertainty in these two environments is handled by OR-HITL using Online-Replanning and I compare against an open-loop planner.

Simulation: I configured the simulation environment in the following way. At each *simulation step* (during execution, not during optimisation) we observe the velocities

Table 4.2: Simulation results with physics uncertainty. Experiments conducted with artificial noise added to the position of objects to simulate uncertainty and test the robustness of the online-replanning strategy proposed. "Adaptive" is the proposed approach with a human-in-the-loop and online-replanning, "Autonomous" is a baseline with online-replanning but without human input, "OL Autonomous" is a baseline *without* online-replanning and without human input. The results suggest that the online-replanning strategy is robust to physics-uncertainty. Errors indicate 95% CI.

	Adaptive	Autonomous	OL Autonomous
Success	58 / 60	35 / 60	23 / 60
Optimisation Failures	1 / 60	16 / 60	23 / 60
Execution Failures	1 / 60	9 / 60	14 / 60
Optimisation Time (s)	62.2 ± 10.5	118.2 ± 17.2	121.0 ± 18.1
Replanning Iterations	2.8 ± 1.2	4.2 ± 3.0	11.9 ± 3.1
Human Time (s)	5.4 ± 0.8	-	-
Total Time (s)	79.6	124.2	124.0

of all the movable objects and if they are beyond some threshold[6] we add a small Gaussian noise to their velocities ($\mathcal{N}(0, 0.02)$ where 0.02 is m/s). I compare Adaptive, Autonomous and I also implemented an Open-Loop Autonomous (Open-Loop (OL) Autonomous) planner as a baseline. OL Autonomous replans only at the end of the trajectory execution instead of the end of each action. Table 4.2 presents the results. Execution Failures indicate the number of times the planner found an initial solution but failed to execute and replan. Replanning Iterations indicate the number of optimisation iterations required to successfully replan a trajectory that failed during execution. Total time is the average time to complete the task (planning time, human time and execution time). It is clear that Adaptive performed better than Autonomous and OL Autonomous in success rate and planning time. Success rate for Adaptive was 97%, 58% for Autonomous and 38% for OL Autonomous. During replanning, Adaptive and Autonomous required on average 3 and 4 iterations respectively to correct the trajectory while the OL Autonomous required on average

[6]To avoid adding uncertainty to objects that have not moved since the previous step.

12 iterations.

Real-world: In a real-world setting I evaluated Adaptive and Open-Loop Adaptive (OL Adaptive). OL Adaptive replans at the end of the trajectory instead of the end of each action. I performed 30 real-world experiments, 15 for each planner in 15 different scenes. The robot was asked to reach for the green object in a small shelf among other 9 obstacles. Some of the scenes are depicted in Figure 4.5. To avoid damage of the physical robot or of the objects in the environment, I stopped the robot if it collided with any static obstacle or forcefully pushed an object against the shelf and I declared the attempt as an Execution Failure due to violation of these safety rules. There are some demonstrations of these experiments in the accompanying video.[7] I present the results in Table 4.3. The success rate for Adaptive is 86% while for the OL Adaptive is 53%. Adaptive failed once during planning and once during execution (due to safety rule violation). OL Adaptive failed once during planning and six times during execution. The main cause of failures for the OL Adaptive planner was that physics uncertainty caused the robot to violate some of the safety rules. The OL Adaptive, since there was no replanning at each step, was in general faster at executing trajectories, but when it failed it required more replanning time. Adaptive on the other hand, replanned for some easy instances of the problem, but the replanning was always very short because of the warm-starting.

Figure 4.5: Real-world scenes

[7] "Adaptive" execution is slower than "OL Adaptive" due to delays with the perception system called after each action execution.

Table 4.3: Real-world results: success rate of proposed approach (Adaptive) compared to open-loop baseline (OL Adaptive). Adaptive uses online-replanning while OL Adaptive does not use online-replanning. "Optimisation Failures" indicate the planning failures, while "Execution Failures" indicate failures during execution. The results suggest that Adaptive is robust to real-world physics uncertainty, with one execution failure compared to six execution failures of the open loop planner.

	Adaptive	OL Adaptive
Success	13 / 15	8 / 15
Optimisation Failures	1 / 15	1 / 15
Execution Failures	1 / 15	6 / 15

4.5.5 Warehouse problem

I consider a scenario where there are 50 orders to pick objects from cluttered shelves in a simulated warehouse. The warehouse operates six robots at the same time. The objective is to increase the success rate of these robots and fulfil the 50 orders as quickly as possible.

In simulation, I compare six autonomous robots (Autonomous) working in parallel trying to fulfil these 50 orders, with the proposed system with a *single* human-operator (Adaptive) guiding these six robots simultaneously. Some of these scenes are shown in Figure 4.1. Each robot attempts to solve the assigned problem within 180 seconds. Once a robot finishes with a problem it is assigned with the next available one. I conducted this experiment on a more powerful computer[8] as 6 simultaneous instances of the physics simulator requires extensive CPU power and memory usage.

The results (Table 4.4) show that the proposed approach was more successful (74% success rate compared to 32% for the Autonomous) and faster in planning solutions per problem, almost a minute faster (95 seconds on average compared to 150 seconds for the baseline). Human time (the time the human spent on each robot) was on average under six seconds. Finally, an important metric for performance is the total time. The total time includes optimisation time, human time if applicable (i.e., only

[8]Intel Xeon E5-2650 v2 CPU @ 2.60GHz, 64GB RAM.

Table 4.4: Warehouse problem: Results for the warehouse problem. Six robots operating in parallel in a virtual warehouse picking objects with a single human operator. The proposed approach (Adaptive) performs better than the baseline (Autonomous) both in success rate and total time. Errors indicate 95% CI.

	Adaptive	Autonomous
Success	37 / 50	16 / 50
Failures	13 / 50	34 / 50
Optimisation Time (s)	94.7 ± 15.1	149.7 ± 15.9
Human Time (s)	5.5 ± 1.0	-
Total Time (s)	112.2	152.7

for the proposed approach), and execution time. It includes the planning time for both the successes and the failures (which is bounded to 180 seconds). This number gives the actual time it takes for the robot to complete a task on average. On average, Adaptive considered four actions per problem (for example, three pushing actions and one reaching action). Since each action is executed individually, there are 3 seconds of execution time per action, which adds in total 12 seconds of execution time per problem on average. On the other side, since Autonomous is planning straight to the goal, there is only one trajectory and hence there are only 3 seconds of execution time per problem added to the total time.

4.6 Conclusions

In this chaper, I proposed OR-HITL, an online-replanning framework with Human-In-The-Loop based on trajectory optimisation. My approach starts solving the problem fully autonomously and decides to ask for human input only when the problem is estimated to be too difficult to be solved autonomously. My system uses an adaptive approach (Adaptive) to take this decision based on the problem difficulty. I demonstrated that this adaptiveness is useful in a simulated warehouse setting where a single human operator manages a fleet of robots at the same time. Finally, my framework showed increased robustness in real-world execution due to

the online-replanning strategy I implemented in the framework.

In the next chapter, I motivate an extension of OR-HITL that permits robots to predict the human contribution and bid for human time. This way, human time can be distributed to robots that seek urgent guidance more effectively.

Chapter 5

Learning When to Ask for Human Help

Chapter Deliverables

Source code: `https://github.com/rpapallas/predictive-guided-framework`

5.1 Introduction

In the previous chapter, I introduced OR-HITL, a trajectory optimisation-based motion planning framework with an adaptive approach to decide when to ask for human help. The adaptive approach allows the system to solve trivial problems without human intervention. This was an improvement over the previous framework (the sampling-based approach in Chapter 3), that required human help before planning. OR-HITL also demonstrated fast planning times and robust execution due to the online-replanning strategy employed. Despite the good performance of OR-HITL, the framework revealed the following weaknesses:

- **Human is queried relatively late:** Currently, OR-HITL, will query the human for help only when the robot hits local minima. It can take several iterations for the robot to realise this (our experiments show 25 iterations on average), and therefore waste planning time unnecessarily. This chapter looks into ways to identify such scenes and query the human earlier.
- **Human is overburden with robot requests:** The current implementation of OR-HITL displays multiple robots that require human help. This can be disturbing for a human, especially when the number of robots scales from six to twenty. This chapter looks into a way to prioritise robots' requests for human

help and display only one robot at a time to the human; the one which needs the help the most.

My observation while using OR-HITL to solve several RTC problems is that looking at certain RTC instances, you can tell if the robot will request human help. For instance, consider Figure 5.1. For a human who knows how the robot will approach the problem (optimizing on a straight line as shown in the figure) and with few minutes operating the system, can tell that the robot will be challenged with the problem on the left, while the problem on the right can most likely be solved fully autonomously. This is due to our intuition of how the objects on the easy problem will move while the robot is approaching the goal object (they will more likely not block the robot's way and they will be pushed out of the way indirectly). In this chapter I am asking the question of whether a system can *learn* to realise the same and query the human for help earlier, before it actually hits local minima.

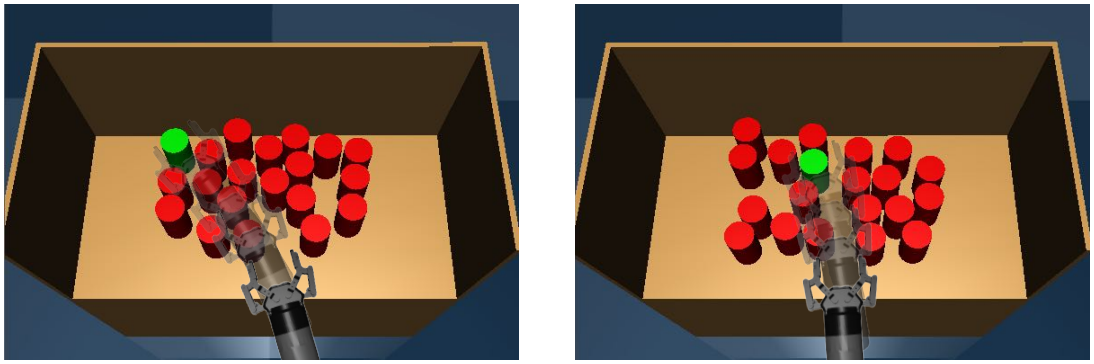


Figure 5.1: Two different instances of an RTC problem. The problem on the left is challenging, and the robot is very likely to ask for help, while the problem on the right is less challenging and the robot might be able to solve it fully autonomously.

This chapter, introduces a *predictive* framework (an extension to OR-HITL) that, given the state of the problem at a certain point in time, predicts how the optimisation will evolve in the future, thereby allowing the system to query the human before the robot hits local minima. Figure 5.2 depicts the high-level idea of this predictive framework. There are n robots each solving a reaching through clutter task. They all use a variant of OR-HITL (more details in Section 5.4) to plan and execute a

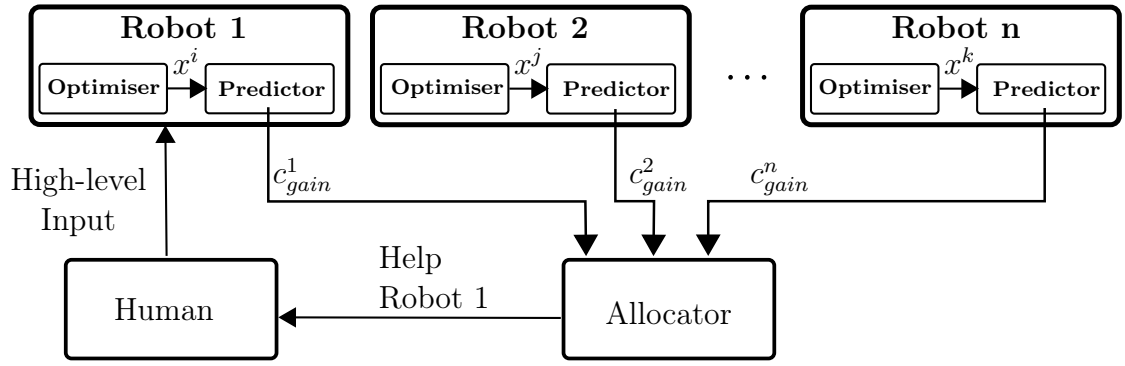


Figure 5.2: The proposed framework. There is a human operator available to guide n robots. Each of the robots, using trajectory optimisation, tries to optimise and find a feasible trajectory fully autonomously. At each iteration, each robot predicts a human gain (c_{gain}) for some l iterations in the future. This human gain indicates how impactful human input will be in the current scene. High gain means that the robot will be able to solve the problem only if human intervenes, while low gain means that the robot might benefit from human input but might also be able to solve the problem autonomously. The "Allocator", observes the predicted human gains from all the robots and assigns the human to the robot with the highest expected gain.

solution and, therefore, they optimise over an initial trajectory. Each robot, at each optimisation iteration, using the "predictor", predicts their cost *with* and *without* human input and calculates a *predicted human gain*. This human gain essentially tells how impactful the human input can be in the current context as opposed to planning fully autonomously. If the predictor anticipates that autonomously the robot will be in local minima but human input is likely to help the robot to solve the problem, the predicted human gain will be high. Robots which anticipate a positive human gain, submit their predicted human gain to the "Allocator". The allocator then considers all the requests and assigns the robot with the highest predicted human gain to the human. As a result, robots which anticipate greater human gain have higher priority over other robots while robots which anticipate that the human will not be able to help or are not of high urgency, continue fully autonomously.

The system I propose here, learns the human gain from past experiences when a human interacted with the system, providing high-level inputs. The system essentially

learns to predict (1) how the optimisation will evolve in the future *without* human input and (2) how the optimisation will evolve in the future *with* human input. Knowing these two values, a robot can predict, with a certain error, the human gain at each iteration and allows the robot to request the human early on, before unnecessary time passes. Having a value representing the human gain is appealing in the setting where multiple robots will be bidding for human time as well.

Next in, Section 5.2, I outline the assumptions, objectives and contributions of the chapter. In Section 5.3, I define the problem formulation. In Section 5.4, I describe the proposed framework, PGF. Finally, in Section 5.6, I present an evaluation of the system both in terms of the prediction performance and with the overall system working in a virtual warehouse with a number of robots and a single human operator guiding them at the same time.

5.2 Assumptions, Objectives and Contributions

In this chapter, I assume the following:

1. The robot is constrained on the plane and that the objects cannot be grasped from the top. Therefore, the robot motion is constrained in $SE(2)$.
2. The goal object is easily graspable by closing the robot fingers.
3. There is a perception system that detects object poses with good accuracy.
4. Scenes with similar object arrangements will have similar cost.

The objective of this chapter is to explore the idea of learning when to ask for human help. In a setting where a human supervisor is required to supervise a large number of robots, choosing the right robot to guide at the right time might be beneficial. The aim of this chapter is to present a possible implementation of such predictive system.

The contribution of this chapter is, therefore, a predictive approach that allows a robot to predict the cost of a future iteration and the integration of this predictive

approach in a motion planning algorithm for the problem of Reaching Through Clutter with a human-in-the-loop for physics-based non-prehensile manipulation.

5.3 Problem Formulation

We consider the kinodynamic problem of Reaching Through Clutter (RTC) introduced in Section 3.3 and the trajectory optimisation formulation and notations introduced in Section 4.3.

We define a prediction model ϕ_{human} that, given the state of the environment of a robot, returns a predicted trajectory optimisation cost l iterations in the future assuming that *human intervened*. Similarly, we define a prediction model $\phi_{autonomous}$, that given the state of the environment of a robot, returns a predicted trajectory optimisation cost l iterations in the future assuming that *human did not intervened*. We define, therefore, c_{gain} as the predicted human gain l iterations in the future.

The problem we are trying to solve is, given the environment, a human operator and n parallel robots, to select the robot that is more likely to require human guidance in the future and is the one that will benefit the most.

5.4 Predictive Guided Framework (PGF)

In this section, I introduce the Predictive Guided Framework (PGF). The overall framework is described in Algorithm 6. PGF is an extension to OR-HITL framework introduced in Section 4.4.1 (Algorithm 4). Here, I outline the changes and key differences between the two. The remaining details (for example how the initial trajectories are created, how human input is integrated in the optimisation etc) can be found in Section 4.4. The only difference in OR-HITL and PGF is the call to the PGF-SOLVE in line 5 instead of SOLVE. This new solver integrates a predictive model to decide when to ask for human help (so the human is not queried in each cycle) and assigns the human to the robot with the highest anticipated gain. The PGF-SOLVE is the one that integrates the predictions and will be discussed next.

Algorithm 6 Predictive Guided Framework

Input: τ: A trajectory
$\mathcal{C}$: The cost function

1: **function** PGF
2: $x^{world} \leftarrow$ observe current real-world state
3: **if** x^{world} reached the goal **then** ***stop***
4: **do**
5: *result* $\leftarrow$ PGF-SOLVE(x^{world}, τ, $\mathcal{C}$)
6: **if** *result* is "human input required" **then**
7: *input* $\leftarrow$ obtain input from human
8: update cost function $\mathcal{C}$ based on *input*
9: update τ based on *input*
10: **while** *result* is not "success"
11: execute $\tau_{[0]}$ in real-world
12: $\tau \leftarrow \tau_{[1,n-1]}$ and expanded with u_{togoal}
13: **return** PGF(τ, $\mathcal{C}$)

5.4.1 Predictive solver

The PGF-SOLVE solver is described in Algorithm 7. It is also an extension to OR-HITL's solver (first introduced in Algorithm 5). The main difference is between the lines 5 and 7. Initially, the planner rolls out the initial trajectory to obtain a sequence of states (line 2). It then obtains the cost of the rollout (line 3). The solver then proceeds until it is successful with the following process. In line 5 we use the current state of the system, x_0, to predict the human gain c_{gain}. The predicted gain is then given to the HUMAN-ALLOCATOR function (line 6) which decides if the robot will get human help. If the allocator decides that the current robot should be assigned the human (line 7) then the solver stops and returns "human input required". Human is prompted for input (Algorithm 6, line 7) and the input is integrated in the optimiser (Algorithm 6, lines 8 and 9). If human input is not required, then we sample k noisy trajectories from τ (line 9) we rollout each of the k trajectories (line 10) and we obtain the cost for each (line 11) and keep the trajectory with the lowest cost (line 12).

The solver also uses a cost function, C. The cost function is similar to the one

Algorithm 7 Predictive Solver

Input: x_0: The initial state of the system
τ: A trajectory
$\mathcal{C}$: The cost function
Output: "Success" with a feasible trajectory otherwise "human input required"

```
1: function PGF-SOLVE
2:   S ← rollout τ from x_0 using R(x_0, τ)
3:   obtain the cost of rollout using C(S)
4:   while not successful do
5:     c_gain ← PREDICT(x_0)
6:     humanAssigned ← HUMAN-ALLOCATOR(r, c_gain)
7:     if humanAssigned then
8:       return "human input required"
9:     sample k noisy trajectories from τ
10:    rollout each of the k trajectories from x_0 using R
11:    obtain cost for each rollout using C
12:    τ ← trajectory with the lowest cost
13:  return "success"
```

introduced in the previous chapter with slight modifications (the inclusion of c_b instead of the cost for the static boundaries). The cost terms of the cost function are listed below:

- $c_d = w_d \cdot \mathrm{d}(\mathbf{q^{ee}}, \mathbf{q^g})$: The Euclidean distance from the robot's end-effector to the goal object
- $c_f(x_i) = \sum_{j=1}^{|\mathcal{O}|} w_f \cdot F(x_i^j)$: For a state x_i we penalize any movable object that applies high forces to any other movable or static obstacle. F is a function that given a state of an object x_i^j returns the contact forces of that object.
- $c_b(x_i) = \sum_{j=1}^{|\mathcal{O}|-1} w_b \cdot \mathbb{1}_b(\mathbf{x_n}, \mathbf{q^{ee}}, \mathbf{q^i})$: For each obstacle we check if there is a blocking obstacle in the robot's end-effector in the last state of the trajectory (x_n). $\mathbb{1}_b$ is an indicator function that returns 1 if the robot, has an obstacle (other than the goal) in the hand, 0 otherwise.

The first cost term is defining how close the robot is to grasp the desired object and

is therefore indicating if the robot is grasping the object at the end of the trajectory. To avoid trajectories that forcefully pushes objects against static obstacles, we use the second term. The third term of the cost function encourages the planner to find trajectories that do not get blocking obstacles in the robot's hand. This cost function was experimentally chosen and provided a good cost function for the problem of Reaching Through Clutter.

When human input is provided, the cost function and initial trajectories change accordingly (as already discussed in Section 4.4.4).

5.4.2 Allocator

Algorithm 8 Allocator

Input: r_i: ith robot
c_{gain}: Anticipated human gain for robot i
Output: True if robot should be supervised, false otherwise

1: **function** HUMAN-ALLOCATOR
2: **if** window not created **then**
3: create window for t_{window} seconds
4: $R_{pool} \leftarrow R_{pool} \cup \{r_i, c_{gain}\}$
5: wait t_{window} seconds for other robots to join
6: **return** true if r_i is the robot with highest c_{gain} in R_{pool}, false otherwise

In Algorithm 7 in line 6, we make a call to HUMAN-ALLOCATOR. This function is described in Algorithm 8. The allocator creates timed windows where robots join a pool of robots and place a bid, their expected human gain. If a window is not currently active, the system creates a window for t_{window} seconds (line 3). Robots join the active window and they are added in the robot pool R_{pool} which consists of the robots and their predicted human gain (line 4). The robot waits for t_{window} seconds (line 5) for other robots to join the bidding. After the time elapses, the robot with the highest predicted gain gets the human while all other robots in the pool are refused access to the human (line 6).

So far, I introduced the PGF framework, the modified solver that employees a

predictive approach to decide when to ask for human help and the allocator that uses the predicted values to allocate the appropriate robot to the human. The only missing component of PGF is the predictors. In the next section, I discussed the predictors and how I have implemented them for the problem of RTC.

5.5 Predictions

In this section, I describe how the predictions are made. First, in the next section I describe the PREDICT algorithm that makes the predictions in Algorithm 7 (Section 5.4.1). In Section 5.5.2, I describe the structure of the dataset that I collected to train the models. Finally, in Section 5.5.3, I describe the network architecture.

5.5.1 The prediction algorithm

Algorithm 9 Predictor

Input: $x_{current}$: Current state of the system
Output: The anticipated human gain

1: **function** PREDICT
2: $c_{human} \leftarrow \phi_{human}(x_{current})$
3: $c_{autonomous} \leftarrow \phi_{autonomous}(x_{current})$
4: $c_{gain} \leftarrow c_{autonomous} - c_{human}$
5: **return** c_{gain}

In Algorithm 7 in line 5, we make a call to a PREDICT function. This function is described in Algorithm 9. Using two prediction models ϕ_{human} and $\phi_{autonomous}$, we predict two future costs. The first cost (line 2) is the prediction of the cost terms of the optimisation l iterations in the future, assuming human input. The second cost (line 3) is the prediction of the cost terms of the optimisation l iterations in the future, assuming no human input. The algorithm then calculates c_{gain} (line 4) as the anticipated gain the robot will get in the future if human input is given. In line 5 the predicted gain is returned to Algorithm 7 and line 5.

The predictors use the current state to inform their decision. However, in the problem of RTC that I consider here, the state of the environment is of high dimensionality (configuration of movable objects) which is challenging to learn. Instead, I use a projection of the state space as an input to the predictors which does not include the configuration of the movable objects. The projection is the number of movable objects in the scene, n_{obj} and the cost terms (c_d, c_f, c_b) in the *last p iterations*. I found that using the cost term from the last iteration was not enough to represent the complexity of a problem and instead I found more successful to look at a history of the cost breakdown.

5.5.2 Structure of the datasets

I generated two different set of data. One dataset for training $\phi_{autonomous}$ (the autonomous dataset) and another dataset for training ϕ_{human} (the human dataset).

The dataset for $\phi_{autonomous}$ consists of the optimisation cost breakdown per scene. I generated a number of distinct scenes (different rearrangements of object positions on the shelf) and ran the autonomous optimiser to solve them and collect data. For *each* of these scenes we have the following: $\langle n_{obj}, \langle 0, c_d, c_f, c_b\rangle, \langle 1, c_d, c_f, c_b\rangle, \ldots, \langle m, c_d, c_f, c_b\rangle\rangle$. That is, $\langle 0, c_d, c_f, c_b\rangle$ is the cost breakdown c_d, c_f and c_b at iteration 0 for one scene. n_{obj} is the number of objects in that scene and m is the last iteration of the optimisation. We have a tuple for each distinct scene (each of which has a number of tuples for each iteration of the optimisation).

Similarly, the dataset for ϕ_{human} consists of the optimisation cost breakdown per scene before and after human input. Again, I generated a number of distinct scenes and ran OR-HITL this time to solve them and collect data. For *each* scene we have the following: $\langle n_{obj}, \langle 0, c_d^a, c_f^a, c_b^a\rangle, \langle 1, c_d^a, c_f^a, c_b^a\rangle, \ldots, \langle m-1, c_d^h, c_f^h, c_b^h\rangle, \langle m, c_d^h, c_f^h, c_b^h\rangle\rangle$. That is, $\langle 0, c_d^a, c_f^a, c_b^a\rangle$ is the cost breakdown c_d, c_f and c_b at iteration 0 *before human input*. $\langle m-1, c_d^h, c_f^h, c_b^h\rangle$ is the cost breakdown at iteration $m-1$ *after human input*. Again, n_{obj} is the number of objects in the scene and m is the last iteration of the optimisation. The only difference here is that we have a key iteration when human input was provided and we treat the cost breakdown differently before and after that

point (i.e., the change in the cost after the human input). Essentially, we want to learn what $\langle c_d^h, c_f^h, c_b^h \rangle$ will be given a sequence of $\langle i, c_d^a, c_f^a, c_b^a \rangle$.

5.5.3 Network architecture

The two sets are used to train ϕ_{human} and $\phi_{autonomous}$. I used Tensorflow for implementing both models [135]. $\phi_{autonomous}$ and ϕ_{human} are two feed-forward deep neural networks with the same architecture. After experimentation with different architectures, the current architecture consists of an input layer, three hidden layers and the output layer (the cost l iterations in the future). The hidden layers contain 64, 32 and 8 neurons with reLU activation function respectively.

The input and output layers for the two models are thus:

- $\phi_{autonomous}$: $\langle n_{obj}, \langle i-p, c_d, c_f, c_b \rangle, \ldots, \langle i-1, c_d, c_f, c_b \rangle, \langle i, c_d, c_f, c_b \rangle \rangle$. That is, some p consecutive iterations and the cost breakdown for each. The output layer is the total cost at the $l = i + 10$ iteration. That is, $c_{autonomous} = c_d + c_f + c_b$ on the lth iteration.

- ϕ_{human}: $\langle n_{obj}, \langle i-p, c_d^a, c_f^a, c_b^a \rangle, \ldots, \langle i-1, c_d^a, c_f^a, c_b^a \rangle, \langle i, c_d^a, c_f^a, c_b^a \rangle \rangle$. That is, some p consecutive iterations and the cost breakdown for each *before human input*. The output layer is the total cost at the $l = i + 10$ iteration *after human input*. That is, $c_{human} = c_d^h + c_f^h + c_b^h$ on the lth iteration.

5.6 Experiments & Results

In this section, I evaluate the performance of the models and of the framework in a virtual warehouse with a sliding number of robots for the problem of Reaching Through Clutter. I used a randomiser to generate random problems. Each problem consists of a scene with 20 movable objects on a shelf and the robot. The randomiser shuffles the movable obstacles in collision-free places including the goal object but ensures that the goal object is always at the back row of the shelf to create more challenging problems.

5.6.1 Hypotheses

The hypotheses I am evaluating in this section are:

1. It is possible to learn to predict future optimisation cost values.
2. Choosing the right robot (from a group of robots) now that will need human guidance in the future, will yield to better overall performance in a setting where a single human operator guides multiple robots.

5.6.2 Experimental setup

Optimiser parameters: For the problem of Reaching Through Clutter, I use MuJoCo [134] and its C++ API to implement the system dynamics. I use $k = 8$ noisy trajectories at each iteration, variance of 0.04 m/s for the robot's linear velocities and 0.04 rad/s for the robot's angular velocity. The cost function's parameters are: $w_d = 4000, w_f = 30, w_b = 450$ and $w_s = 300$. The success threshold is 20.0.

Predictors: The prediction is $l = 10$ iterations in the future. The input layer of the neural network uses $p = 3$, that is, we look at the last three iterations: $\langle n_{obj}, \langle i-2, c_d, c_f, c_b\rangle, \langle i-1, c_d, c_f, c_b\rangle, \langle i, c_d, c_f, c_b\rangle\rangle$. I trained the networks on a computer with Intel© Core™ i7-4790 CPU @ 3.60GHz, 16GB RAM. I used TensorFlow 2.0 [135] with Keras. The training batch size is 50 and used Keras' EarlyStopping callback for adaptive number of epochs. I used the "Adam" optimiser with Root Mean Squared Error (RMSE) as the loss function. $t_{windows}$ for HUMAN-ALLOCATOR is 2 seconds.

The evaluation proceeds as follows. First, in the next section I evaluate the performance of the models and in Section 5.6.4, I evaluate the proposed PGF framework in a virtual warehouse with a sliding number of robots.

5.6.3 Model performance

For the human dataset, I collected data from 3 participants and myself from 1,500 distinct scenes. OR-HITL was used in these scenes and the data collected were used

to train ϕ_{human}. For the autonomous dataset, I collected data from 4,000 distinct scenes to train $\phi_{autonomous}$. In these 4,000 problems the robot tried to solve RTC problems fully autonomously, without any human intervention. These datasets were split 80-20% for training and validation respectively.

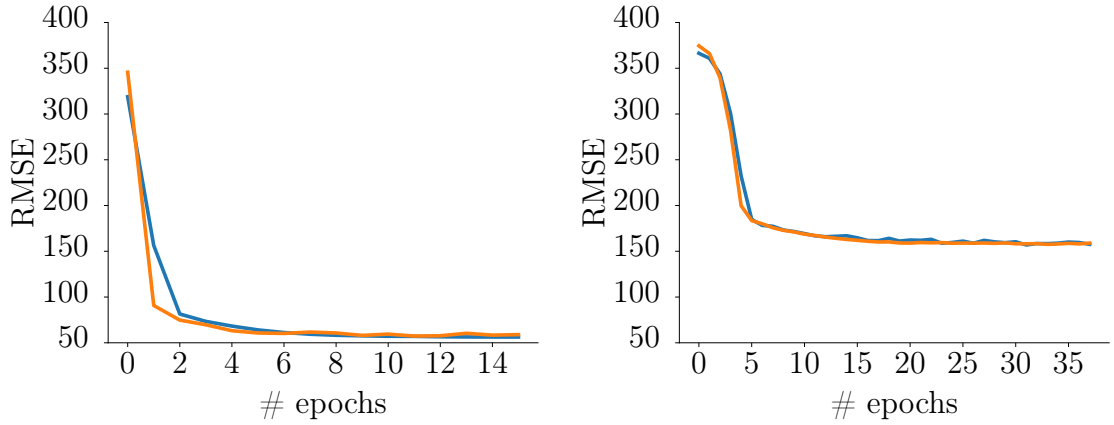
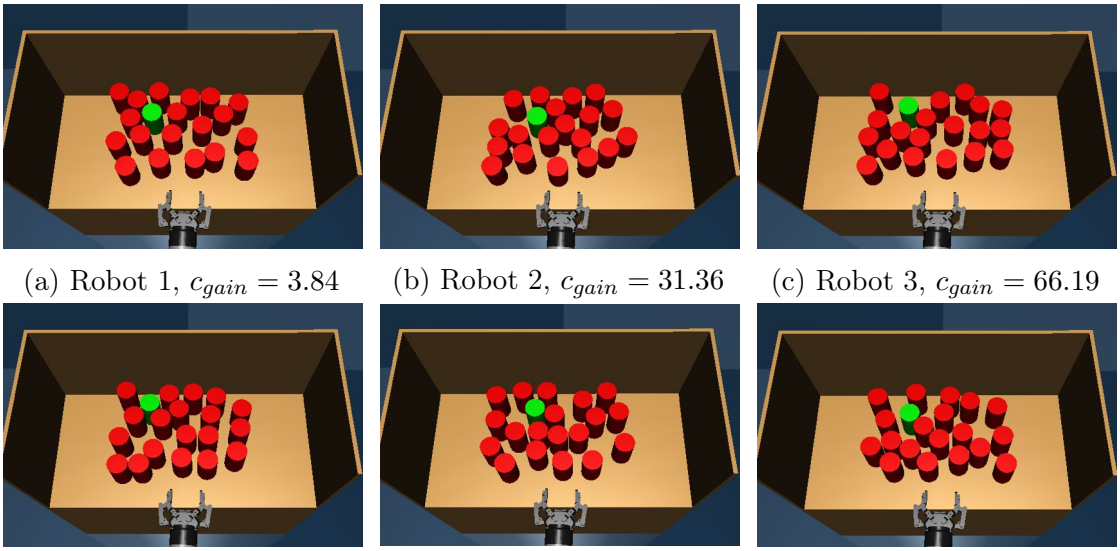


Figure 5.3: $\phi_{autonomous}$ (left) and ϕ_{human} (right). Blue line is the loss function (Root Mean Squared Error) on the training set and orange line is the loss function on the validation set.

Figure 5.3 shows the performance of these models. The error for the autonomous predictor is much lower due to the availability of more data points. The RMSE of the autonomous predictor is 58.56 and for the human predictor is 157.49. To put these numbers into a perspective, for the two scenes in Figure 5.1 the cost of the initial trajectory for the left and right problems is 749.56 and 0 (initial trajectory was a valid solution; no optimisation iteration was required) respectively.

In the next section, using these predictors I evaluate the complete system in a virtual warehouse with a number of robots.



(a) Robot 1, $c_{gain} = 3.84$ (b) Robot 2, $c_{gain} = 31.36$ (c) Robot 3, $c_{gain} = 66.19$

(d) Robot 4, $c_{gain} = 66.64$ (e) Robot 5, $c_{gain} = 49.52$ (f) Robot 6, $c_{gain} = 51.06$

Figure 5.4: Example scenes from the experiments in a bidding window. There are twenty potential robots. Six of them join the bidding window (a-f) at the same time. c_{gain} is their predicted gain.

5.6.4 Coping with a fleet of demanding robots

Next, I set-up a virtual warehouse[1] of 1-30 robots picking objects from shelves. I generated 1,500 new random scenes equally distributed among the thirty robots[2]. PGF is the proposed algorithm of this chapter and OR-HITL is the algorithm from the previous chapter. The main difference between the two is that PGF uses the predictive approach to predict future cost and queries human help earlier, while OR-HITL queries the human when stuck in local minima and there is no precedence; the human is allocated to the first robot to enter the pool.

Some demonstration scenes are presented in Figure 5.4. The figure presents six robots, out of the twenty parallel robots in the virtual warehouse, that joined the

[1]Using a powerful EC2 instance on Amazon Web Services ©: c5.24xlarge with 96 vCPUs and 192GiB of RAM

[2]With the exception of one robot experiments that the number was increased to 100 instead of 50.

same window with different predicted gains. Among the six robots, robot 4 was the one with the highest predicted gain and the one assigned to the human. I compare their predicted values with the final costs at the final iteration when the same problem was solved using OR-HITL. For robot 1, OR-HITL solved the problem autonomously in 4 iterations, robot 2 solved it autonomously in 8 iterations, robot 3 hit a local minimum and queried the human on the 7th iteration, robot 4 failed to solve the problem altogether, robot 5 solved it autonomously in 20 iterations, and robot 6 hit a local minimum early on (on the 4th iteration) and queried the human.

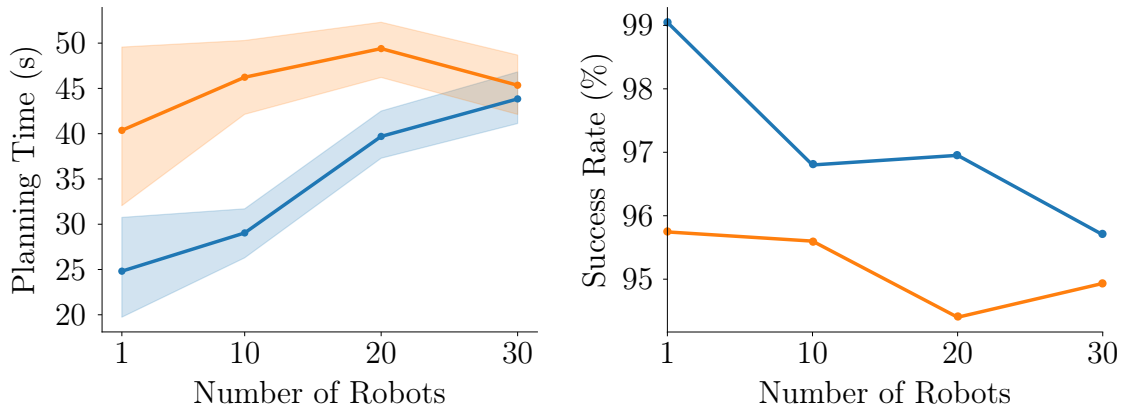


Figure 5.5: Planning times (left) and success rate (right) for PGF (blue) and OR-HITL (orange) as the number of robots under the supervision of a single human-operator increases. Error shadow indicates the 95% Confidence Interval. The results suggest that a single human operator can guide efficiently up to 20 robots using the proposed approach (PGF). As the number of robots under the human supervision increases from 20 to 30, the performance starts to become equal to the baseline (OR-HITL).

Next, I measure the performance of these systems and present results in Figure 5.5. The data in Figure 5.5-Left suggest that the planning time of the robots increases as the number of robots under the human supervision increases as well, however there is a statistically significant improvement in planning times when a human supervises from one to twenty robots when using the PGF system. In these cases the system correctly queried the human earlier and therefore allowed the robots to receive high-level human input earlier on. The data suggest that once the number of robots

increases from twenty to thirty this significance shrinks. A possible explanation is that as the number of robots increases, there are more robots waiting for help that are not able to take human time and therefore these robots perform similar to OR-HITL (where help comes much later or not at all). The success rate in Figure 5.5-Right shows a similar trend but in general the success rate of both approaches is relatively high compared to the task difficulty. Considering the total experiments, OR-HITL queried the human on the 25th iteration on average, while PGF queried the human on the 6th iteration on average.

5.7 Conclusions

In this chapter, I proposed an extension to OR-HITL, the PGF. By incorporating predictions of future costs, this system is able to adaptively query the human for help earlier, before unnecessary planning time is wasted. The predictions act also as a metric to sort robot requests based on expected gain, allowing the system to allocate the most urgent robot to the human. The results suggest that the system improves OR-HITL significantly while the human is not overburdened by multiple robot requests. However, the results show that the number of robots a single human can effectively guide at the same time has a cap of 20 robots. In settings with more than 20 robots, the data shows that PGF has no significant difference than OR-HITL, suggesting that a single human-operator can effectively guide up to 20 robots at a time.

Chapter 6

Conclusions & Future Work

In this chapter, I conclude and summarise the content of the thesis and suggest directions for future work.

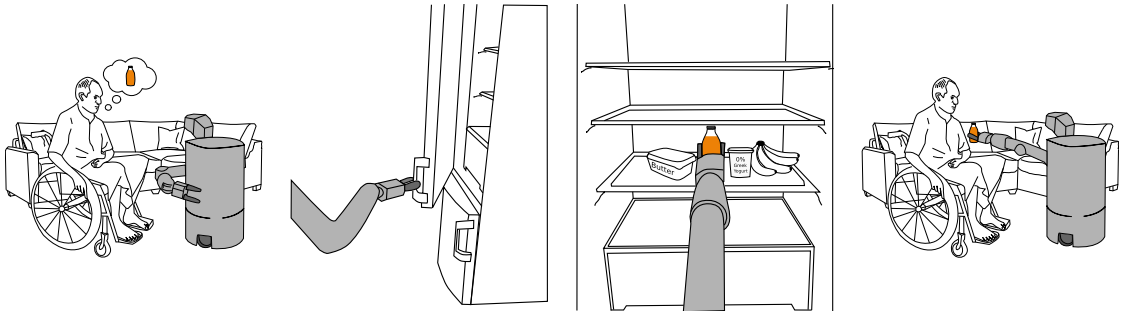


Figure 6.1: Which are some of the remaining challenges with respect to this task?

6.1 Conclusions

The focus of this thesis is on physics-based non-prehensile manipulation in clutter. Such problems occur frequently in different cases. For example, as shown in Figure 6.1, a home care robot that assists a motor-impaired person or in warehouse management where robots pick objects from shelves. In these cases, the robots need to reach and work in cluttered environments and reason about which objects to push and where to make space to reach for a desired object. The problem is challenging for the following reasons:

1. The problem is a high-dimensional one, where the system needs to consider the state of all the movable objects and itself.

2. The problem is an under-actuated system; the robot needs to control the state of movable objects through contact.

3. Physics-based motion planning requires the use of physics simulator which is a computationally expensive process.

4. Physics-uncertainty causes valid solutions obtain in simulation to fail during execution in the real world.

All these challenges make this problem a challenging one, and existing methods suffer from long planning times or low success rate for realistic instances of the problem.

In this thesis, I investigate how human input can be integrated in motion planning algorithms to address these challenges. My focus was mainly on sampling-based kinodynamic planning and trajectory optimisation-based planning.

Initially, my contribution to sampling-based kinodynamic planning in Chapter 3, presents a framework that considers high-level inputs from a human-operator. The human operator input is a triple of an object to be manipulated and an approximate position of where to be pushed. With these inputs, the robot can plan for sub-goals to more effectively reach and grasp the goal object. The approach builds on existing motion planning sampling-based planners (kinodynamic RRT and KPIECE) and proposes a framework for non-prehensile manipulation with human-in-the-loop. I compare the proposed system with autonomous planners (that consider no guidance) and with a baseline that automatically tries to generate high-level actions (similar to the ones that are suggested by a human). The results suggest that human input improves the performance of the proposed system compared to state-of-the-art baselines, and show that human intuition is effective and easy to obtain. This initial framework demonstrated that human-input is effective in these problems, but shown to be pruned to long and noisy solutions.

Next, in Chapter 4, I proposed a trajectory-optimisation-based planner to alleviate shortcomings of sampling-based planning. Specifically, the new approach, OR-HITL, allows robots to start tackling problems fully-autonomously and only query a human when needed. This is achieved by using an "Adaptive" approach to decide when to ask for human help. In specific, the system tries to detect when the planner is stuck in local minima and queries the human at that point. This permits the robots

to solve trivial problems fully autonomously, yet fallback to a human when they are stuck at a local minimum. The use of trajectory-optimisation also permits to more easily integrate the human-input in the optimisation by injecting the high-level input in the cost function. Finally, trajectory optimisation lends itself more easily to online-replanning, and I proposed a novel online-replanning strategy to tackle the problem of physics-uncertainty. This online-replanning strategy takes advantage of trajectory optimisation and optimises the trajectory at the end of each action execution in the real world. As a result, if physics-uncertainty causes the trajectory to be invalid after a number of step execution in the real-world, the planner will re-optimise and improve the trajectory. The planner interleaves optimisation and execution and warm-starts the optimisation with the trajectory from the previous iteration to achieve convergence faster. This new system, due to the effective use of human-input (human only supervises robots dealing with hard problems as opposed to supervise *all* robots as with the system in Chapter 3), allows a single human to easily guide six robots simultaneously and provides significant improvements in terms of the planning times and success rates than the baselines. The results demonstrated that this new system is solving RTC problems faster, saves human time by solving problems autonomously and also that is robust to physics-uncertainty.

Finally, the last contribution is PGF in Chapter 5, a novel predictive system, extension of OR-HITL, that permits robots to learn the human contribution and predict the course of the optimisation to identify early on when to ask for help, before time is lost unnecessarily waiting to hit the local minima. The system trains a model from past experiences when a human supervised robots and when the robot solved problems fully autonomously. The model learns to predict the future optimisation cost of a given problem with and without human help to derive a predicted human gain (what is the effect of human input in the future if I were to request help now?). This predicted gain is then used to decide when to ask for help, and also to order the requests of multiple robots. This predictive system considers multiple robots (up to thirty) all of which "bid" for human time, allowing the system to assign the human to the robot with the highest anticipated gain. PGF was effective in guiding 1-20 robots, but shown to perform equally with OR-HITL when the number of robots was

increased to thirty.

The work, however, has a number of limitations. In specific, I consider a planar problem where the robot motion is constraint in the plane. This simplifies the problem, but might not be ideal for certain instances of the problem that require motion planning in a 3D space. Additionally, a challenge of the problem is related to perception. In this thesis, I do not focus on the problem of perception and I used a Motion Capture System to track the position of objects. Furthermore, in this thesis I assume a fully-observable environment and that the robot is aware of the position of all the objects in the environment. In realistic problems, however, this is not always the case, as objects might be occluded from the robot's view. Finally, I consider pushing-based actions only, which again can be limited for solving challenging instances of the problem. In the next section, I consider all the above limitations and propose possible future directions of the work.

6.2 Future Work

In the next sections, I consider several future directions and how this work can be improved within the scope of non-prehensile manipulation in clutter.

6.2.1 Computer vision and partially-observable environment

In the introduction of this thesis, I noted that manipulation is challenging for several reasons, one of which is computer vision. Throughout this thesis I excluded this challenge to focus on the computational problem of robotic manipulation planning and control. I, therefore, assumed a fully-observable environment. That is, the robot is aware of the pose of all the objects.

In a realistic problem like the one in Figure 6.1, however, there are two challenges that we need to consider:

1. Detecting objects and their pose with respect to the robot.

2. Assume partial observability of the environment and that the robot vision is also constrained depending on the robot's position.

There were great advancements recently with respect to the first problem using machine learning and depth-sensors [136–139], however it still remains challenging to accurately detect a variety of different or novel objects. Moreover, a system that detects objects and their poses through a depth-sensor is subject to pose errors that could create further planning challenges due to the problem of pose/physics-uncertainty. For the second problem, there was a recent work tackling this specific problem for manipulation in clutter, but limited to 2D planar actions [73].

6.2.2 Handling deformable objects and non-standard geometry objects

The objects I consider in this thesis are standard rigid bodies (cylinders or boxes) and assume they are easily graspable by the robot. In a real-environment, however, there are objects of non-standard geometry or deformable objects, especially in industrial shelves. Such objects pose challenges for computer vision, pose estimation and for physics simulation. Grasping, is also a research area by itself [140–142], and I did not consider the problem of grasping in this thesis. The objects that the robot had to mainly grasp where small and graspable objects. I assumed that once the object is within the robot's end-effector the robot can close its fingers and grasp the object. It is an interesting direction, especially from an industrial point of view, to investigate how this work can be extended to handle such objects.

6.2.3 Handling more complex non-prehensile interactions

Throughout this thesis, I consider pushing-based actions and I refer to non-prehensile manipulation just for pushing-based actions. Certain clutter configurations might require more complex non-prehensile interaction or a sequence of such non-prehensile actions. Consider Figure 6.2 as an example problem. Retrieving the book from the shelf requires first to tilt it, to create a graspable surface, and second to grasp

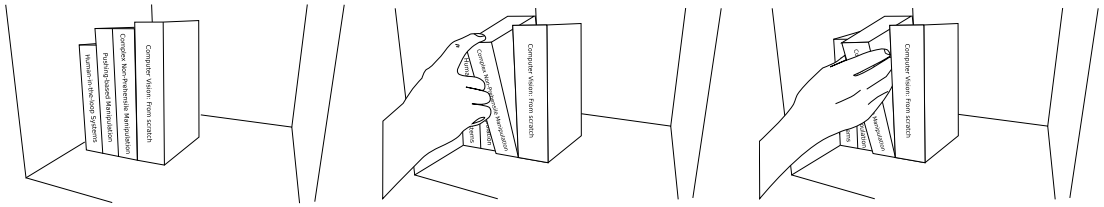


Figure 6.2: Grasping the goal book requires different non-prehensile actions. A similar example and figure was first introduced in [3] which inspired the illustration of this figure.

and pull the object. Such problems could occur frequently in cluttered scenes and robots should have the required skills to tackle such problems where a sequence of non-prehensile actions need to be performed in the right order.

Although throughout this thesis I motivated non-prehensile over prehensile manipulation to manipulate cluttered scenes, the two are and should not be mutually exclusive. In certain environments, prehensile manipulation is actually an optimal choice. Consider the case when a fridge is highly cluttered and objects are stacked over others. There are cases when we need to pick and place a key obstacle out of the fridge, on a temporary surface, to free up space or to ensure the stability and safety of that obstacle before we can reach for the desired object at the back of the fridge. Not doing so, could result in damaging the obstacle while pushing other objects. Such problems require synergies between prehensile and non-prehensile manipulation. Such systems have been studied before [19] but not for the problem of reaching through clutter.

The challenges of such hybrid approaches is the high-dimensionality of both the task and state space. A recent approach used a hierarchical planning approach for planning a sequence of non-prehensile actions [110]. The work is limited to 2D problems with two robot contacts only. In the context of human-in-the-loop planning, it will be interesting to see how human input can more effectively used to guide the task planner and/or the motion planner.

6.2.4 Moving from a 2-dimensional space to a 3-dimensional

To simplify the problem and focus on the pushing-based actions of reaching through clutter I assumed that the robot is moving on the plane. Although a great number of problems can be solved in this space, certain problems, like the one in Figure 6.2 for example, require planning and acting in the 3-dimensional space. Moving to a 3-dimensional space will increase the dimensionality of the state and control space. It will also require the use of an inverse-kinematics (IK) solver to calculate the required robot configurations and to plan in the robot configuration space instead of the end-effector space. The frameworks I proposed in this thesis, however, are not particularly limited to a 2D space. It will require changes to the control sampling, state space representation and to the roll-out function.

6.2.5 Different types of human input

This thesis mainly focused on pushing-based actions and the human input was constrained to this specific type of action while trying to capture information from a high-level. There is space for experimentation with different types of human input. A possibility is to implement a system with different Levels of Robot Autonomy (LORA) [143], that will allow the system to ask different types of inputs from a human. For example, a higher-level of autonomy might require minimal input from the human, like the one I studied in this thesis, yet a lower-level of autonomy might allow the human to specify a complete configuration of the robot. The latter requires more work from the human, but might be more effective in very challenging instances of the problem. A challenge with such a system is to know when to query the human with higher level of autonomy or lower level of autonomy. One idea is to extend the work in Chapter 5 to handle not only when to ask for human help but also what control level to give to the human.

Finally, in this work I considered conventional interfaces and input devices like a computer monitor with a graphical user interface, a mouse and a keyboard. Although such devices are highly available, cheap and effective in most cases, there is possibility for improvement and innovation considering more sophisticated user interfaces with

more tailored or higher-dimensional input devices.

6.2.6 Learning from human input

An argument one can make when it comes to shared-autonomy and a system that receives input from a human is whether the robot can learn from the human input. This is an interesting and active research question. There are recent works in imitation learning that consider similar problems [69, 80, 144, 145]. Given a number of demonstrations from a human, a system can learn to imitate this behaviour. The challenges for such system are mainly the number of demonstrations needed for a system to generalise and how easy will be for the robot to learn from different sources in unstructured and diverse environments. In our setting, the system can be used by different human-operators. The high-level input, most of the time, is a user preference. There are multiple good high-level inputs that a human can suggest. Learning in such diverse space could be challenging for existing methods, especially in a continuous space (position where the object should be pushed).

6.3 Final Remarks

In this chapter, I summarised the work and the results. It is clear that human-in-the-loop systems for non-prehensile manipulation in clutter are advantageous, and this advantage comes with small overhead to a human operator who can guide multiple robots simultaneously. The work considers multiple relaxations of the original problem and focuses on the motion planning and control aspect of pushing-based non-prehensile manipulation. I identified several limitations of the work and possible future directions to improve it.

Bibliography

[1] Gordon Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3(1):131–150, 1991.

[2] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[3] Francois Robert Hogan, Eudald Romo Grau, and Alberto Rodriguez. Reactive planar manipulation with convex hybrid mpc. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 247–253. IEEE, 2018.

[4] Mustafa Ersen, Erhan Oztop, and Sanem Sariel. Cognition-enabled robot manipulation in human environments: requirements, recent work, and open problems. *IEEE Robotics & Automation Magazine*, 24(3):108–122, 2017.

[5] Matthew T Mason. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:1–28, 2018.

[6] Qaid Mohammed Marwan, Shing Chyi Chua, and Lee Chung Kwek. Comprehensive review on reaching and grasping of objects in robotics. *Robotica*, pages 1–34, 2021.

[7] Nikita Kitaev, Igor Mordatch, Sachin Patil, and Pieter Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3102–3109. IEEE, 2015.

[8] Wisdom C Agboh and Mehmet R Dogar. Robust physics-based manipulation by interleaving open and closed-loop execution. *arXiv preprint arXiv:2105.08325*, 2021.

[9] Muhayy ud din, Mark Moll, Lydia Kavraki, Jan Rosell, et al. Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719, 2018.

[10] Sebastian Muszynski, Jörg Stückler, and Sven Behnke. Adjustable autonomy for mobile teleoperation of personal service robots. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 933–940. IEEE, 2012.

[11] Adam Eric Leeper, Kaijen Hsiao, Matei Ciocarlie, Leila Takayama, and David Gossow. Strategies for human-in-the-loop robotic grasping. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 1–8. ACM, 2012.

[12] Matei Ciocarlie, Kaijen Hsiao, Adam Leeper, and David Gossow. Mobile manipulation through an assistive home robot. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5313–5320. IEEE, 2012.

[13] Andrew Kimmel, Rahul Shome, and Kostas Bekris. Anytime motion planning for prehensile manipulation in dense clutter. *Advanced Robotics*, 33(22):1175–1193, 2019.

[14] Muhammad Suhail Saleem and Maxim Likhachev. Planning with selective physics-based simulation for manipulation among movable objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6752–6758. IEEE, 2020.

[15] A Akbari Muhayy ud din and Jan Rosell. Knowledge-oriented physics-based motion planning for grasping under uncertainty, 2017.

[16] Lillian Y Chang, Garth J Zeglin, and Nancy S Pollard. Preparatory object rotation as a human-inspired grasping strategy. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 527–534. IEEE, 2008.

[17] Matthew T Mason. Progress in nonprehensile manipulation. *The International Journal of Robotics Research*, 18(11):1129–1141, 1999.

[18] Fabio Ruggiero, Vincenzo Lippiello, and Bruno Siciliano. Nonprehensile dynamic manipulation: A survey. *IEEE Robotics and Automation Letters*, 3(3):1711–1718, 2018.

[19] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[20] M Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha Srinivasa. Physics-based grasp planning through clutter. In *Robotics: Science and Systems*, 2012.

[21] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martín-Martín, Arne Sieverling, Vincent Wall, and Oliver Brock. Lessons from the amazon picking challenge: Four aspects of building robotic systems. In *Robotics: Science and Systems*, 2016.

[22] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano, and Peter R Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1): 172–188, 2016.

[23] Rafael Papallas and Mehmet R Dogar. Non-prehensile manipulation in clutter with human-in-the-loop. In *IEEE International Conference on Robotics and Automation*, 2020.

[24] Rafael Papallas and Mehmet R Dogar. Human-guided planner for non-prehensile manipulation. In *IROS 2019: Workshop Factory of the Future*, 2019.

[25] Rafael Papallas and Mehmet R Dogar. Online replanning with human-in-the-loop for non-prehensile manipulation in clutter — a trajectory optimization based approach. *IEEE Robotics and Automation Letters*, 2020.

[26] Rafael Papallas, Anthony G Cohn, and Mehmet R Dogar. Optimization-based motion planning with human in the loop for non-prehensile manipulation. In *ICRA 2020: Workshop Shared autonomy: Learning and Control*, 2020.

[27] Lydia E Kavraki and Steven M LaValle. Motion planning. In *Springer Handbook of Robotics*, pages 139–162. Springer, 2016.

[28] Wisama Khalil and Etienne Dombre. *Modeling identification and control of robots*. CRC Press, 2002.

[29] Kenneth J Waldron and James Schmiedeler. Kinematics. In *Springer handbook of robotics*, pages 11–36. Springer, 2016.

[30] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[31] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[32] James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[33] Olzhas Adiyatov and Huseyin Atakan Varol. Rapidly-exploring random tree based memory efficient motion planning. In *2013 IEEE international conference on mechatronics and automation*, pages 354–359. IEEE, 2013.

[34] Giray Havur, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid rea-

soning approach. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 445–452. IEEE, 2014.

[35] Wissam Bejjani, Mehmet R Dogar, and Matteo Leonetti. 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)*. IEEE, 2019.

[36] Nikita Kitaev, Igor Mordatch, Sachin Patil, and Pieter Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3102–3109. IEEE, 2015.

[37] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Sampling-based methods for motion planning with constraints. *Annual review of control, robotics, and autonomous systems*, 1:159–185, 2018.

[38] Jory Denny, Read Sandström, and Nancy M Amato. A general region-based framework for collaborative planning. In *Robotics Research*, pages 563–579. Springer, 2018.

[39] Xinda Wang, Xiao Luo, Baoling Han, Yuhan Chen, Guanhao Liang, and Kailin Zheng. Collision-free path planning method for robots based on an improved rapidly-exploring random tree algorithm. *Applied Sciences*, 10(4):1381, 2020.

[40] Vojtěch Vonásek, Jan Faigl, Tomáš Krajník, and Libor Přeučil. Rrt-path–a guided rapidly exploring random tree. In *Robot Motion and Control 2009*, pages 307–316. Springer, 2009.

[41] Matthew Jordan and Alejandro Perez. Optimal bidirectional rapidly-exploring random trees, 2013.

[42] Xin Gao, Haoxin Wu, Lin Zhai, Hanxu Sun, Qingxuan Jia, Yifan Wang, and Likai Wu. A rapidly exploring random tree optimization algorithm for space robotic manipulators guided by obstacle avoidance independent potential field.

International Journal of Advanced Robotic Systems, 15(3):1729881418782240, 2018.

[43] Emese Szádeczky-Kardoss and Bálint Kiss. Extension of the rapidly exploring random tree algorithm with key configurations for nonholonomic motion planning. In *2006 IEEE International Conference on Mechatronics*, pages 363–368. IEEE, 2006.

[44] Daqing Yi, Michael A Goodrich, and Kevin D Seppi. Homotopy-aware RRT*: Toward human-robot topological path-planning. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pages 279–286. IEEE Press, 2016.

[45] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[46] Joshua A Haustein, Jennifer King, Siddhartha S Srinivasa, and Tamim Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3075–3082. IEEE, 2015.

[47] Jennifer E King, Marco Cognetti, and Siddhartha S Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3940–3947. IEEE, 2016.

[48] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. *IEEE Robotics and Automation Letters*, 4(4): 4298–4305, 2019.

[49] Chang-bae Moon and Woojin Chung. Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree. *IEEE Transactions on industrial electronics*, 62(2):1080–1090, 2014.

[50] Jennifer E King, Joshua A Haustein, Siddhartha S Srinivasa, and Tamim Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2508–2515. IEEE, 2015.

[51] Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE international conference on robotics and automation*, pages 2493–2498. IEEE, 2010.

[52] Ahmed Hussain Qureshi and Yasar Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 40(6):1079–1093, 2016.

[53] Zaid Tahir, Ahmed H Qureshi, Yasar Ayaz, and Raheel Nawaz. Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments. *Robotics and Autonomous Systems*, 108:13–27, 2018.

[54] Ioan A Şucan and Lydia E Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009.

[55] Radu Bogdan Rusu, Ioan Alexandru Şucan, Brian Gerkey, Sachin Chitta, Michael Beetz, and Lydia E Kavraki. Real-time perception-guided motion planning for a personal robot. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4245–4252. IEEE, 2009.

[56] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `http://ompl.kavrakilab.org`.

[57] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[58] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[59] Chonhyon Park, Jia Pan, and Dinesh Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of the international conference on automated planning and scheduling*, volume 22, 2012.

[60] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer, 2013.

[61] Jeffrey Ichnowski, Michael Danielczuk, Jingyi Xu, Vishal Satish, and Ken Goldberg. Gomp: Grasp-optimized motion planning for bin picking. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5270–5277. IEEE, 2020.

[62] Yinxiao Li, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6000–6006. IEEE, 2015.

[63] Balakumar Sundaralingam and Tucker Hermans. Relaxed-rigidity constraints: kinematic trajectory optimization and collision avoidance for in-grasp manipulation. *Autonomous Robots*, 43(2):469–483, 2019.

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

[65] Wisdom C Agboh, Daniel Ruprecht, and Mehmet R Dogar. Combining coarse and fine physics for manipulation using parallel-in-time integration. *arXiv preprint arXiv:1903.08470*, 2019.

[66] Wisdom C Agboh and Mehmet R Dogar. Real-time online re-planning for grasping under clutter and uncertainty. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2018.

[67] Marwan Qaid Mohammed, Kwek Lee Chung, and Chua Shing Chyi. Review of deep reinforcement learning-based object grasping: Techniques, open challenges and recommendations. *IEEE Access*, 2020.

[68] Jinwook Huh and Daniel D Lee. Efficient sampling with q-learning to guide rapidly exploring random trees. *IEEE Robotics and Automation Letters*, 3(4): 3868–3875, 2018.

[69] Mohamed Hasan, Matthew Warburton, Wisdom C Agboh, Mehmet R Dogar, Matteo Leonetti, He Wang, Faisal Mushtaq, Mark Mon-Williams, and Anthony G Cohn. Human-like planning for reaching in cluttered environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7784–7790. IEEE, 2020.

[70] Andrew M Wells, Neil T Dantam, Anshumali Shrivastava, and Lydia E Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE robotics and automation letters*, 4(2):1255–1262, 2019.

[71] Wissam Bejjani, Rafael Papallas, Matteo Leonetti, and Mehmet R Dogar. Planning with a receding horizon for manipulation in clutter using a learned value function. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE, 2018.

[72] Wissam Bejjani, Matteo Leonetti, and Mehmet R Dogar. Learning image-based receding horizon planning for manipulation in clutter. *Robotics and Autonomous Systems*, 138:103730, 2021.

[73] Wissam Bejjani, Wisdom C Agboh, Mehmet R Dogar, and Matteo Leonetti. Occlusion-aware search for object retrieval in clutter. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[74] Joshua A Haustein, Isac Arnekvist, Johannes Stork, Kaiyu Hang, and Danica Kragic. Learning manipulation states and actions for efficient non-prehensile rearrangement planning. *arXiv preprint arXiv:1901.03557*, 2019.

[75] Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning robot objectives from physical human interaction. In *Conference on Robot Learning*, pages 217–226. PMLR, 2017.

[76] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016.

[77] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[78] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[79] Frederik Ebert, Sudeep Dasari, Alex X Lee, Sergey Levine, and Chelsea Finn. Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning. In *Conference on Robot Learning*, pages 983–993. PMLR, 2018.

[80] Michael Laskey, Jonathan Lee, Caleb Chuck, David Gealy, Wesley Hsieh, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In

2016 IEEE International Conference on Automation Science and Engineering (CASE), pages 827–834. IEEE, 2016.

[81] Weihao Yuan, Kaiyu Hang, Danica Kragic, Michael Y Wang, and Johannes A Stork. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems*, 119: 119–134, 2019.

[82] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[83] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 33(14):1726–1747, 2014.

[84] Athanasios Krontiris and Kostas E Bekris. Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3924–3931. IEEE, 2016.

[85] Michael Beetz, Raja Chatila, Joachim Hertzberg, and Federico Pecora. Ai reasoning methods for robotics. In *Springer Handbook of Robotics*, pages 329–356. Springer, 2016.

[86] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*. IEEE, 2010.

[87] Aliakbar Akbari, Jan Rosell, et al. Task and motion planning using physics-based reasoning. In *2015 IEEE 20th conference on Emerging technologies & factory automation (ETFA)*, pages 1–7. IEEE, 2015.

[88] KiBeom Kim, Jinhwi Lee, ChangHwan Kim, and Changjoo Nam. Retrieving objects from clutter using a mobile robotic manipulator. In *2019 16th International Conference on Ubiquitous Robots (UR)*, pages 44–48. IEEE, 2019.

[89] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3327–3332. IEEE, 2007.

[90] Jinhwi Lee, Younggil Cho, Changjoo Nam, Jonghyeon Park, and Changhwan Kim. Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. *arXiv preprint arXiv:1902.06907*, 2019.

[91] Changjoo Nam, Jinhwi Lee, Sang Hun Cheong, Brian Y Cho, and ChangHwan Kim. Fast and resilient manipulation planning for target retrieval in clutter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3777–3783. IEEE, 2020.

[92] Shiqi Zhang, Mohan Sridharan, and Jeremy L Wyatt. Mixed logical inference and probabilistic planning for robots in unreliable worlds. *IEEE Transactions on Robotics*, 31(3):699–713, 2015.

[93] Shiqi Zhang, Mohan Sridharan, and Christian Washington. Active visual planning for mobile robot teams using hierarchical pomdps. *IEEE Transactions on Robotics*, 29(4):975–985, 2013.

[94] Jochen Stüber, Claudio Zito, and Rustam Stolkin. Let's push things forward: A survey on robot pushing. *Frontiers in Robotics and AI*, 7:8, 2020.

[95] Michael A Erdmann and Matthew T Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):369–379, 1988.

[96] Nik A Melchior and Reid Simmons. Particle rrt for path planning with uncertainty. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1617–1624. IEEE, 2007.

[97] Mehmet R Dogar and Siddhartha S Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2123–2130. IEEE, 2010.

[98] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 1, 2011.

[99] Aaron M Johnson, Jennifer E King, and Siddhartha Srinivasa. Convergent planning. *IEEE Robotics and Automation Letters*, 1(2):1044–1051, 2016.

[100] Elbrous M Jafarov, Mehmet Nur Alpaslan Parlakçi, and Yorgo Istefanopulos. A new variable structure pid-controller design for robot manipulators. *IEEE Transactions on Control Systems Technology*, 13(1):122–130, 2004.

[101] Rakesh P Borase, DK Maghade, SY Sondkar, and SN Pawar. A review of pid control, tuning methods and applications. *International Journal of Dynamics and Control*, pages 1–10, 2020.

[102] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[103] Ermano Arruda, Michael J Mathew, Marek Kopicki, Michael Mistry, Morteza Azad, and Jeremy L Wyatt. Uncertainty averse pushing with model predictive path integral control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 497–502. IEEE, 2017.

[104] François Robert Hogan and Alberto Rodriguez. Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. In *Algorithmic Foundations of Robotics XII*, pages 800–815. Springer, 2020.

[105] Max Schwarz, Anton Milan, Arul Selvam Periyasamy, and Sven Behnke. Multi-class rgb-d object detection and semantic segmentation for autonomous manipulation in clutter, 2017.

[106] Bingjie Tang, Matthew Corsaro, George Konidaris, Stefanos Nikolaidis, and Stefanie Tellex. Learning collaborative pushing and grasping policies in dense clutter. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[107] Andrew Kimmel, Rahul Shome, Zakary Littlefield, and Kostas Bekris. Fast, anytime motion planning for prehensile manipulation in clutter. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE, 2018.

[108] Mehmet Dogar, Andrew Spielberg, Stuart Baker, and Daniela Rus. Multi-robot grasp planning for sequential assembly operations. *Autonomous Robots*, 43(3): 649–664, 2019.

[109] Guy Rosman, Changhyun Choi, Mehmet Dogar, John W Fisher, and Daniela Rus. Task-specific sensor planning for robotic assembly tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2932–2939. IEEE, 2018.

[110] Gilwoo Lee, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Hierarchical planning for multi-contact non-prehensile manipulation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 264–271. IEEE, 2015.

[111] Wissam Bejjani. Automated planning of whole-body motions for everyday household chores with a humanoid service robot, 2015.

[112] Francois R Hogan and Alberto Rodriguez. Reactive planar non-prehensile manipulation with hybrid model predictive control. *The International Journal of Robotics Research*, 39(7):755–773, 2020.

[113] Ziyan Gao, Armagan Elibol, and Nak Young Chong. Non-prehensile manipulation learning through self-supervision. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 93–99. IEEE, 2020.

[114] Wisdom Agboh, Oliver Grainger, Daniel Ruprecht, and Mehmet Dogar. Parareal with a learned coarse model for robotic manipulation. *Computing and Visualization in Science*, 23(1):1–10, 2020.

[115] Eric Huang, Zhenzhong Jia, and Matthew T Mason. Large-scale multi-object rearrangement. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 211–218. IEEE, 2019.

[116] Günter Niemeyer, Carsten Preusche, Stefano Stramigioli, and Dongjun Lee. Telerobotics. In *Springer handbook of robotics*, pages 1085–1108. Springer, 2016.

[117] Cyrill Stachniss, John J Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer, 2016.

[118] Jory Denny, Jonathan Colbert, Hongsen Qin, and Nancy M Amato. On the theory of user-guided planning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4794–4801. IEEE, 2016.

[119] Michel Taïx, David Flavigné, and Etienne Ferré. Human interaction with motion planning algorithm. *Journal of Intelligent & Robotic Systems*, 67(3-4): 285–306, 2012.

[120] Yong K Hwang, Kyoung R Cho, Sooyong Lee, Sang M Park, and Sungchul Kang. Human computer cooperation in interactive motion planning. In *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pages 571–576. IEEE, 1997.

[121] Benjamin Pitzer, Michael Styer, Christian Bersch, Charles DuHadway, and Jan Becker. Towards perceptual shared autonomy for robotic mobile manipulation. In *2011 IEEE International Conference on Robotics and Automation*, pages 6245–6251. IEEE, 2011.

[122] Daniel J Butler, Sarah Elliot, and Maya Cakmak. Interactive scene segmentation for efficient human-in-the-loop robot manipulation. In *2017 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 2572–2579. IEEE, 2017.

[123] Zdeněk Materna, Michal Španěl, Marcus Mast, Vítězslav Beran, Florian Weisshardt, Michael Burmester, and Pavel Smrž. Teleoperating assistive robots: A novel user interface relying on semi-autonomy and 3d environment mapping. *Journal of Robotics and Mechatronics*, 29(2):381–394, 2017.

[124] Thomas Witzig, J Marius Zöllner, Dejan Pangercic, Sarah Osentoski, Rainer Jäkel, and Rüdiger Dillmann. Context aware shared autonomy for robotic manipulation tasks. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5686–5693. IEEE, 2013.

[125] Christine Bringes, Yun Lin, Yu Sun, and Redwan Alqasemi. Determining the benefit of human input in human-in-the-loop robotic systems. In *2013 IEEE RO-MAN*, pages 210–215, 2013. doi: 10.1109/ROMAN.2013.6628447.

[126] O Burchan Bayazit, Guang Song, and Nancy M Amato. Enhancing randomized motion planners: Exploring with haptic hints. *Autonomous Robots*, 10(2): 163–174, 2001.

[127] Bharath Sankaran, Benjamin Pitzer, and Sarah Osentoski. Failure recovery with shared autonomy. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 349–355. IEEE, 2012.

[128] Fahad Islam, Oren Salzman, and Maxim Likhachev. Online, interactive user guidance for high-dimensional, constrained motion planning. *arXiv preprint arXiv:1710.03873*, 2017.

[129] Gokul Swamy, Siddharth Reddy, Sergey Levine, and Anca D Dragan. Scaled autonomy: Enabling human operators to control robot fleets. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5942–5948. IEEE, 2020.

[130] M Bernardine Dias, Balajee Kannan, Brett Browning, E Jones, Brenna Argall, M Freddie Dias, Marc Zinck, Manuela M Veloso, and Anthony Stentz. Sliding autonomy for peer-to-peer human-robot teams. In *Proceedings of the international conference on intelligent autonomous systems*, pages 332–341, 2008.

[131] David J Bruemmer, Donald D Dudenhoeffer, and Julie L Marble. Dynamic-autonomy for urban search and rescue. In *AAAI mobile robot competition*, pages 33–37. Menlo Park, CA, 2002.

[132] Brennan Sellner, Reid Simmons, and Sanjiv Singh. User modelling for principled sliding autonomy in human-robot teams. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 197–208. Springer, 2005.

[133] Katherine M Tsui, Dae-Jin Kim, Aman Behal, David Kontak, and Holly A Yanco. "i want that": Human-in-the-loop control of a wheelchair-mounted robotic arm. *Applied Bionics and Biomechanics*, 8(1):127–147, 2011.

[134] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[135] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[136] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[137] Chaitanya Mitash, Kostas E Bekris, and Abdeslam Boularias. A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 545–551. IEEE, 2017.

[138] Chaitanya Mitash, Abdeslam Boularias, and Kostas E Bekris. Improving 6d pose estimation of objects in clutter via physics-aware monte carlo tree search. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3331–3338. IEEE, 2018.

[139] Qiang Bai, Shaobo Li, Jing Yang, Qisong Song, Zhiang Li, and Xingxing Zhang. Object detection recognition and robot grasping based on machine learning: A survey. *IEEE Access*, 8:181855–181879, 2020.

[140] Abhijit Makhal, Federico Thomas, and Alba Perez Gracia. Grasping unknown objects in clutter by superquadric representation. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 292–299. IEEE, 2018.

[141] Peiyuan Ni, Wenguang Zhang, Weibang Bai, Minjie Lin, and Qixin Cao. A new approach based on two-stream cnns for novel objects grasping in clutter. *J. Intell. Robotic Syst.*, 94(1):161–177, 2019.

[142] Marios Kiatos, Sotiris Malassiotis, and Iason Sarantopoulos. A geometric approach for grasping unknown objects with multifingered hands. *IEEE Transactions on Robotics*, 37(3):735–746, 2020.

[143] Jenay M Beer, Arthur D Fisk, and Wendy A Rogers. Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of human-robot interaction*, 3(2):74–99, 2014.

[144] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[145] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635. IEEE, 2018.