Planning algorithms for complex manipulation tasks

Lertkultanon, Puttichai

2017

Lertkultanon, P. (2017). Planning algorithms for complex manipulation tasks. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73018>

<https://doi.org/10.32657/10356/73018>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**PLANNING ALGORITHMS FOR
COMPLEX MANIPULATION TASKS**

**PUTTICHA LERTKULTANON
SCHOOL OF MECHANICAL AND AEROSPACE
ENGINEERING**

2017

PLANNING ALGORITHMS FOR COMPLEX MANIPULATION TASKS

PUTTICHA LERTKULTANON

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2017

PUTTICHA LERTKULTANON

Planning Algorithms for Complex Manipulation Tasks

Nanyang Technological University

Control Robotics Intelligence Group (CRI)

School of Mechanical and Aerospace Engineering

Robotics Research Centre

50 Nanyang Avenue

N3-01a-01

Singapore 639798

Abstract

Solving manipulation tasks requires planning not only robot motions but also various interaction such as grasps (robot-object) and placements (object-environment). This indispensable interaction imparts extra complexity to the problems such that solving complex manipulation tasks, which require a number of regrasping operations, remains elusive. In this thesis, we advance the state of the art by presenting novel unimanual and bimanual manipulation planning algorithms capable of planning manipulation motions with multiple regrasping. First, we introduce a unimanual manipulation planner that explores the composite configuration space efficiently and systematically, thanks to the guidance of the novel high-level grasp-placement graph. Unlike existing methods, the graph construction does not require heavy preprocessing and is specific to only the gripper and the manipulated object. Next, we present two bimanual manipulation planners. The first one addresses specific, yet challenging, cases when bimanual grasps remain the same throughout. With the novel characterization of configuration space with closed-chain constraints, the proposed planner can plan motions across different closed-chain connected components. The second one addresses more general cases when the object can be moved only when grasped by both robots. We present a planner with certified completeness property, which guarantees that when a certificate is available for a given object and environment, the planner will find a solution to any bimanual manipulation query whenever one exists. The hardware experiment demonstrates the planner’s capability and is, to the best of our knowledge, the first to illustrate such regrasping capability, solving complex bimanual manipulation task on an actual system. Furthermore, we also present two improvements to motion planning, which indeed is a crucial component in any manipulation planning algorithm. The first improvement is an algorithm for generating time-optimal second-order trajectories subject to velocity, acceleration, and minimum-switch-time constraints. The latter constraint helps prevent concentrated acceleration switching in trajectories. The second improvement is a new bidirectional motion planner called AVP-BiRRT. The integration of the Admissible Velocity Propagation (AVP) algorithm, which enables a geometric path planner to find dynamically feasible paths, into a bidirectional planner is made possible by our newly proposed extension, AVP-Backward.

Acknowledgments

First and foremost, I would like to thank my supervisor Quang-Cuong Pham. It is he who steers me in the right direction throughout my study. I am thankful for his support, his care, and his time he has given to me, which are, I truly believe, among the best a PhD student could hope for. I appreciate how his perpetual energy and enthusiasm keep pushing me forward. We have had countless fruitful discussions and I have learned a lot from him.

I am grateful to the good friends I have here in CRI group—Huy Nguyen, Francisco Suárez-Ruiz, Ahmad Bin Anwar, Tien Hung Pham, Aditya Kapoor, and many more—for making the lab more than just a workplace, for making my PhD life healthier. We have had a lot of great discussions about both research and non-research. I would like to also thank Francisco specially for the effort he put into making the hardware work decently. It has never been easier to command the robots to move.

I am grateful to my Thai friends I have here in NTU. You have brought a great deal of fun into my PhD life. I also enjoy playing badminton with you guys.

I am eternally grateful to my parents for their unconditional, persistent love and support—from the first day of my life. Without you, I would not have come this far.

Contents

1	Introduction	11
1.1	Motivation	11
1.2	Contributions	13
1.2.1	Unimanual Manipulation Planning	13
1.2.2	Bimanual Manipulation Planning	14
1.2.3	Improvements to Motion Planning	15
1.3	Outline of the Thesis	16
2	Literature Review	19
2.1	Unimanual Manipulation Planning	19
2.1.1	Related Problems	23
2.2	Bimanual Manipulation Planning	24
2.2.1	Problem Classification	24
2.2.2	Moving an Object with Two Hands Grasping	26
2.2.3	Completeness Results in Manipulation Planning	29
2.3	Motion Planning	30
2.3.1	Approaches to Kinodynamic Motion Planning	30
2.3.2	Path-Velocity Decomposition	31
3	Fundamentals of Manipulation Planning	37
3.1	Composite Configuration Space	38
3.2	Grasps and Placements	39
3.2.1	Definitions	39
3.2.2	Stable Placement Computation	42
3.2.3	Grasp Computation	43
3.3	Trajectories	46

4	Unimanual Manipulation Planning: A Single-Query Manipulation Planner	49
4.1	High-Level Grasp-Placement Graph	50
4.1.1	Overview	50
4.1.2	Graph Construction	51
4.1.3	Guiding a Manipulation Planner via Task Plans	52
4.2	Algorithm	53
4.2.1	Example	55
4.2.2	Remarks	56
4.3	Experiments	56
4.3.1	Task Details	57
4.3.2	Descriptions of Alternative Manipulation Planners	57
4.3.3	Simulation Results and Comparisons	60
4.3.4	Hardware Experiment	62
5	Bimanual Manipulation Planning: Regrasping via <i>IK-Switch</i>	63
5.1	Problem Formulation and Analysis	64
5.1.1	Essentially Mutually Disconnectedness	65
5.1.2	Problem Formulation	66
5.1.3	Problem Analysis	66
5.1.4	<i>IK-Switch</i>	68
5.2	Algorithm	69
5.2.1	Stage 1: Planning Global Path	70
5.2.2	Stage 2: Planning <i>IK-Switch</i> Moves	73
5.2.3	Properties of the Planner	75
5.3	Experiments	75
6	Bimanual Manipulation Planning: A Certified-Complete Manipulation Planner	79
6.1	The Proposed Approach	79
6.1.1	Overview of Trajectory Generation	81
6.1.2	Main Algorithm	81
6.2	Algorithm	82
6.2.1	Generating Trajectories within a Placement Class	83
6.2.2	Generating Trajectories between Placement Classes	88
6.3	Experiments	93

6.3.1	Computing Certificates	94
6.3.2	Solving Bimanual Queries	95
6.4	Discussion	98
6.4.1	Completeness Property	98
6.4.2	Limitations and Future Works	99
7	Improvement to Motion Planning:	
	Minimum-Switch-Time Constraints	103
7.1	Trajectory Planning Pipeline	103
7.1.1	Overview	103
7.1.2	Trajectory Planning without Minimum-Switch-Time Constraints	105
7.2	Trajectory Planning with Minimum-Switch-Time Constraints . . .	110
7.2.1	Simultaneous Interpolation with Zero Terminal Velocities .	111
7.2.2	Independent Interpolation with Arbitrary Terminal Velocities	113
7.3	Simulations	122
7.3.1	Comparison with an Optimization-Based Approach	123
7.3.2	Trajectory Planning and Shortcutting with Minimum-Switch- Time Constraint	125
7.3.3	Comparison with an Existing Method	126
8	Improvement to Motion Planning:	
	Extension to Admissible Velocity Propagation Algorithm	131
8.1	Time-Optimal Path Parameterization (TOPP)	132
8.2	Admissible Velocity Propagation	135
8.2.1	Overview	135
8.2.2	Combining AVP with (Unidirectional) Sampling-Based Planners	136
8.3	AVP-Backward: The Proposed Extension	138
8.3.1	Algorithm	138
8.3.2	Integration into Bidirectional Planners	142
8.4	Example Application: Dynamic Non-Prehensile Object Transportation	144
8.4.1	Task Description	144
8.4.2	Solution using AVP-BiRRT	145
9	Conclusion	151
9.1	Summary	151

9.2	Future Works	153
A	Closed-Chain Trajectory Execution and Control	155
B	Appendix to Minimum-Switch-Time Constraints	157
B.1	Proof of Time-Optimality	157
B.2	Velocity Profile Re-Interpolation with Fixed Total Duration	159
B.2.1	Velocity Profile Re-Interpolation into Two-Ramp	160
B.2.2	Velocity Profile Re-Interpolation into Three-Ramp	160
C	Dynamic Non-Prehensile Object Transportation:	
	Reduction of Constraints into the Canonical Form	163
C.1	Unilaterality Constraint	164
C.2	Non-Slippage Constraint	165
C.3	ZMP Constraint	167
	Bibliography	171

Chapter 1

Introduction

1.1 Motivation

Industrial automation by robots has been playing a considerable role in driving economic growth across the globe in the past decades (Graetz and Michaels 2015). Recent statistics also reveal steadily growing demands of industrial robot supplies worldwide (International Federation of Robotics 2016). However, the automation is still limited to relatively simple and repetitive tasks such as welding, spray painting, and ones in electronics or automotive production lines. Each robot needs only to perform a single task repeatedly in a strictly controlled environment. Automation of manipulation tasks such as furniture assembly, on the other hand, has thus far remained elusive. Such automation, if achieved, will undoubtedly boost productivity and production standards in various industries world wide. There are a number of major hurdles to overcome along the road towards full automation of manipulation tasks. One of them is indeed the planning of robot manipulation motions.

Unlike general robot motion planning, which concerns generation of *collision-free* robot motions, manipulation planning consists in planning robot motions to actually *interact* – through contacts – with movable objects. This required interactions present major challenges in planning since any robotic system tackling a manipulation task would, at least, necessitate the abilities to (i) reason about the geometries of the movable object to be manipulated, as well as the environment; and (ii) plan appropriate sequence of interactions with the object to achieve the desired result, usually specified as a goal object pose.

In this thesis, we are interested in developing capabilities of automatic and efficient generation of *regrasping motions*—sequences of ungrasping and grasping operations—for a robotic system to manipulate an object in order to complete *complex* manipulation tasks—the tasks that require one or more regrasping operations.

Figure 1.1: A scenario where a robot needs to grab an assembly part on a table (a) in such a way that allows another robot to perform some operations, such as screwing, on the same piece (b).

Figure 1.2: A scenario where two robots have to move a piece of furniture (a) to some particular pose (b) to facilitate further process.

Imagine an assembly step where a robot needs to grab an assembly part in such a way that allows another robot to perform some operations, such as screwing, on the same piece (see, for example, Figure 1.1). Or Imagine a situation where two robots have to move a piece of furniture to some particular pose for a finishing process (see, for example, Figure 1.2). Notice that in both scenarios shown in Figure 1.1 and Figure 1.2, there exists no single grasp that allows the system to move the object to the designated pose in one step. Hence, the robot (or robots) needs to perform *regrasping several times* until the desired object pose is attained. From these examples, one can see that a system capable of planning and executing such complex manipulation operations therefore requires

- (i) the ability to reason about the geometries of the movable object to be manipulated, as well as the environment, so as to know how to grasp the object and how the object can be stably placed down; and
- (ii) the ability to plan appropriate sequences of interactions with the object to achieve the desired result since the current choice of grasp affects how the object can be placed down in the next step, and how the object is placed down affects how the robot (or robots) can grasp and pick it up, and so on.

Although the manipulation problem with regrasping has been first studied three decades ago (Tournassoud, Lozano-Pérez, and Mazer 1987), it has not received much attention until these recent years. Existing regrasping planners rely on some form of *discretization* of the sets of grasps and object placements or random roadmaps, which entails the problem of heavy pre-processing as well as potential loss in crucial grasp and placement information. Moreover, although there are several hardware experiments on unimanual systems regrasping objects (see, e.g., Lozano-Pérez et al. 1987; Stoeter et al. 1999; Cho, Kim, and Song 2003; Wan and Harada 2015b), there has not been any system – either unimanual or bimanual – that exhibits the regrasping capability that can solve regrasping tasks in the same level of complexity such as the ones shown in Figure 1.1 and Figure 1.2, in which multiple regrasping operations are required. This lack of effective and efficient solutions to regrasping problems motivates us to study and propose new manipulation planners for solving complex manipulation tasks. The capabilities enabled by our proposed planners will definitely be a firm step towards full automation of manipulation tasks.

1.2 Contributions

Due to its nature, a manipulation task with regrasping operations can be addressed in two planning layers: high-level *task planning* and low-level *motion planning*. In task planning, a regrasping sequence that solves a particular task is found while in motion planning, an actual motion realizing each regrasping step is found. While the two main contributions of this thesis lie in the task planning layer, we also made contributions in the motion planning layer.

1.2.1 Unimanual Manipulation Planning

To solve a regrasping problem, a planner necessarily needs information on how the object can be picked up (i.e., the grasp set), how the object can be placed down (i.e.,

the placement set), and interrelation between them. As both grasp and placement sets are usually infinite as the robot can slide along the grasped link and the object can continuously translate and/or rotate on the support surface, existing works approximate the interrelation, or the connectivity of the composite configuration space, by means of discretization of grasp and placement sets or creating random roadmaps. Such approaches suffer from extensive pre-processing, which furthermore has to be redone each time the environment changes. **The first contribution of this thesis is a single-query manipulation planner** that

- explores the composite configuration space systematically and efficiently by the guidance of the novel *high-level Grasp-Placement Graph*; and
- has light pre-processing, which does not involve any discretization, and is specific to only the gripper and the movable object.

1.2.2 Bimanual Manipulation Planning

Large and/or heavy objects are best manipulated using two hands. While there are a large number of research works addressing bimanual manipulation planning, the problem when large/heavy objects can only be moved when grasped by both hands has received scarcely any attention.

Regrasping via IK-Switch

First we consider a more specific class of the aforementioned problems where the bimanual grasp remains the same throughout the motion. When two robot hands simultaneously grasp a single object, a closed kinematic chain is formed. The closed-chain constraint restricts the robot motions to be on a set of manifolds of lower dimension that may be further disconnected. **This thesis presents a novel concept of essentially mutually disconnectedness of closed-chain configurations and a move called IK-switch**—a regrasp operation that connects two configurations from different essentially mutually disconnected sets.

Certified-Complete Planner

For more general bimanual manipulation problems, existing works dealt with low-DOF systems by means of discretization or assumed discrete grasp and/or placement sets. They therefore do not scale to complex regrasping problems in more practical settings, such as the scenario shown in Figure 1.2. **The third contribution of**

this thesis is a *certified-complete* bimanual manipulation planner: when a certificate, which is specific to the movable object and the environment, is available, the planner is guaranteed to find a solution to *any* bimanual manipulation query whenever one exists.

1.2.3 Improvements to Motion Planning

While our main contributions lie in the task planning layer, we also make two improvements in motion planning layer. When the next grasp or placement has been decided in the task planning level, it is the motion planner that fulfills the request. Generally, there are two types of manipulation motions: *transfer motions*, along which the robot (or robots) moves while grasping the object; and *transit motions*, along which the robot (or robots) moves alone to change its grasp.

In the unimanual case, generating a transfer path is essentially similar to generating a transit path as grasping an object only slightly affect the geometry of the robot. Generating a transfer path in the bimanual case, on the other hand, is significantly different due to the so-called closed-chain constraints. Our work on bimanual manipulation planning for the cases when bimanual grasps remain the same throughout the motions can also be considered as a component for generating transfer motions in a more general bimanual manipulation planner, such as our certified-complete planner.

For transit motion generation, the problem becomes the usual motion planning problem. Since execution time is a determining factor in industrial productivity, it is desirable that a robot traverses a path fast while also respecting both *kinematic constraints*, such as joint limits and obstacle avoidance, and *dynamic constraints*, which are related to time-derivatives of robot configurations. The problem is also known as a *kinodynamic motion planning* problem.

Path-velocity decomposition is a powerful approach to addressing kinodynamic motion planning problems. The problem is decomposed into two smaller subproblems: path planning, where a geometric path is found, and time-parameterization, where a velocity profile is found. Apart from the beneficial dimensionality reduction, another appeal of this approach is that *time-optimal* path parameterization can be attained, numerically or even *analytically* in some cases. When the path is piecewise linear – which is usually the case when a sampling-based path planner is deployed – and the time-parameterization is subject to only velocity and acceleration constraints, the time-optimal parameterization can actually be com-

puted *analytically* (Kröger, Tomiczek, and Wahl 2006; Hauser and Ng-Thow-Hing 2010; Kunz and Stilman 2014). Nevertheless, time-optimal velocity profiles computed from existing methods tend to have frequent acceleration switches that are sometimes also concentrated in short time interval. Those frequent acceleration switching can potentially hamper the robot performance as well as accelerate motor wear. To alleviate the problem, **this thesis presents a new constraint called *minimum-switch-time* and a *time-optimal* second-order trajectory interpolation method subject to not only velocity and acceleration constraints but also the new minimum-switch-time constraint.**

However, one major drawback of the path-velocity decomposition approach is that the path found in the first stage must be *time-parameterizable*. To fulfill this requirement, Pham, Caron, and Nakamura (2013) proposed an algorithm called Admissible Velocity Propagation (AVP), which when integrated into a path planner, allows the planner to discover dynamically feasible motions while staying in the configuration space. The originally proposed AVP algorithm is, however, integrable into only unidirectional path planners. **This thesis presents a bidirectional planner called AVP-BiRRT, which combines the benefits of the decomposition approach and the efficiency of bidirectional planning.** Such integration of the velocity propagation method into a class of bidirectional planners is rendered possible by the newly proposed extension to AVP, called AVP-Backward.

1.3 Outline of the Thesis

This thesis consists of 9 chapters, including this introductory chapter. The rest of the thesis is organized as follows.

In Chapter 2, we review literature on manipulation planning as well as motion planning. This chapter provides a bird’s-eye view of the existing literature in the fields and describe how our contributions fit in the literature.

In Chapter 3, we give definitions and fundamentals of manipulation planning, which serve as a foundation for discussions in subsequent chapters.

The next three chapters discuss our works on manipulation planning, specifically in the task planning layer. We present in Chapter 4 a single-query unimanual manipulation planner that solves manipulation queries efficiently and systematically by utilizing the proposed novel high-level Grasp-Placement Graph. Next, in Chapter 5, we address a particular, nevertheless challenging, case of bimanual manipulation planning where bimanual grasps remain the same throughout the mo-

tions. We present a new concept of essentially mutually disconnectedness of closed-chain configurations and a closed-chain planner with a new move called IK-switch. Then in Chapter 6, we address a more general problem of bimanual manipulation planning. We present a certified-complete manipulation planner that can solve a more general class of bimanual manipulation planning where the object needs two hands grasping before it can be moved.

The subsequent two chapters are devoted to the improvements we made to motion planning. In Chapter 7, we present a new type of constraints called minimum-switch-time, which can prevent concentrated acceleration switching in trajectories. Then we present an algorithm that generates time-optimal parabolic trajectories subject to velocity, acceleration, and the minimum-switch-time constraints. In Chapter 8, we propose an extension to AVP algorithm, called AVP-Backward, that enables AVP algorithm to be integrated into bidirectional path planners. Consequently, we introduce a new bidirectional planner—AVP-BiRRT.

Finally, we conclude this thesis in Chapter 9.

Chapter 2

Literature Review

This Chapter consists of three Sections: unimanual manipulation planning, bimanual manipulation planning, and motion planning. In each Section, we review related research works and describe how the contributions of this thesis fit in the literature.

2.1 Unimanual Manipulation Planning

We focus here on industrial assembly where parallel jaw grippers are pervasive in industrial applications owing to their strength, robustness, cost-effectiveness, ease of integration, and many more. However, unlike multi-fingered hands, parallel grippers cannot realize in-hand manipulation. Once grasped, the object cannot change its relative transformation with respect to the gripper unless it is ungrasped first. To move the object from one pose to another, there might exist no single grasp that can achieve the task. Therefore, the robot equipped with a parallel gripper must *transfer* the object to some intermediate stable placement, then *transit* to a new grasp. Those operations are collectively termed **regrasping**.

A regrasping problem amounts to finding an alternating sequence of transit and transfer paths that brings the object to the desired pose. Since along a transit path, the robot moves alone, while along a transfer path, the robot moves while grasping the object, the two types of paths can only be joined together at a configuration where the robot grasps the object while the object itself is resting on a support surface. Let $\mathcal{G}$ be the set of composite configurations at which the robot is grasping the object with a valid grasp and $\mathcal{P}$ the set of composite configurations at which the object is resting stably on a support surface (for detailed definitions and treatments, see Chapter 3). The regrasping problem can then be considered as consisting in finding a (finite) sequence of composite configurations in $\mathcal{G} \cap \mathcal{P}$, at which the object

is being grasped while resting. The immediate challenge is then how to obtain a finite sequence of $\mathcal{G} \cap \mathcal{P}$ configurations that solves the problem among *infinitely* many possibilities of grasps and placements.

It has been observed, since the time this regrasping problem emerged, that infinite grasps and placements can actually be *grouped* into a finite number of grasp classes and placement classes, each class may be parameterized by a certain set of parameters (see, e.g., Tournassoud, Lozano-Pérez, and Mazer 1987; Röhrdanz and Wahl 1997; Stoeter et al. 1999). However, most of the existing works on regrasping made use of the fact only as a means to *discretize* grasp and placement sets so as to eventually get *manageable, finite* sets of grasps and placements.

In the pioneering work on regrasping (Tournassoud, Lozano-Pérez, and Mazer 1987), each grasp class was constrained to have only one varying parameter ϑ , the gripper rotation, while each placement class was constrained to have only one varying parameter φ , the object vertical rotation. This constraining served as a means to limit the complexity of the problem. Then the authors discretized the space of ϑ and φ to get a finite set of discretized points (ϑ, φ) , each point represented a combination of grasp and placement. Each point was also checked for collision and validity of robot inverse kinematics (IK) solutions before being included into a feasible set. Finally, a regrasping sequence was searched within the feasible set of (ϑ, φ) by backward chaining from the goal configuration. In Röhrdanz and Wahl (1997), the authors also constrained each grasp class and placement class to have one varying parameter and performed discretization. The authors then presented a notion of grasp-placement-grasp (gpg) triple, where each triple represented a simple regrasping operation. The first and the second grasps in a triple represented a put-down and a pick-up grasps, respectively. A solution regrasping sequence was searched over the finite set of gpg triples by forward chaining from the initial configurations. Discretization of grasps and placements has also been utilized in recent works. Cho, Kim, and Song (2003) used discretization to build a look-up table of grasps and placements then searched for a solution in the table. Wan et al. (2015), Wan and Harada (2015a), and Wan and Harada (2015b) constructed a two-layer regrasp graph where the first layer indicated how different placement classes could be connected to one another, and the second layer indicated how different grasps could be connected to one another.

Although discretization works for simple regrasping problems, it compromises the quality or even the existence of solutions to regrasping problems. Crucial grasps and placements that are necessary to solve particular regrasping queries may po-

tentially be discretized out if the resolution is too coarse. On the other hand, the computational complexity escalates quickly when discretization resolution becomes finer. Some of existing works used random sampling instead. Stoeter et al. (1999) extended the idea of gpg triples, originally presented in Röhrdanz and Wahl (1997) while using random sampling to generate gpg triples instead of discretization. However, they still considered discrete placements.

Another closely related line of research on regrasping revolves around manipulation graph (Alami, Laumond, and Siméon 1994). A manipulation graph is a graph whose nodes represent connected components of $\mathcal{G} \cap \mathcal{P}$. Two nodes have a common edge if there exists a feasible transit or transfer path connecting two configurations in the respective connected components. A manipulation graph is then considered as a tool to represent (estimated or captured) connectivity of $\mathcal{G} \cap \mathcal{P}$ connected components. In Alami, Laumond, and Siméon (1994), the authors introduced the notion of manipulation graph and presented two manipulation planning algorithms. The first algorithm addressed the cases with discrete grasps and placements. Each node of the constructed manipulation graph was directly a single $\mathcal{G} \cap \mathcal{P}$ configuration¹. An edge was a path connecting two configurations. A manipulation query was solved by searching for a solution on the graph using A^* algorithm. The second algorithm could handle infinite grasps and placements. Since the authors considered specifically the problems with a polygonal robot manipulating (grasping) a polygonal object and moving in a two dimensional space, they could construct a manipulation graph *explicitly* by using cell decomposition. Nielsen and Kavraki (2000) tackled manipulation planning problems by constructing a fuzzy probabilistic roadmap (PRM), which also played a role of a manipulation graph. The algorithm leveraged fuzzy roadmap edges to reduce the number of necessary collision checking along paths and hence computation time reduction. The planner, however, could only address discrete grasps and placements and also relied on some *a priori* information input by users. Siméon et al. (2004) also presented a probabilistic roadmap manipulation planner. The constructed roadmap also played a role of a manipulation graph, as in Nielsen and Kavraki (2000). The difference is that, unlike previously proposed planners, their planner could handle infinite grasps and placements.

Table 2.1 categorizes existing works on unimanual regrasping based on how they treated grasp and placement sets. One can see that only the second algorithm presented in Alami, Laumond, and Siméon (1994) and the PRM planner presented in Siméon et al. (2004) can handle infinite grasps and placements. Nevertheless, the

¹The construction was possible since there were only a finite number of grasps and placements.

Table 2.1: How various existing works on regrasping treated grasp and placement sets.

	discrete placement set	continuous placement set
discrete grasp set	Tournassoud, Lozano-Pérez, and Mazer (1987); Alami, Laumond, and Siméon (1994) (Algorithm 1); Röhrdanz and Wahl (1997); Nielsen and Kavraki (2000); Cho, Kim, and Song (2003); Wan et al. (2015)	-
continuous grasp set	Stoeter et al. (1999)	Alami, Laumond, and Siméon (1994) (Algorithm 2); Siméon et al. (2004)

algorithm in Alami, Laumond, and Siméon (1994) addressed only the specific case of a polygonal robot with a polygonal object in two dimensional space.

While being able to handle infinite grasp and placement sets, the planner in Siméon et al. (2004) explored the connectivity between different connected components of $\mathcal{G} \cap \mathcal{P}$ in a rather unsupervised manner. At a composite configuration in a connected component of $\mathcal{G} \cap \mathcal{P}$, the planner only had information of adjacent $\mathcal{G} \cap \mathcal{P}$ components, to which an edge from the current configuration could possibly be generated. However, it had no information as to how the attempted extension would contribute to any progress towards the goal configuration. This approach is suitable in the cases when the planning environment is relatively constrained such as the example problem presented in Siméon et al. (2004), where the robot needed to repeatedly execute transfer paths within the same connected component of $\mathcal{G} \cap \mathcal{P}$, in order to move the object out of the cage attached to the floor. However, in more realistic industrial settings, it is arguable that the environment may be cluttered but not tightly constrained. Repetitive motions lying within the same connected component of $\mathcal{G} \cap \mathcal{P}$ can potentially be considered unnecessary. Using this planner then tends to spend more time exploring the composite configurations space and generate a longer regrasping sequence, due to unsupervised composite configuration space exploration.

To address this problem, we present in Chapter 4 a regrasping planner that explores the composite configuration space systematically and efficiently thanks to the guidance of the novel *high-level grasp-placement graph*. The graph captures potential connectivity between different connected components of $\mathcal{G} \cap \mathcal{P}$ while not requiring discretization nor heavy pre-processing as in previous methods. There-

fore, the proposed planner can also handle infinite grasps and placements. Moreover, the constructed high-level grasp-placement graph is specific to only the robot and the object and hence does not require to be recomputed when the environment changes.

2.1.1 Related Problems

There are some problems that are closely related to the regrasping problem. Some can be considered as variants of the regrasping problem while others are bigger classes of problems that contain regrasping as a special case. We review some of them here.

Motion Planning with Movable Obstacles

The problems in this domain generally deal with the scenario of one robot manipulating multiple movable bodies. Challenges in solving these problems arise from the large number of possibilities of moving objects sequentially. Complexity varies depending on whether or not the goal positions of the objects are specified (Wilfong 1988). An example where some or none of the goal positions are specified is the navigation among movable obstacles (Stilman et al. 2007), where the robot needs to reach for a particular object in a cluttered environment. In order to do so, the robot needs to remove some obstructing objects out of the way. Problems with specified goal positions for all objects are also called object rearrangement problems (Ota 2004; Krout and Bekris 2015).

These aforementioned problems, however, do not consider different placement classes of the objects: they are picked up from and put down on the same placement class. The focus is more on how moving one object affects the possibilities of moving other objects and how to move them sequentially so as to achieve the goal.

Motion Planning with Multiple Actions

A regrasping problem can be considered as a special case of motion planning with multiple actions where there are two types of available actions that the robot can execute: transit and transfer. These problems can be solved in the framework of *task and motion planning* (see, e.g., Cambon, Alami, and Gravot 2009; Kaelbling and Lozano-Pérez 2010; Garrett, Lozano-Pérez, and Kaelbling 2015), where the problems are modelled symbolically. A planner solves a manipulation planning problem by selecting actions such as pick, move, or place, verifying their pre-conditions, and

reasoning out their consequences. On the other hand, Hauser et al. (Hauser and Latombe 2010; Hauser and Ng-Thow-Hing 2011) proposed a framework for multi-modal manipulation planning in which a system is capable of executing multiple (family of) modes of motions. Within this framework, a regrasping system is capable of executing two families of modes: transit family and transfer family. Each family consists of infinitely many modes².

Although a regrasping problem can also be solved using these approaches for more general classes of problems, these approaches do not capture sufficiently the intricacy of regrasping problems, such as the special topology of the set $\mathcal{G} \cap \mathcal{P}$. To effectively and efficiently solve a regrasping problem, a more specific reasoning about grasps and placements, as well as interrelation between them, is required.

2.2 Bimanual Manipulation Planning

2.2.1 Problem Classification

Bimanual manipulation concerns the use of two robotic arms to manipulate movable objects. There are a wide spectrum of works on bimanual manipulation considering different types of objects, different kinds of manipulation tasks, etc. To localize our works and contributions in this broad field of research, we use a dichotomous key, as shown in Figure 2.1. Keys for distinguishing features are labeled in red, while the category where our works fall into is labeled in blue.

Here, we localize our works using dichotomous key based on the qualities and quantity of the objects manipulated by a bimanual system (see Figure 2.1). We briefly describe some related works below.

- (i) Non-rigid objects. Example applications include cloth folding (Maitin-Shepard et al. 2010; van den Berg et al. 2011; Bersch, Pitzer, and Kammel 2011) and metal sheet bending (Zheng and Chen 1994).
- (ii) Rigid, articulated objects. Pflueger and Sukhatme (2015) discussed a problem of a bimanual robot manipulating a foldable chair. The task was solved via searching over discretized chair states. In Liu and Dai (2003), the authors modelled cartons as articulated mechanisms and presented a trajectory generation method for carton folding tasks.

²For example, each mode in a transit family corresponds to one object placement; and there are infinitely many placements.

Figure 2.1: Dichotomous key based on the qualities and quantity of the movable objects.

(iii) Multiple objects. There are a few subcategories including:

- Bimanual assembly (Edsinger and Kemp 2008; Hwang, Lee, and Chung 2007; Surdilovic et al. 2010; Shauri and Nonami 2011). Works in this area focus more on control of interactions between assembly parts.
- Two robots sharing the same workspace (Li and Latombe 1997; Zacharias et al. 2010).

With the keys presented in Figure 2.1, our works have been classified into bimanual manipulation with single rigid non-articulated object. Existing works in this area may be categorized into one of the following paradigms:

- (a) Discussing only reaching motions. Works in this area are interested in planning motions for two robot arms to reach their grasping positions and hence more related to multi-arm path planning where robot arms move independently but coordinatively to reach their goals without colliding with one another. In Gharbi, Cortés, and Siméon (2009), a multi-arm system is decomposed into parts and a roadmap is computed for each part. Then they are combined together to get a multi-robot roadmap. Vahrenkamp et al. (2009b) used visual servoing to drive

arms to grasping positions. Various planners for dual-arm systems were presented in Vahrenkamp et al. (2009a) and Vahrenkamp et al. (2010).

- (b) Passing an object from one hand to the other. These motions can be seen as a way to increase the workspace volume of the system or to help solve single-arm manipulation problems more easily. In Saut et al. (2010), the task was to grasp the object from the initial pose by one hand, pass it to the other, and put it down. The task was actually *pre-decomposed* into four subtasks. To find a regrasping object pose, optimization was performed on the 3D environment grid to minimize wrist motions. Balaguer and Carpin (2012) addressed a similar task. The authors combined optimization, image processing, and machine learning to find the object passing configurations that minimized the execution time. Harada, Tsuji, and Laumond (2014) extended manipulation graph (Siméon et al. 2004) to bimanual systems while Wan and Harada (2016) extended regrasp graph (Wan et al. 2015) to bimanual systems.
- (c) Two arms moving an object. The movable object is large and/or heavy such that it can only be moved when grasped by both arms. This is the category our works fall into. We will discuss this topic in details in the following.

2.2.2 Moving an Object with Two Hands Grasping

First we discuss a smaller, nevertheless challenging, class of bimanual manipulation planning problems where the (bimanual) grasp remains the same throughout the motion (Gharbi, Cortés, and Siméon 2008; Cohen, Chitta, and Likhachev 2013). As a closed kinematic chain is formed whenever two robot arms concurrently grasp a single object, this class of problems can potentially be considered as an instance of closed-chain motion planning.

Closed-chain constraints restrict valid closed-chain configurations to be in a set of submanifolds lying in a higher dimensional ambient configuration space (Burdick 1989). These manifolds may be connected via the set of singularities that is of lower dimensions (Burdick 1989; Thomas 1993; Gharbi, Cortés, and Siméon 2008). Most existing works have been focusing on generating closed-chain configurations and closed-chain motions in this manifold – or connected component – level (see Chapter 5 for more elaborated details on this so-called *level* of closed-chain planning). To generate a closed-chain configuration, Han and Amato (2000) proposed to break the chain into two subchains. A configuration of one subchain can then be sampled

freely while the configuration of another subchain is computed via inverse kinematics. This sampling method was further refined in Cortés, Siméon, and Laumond (2002). Yakey, LaValle, and Kavraki (2001) presented gradient descent-based methods to sample closed-chain configurations and to generate closed-chain motions. In Trinkle and Milgram (2002), the authors presented a complete closed-chain planner that, however, did not take into account any obstacles. In Tang et al. (2010), the authors proposed a new method for sampling closed-chain configurations based on a new representation of a closed-chain system as a hierarchy of (variable-length) virtual links. A sampler based on robot reachable volumes was proposed in McMahon, Thomas, and Amato (2014), which can be applied to sample configurations with closure constraints. Porta, Jaillet, and Bohigas (2012) presented a path planner that operates on a manifold by constructing charts, each parameterizing a local region on the manifold. A search-based planner with motion primitives was presented in Cohen, Chitta, and Likhachev (2013), which can solve simple bimanual tasks.

While planning closed-chain motions in this so-called component level is relatively well-understood, the planning in the *global* level, i.e., across different connected components, remains a major hurdle when deploying a bimanual system to address practical tasks. Most results on the global-level planning exist only for planar parallel mechanisms (see, e.g., Au, Chen, and Chung 2012; Au, Chung, and Chen 2014). Motion planning presented in these works relied on discretization of the configuration space, the complexity of which escalates exponentially with respect to the number of DOFs. Gharbi, Cortés, and Siméon (2008) proposed a planner that could plan closed-chain motions across different components by exploiting singular configurations of the system. This specific method, however, limits possible solutions that the planner can discover since such motions across different components can be singularity-free (Innocenti and Parenti-Castelli 1998). Furthermore, although the planner did not utilize discretization, it assumed the availability of analytic inverse kinematic and the definite knowledge of the components structure, which allows identifying *which* component a given configuration belongs to. Unfortunately, to the best of our knowledge, there exists no *efficient* method for general systems that can identify if two given closed-chain configurations belong to the same connected component. To address this issue, we propose a notion of *single-transfer connectedness*, which characterizes the structure of the set of closed-chain configurations in a practical way. Then we present a new move called IK-switch that helps bridge different single-transfer connected components together. The move exploits the fact that the closed-chain formed by two robot grasping an object can be *broken* and

re-formed. Finally, we present a closed-chain planner that is able to plan motions across different single-transfer connected components.

When addressing more complex bimanual manipulation queries, in many cases it is necessary for the robots to *regrasp* the object before they could bring the object to the goal pose. In a pioneering work, Koga and Latombe (1992) considered a two-dimensional problem setting with two 3-DOF robots manipulating one object. They presented three bimanual manipulation planners. The first planner assumed no robot self-collision. It planned a trajectory for the object first then planned the arms motions to track the object accordingly. The second planner removed the assumption made in the first planner. It constructed a manipulation graph (Alami, Laumond, and Siméon 1994) by means of discretization then searched on the graph to find a solution path. The third planner was based on a potential field method. In Koga et al. (1994), the authors addressed planning of two human arms to grasp and regrasp an object in a three-dimensional world. The proposed planner operates by first generating an object path, then generating the arms motions to transfer the object along the path, similar to the work in Koga and Latombe (1992). One implicit assumption made in the aforementioned works was that regrasping operations could be performed any time when needed, i.e., an object can be grasp and hold *in-air* with only one arm. This is not the case when a bimanual system manipulates a large piece of furniture, for example. In Lee, Lozano-Pérez, and Kaelbling (2015), the authors solved a manipulation planning problem by sequentially generating a sequence of object contact states, object poses, and manipulator contact points. However, they did not take into account robot kinematics.

One can see that while unimanual manipulation is a relatively established research field with solid theoretical foundations and a number of working demonstrations (see, e.g., Siméon et al. 2004; Lertkultanon and Pham 2015) and even other aspects of bimanual manipulation have been much explored, results regarding cases when a movable object needs to be handled by two hands are still scarce. Considering this, we propose a new bimanual manipulation planner to specifically address this case. The key specificity of our algorithm is that it is *certified-complete*: for a given object and a given environment, we provide a certificate that the algorithm will find a solution to any bimanual manipulation query whenever one exists. Moreover, the certificate is constructive: at run time, it can be used to quickly construct a solution to a given query. In the next Section, we review completeness results of existing manipulation planning algorithms.

2.2.3 Completeness Results in Manipulation Planning

Proofs of probabilistic completeness have been obtained for some classes of motion planning algorithms, under more or less restrictive and verifiable assumptions (see, e.g., LaValle 2006; Caron, Pham, and Nakamura 2017). For manipulation planning, it becomes much more difficult because, in addition to the motion planning level, manipulation planners include task planning level, whose completeness properties are difficult to formalize or prove. Nevertheless, there are some completeness results related to a broader class of problems, so-called *multi-modal motion planning*. In this class of problems, configuration spaces have multi-modal structure where each mode limits possible motions of the system to a submanifold. From the perspective of multi-modal planning, the manipulation planning problem that we address here has two available *family of modes* of motions consist of *transit* and *transfer* and each family comprises infinitely many modes. For example, each mode in the transit family corresponds to one object placement.

In Hauser and Latombe (2010), the authors presented a probabilistically complete multi-modal planner, which, however, applicable to only problems with *finite* number of modes. Later in Hauser and Ng-Thow-Hing (2011), the authors presented a planner called Random-MMP, which can deal with infinite number of modes. Despite the probabilistic completeness guarantee, using Random-MMP directly to solve these bimanual regrasping tasks poses critical disadvantages. This is mainly due to unsupervised mode switches: consider for example when the object is at rest on a supporting surface without being grasped by any robot (i.e., the configuration is in a transit mode), Random-MMP will proceed by simply sampling an adjacent mode, which is essentially sampling any grasp (transfer mode). Without utilizing knowledge of how grasps and placements correlate, as is done in, e.g., Lertkultanon and Pham (2015), the planner can be very slow since it indeed needs to randomly sample a correct order of a correct combinations of grasps and placements before it can eventually reach the goal.

Nevertheless, completeness results are crucial for automation, where time is a valuable asset. Certified-completeness, for example, eliminates the need to spend computation time searching for non-existent manipulation paths. The computed certificates also help the algorithm to find shorter manipulation paths since the robots will only bring the object to different placements only if necessary.

2.3 Motion Planning

As execution time is one of the determining factors in industrial productivity, apart from being safe, the generated robot motions should be fast. Those motions are subject to both *kinematic constraints*, such as robot joint limits and obstacle avoidance, and *dynamic constraints*, which are related to time-derivatives of robot configurations. This problem is called kinodynamic motion planning problem, which is a term introduced in Donald et al. (1993). We briefly review existing approaches to kinodynamic motion planning before going into more details on the configuration space approach, which we have taken to solving these challenging planning problems.

2.3.1 Approaches to Kinodynamic Motion Planning

Trajectory Optimization

This approach iteratively *modifies* an initial trajectory so as to decrease a cost of the trajectory to some certain threshold. The initial trajectory can be any trajectory connecting the start and goal states and does not have to be feasible, i.e., it can be in collision or violate some kinodynamic constraints. The cost encodes, in particular, how much constraints are violated. The iterative trajectory modification may be done deterministically (Zucker et al. 2013) or stochastically (Kalakrishnan et al. 2011; Park, Pan, and Manocha 2012). Trajectory optimization can also be used on problems with discontinuous dynamics such as legged robot locomotion Harada et al. (2006) and Posa and Tedrake (2013). However, for long time-horizon and high-DOF systems, this approach requires solving a large nonlinear optimization problem, which is computationally challenging.

State Space Approach

The planning takes place in the state space. Each state is a robot configuration augmented with velocity coordinates. Traditional sampling-based path planners such as Probabilistic Roadmaps (PRM) (Kavraki et al. 1996) and Randomly-exploring Random Trees (RRT) (LaValle and Kuffner 2000) have been successfully adapted to directly work in the state space (see, e.g., Hsu et al. 2002; LaValle and Kuffner 2001). However, a state-space planner consequently suffers from an inherent problem of sensitivity of its performance to the deployed distance metric (Amato et al. 1998; LaValle and Kuffner 2001). For example, Euclidean distance metric can lead to poor performance even for a low-dimensional system such as a 1-DOF pendulum sub-

ject to joint torque limits (Shkolnik, Walter, and Tedrake 2009). Furthermore, unlike connecting two configurations subject to kinematic constraints, where a linear segment, for example, can be used to directly interpolate the two configurations, the manifold structure of the state space makes connecting two states subject to kinodynamic constraints significantly more challenging. Apart from that, the high dimensionality of the state space contributes to significant increase in algorithmic complexity of the problem (as the complexity is exponential in the number of DOFs). Therefore, to efficiently generate fast trajectories for manipulation tasks, we follow the configuration space approach.

Configuration Space Approach

In this approach, a kinodynamic motion planning problem is decomposed into two much smaller subproblems, namely path planning and time-parameterization. The path planning problem amounts to finding a geometric path subject to only kinematic constraints. This subproblem is purely geometric and thus can be solved in the configuration space, which is of only half the dimensionality of the state space. The second subproblem, whose dimension is only *two* (see Chapter 8 for more details), seeks to assign to a *time-parameterization function*, or equivalently a *velocity profile*, to a given geometric path such that when traversing the path with the specified velocities, the robot does not violate any given kinodynamic constraints. Apart from the beneficial dimensionality reduction, another appeal of the configuration space approach is that *time-optimal* path parameterization can also be attained, numerically or even *analytically* in some certain cases. This approach is sometimes referred to as *path-velocity decomposition* approach (Kant and Zucker 1986; Pham et al. 2016). As we follow this approach to generate trajectories, we review more thoroughly the related literature in the next Section.

2.3.2 Path-Velocity Decomposition

Computing Velocity Information

Consider a path P , generated in the path planning subproblem. A time-parameterization of P is an increasing *scalar* function $s : [0, T] \rightarrow [0, s_{\text{end}}]$. It can be alternatively seen as a velocity profile, which is the curve $\dot{s}(s)$, $s \in [0, s_{\text{end}}]$ in the s - $\dot{s}$ plane. We review here two approaches to generating the velocity information.

The first approach aims at generating a time-parameterization function $s(t)$, subject to constraints on velocity, acceleration, and/or higher-order time-derivatives

of the position³. Note that a time-parameterization function s can also be viewed as a *trajectory* of a single-axis system. Therefore, we may also refer to s as a trajectory here. Although this approach has the main limitation in that it could handle only relatively simple kinodynamic constraints, namely the direct time-derivatives of the position, it is still attractive in that a trajectory is generated analytically, and hence presumably computationally light when compared to the other approach.

Polynomial is a family of curves that are often used in trajectory generation. Different approaches to generating $s(t)$ differ on boundary conditions (for example, initial/final velocity/acceleration) and optimization objectives. Jerk-boundedness is one of the objectives that has been receiving much attention in the literature for it has been shown to induce some desirable behaviors to the resulting trajectories (Kyriakopoulos and Saridis 1988; Gasparetto and Zanotto 2008). To achieve jerk-boundedness, a polynomial needs to be of order three or more. In Kyriakopoulos and Saridis (1988), the authors generated rest-to-rest minimum-jerk trajectories by using cubic polynomials. This class of polynomials was also utilized in Haschke, Weitnauer, and Ritter (2008), where the authors proposed a method for generating time-optimal trajectories, given that the trajectories end at rest. Liu (Liu 2002) presented a seven-segment method for generating cubic trajectories. A trajectory was divided into seven segments and was limited to be one of several forms and the exact trajectory was calculated once the best form of the trajectory had been chosen. This method is, however, limited to zero boundary conditions. The method was later extended for multi-axis systems Broquére, Sidobre, and Herrera-Aguilar 2008. Quartic (fourth-order) polynomials were used in Kröger and Wahl (2010) subject to zero final acceleration. Ahn, Chung, and Youm (2004) also proposed a framework for quartic polynomial trajectory generation. The method was claimed to be computationally light. However, it did not take into account velocity bounds. Quintic (fifth-order) polynomials were used in Macfarlane and Croft (2003) subject to zero terminal acceleration. Recently, there was also the work addressing generation of polynomial trajectories of any order (Ezair, Tassa, and Shiller 2014) subject to general boundary conditions. Although high-order polynomials generate smooth trajectories, polynomials with order higher than three are not generally used due to their tendency to produce oscillation in joint positions (Piazzi and Visioli 2000; Macfarlane and Croft 2003).

Splines, a particular class of polynomials, are also prevalent in industrial robot

³Examples include jerk, which is the third-order time-derivative of the position and snap (or sometimes jounce) (Visser 2004), which is the forth-order time-derivative of the position.

trajectory generation. B-splines were used in Thompson and Patel (1987) to interpolate trajectories with zero terminal velocity and acceleration. In Piazzzi and Visioli (2000), the authors used cubic splines to interpolate minimum-jerk trajectories. The traveling time was required as an input. Cubic splines were also used in Gasparetto and Zanotto (2008), where the trajectory generation problem was formulated as an optimization problem. The objective function was set to be a weighted sum between traveling time and integral of squared jerk. In Erkorkmaz and Altintas (2001), the authors proposed a quintic spline interpolation method, which gave continuous acceleration profiles. Apart from polynomial splines, the use of trigonometric splines were also investigated in Simon and Isik (1993) to generate smooth trajectories.

Generating a time-parameterization function considering both time-optimality and jerk bounds is, however, too complex to be solved without resorting to more restricted boundary conditions (Kyriakopoulos and Saridis 1988; Haschke, Weitnauer, and Ritter 2008) or computationally expensive optimization routines (Gasparetto and Zanotto 2008). Although jerk-boundedness is a desirable property, it is *not* a necessary condition for successful trajectory execution. Therefore, in many cases, it can be preferable to trade off jerk-boundedness for time-optimality *and* computational efficiency. This can be achieved by deploying *parabolic* interpolation, which produces quadratic (second-order) trajectories, since the time-optimal parabolic interpolation method subject to velocity and acceleration bounds can be computed *analytically*. Another reason quadratic trajectories are acceptable is that a significant proportion of industrial robots are controlled in acceleration, i.e., their inputs are time-series of the joint accelerations.

Hauser and Ng-Thow-Hing (Hauser and Ng-Thow-Hing 2010) and Kröger et al. (Kröger, Tomiczek, and Wahl 2006) investigated the problem of generating time-optimal piecewise parabolic trajectories subject to velocity and acceleration bounds (note that in this case the parabolicity of the optimal trajectories is *a posteriori*). However, we have noticed that trajectories computed by the aforementioned methods tend to have a large number of switch points, time instants when any joint changes its acceleration. Furthermore, these switch points are sometimes concentrated in a short time interval, especially after a shortcutting procedure (Geraerts and Overmars 2007; Hauser and Ng-Thow-Hing 2010) has been applied. Yet, high concentration of switch points might hamper the performance of the system (for more discussion on the topic, see Chapter 7). To address this issue, we present in Chapter 7 a new constraint called minimum-switch-time, which ensures that any two consecutive switch points are separated by at least some given duration, and an

efficient, analytic algorithm for planning piecewise parabolic trajectories subject to velocity and acceleration bounds, *and* the newly introduced minimum-switch-time constraint.

The second approach aims at generating a velocity profile $\dot{s}(s)$, which can be viewed as a *curve* in the s - $\dot{s}$ plane. The problem is referred to as the Time-Optimal Path Parameterization (TOPP) problem. There are three families of methods:

- *Dynamic Programming* (Shin and McKay 1986a)—The s - $\dot{s}$ plane is divided into a grid. Then dynamic programming is used to find the time-optimal velocity profile. The complexity of the approach is then $O(N^3)$, where N is the number of discretization points on each axis.
- *Convex Optimization* (Verscheure et al. 2009; Suleiman et al. 2010; Hauser 2014)—The path is discretized into N segments and the problem of computing a velocity profile is formulated as a convex optimization of $O(N)$ variables subject to $O(N)$ equality and inequality constraints. Although being versatile, i.e., the optimization adjective can be adjusted to trade of the time-optimality with other criteria such as energy consumption, with only a fair number of discretized points along a path, it can be relatively slow compared to the numerical integration approach (Pham 2014).
- *Numerical Integration*—This family of methods is based on the Pontryagin Maximum Principle, i.e., the knowledge that the time-optimal trajectory is known to follow either maximum or minimum acceleration (Bobrow, Dubowsky, and Gibson 1985; Shin and McKay 1985; Pfeiffer and Johanni 1987). The time-optimal velocity profile can then be computed by *integrating* successively the maximum and minimum acceleration bounds (see Chapter 8, Section 8.1 for technical details of the approach). This method is generally faster than the aforementioned methods and an efficient and robust implementation of the algorithm is also openly available (Pham 2014).

Overcoming a Limitation of Path-Velocity Decomposition

In the first subproblem – path planning problem – only kinematic constraints are taken into account. Consequently, there is no guarantee that the obtained geometric path will be *time-parameterizable*. This is a critical issue since a non-parameterizable path may be discarded entirely and the planning process needs to restart, hence

inevitable increase in running time. There are two common directions to resolving this issue:

- *Modifying the path such that it is time-parameterizable.* Huang, Tanie, and Sugano (1999) followed this direction and proposed a trajectory modification scheme for trajectory planning for mobile robots under Zero-Moment Point (ZMP) constraints (Vukobratovic, Borovac, and Surdilovic 2001). A trajectory is modified so as to *pull* the ZMP to stay inside the support area of the robot. However, the applicability of the method is limited to the specific case. Moreover, it suffers from additional collision checking, which is expensive and also needs to be done every time the path is modified.
- *Enforcing, in path planning stage, quasi-static constraints to ensure the existence of time-parameterization functions.* A quasi-static path is always executable at *arbitrarily low speed*, hence automatically time-parameterizable. For example, in trajectory planning for legged robots (see, e.g., Kuffner et al. 2002), imposing quasi-static constraints amounts to enforcing the projection of the center of mass (COM) of the robot to lie in the support area at any configuration. This constraint is purely geometric and thus could be enforced in planning without leaving the configuration space, as desired.

From the above, one may see that enforcing quasi-static constraints can be a way to resolve the aforementioned problem. However, this also poses another major drawback that all *dynamically feasible* motions that are not quasi-statically feasible are overlooked. In some challenging problems, this essentially excludes *all* solution trajectories (see, for example, the experiment presented in Chapter 8, Section 8.4).

To overcome this limitation, Pham, Caron, and Nakamura (2013) proposed an algorithm, based on the numerical integration approach to TOPP, called Admissible Velocity Propagation (AVP). The algorithm, when integrated into a geometric path planner, allows the planner to discover time-parameterizable path while staying in the configuration space. The underlying mechanism of the originally proposed AVP algorithm, however, makes it applicable to only unidirectional path planners (see Chapter 8, Section 8.2 for the details of the algorithm). To address this issue, we present in Chapter 8 an extension to AVP algorithm towards the class of bidirectional planners.

Chapter 3

Fundamentals of Manipulation Planning

Manipulation Planning has emerged as a research area since at least the 1980s and is still active nowadays. Despite its long-standing existence, there have been not many attempts to mathematically formalize manipulation-related definitions and notations. Although high-level definitions of terms such as a grasp or a placement, as often given in works in manipulation planning, are adequate when, for example, discussing algorithms and their underlying ideas or intuition, they lack precision and rigor, which is ultimately crucial for deeper understanding and further *theoretical* studies of the topic.

In Alami, Laumond, and Siméon (1994), in which an important result called the reduction property was proven, although the authors gave definitions sufficient for stating the proof, those definitions were given specifically for two-dimensional problems, hence not suitable for generalization to problems with higher dimensions. On the other hand, Hauser and Latombe (2010) and Hauser and Ng-Thow-Hing (2011) presented a mathematical framework for multi-modal planning problems. The framework is general and applicable to various multi-modal problems, where the system is capable of exhibiting different *modes* or types of motions (for more discussion, see Chapter 6). Although pick-and-place manipulation can be considered as one category of multi-modal problems, the general multi-modal framework does not capture intricacy of the composite configuration space and its underlying structure specific to the task (e.g., grasp and placement classes and so on).

In this Chapter, we propose definitions and fundamentals of manipulation planning, formulated mathematically. They serve as the foundation for discussions on manipulation planning—both unimanual and bimanual. Definitions presented here, although formulated for unimanual manipulation systems, can be extended to bimanual systems without any difficulties.

3.1 Composite Configuration Space

Consider the 3D space where the manipulation system is located, called world. The world $\mathcal{W}$ consists of a robot $\mathcal{R}$, a movable object $\mathcal{O}$, and the environment $\mathcal{E}$. In this thesis, we assume that the robot is equipped with a parallel jaw gripper. The environment also includes *support surface(s)*, on which the object is allowed to rest.

Let $\mathcal{C}_{\mathcal{R}} \subseteq \mathbf{R}^n$ be the configuration space of the robot, where n is the number of DOFs of the robot, and $\mathcal{C}_{\mathcal{O}} \subseteq SE(3)^1$ be the configuration space of the object. The configuration space of the system, $\mathcal{C}$, is then a *composite* one, which is the Cartesian product of the configuration spaces of the robot and the object, $\mathcal{C} = \mathcal{C}_{\mathcal{R}} \times \mathcal{C}_{\mathcal{O}}$. Each composite configuration $c \in \mathcal{C}$ can then be represented as a pair (q, T) , where $q \in \mathcal{C}_{\mathcal{R}}$ is a robot configuration and $T \in \mathcal{C}_{\mathcal{O}}$ is an object transformation matrix.

We equip with the composite configuration space a metric $d : \mathcal{C} \times \mathcal{C} \rightarrow \mathbf{R}$ defined as a linear combination of the Euclidean distance between robot configurations and a distance between two rigid body transformations. In particular,

$$d(c_a, c_b) = \sqrt{\lambda \|q_a - q_b\|_2^2 + (1 - \lambda) w^2(T_a, T_b)},$$

where $\lambda \in [0, 1]$ and w is a distance metric in $SE(3)$. Let a point $T \in SE(3)$ be written as

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}.$$

We choose w to be

$$w(T_a, T_b) = \sqrt{\gamma \|\log(R_a^{-1} R_b)\|_2^2 + (1 - \gamma) \|p_a - p_b\|_2^2}, \quad (3.1)$$

where $\gamma \in [0, 1]$. That is, w is the weighted sum of the minimal geodesic distance between two rotations (see Park and Ravani 1997, for more details) and the Euclidean distance between two displacements. This metric is also scale-dependent left-invariant (Park 1995).

¹ $SE(n)$ is called the special Euclidean group, which is a group of rigid body transformations.

Figure 3.1: A parallel gripper with its local frame. The lateral direction is orthogonal to both finger surfaces. The sliding direction is parallel to both finger surfaces and is defined such that the approaching direction is pointing *outwards* from the gripper.

3.2 Grasps and Placements

3.2.1 Definitions

Grasps and placements are important notions in pick-and-place manipulation planning. We define them, together with other related terms and concepts, here.

Definition 1. A *grasp* is a relation between the object pose and the gripper pose.

One can represent a grasp by, e.g., a pair of relative transformations between the gripper and the object. Note that from the definition, any pair of the relative transformations *is* a grasp. However, the object can be moved only when grasped by a *valid grasp*. The set of all valid grasps are to be determined by the users, either explicitly (e.g., as a set of grasps) or implicitly (e.g., as a set of conditions to be satisfied by the gripper). Note also that there might exist many robot configurations corresponding to exactly the same grasp due to multiplicity of inverse kinematic (IK) solutions associated with the same gripper pose.

With a parallel jaw gripper, the set of all valid grasps can be parameterized by a finite set of parameters, which is not necessarily unique (Lertkultanon and Pham 2015). Consider, for example, an object composed entirely of boxes and a parallel jaw gripper such as the one shown in Figure 3.1. Grasp parameters may be defined as follows.

- l : an integer indicating the index of the link (box) that the gripper is grasping.
- a : an integer in $\{0, 1, 2, \dots, 5\}$ indicating how the gripper is approaching the object. For example, if the gripper's approaching direction is aligned with $+x$ -axis of the box's local reference frame (RF), $a = 0$; if it is aligned with $+z$ -axis of the box's RF, $a = 2$, etc.
- b : an integer in $\{0, 1, 2, \dots, 5\}$ indicating which axis of the box's local RF the gripper's sliding direction is aligned with.

- δ : a real number indicating the position of the gripper along the sliding direction. The range of δ is arbitrary. However, one may, for example, normalize the range such that δ always takes a value in the interval $(0, 1)$.

Therefore, a grasp may be written as a vector $\mathbf{g} = [l \ a \ b \ \delta]^\top$. For grasp parameterization of other classes of objects, see discussion in Section 3.2.3.

Definition 2. A **placement** refers to an object transformation at which the object is in contact with a support surface. A placement is said to be **stable** if when not in contact with any robot, the object remains stationary.

More details of stable placement computation are given in Section 3.2.2.

With the above definitions of a grasp and a placement, we can now define a *collision-free* configuration and a *feasible* configuration.

Definition 3. A composite configuration is said to be **collision-free** if there are no collisions in the world except the ones induced by valid grasps and the ones induced by placements.

Definition 4. A composite configuration is said to be **feasible** if it is collision-free and at least one of the following holds:

1. The robot is grasping the object with a valid grasp;
2. The object is at a stable placement.

Let us now consider intrinsic structure of $\mathcal{C}$. For convenience, we define a function $\pi_p : \mathcal{C} \rightarrow \mathcal{C}_O$ that projects a composite configuration $\mathbf{c} = (\mathbf{q}, \mathbf{T})$ into $SE(3)$ such that $\pi_p(\mathbf{c}) = \mathbf{T}$. There are two subsets of $\mathcal{C}$ induced by valid grasps and stable placements.

Definition 5. Grasp configuration set, $\mathcal{G}$, is the set of feasible composite configurations where the robot is grasping the object with a valid grasp.

Definition 6. Placement configuration set, $\mathcal{P}$, is the set of feasible composite configurations such that

1. $\forall \mathbf{c} \in \mathcal{P} \ \pi_p(\mathbf{c})$ is a stable placement and
2. $\forall \mathbf{c} \in \mathcal{P} \ \exists \mathbf{c}' \in \mathcal{G} \ \pi_p(\mathbf{c}') = \pi_p(\mathbf{c})$.

Figure 3.2: The gripper is set at different grasping poses from some of the available grasp classes for this L-shaped object.

Figure 3.3: The object is placed at transformations from different placement classes. For this particular object, there are 5 available placement classes as shown.

Note that this definition of the placement configuration set slightly differs from previous definitions (such as the one given in Siméon et al. (2004)) by the second requirement above. It ensures that for any placement configuration $c \in \mathcal{P}$, its corresponding placement is always reachable by some grasp.

Both $\mathcal{G}$ and $\mathcal{P}$ can be partitioned into a *finite* number of subsets, called *grasp classes* and *placement classes*, respectively (Lertkultanon and Pham 2015).

Definition 7. A **grasp class** is a subset of $\mathcal{G}$ whose configurations have the same grasp parameters l (link index) and a (approaching direction).

Figure 3.2 illustrates the concept of grasp classes by showing the gripper at different grasping poses from some available grasp classes for the object.

Now let $\mathcal{H}$ be the convex hull of the object. All stable placements can be grouped based on which surface of $\mathcal{H}$ is in contact with the support surface. Therefore, a placement class is defined as follows.

Definition 8. A **placement class** is a subset of $\mathcal{P}$ where at each configuration, the same face of $\mathcal{H}$ is in contact with the support surface.

Figure 3.3 shows the object placed at transformations from five different placement classes.

Figure 3.4: (a) A trimesh model of an IKEA chair. This model is also used in simulations presented later in Chapter 6. (b) The convex hull of the chair computed via OpenRAVE (Diankov 2010).

Although a placement class is a set of composite configurations, for convenience, we will say that two object transformations are *in the same placement class* if at both transformations, the same face of $\mathcal{H}$ is in contact with the support surface.

3.2.2 Stable Placement Computation

To compute stable placements of an object, one starts by computing *the* convex hull of the object, which is the smallest convex set that contains the object. In computer simulations, objects are usually modeled using a triangle mesh (trimesh), which comprises a set of triangles connected by their common edges or vertices. In such cases, one gives the collection of all vertices from the model as input to a convex hull algorithm (see, e.g., Chazelle (1993) and Chan (1996)). A number of robust implementations of convex hull algorithms are also openly available in, for example, *CGAL: Computational Geometry Algorithms Library* and OpenRAVE (Diankov 2010)). The output of a convex hull algorithm will be a convex 3-polytope, $\mathcal{H}$, which will be used to determine stable placement classes. Figure 3.4 shows an IKEA chair and its convex hull.

Placement Static Equilibrium Checking

We determine a placement class based on which face of the object's convex hull is in contact with the support surface.

Suppose the object $\mathcal{O}$ is placed on a support surface such that the face f of the convex hull $\mathcal{H}$ is in contact with the support surface. We will then call f the support

region. In order for f to correspond to a (stable) placement class, the object must be in static equilibrium without external help from any robot. This can be easily examined by checking if the projection of the object's center of mass (COM) along the direction of gravity onto the support surface falls *inside* the support region f (see, e.g., Bretl and Lall (2008)).

When an external wrench acts on the object at a stable placement, it may cause the object to tip over one of the edges of $\mathcal{H}$. This, in turn, displaces the projection of the COM of the object. If the original position of the projection is close to a boundary of the support region, even small disturbances can potentially displace the projection such that it lies outside of the support region; hence the object falls down. To make sure that the object at a stable placement can resist some degrees of external disturbances, one may declare a placement stable only if at this placement, the projection of the COM of the object lies inside the support region f with some clearance from any boundary of f .

Remarks

Convex hulls are not always polytopes, i.e., they may contain non-flat faces. These cases, however, does not affect stable placement computation discussed above. To see why, consider a convex hull $\mathcal{H}$ that has a non-flat face f . When the object is placed on a support surface such that the face f is in contact with the support surface, the support region will be either a point or a line. Static equilibrium can still be checked in the same way by examining if the projection of the COM of the object lies in the support region (a point or a line). Notice, however, that in this case, such a stable placement can be at most only marginally stable since the COM projection will always be on a boundary of the support region.

3.2.3 Grasp Computation

Parameterization enables categorizing infinitely many grasps into a finite number of grasp classes, where each grasp class contains grasps of similar properties. This parameterization in turn obviates the needs for discretizing the set of grasps.

With objects that are boxes or composed entirely of boxes, grasp classes can be easily computed as discussed previously in Section 3.2.1. To obtain grasp-parameterization for a general object, one may proceed by approximating each link of the object as a geometric primitive such as a box (Huebner, Ruthotto, and Kragic 2008), a cylinder, a sphere, or others (Miller et al. 2003). Then grasp classes and their cor-

Figure 3.5: Three possible grasp classes for a parallel gripper grasping a bottle. Arrows show gripper motions that result in grasps contained in the same class.

responding grasp parameters can be obtained accordingly (see, e.g., Yamanobe and Nagata 2010).

As an example, consider a bottle shown in Figure 3.5. There are three grasp classes available for the object. One grasp class is when the gripper grasps the bottle along its length. This grasp class can then be parameterized by two parameters θ and δ , where θ indicates the rotation of the gripper around the axis of the bottle and δ indicates the position, along the length of the bottle, where the gripper grasps. The other two grasp classes are when the gripper approaches the bottle from either end. Both grasp classes has only one grasp parameter θ indicating the gripper rotation.

Grasp Stability Checking

Although no grasp by a parallel gripper can be force-closed (see, e.g., Bicchi (1995) and references therein for more details on the force closure property), one can determine if a grasp is able to resist the gravitational wrench such that the grasped object remains in static equilibrium.

The problem can be formulated as a feasibility problem as follows. First, one starts with Newton-Euler equations

$$m\mathbf{g} + \sum_{i=1}^k \mathbf{f}_i = \mathbf{0}_3 \quad (3.2)$$

$$\sum_{i=1}^k (\mathbf{p}_i - \mathbf{p}_{\text{COM}}) \times \mathbf{f}_i = \mathbf{0}_3, \quad (3.3)$$

where m is the object mass, $\mathbf{g}$ the acceleration due to gravity, $\mathbf{p}_{\text{COM}}$ the position of the COM of the object, k the number of contact points on the object (in this case, the contact points between the gripper and the object), $\mathbf{p}_i$ the position of the i^{th} contact point, and $\mathbf{f}_i$ the contact force exerted at $\mathbf{p}_i$. Note that in case when the

contact is a surface contact, one can still suppose that the contact forces are acting at the vertices of the contact polygon (Caron, Pham, and Nakamura 2015b). From the equations above, one can reformulate them as a matrix equation

$$\mathbf{w}_{\text{GI}} = \mathbf{G}\mathbf{f}_{\text{all}}, \quad (3.4)$$

where $\mathbf{w}_{\text{GI}} = \begin{bmatrix} -m\mathbf{g} \\ -m(\mathbf{p}_{\text{COM}} \times \mathbf{g}) \end{bmatrix}$, $\mathbf{G} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{I}_3 & \cdots & \mathbf{I}_3 \\ [\mathbf{p}_1] & [\mathbf{p}_2] & \cdots & [\mathbf{p}_k] \end{bmatrix}$, and

$\mathbf{f}_{\text{all}} = [\mathbf{f}_1^\top \mathbf{f}_2^\top \cdots \mathbf{f}_k^\top]^\top$. The vector $\mathbf{w}_{\text{GI}}$ is the gravito-inertial wrench (Caron, Pham, and Nakamura 2015a). The operator $[\cdot]$ maps a three-dimensional vector to a unique 3×3 skew-symmetric matrix. Now one has that the current configuration is in static equilibrium if and only if there exists a vector of contact forces $\mathbf{f}$ that verifies Equation (3.4) subject to constraints on contact forces $\mathbf{U}\mathbf{f} \preceq \mathbf{b}$ (e.g., max grip force, linearized friction cone constraints). This feasibility problem can be solved via a linear programming solver or other available methods (see, e.g., Caron, Pham, and Nakamura 2015a; Prete, Tonneau, and Mansard 2016).

Apart from grasp static equilibrium checking, one can also determine if a grasp is able to resist external wrenches caused by the grasping robot moving along some given robot trajectory by using the same procedure as discussed above.

Remarks

Note also that to further increase the size of the grasp set, one can include more grasp parameters to allow more degrees-of-freedom between the gripper and the object. For example, the gripper may be allowed, after approaching the object, to rotate about its sliding axis (see Figure 3.1). This can be done easily by including one more grasp parameter α that indicates the angle of such rotation. While such inclusion of more grasp parameters allows more grasping freedom, it also increases the size of the search space of a manipulation planner, hence it may potentially increase the running time of the planner. From our simulations and experiments, to be presented later in Chapter 4 and Chapter 6, our manipulation planners work well and are able to find solutions to manipulation queries even without the help of such extra grasp parameters.

3.3 Trajectories

A **single-mode** path is defined as a continuous function P from the unit interval $[0, 1]$ to a level set of $\mathcal{G}$ or $\mathcal{P}$. There are two types of valid *single-mode* paths in the composite configuration space.

Definition 9. A **transit path** maps the unit interval $[0, 1]$ into a level set of $\mathcal{P}$.

For any two composite configurations (q_1, T_1) and (q_2, T_2) on the path, one always has that $T_1 = T_2$. Along a transit path, the robot moves alone while the object is placed at some stable configuration.

Definition 10. A **transfer path** maps the unit interval $[0, 1]$ into a level set of $\mathcal{G}$.

Along a transfer path, the robot moves while rigidly grasping the object with a valid grasp.

Next, we define a binary operation called **composition**. The *composition* of two single-mode paths P_1 and P_2 is denoted as $P_1 * P_2$ and is defined by

$$(P_1 * P_2)(s) = \begin{cases} P_1\left(\frac{2s}{t}\right) & 0 \leq s \leq \frac{t}{2}, \\ P_2\left(\frac{2s}{t} - 1\right) & \frac{t}{2} \leq s \leq t, \end{cases} \quad (3.5)$$

where $t = 1$ if both paths are of the same type and $t = 2$ otherwise. Note that the composition operation is only defined when $P_1(1) = P_2(0)$. From the above definition, when P_1 and P_2 are of the same type, the resulting path $M = P_1 * P_2$ is also a single-mode path of the same type (and hence is still parameterized by $s \in [0, 1]$). When P_1 and P_2 are of different types, the resulting path M is, in a sense, *longer*, i.e., it is parameterized by $s \in [0, 2]$. We use $|M|$ to denote the **parameterization domain length** or simply **domain length** of M . By definitions, all single-mode paths have unit domain length.

Let $\prod_{i=a}^b P_i = P_a * P_{a+1} * \dots * P_{b-1} * P_b$, where $b \geq a$. Since the composition operation is associative², a composition of k single-mode paths $\prod_{i=1}^k P_i$ can be

²Associative property of composition operation is inherited from the composition of relations.

computed, recursively, as

$$\begin{aligned} \left(\prod_{i=1}^k P_i \right) (s) &= \begin{cases} (P_1 * P_2)(s) & 0 \leq s \leq t_1 \\ M_2(s - t_2) & t_2 \leq s \leq |M_2| + t_2 \end{cases} \\ &= \begin{cases} M_1(s) & 0 \leq s \leq |M_1| \\ (P_{k-1} * P_k)(s) & |M_1| \leq s \leq |M_1| + t_1, \end{cases} \end{aligned}$$

where $M_1 = \prod_{i=1}^{k-2} P_i$, $M_2 = \prod_{i=3}^k P_i$, and t_1 and t_2 are 1 if both single-mode paths are of the same type and are 2 otherwise.

We are now ready to define a manipulation path.

Definition 11. A **manipulation path**, M , is a composition of single-mode paths, i.e., $M = \prod_{i=1}^k P_i$, $k \geq 1$. A manipulation path is said to be **irreducible** if no two consecutive single-mode paths are of the same type, i.e., it is strictly an alternating composition of transit and transfer paths.

Any composition $\prod_{i=1}^k P_i$ can always be written as an irreducible manipulation path $\prod_{j=1}^l P'_j$ with $l \leq k$. For an irreducible manipulation path M' , we define the number of **transitions** (between consecutive transit and transfer paths or vice versa) as $|M'| - 1$.

Next, we introduce a notion of *single-transfer connectedness*.

Definition 12. Two composite configurations c_1 and c_2 are **single-transfer connected** if there exists a transfer path whose terminal configurations are c_1 and c_2 .

Definition 13. A **single-transfer connected set** $\mathcal{F}$ is a set in which any $c_1, c_2 \in \mathcal{F}$ are single-transfer connected.

Single-transfer connected sets are indeed sets induced by transfer paths. Therefore, they are specific to grasps of the transfer paths. We then define $\mathcal{F}(g)$ as the union of all $\mathcal{F}$ where the grasp associated with any composite configuration $c \in \mathcal{F}$ is specified by the grasp vector g .

In single-arm (unimanual) cases, suppose that the environment is *not* relatively constrained³, for example, when the world consists of only the robot, the movable object, and the support surface, the set $\mathcal{F}(g)$ will comprise only one single connected component. Given two composite configurations c_1 and c_2 , where $c_1 \neq c_2$

³An example of constrained environments is the one presented in Siméon et al. (2004), where the movable object was placed within a cage such that it could not be taken out of the cage by a single transfer path.

and they have different object transformations and grasps, i.e., two composite configurations where the robot is grasping the object with the same grasp but different IK solutions, it is not difficult to find a transfer path to move from one configuration to another. This is not the case, however, for bimanual systems since the system is severely constrained by *closed-chain* constraints as well as joint limits. Characterizing these single-transfer connected sets plays a crucial role in bimanual manipulation planning, as we shall see in Chapter 5 and Chapter 6.

Conclusion

In this Chapter, we mathematically formulate definitions of terms related to pick-and-place manipulation planning, which will later on facilitate discussions on manipulation planning in the sequel.

Chapter 4

Unimanual Manipulation Planning: A Single-Query Manipulation Planner

In manipulation tasks, the configuration space is the Cartesian product of the configuration space of the robot and that of the movable object. It is the complex structure of such a composite configuration space that makes manipulation planning particularly challenging. Previous works approximate the connectivity of the composite configuration space by means of discretization or by creating random roadmaps. Such approaches involve extensive pre-processing, which furthermore has to be redone each time the environment changes. In this Chapter, we propose a **high-level Grasp-Placement Graph**, similar to that proposed by Tournassoud, Lozano-Pérez, and Mazer (1987), but which does not require any discretization or heavy pre-processing. The graph captures the potential connectivity of the composite configuration space while being specific only to the movable object. In particular, it does not require to be recomputed when the environment changes. During the query phase, the graph is used to guide a tree-based planner to explore the space systematically. Our simulations and experiments show that the proposed method enables improvements in both running time and trajectory quality as compared to existing approaches. This work has been published in **Lertkultanon, Puttichai and Quang-Cuong Pham (2015)**. “A single-query manipulation planner”. In: *IEEE Robotics and Automation Letters* 1.1, pp. 198–205. DOI: 10.1109/LRA.2015.2513731.

4.1 High-Level Grasp-Placement Graph

Any transit path, along which the robot moves alone while the object remains at a stable placement, lies entirely in the placement configuration set $\mathcal{P}$. Similarly, any transfer path, along which the robot moves while grasping the object with a valid grasp, lies entirely in the grasp configuration set $\mathcal{G}$. The subset $\mathcal{G} \cap \mathcal{P}$ of the composite configuration space is therefore central to solving pick-and-place manipulation planning problems because it is the only place where transitions between transit and transfer paths may occur.

The high-level Grasp-Placement Graph is constructed to capture potential connectivity of the subset $\mathcal{G} \cap \mathcal{P}$. By constructing and using the graph, we decouple a pick-and-place manipulation planning problem into two layers of planning. The *high-level* task planning layer consists in finding a sequence of $\mathcal{G} \cap \mathcal{P}$ configurations that answers the query, while the *low-level* motion planning layer consists in finding *actual* motions between the $\mathcal{G} \cap \mathcal{P}$ configurations. Note that our emphasis is on the high-level planning, i.e., how we construct and use a high-level Grasp-Placement Graph, addressing uncertainty, which takes place at the second layer, is beyond the scope of this work.

4.1.1 Overview

A high-level Grasp-Placement Graph is an undirected, unweighted graph whose nodes represent different subsets of $\mathcal{G} \cap \mathcal{P}$. According to the partitioning of $\mathcal{G}$ and $\mathcal{P}$ into grasp and placement classes (as discussed in Chapter 3), the set $\mathcal{G} \cap \mathcal{P}$ is then partitioned into finite disjoint subsets. Each subset of $\mathcal{G} \cap \mathcal{P}$ is the intersection between a particular pair of placement and grasp classes. Therefore, a node in the graph can be represented by a pair of integers – a placement class index and a grasp class index. We visualize a high-level Grasp-Placement Graph in the similar way as the authors of Tournassoud, Lozano-Pérez, and Mazer (1987) did for their Grasp-Placement Table. Note that while each node in a Grasp-Placement Table presented in Tournassoud, Lozano-Pérez, and Mazer (1987) represents a pair of *particular* grasp and placement, each node in our high-level Grasp-Placement Graph represents a pair of grasp and placement *classes*. Our graph is plotted on a two-dimensional grid. Each vertical line corresponds to a placement class whereas each horizontal line corresponds to a grasp class. Intersections of vertical with horizontal lines then represent subsets of $\mathcal{G} \cap \mathcal{P}$.

If there were no collision avoidance constraints and kinematic reachability constraints, any combination of grasp and placement classes would have been possible. Any pair of nodes on the same vertical or horizontal line will be connected. Imposing those constraints makes some nodes and edges infeasible. However, verifying all constraints at the graph construction phase entails costly computations. Also, by checking kinematic reachability, the graph will become robot- and environment-dependent: it will need to be re-computed when the environment changes. Therefore, we propose to construct the graph by verifying only collision avoidance constraints (between the gripper and the support surface). Robot kinematic reachability constraint verification (IK computation) is postponed until the planning phase.

4.1.2 Graph Construction

In the first step, we check for feasibility of graph nodes. For each placement class, we place the object at a nominal location on the support surface. Then for each box composing the object, we check whether the gripper can approach the box from any of the 6 directions, i.e., $+x$ -, $+y$ -, $+z$ -, $-x$ -, $-y$ -, and $-z$ -directions, of the box's local frame, without colliding with the support surface. If there is no collision, we add a node that represents the respective placement and grasp classes into the graph. We continue this procedure for all placement classes. Since we do not consider the robot kinematics here, the actual location of the object on the table, i.e., how far the object is from the robot, does not affect the resulting graph.

After we obtain all feasible nodes in the first step, we display them on a grid. Then we connect *every pair* of nodes on the same vertical or horizontal line. These edges represent *potential* connections between nodes. An edge connecting a pair of nodes on the same vertical line represents *transit* paths while an edge connecting a pair of nodes on the same horizontal line represents *transfer* paths. Every node also has two self-loops. One loop corresponds to a transit path; the other loop corresponds to a transfer path. Figure 4.1 shows an example of a high-level Grasp-Placement Graph for a box of dimension 28.0 cm. $\times$ 4.9 cm. $\times$ 2.5 cm. For simplicity, we do not show self-loops in the plot.

The construction can be generalized to handle start and goal configurations in $\mathcal{G} \setminus \mathcal{P}$ and/or $\mathcal{P} \setminus \mathcal{G}$ by adding special nodes into the graph. To handle configurations in $\mathcal{P} \setminus \mathcal{G}$, we can add a grasp class index 0 to the graph. For example, in the above example of the box, we will obtain 6 new nodes, $(1, 0), (2, 0), \dots, (6, 0)$. These nodes are connected to all other nodes with the same placement class. Note,

Figure 4.1: An example of a high-level Grasp-Placement Graph for a box of dimension 28.0 cm. $\times$ 4.9 cm. $\times$ 2.5 cm. For clarity, self-loops are not depicted. Note that every node on the same line is connected to all other nodes although we do not draw separate lines for them. For example, the node (1, 6) is reachable from (6, 1) in one step.

however, that there are no horizontal edges between them since it is not possible to travel directly from one placement class to another in $\mathcal{P}$. To handle configurations in $\mathcal{G} \setminus \mathcal{P}$, we can add a placement class index 0, similarly to the previous case.

4.1.3 Guiding a Manipulation Planner via Task Plans

One of quality measures of manipulation paths is the number of transitions (as defined in Chapter 3). Fewer transitions means fewer grasp/ungrasp operations and fewer single-mode paths. This can therefore potentially lead to shorter overall execution time.

From the constructed graph we can extract *task plans* of a specific length¹, which will later serve as a guide for the planner about how to travel between different subsets of $\mathcal{G} \cap \mathcal{P}$. The planner, instead of exploring randomly how to go from one subset of $\mathcal{G} \cap \mathcal{P}$ to another, will follow those plans to search for a manipulation path of a specific number of transitions. Here a **task plan** is a sequence of graph nodes connected by graph edges. For example, from Figure 4.1, a task plan of length 3 to travel from (6, 6) to (2, 2) is given by $(6, 6) \rightarrow (4, 6) \rightarrow (4, 2) \rightarrow (2, 2)$.

The task plans extracted from the graph only provide high-level information on how to pick and place the object. Exact values of parameters for each grasp and placement classes will be assigned by the planner, i.e., via random sampling, in the planning phase (see Section 4.2).

¹A task plan of length k has $k - 1$ transitions.

4.2 Algorithm

Our pick-and-place manipulation planner proceeds in two phases: pre-processing phase and planning phase. In the pre-processing phase, it constructs a high-level Grasp-Placement Graph based on the models of the object and of the gripper (the latter is needed for collision checking). Note that neither the kinematic model of the robot nor the environment is needed at this phase. Then this high-level Grasp-Placement Graph is used in the planning phase to guide a bidirectional tree-based planner, similar to Lavalle and Kuffner (2001), to search for a manipulation path. The main algorithm is listed in Algorithm 1.

Algorithm 1: Main algorithm

Input: $\mathcal{R}$, $\mathcal{O}$, q_{start} , T_{start} , q_{goal} , T_{goal} , and t_{max}

Output: A valid manipulation path connecting $(q_{\text{start}}, T_{\text{start}})$ and $q_{\text{goal}}, T_{\text{goal}}$ or Failure

```

1  $G \leftarrow \text{Preprocess}(\mathcal{R}, \mathcal{O})$ 
2  $\mathcal{V}_{\text{start}} \leftarrow \text{Vertex}(q_{\text{start}}, T_{\text{start}})$ 
3  $\mathcal{V}_{\text{goal}} \leftarrow \text{Vertex}(q_{\text{goal}}, T_{\text{goal}})$ 
4  $l \leftarrow \text{FindShortestPathLength}(G, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}})$ 
5  $t \leftarrow 0; k \leftarrow l; \text{found} \leftarrow \text{False}$ 
6  $\Delta \leftarrow \text{ChoosePathLengthIncrement}(\mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}})$ 
7 while  $(t < t_{\text{max}})$  and (not found) do
8    $t_{\text{begin}} \leftarrow \text{GetTime}()$ 
9    $\Pi \leftarrow \text{FindPlansOfGivenLength}(k, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}})$ 
10   $Q \leftarrow \text{CreateDirectedGraph}(\Pi)$ 
11  found,  $\pi \leftarrow \text{PlanPath}(Q, \mathcal{R}, \mathcal{O}, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}}, t_{\text{max}} - t)$ 
12   $t_{\text{end}} \leftarrow \text{GetTime}()$ 
13   $t \leftarrow t + (t_{\text{end}} - t_{\text{begin}})$ 
14  if found then
15    return  $\pi$ 
16  end if
17   $k \leftarrow k + \Delta$ 
18 end while
19 return Failure

```

The main algorithm takes as its input a robot model $\mathcal{R}$; an object model $\mathcal{O}$; start and goal robot configurations q_{start} and q_{goal} ; start and goal object transformations T_{start} and T_{goal} ; and a maximum allowed computation time for the planner t_{max} . Details of functions in Algorithm 1 are listed below.

- $\text{Preprocess}(\mathcal{R}, \mathcal{O})$ gathers geometric information of the gripper and the object

Algorithm 2: PlanPath

Input: $Q, \mathcal{R}, \mathcal{O}, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}},$ and t_{max}

Output: The planning result of the given query (True/False) and the corresponding solution path (if any)

```
1  $\mathcal{T}_{\text{FW}} \leftarrow \text{Tree}(); \mathcal{T}_{\text{BW}} \leftarrow \text{Tree}()$ 
2  $\text{InitializeTree}(\mathcal{T}_{\text{FW}}, \mathcal{V}_{\text{start}})$ 
3  $\text{InitializeTree}(\mathcal{T}_{\text{BW}}, \mathcal{V}_{\text{goal}})$ 
4  $\mathcal{T}_a \leftarrow \mathcal{T}_{\text{FW}}; \mathcal{T}_b \leftarrow \mathcal{T}_{\text{BW}}$ 
5  $t \leftarrow 0$ 
6 while  $t < t_{\text{max}}$  or  $Q.\text{haspath}$  do
7    $t_{\text{begin}} \leftarrow \text{GetTime}()$ 
8    $\mathcal{V} \leftarrow \text{SampleTree}(\mathcal{T}_a)$ 
9    $\text{RemoveInfeasibleEdges}(Q)$ 
10   $\text{result} \leftarrow \text{ExtendFrom}(\mathcal{V}, Q)$ 
11  if  $\text{result} == \text{REACHED}$  then
12     $\pi \leftarrow \text{ExtractPath}(\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}})$ 
13    return True,  $\pi$ 
14  end if
15   $t_{\text{end}} \leftarrow \text{GetTime}()$ 
16   $t \leftarrow t + (t_{\text{end}} - t_{\text{begin}})$ 
17   $\text{Swap}(\mathcal{T}_a, \mathcal{T}_b)$ 
18 end while
19 return False, None
```

to construct a high-level Grasp-Placement Graph G , as described in Section 4.1.2.

- $\text{Vertex}()$ initializes a new tree vertex to store information of a composite configuration. Note that this function also examines G to find the graph node to which the input composite configuration belongs.
- $\text{FindShortestPathLength}(G, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}})$ finds the length l of the shortest task plan(s) to go from the graph node containing $\mathcal{V}_{\text{start}}$ to the graph node containing $\mathcal{V}_{\text{goal}}$. Here $l - 1$ serves as the *lower bound* of the number of transitions of manipulation paths connecting $(q_{\text{start}}, T_{\text{start}})$ and $(q_{\text{goal}}, T_{\text{goal}})$.
- $\text{ChoosePathLengthIncrement}(\mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}})$ chooses the suitable value of Δ for the query. If either $(q_{\text{start}}, T_{\text{start}})$ or $(q_{\text{goal}}, T_{\text{goal}})$ is in $\mathcal{G} \cap \mathcal{P}$, Δ can be 1. Otherwise, Δ has to be 2 in order to keep the resulting manipulation path irreducible.
- $\text{FindPlansOfGivenLength}(k, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}})$ collects all paths (i.e., task plans) of the specific length k on the graph G that connects the node containing $\mathcal{V}_{\text{start}}$ with the node containing $\mathcal{V}_{\text{goal}}$. It returns the set Π of such paths.

- `CreateDirectedGraph( $\Pi$ )` creates a directed graph Q from task plans in Π . A node of Q is encoded as a couple (d_i, c_i) , where c_i is the corresponding node of G and d_i indicates the number of steps c_i is away from c_{start} ².
- `PlanPath( $Q, \mathcal{R}, \mathcal{O}, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{goal}}, t_{\text{max}} - t$ )`, which acts as a motion planner, searches for a manipulation path according to information provided by Q . `PlanPath` is listed in Algorithm 2.
- `ExtendFrom( $\mathcal{V}, Q$ )` first randomly samples a new composite configuration in the grasp and placement classes suggested by Q . Then it will attempt to connect the composite configuration contained in $\mathcal{V}$ with the newly sampled one. The function returns True if the attempt is successful. In case the connection attempt fails, the record is also added to the corresponding edge of Q .
- `RemoveInfeasibleEdges( $Q$ )` removes edges in Q that lead to more than N failed attempts by `ExtendFrom`, where N is a threshold value set by the user.

4.2.1 Example

Consider the task of moving the box, whose high-level Grasp-Placement Table is shown in Figure 4.1, from the placement and grasp $c_{\text{start}} = (6, 6)$ to $c_{\text{goal}} = (2, 2)$. In the first planning loop in Algorithm 1, the algorithm will try to find a manipulation path of length $k = 3$. The set Π will store two task plans: $(6, 6) \rightarrow (4, 6) \rightarrow (4, 2) \rightarrow (2, 2)$ and $(6, 6) \rightarrow (1, 6) \rightarrow (1, 2) \rightarrow (2, 2)$. The directed graph Q , constructed from Π , will serve as a guidance for the planner as follows.

In each while loop in Algorithm 2, the planner will randomly pick an existing vertex $\mathcal{V}_{\text{sample}}$ on a tree. Suppose $\mathcal{V}_{\text{sample}}$ contains a composite configuration in placement and grasp classes $(4, 6)$. From Q , the planner then knows that the next extension should be attempted towards a composite configuration in the placement and grasp classes $(4, 2)$. `ExtendFrom` will then sample a composite configuration in that subset of $\mathcal{G} \cap \mathcal{P}$ and call a local planner, e.g., a bidirectional RRT planner (Lavalle and Kuffner 2001), to attempt the connection.

²We need to encode the level d_i of c_i into each node of Q since the node c_i may appear at different steps in different task plans. This helps the planner not to get lost when many task plans contain the same node c_i .

4.2.2 Remarks

1. The high-level Grasp-Placement Graph G , and hence Q , is constructed without considering kinematic constraints. Failed attempts by `ExtendFrom` are likely due to kinematic infeasibility. Therefore, we need to set a threshold N to prevent the planner from repetitively attempting infeasible connections.

This threshold also affects the running time of our planner. If we set N too low, feasible edges of Q can also be removed due to false negative reports of kinematic infeasibility. This will lead to obtaining solutions with redundant transitions. On the other hand, if the threshold is set too high, it will take longer time for the planner to declare an infeasible edge and hence longer overall running time.

2. We can tailor the sampling of vertices from a tree, done in `SampleTree`, by putting different sampling weight on different vertices. For example, we can use weights proportional to the level of c_{sample} in Q , i.e., d_{sample} . In this way, vertices farther away from the root will be more likely to be selected.
3. Our planner tends to work better in a less constrained environment. In a very constrained environment³ in which the robot needs to grasp and ungrasp the object a number of times before it can move the object to the final pose, our planner will spend a considerable amount of time verifying infeasible edges. However, in actual industrial settings, although the environment might be cluttered, it is usually not tightly constrained.

4.3 Experiments

In this section, we present details of comparisons between our planner and two other planners: Primitive Manipulation Planner (Section 4.3.2) and Discretization-Based Manipulation Planner (Section 4.3.2). We implemented all planners in Python and used OpenRAVE (Diankov 2010) as a simulation environment. The local planner employed in all manipulation planners was the OpenRAVE built-in bidirectional RRT. The robot was a 6-DOF industrial manipulator Denso VS-060 equipped with a 2-finger Robotiq gripper 85. All simulations were run on a 3.2 GHz Intel® Core™ desktop with 3.8 GB RAM.

³This may be seen as a narrow passage in $\mathcal{C}$. An example of such cases is the problem considered in Siméon et al. (2004) where the robot needs to pull a bar out of a tight cage mounted on the floor. Note, however, that the scenario was intentionally constructed so as to induce narrow passage in $\mathcal{C}$.

Figure 4.2: Scenes used in our experiments. There are two identical (movable) objects shown in each figure. The objects on the left are at the initial transformations. The objects on the right are at the final transformations.

4.3.1 Task Details

The planners had to plan pick-and-place motions for three objects: a box, an L-shaped object, and a small chair. The robot was to move from Home configuration, i.e., $q = \mathbf{0}_6$, pick up the object at a given object transformation, place it at another given transformation, and finally return to Home. Snapshots of all three settings are shown in Figure 4.2. The Grasp-Placement Graphs of the L-shaped object and of the chair are shown in Figure 4.3(a) and Figure 4.3(b), respectively. (The Grasp-Placement Graph of the box is similar to the one in Figure 4.1.)

These tasks are useful especially for assembly operations, such as furniture assembly where the robot needs to grasp a furniture part at some placement and move it to some desired transformation to attach the part to other parts. We set the start and goal object transformations such that the robot needed to perform regrasp operations at least once in order to complete the tasks.

4.3.2 Descriptions of Alternative Manipulation Planners

We implemented two alternative manipulation planners, each of which lies on opposite ends of the spectrum of Manipulation Graph construction. The first planner does not explicitly construct the graph while the second constructs the graph by means of discretization. Our planner lies midway between the two.

Primitive Manipulation Planner

Primitive Manipulation Planner (PMP) has minimal knowledge of the structure of $\mathcal{C}$ as it has no pre-processing stage. It explores the space according to the transition diagram, similar to Lamiraux and Mirabel (2015), shown in Figure 4.4.

(a) The high-level Grasp-Placement Graph for the L-shaped object

(b) The high-level Grasp-Placement Graph for the small chair

Figure 4.3: High-level Grasp-Placement Graphs of objects used in our simulations. For clarity, self-loops are omitted.

Figure 4.4: A transition diagram used in Primitive Manipulation Planner.

We implemented PMP as a bidirectional tree-based planner. It grows one tree rooted at $(q_{\text{start}}, T_{\text{start}})$, the other rooted at $(q_{\text{goal}}, T_{\text{goal}})$. In each iteration, it samples a new composite configuration and tries to connect it with existing vertices on a tree. For example, if the newly sampled configuration is in $G \cap P$ and is to be connected with a configuration in P on a tree, then the local planner will try connecting them with a transit path. The distance metric used in our implementation was as discussed in Chapter 3.

Discretization-Based Manipulation Planner

In contrast with PMP, Discretization-Based Manipulation Planner (DBMP) constructs its variant of Manipulation Graph – two-layer regrasp graph (Wan et al. 2015) – by discretizing placements and grasps

We implemented DBMP following the descriptions provided in Wan et al. (2015). The planner starts by constructing the set of possible grasps by means of sampling. Then it builds a two-layer regrasp graph: the first layer composes of placements and the second layer composes of grasps. Two placements in the first layer are connected together if they share at least one common valid grasp. To solve a query, DBMP first searches for placement sequences that connect the start and goal placements. Then it examines each placement sequence and searches for feasible grasps associated with it.

However, since all the searches are deterministic and proceed in a depth-first fashion, the time-complexity of DBMP is significantly large. Instead of having three varying parameters for each placement class: two parameters for a location on the table and one parameter for the rotation about an axis normal to the table’s surface, we constrained each placement class to have only a single varying parameter, the rotation. The same was also done implicitly in Wan et al. (2015). Furthermore, the deterministic search is also much sensitive to indexing of grasps and placements. To make the comparisons fairer, we reduced this effect by shuffling grasp and placement indices before each run.

Note that our implementation of grasp set computation was different from the method used in Wan et al. (2015), which was basically a discretization. Therefore, the pre-processing time put in Table 4.1 is intended only for reference. The numbers of total grasps computed in the cases of a box, an L-shaped object, and a chair were 124, 242, and 331, respectively.

4.3.3 Simulation Results and Comparisons

For each object, we ran each of the three planners 100 times. The data collected are pre-processing time, planning time, numbers of transitions, and success rate. (If no solution was found in 100 s., then the run was considered failed.) Data were averaged over successful runs and reported in Table 4.1.

Comparison with Primitive Manipulation Planner

From the data reported in Table 4.1, we can see that by exploiting more information about the connectivity of subsets of $\mathcal{G} \cap \mathcal{P}$, i.e., by constructing high-level Grasp-Placement Graphs, our planner was able to search for manipulation paths more systematically and efficiently, as reflected through the running time and the path quality achieved by our planner. As PMP had no information about the connectivity, much planning time was spent attempting infeasible connections between $\mathcal{G} \cap \mathcal{P}$ configurations. Furthermore, when the object was composed of more boxes, connection attempts were even more prone to failure since from any subset of $\mathcal{G} \cap \mathcal{P}$, the ratio of the number of reachable $\mathcal{G} \cap \mathcal{P}$ subsets to the number of all the subsets decreased.

Comparison with Discretization-Based Manipulation Planner

In fact, the two-layer regrasp graph (Wan et al. 2015) is similar to ours. One major difference is that they did not exploit any grasp parameterization in building the graph. Therefore, in their case, each grasp class contains exactly one grasp, hence numerous grasp classes. Also, the deterministic search employed in DBMP makes its capability relatively limited. Firstly, all placements have to be constrained to only one location on the table, or at most a few, due to time complexity of the search. Second, a significant amount of time is spent on exploring infeasible placement and grasp sequences due to the depth-first fashion of the search. This makes the success rate of DBMP in the given time much lower than the other planners.

	Problem 1 (Box)				Problem 2 (L-shaped object)				Problem 3 (Small chair)			
	prep. time (s.)	plan. time (s.)	# transitions	success rate	prep. time (s.)	plan. time (s.)	# transitions	success rate	prep. time (s.)	plan. time (s.)	# transitions	success rate
Our planner	0.24	12.18	4.0	100%	0.50	15.32	4.0	100%	1.95	29.19	4.0	100%
PMP	–	32.02	5.52	100%	–	46.83	5.07	73%	–	68.82	5.88	50%
DBMP	35.53	17.99	4.0	72%	55.18	39.76	4.0	53%	102.38	37.08	4.0	46%

Table 4.1: Pre-Processing time, planning time, and numbers of transitions from the three problems, averaged over 100 runs.

Figure 4.5: Snapshots from the experiment on the real robot. (a) The start configuration. (b)–(f) The robot performed pick-and-place motions to move the object to intermediate stable placements that eventually allowed grasping with the desired final grasp. (g) The goal configuration. The video can be found at <https://youtu.be/tLouwJ0wITQ>.

4.3.4 Hardware Experiment

Besides simulations, we also conducted a hardware experiment. In this experiment, the robot must pick up the leg of a stool, which we approximated by three boxes, and hold it with a given grasp to facilitate a subsequent screwing operation (to be performed by another robot; not included in this experiment). Snapshots of the start and goal configurations are shown in Figure 4.5. One can see that the initial pose of the object does not allow grasping with the desired grasp. Thus, the robot needs to regrasp the object several times. A video of the robot performing the task can be found at <https://youtu.be/tLouwj0wITQ>.

Conclusion

We have presented a new manipulation planner to tackle unimanual pick-and-place planning problems. We first proposed a method to construct a high-level Grasp-Placement Graph based on the models of the robot, the object, and the environment, without resorting to discretization of $\mathcal{G} \cap \mathcal{P}$. Our construction is therefore associated with a full parameterization of $\mathcal{G} \cap \mathcal{P}$, in contrast to previously proposed methods. The Graph then serves as a guide for the planner to explore the composite configuration space efficiently and systematically.

Our method to construct the Grasp-Placement Graph readily applies to that are boxes or composed of boxes. In such cases, assuming that the gripper can exert large enough forces with its fingers, all grasp classes considered here are also force-closure. In fact, the method can also be applied to non-box objects in which grasp parameterizations are available (see, e.g., the discussion in Chapter 3, Section 3.2.3).

The experimental results presented in Section 4.3 confirmed that the high-level Grasp-Placement Graph helps significantly improve both running time and manipulation path quality as compared to existing manipulation planners.

Chapter 5

Bimanual Manipulation Planning: Regrasping via *IK-Switch*

Bimanual or, more generally, multi-arm robotic systems are necessary to manipulate large/heavy objects. It is, however, much more challenging to plan and control multi-arm motions than single-arm motions due to *closed kinematic chains*, which are formed whenever multiple robots concurrently grasp a single object.

Closed-chain constraints induce sub-manifold structure in the configuration space and therefore affect multi-arm motions at different levels:

- Local level: connecting nearby configurations with a path is non-trivial and requires projections or other techniques (see, e.g., Yakey, LaValle, and Kavraki 2001; Pham and Stasse 2015; Mirabel et al. 2016) This is because any feasible path must lie entirely on some lower-dimensional sub-manifold.
- Connected-component level: random sampling in the ambient, i.e., configuration, space will fail to obtain a feasible configuration *almost surely* (Yakey, LaValle, and Kavraki 2001). This is because closed-chain induced sub-manifolds have zero measure relative to the ambient configuration space. Non-trivial modifications to a sampling method are consequently necessary.
- Global level: there can be multiple connected components, formed by closed-chain constraints, that are *mutually disconnected*. Solving a manipulation query might require navigation through these connected component.

With the local and component levels are relatively well understood (see, e.g., Yakey, LaValle, and Kavraki 2001; Han and Amato 2000; Jaillet and Porta 2013, and references therein), it is the *global* level that constitutes a major hurdle when de-

Figure 5.1: There exists no continuous closed-chain motions to flip the chair frame between the start configuration (a) and the goal configuration (b). The two configurations are residing in different *connected components*.

ploying multi-arm systems to address practical tasks such as flipping a furniture part (see Figure 5.1).

In this Chapter, we address closed-chain manipulation planning in the *global* level, as identified above. A novel move, called *IK-switch*, is proposed to help connect different components that are otherwise mutually disconnected. We also present a method to use the environment to help stabilize the (large/heavy) manipulated object during IK-switch moves. Finally, we integrate the IK-switch move into a sampling-based planner with a new tree structure, adapted specifically to suit the IK-switch move, that allows accelerating the planning. The proposed approach is validated by software and hardware experiments.

This work has been published in Xian, Zhou, **Puttichai Lertkultanon**, and Quang-Cuong Pham (2017). “Closed-chain manipulation of large objects by multi-arm robotic systems”. In: *IEEE Robotics and Automation Letters* 2.4, pp. 1832–1839. DOI: 10.1109/LRA.2017.2708134.

5.1 Problem Formulation and Analysis

When two robots $\mathcal{R}_1$ and $\mathcal{R}_2$ are grasping a single object, a closed kinematic chain is formed. In this case, a grasp $\mathbf{g}$ can be uniquely deduced directly from the composite configuration $\mathbf{c} = (\mathbf{q}_1, \mathbf{q}_2, \mathbf{T})$. This closed-chain is subject to the dimension-reducing constraint of the form

$$\mathbf{f}_{\mathbf{g}}(\mathbf{c}) = \mathbf{0}, \quad (5.1)$$

where $\mathbf{0}$ is of an appropriate dimension. Now let $\mathcal{C}_{cc}$ be the set of composite configurations satisfying Constraint (5.1), i.e.,

$$\mathcal{C}_{cc} = \{\mathbf{c} \in \mathcal{C} \mid \mathbf{f}_g(\mathbf{c}) = \mathbf{0}\}. \quad (5.2)$$

Note that 1. $\mathcal{C}_{cc}$ is a set that is specific to a bimanual grasp $\mathbf{g}$. Different grasps lead to different set of valid closed-chain configurations; and 2. excluding singularities, $\mathcal{C}_{cc}$ is actually a set of sub-manifolds of $\mathcal{C}$ (Burdick 1989).

5.1.1 Essentially Mutually Disconnectedness

The subset of collision-free composite configurations of $\mathcal{C}_{cc}$ is indeed a collection of *single-transfer connected sets* (see Chapter 3, Definition 13). Here for convenience, we will write $\mathcal{F}(\mathbf{c})$ as a single-transfer connected set that contains the composite configuration $\mathbf{c}$. Verifying whether two given (bimanual) composite configurations are in the same set $\mathcal{F}$ is actually a difficult problem. To the best of our knowledge, there currently exists no analytical method for the verification, except for (low-dimensional) systems where inverse kinematics is available analytically. To verify the single-transfer connectedness, one needs to directly attempt planning a transfer path between the two configurations. This inspires us to formulate the following definition.

Definition 14. *Two single-transfer connected sets $\mathcal{F}(\mathbf{c}_1)$ and $\mathcal{F}(\mathbf{c}_2)$ are **essentially mutually disconnected (EMD)** if they are topologically disconnected or if, in practice, one cannot find any connection between the two within a reasonable amount of time.*

Note that we use the term “essentially” in the above definition as it is difficulty, in an actual problem instance, to provide a *rigorous* certificate that two sets $\mathcal{F}(\mathbf{c}_1)$ and $\mathcal{F}(\mathbf{c}_2)$ are disconnected. Consider for example the bimanual system in Figure 5.2. One can clearly see that the single-transfer connected sets containing the lower and upper configurations are disconnected. Yet, a certificate of this disconnectedness would involve complex trigonometry formulas. Such certificates are even more difficult to obtain in high-DOF systems, such as the one shown in Figure 5.1. One can, however, say that the composite configurations in Figure 5.1(a) and Figure 5.1(b) are *essentially mutually disconnected* after attempting connecting them by a state-of-the-art planner – without regrasping – for hours without success.

Figure 5.2: Suppose the small arm (shown in red) cannot bend *backward*. Then there exists no continuous motion (i.e., without regrasping) that brings the system from the lower configuration (a) to the upper configuration (b).

In order to *bridge* different EMD components, our proposed algorithm plans not only the component-level closed-chain motions, as most of existing works do, but also *regrasping moves* that help the system *jump* across different EMD components. This significantly enlarges the size of the solution space.

5.1.2 Problem Formulation

In a general closed-chain motion planning problem for a bimanual system, typically one is given a start composite configuration $\mathbf{c}_{\text{start}} \in \mathcal{C}_{\text{cc}}$, which imposes a closure constraint on the system, and a goal object configuration $\mathbf{T}_{\text{goal}}$. The grasp at the goal is pre-determined by the start composite configuration, while the specific goal robot configurations are unknown.

Using the presented notations, the problem can be stated as follows.

Problem 1. *Given a start composite configuration $\mathbf{c}_{\text{start}} \in \mathcal{C}_{\text{cc}}$ and a goal object configuration $\mathbf{T}_{\text{goal}}$, find a path $P : [0, 1] \rightarrow \mathcal{C}_{\text{cc}}$ such that $P(0) = \mathbf{c}_{\text{start}}$ and $\pi_p(P(1)) = \mathbf{T}_{\text{goal}}$.*

5.1.3 Problem Analysis

Consider Problem 1. There exist multiple goal composite configurations $\mathbf{c}_{\text{goal}}$ satisfying $\pi_p(\mathbf{c}_{\text{goal}}) = \mathbf{T}_{\text{goal}}$ due to multiplicity of IK solutions associated to a given grasp. Let $\mathfrak{C}_{\text{goal}} = \{\mathbf{c}_{\text{goal}}^1, \mathbf{c}_{\text{goal}}^2, \dots, \mathbf{c}_{\text{goal}}^m\}$ denote the collection of all m possible goal composite configurations. There exists two possible cases, based on the single-transfer connected set containing $\mathbf{c}_{\text{start}}$, as follows.

Case 1 $\exists \mathbf{c}_{\text{goal}} \in \mathfrak{C}_{\text{goal}} \mathbf{c}_{\text{goal}} \in \mathcal{F}(\mathbf{c}_{\text{start}})$.

In this case, there exists a feasible *transfer* path P that starts at $\mathbf{c}_{\text{start}}$ and ends at some $\mathbf{c}_{\text{goal}}$. (Note, however, that there can still be some composite configurations in $\mathfrak{C}_{\text{goal}}$ that are in single-transfer connected sets essentially mutually disconnected from $\mathcal{F}(\mathbf{c}_{\text{start}})$.) One may be able to obtain such a path using a unidirectional planner.

In case a bidirectional planner such as BiRRT (LaValle and Kuffner 2000) is deployed, one also needs to manually specify a full goal composite configuration that is contained in $\mathcal{F}(\mathbf{c}_{\text{start}})$. Since, to the best of our knowledge, there is currently no method available for such selection, it is likely that the manually chosen $\mathbf{c}_{\text{goal}}$ are essentially mutually disconnected from $\mathcal{F}(\mathbf{c}_{\text{start}})$. This scenario falls into **Case 2**.

One possible approach to avoid selecting a goal composite configuration when using a bidirectional planner is to extend the idea of BiSpace planning (Diankov et al. 2008) to such closed-chain systems. Although plausible, this approach is still limited to the condition of this case, i.e., $\exists \mathbf{c}_{\text{goal}} \in \mathfrak{C}_{\text{goal}} \mathbf{c}_{\text{goal}} \in \mathcal{F}(\mathbf{c}_{\text{start}})$, and cannot handle **Case 2** presented below.

Case 2 $\forall \mathbf{c}_{\text{goal}} \in \mathfrak{C}_{\text{goal}} \mathcal{F}(\mathbf{c}_{\text{goal}})$ and $\mathcal{F}(\mathbf{c}_{\text{start}})$ are essentially mutually disconnected.

In this case, regrasping is necessary for the system to traverse different EMD components to reach $\mathcal{F}(\mathbf{c}_{\text{start}})$. Now $\mathbf{c}_{\text{goal}}$ may be chosen at random. There are two possible subcases:

Case 2.1 $\pi_p(\mathcal{F}(\mathbf{c}_{\text{start}})) \cap \pi_p(\mathcal{F}(\mathbf{c}_{\text{goal}})) \neq \emptyset$.

This means that there exists a path for the system to bring the object from $\pi_p(\mathbf{c}_{\text{start}})$ and T_{goal} with one regrasping. The required regrasping can be performed once the object reaches a configuration $T \in \pi_p(\mathcal{F}(\mathbf{c}_{\text{start}})) \cap \pi_p(\mathcal{F}(\mathbf{c}_{\text{goal}}))$, at which configuration the kinematic chain can jump from $\mathcal{F}(\mathbf{c}_{\text{start}})$ to $\mathcal{F}(\mathbf{c}_{\text{goal}})$.

Case 2.2 $\pi_p(\mathcal{F}(\mathbf{c}_{\text{start}})) \cap \pi_p(\mathcal{F}(\mathbf{c}_{\text{goal}})) = \emptyset$.

Let $\mathcal{F}_{\text{int}}^0 = \mathcal{F}_{\text{start}}$ and $\mathcal{F}_{\text{int}}^{p+1} = \mathcal{F}_{\text{goal}}$. The problem is solvable if and only if there exists $p \geq 1$ *intermediate* EMD components, denoted by $\mathcal{F}_{\text{int}}^1, \mathcal{F}_{\text{int}}^2, \dots, \mathcal{F}_{\text{int}}^p$, such that

$$\pi_p(\mathcal{F}_{\text{int}}^i) \cap \pi_p(\mathcal{F}_{\text{int}}^{i+1}) \neq \emptyset \quad \forall i \in \{0, 1, \dots, p\}.$$

Figure 5.3: Visualization of bridging different EMD single-transfer connected sets via IK-switch moves. Switching IK solutions corresponding to the same grasp constitutes *jumping* between different single-transfer connected sets.

The intersections between the projections of different EMD components provide shared regions to bridge EMD components and in turn, connect $\mathcal{F}(c_{\text{start}})$ with $\mathcal{F}(c_{\text{goal}})$.

The discussion above summarizes all possible scenarios that can be encountered in a closed-chain planning problem. To effectively *jump* between different single-transfer connected sets that are EMD, we present the *IK-switch* move in the next Section.

5.1.4 IK-Switch

For a given unimanual grasp, hence a 6D end-effector pose, a 6-DOF robot arm may have up to 16 IK solutions (Primrose 1986). For a bimanual grasp, there are consequently even more IK solutions and they may belong to single-transfer connected components that are EMD. “Jumping” between different IK solutions corresponding to the same grasp can therefore *bridge* different EMD components (see Figure 5.3).

Specifically, an *actual* IK-switch move consists of three steps:

1. A manipulator $\mathcal{R}_i$ releases the grasp;
2. $\mathcal{R}_i$ moves along a transit path from the original IK solution to another IK solution corresponding to the same grasp;

3. $\mathcal{R}_i$ grasps the object.

The main difference from the usual *regrasping* move, such as the one presented in Chapter 4, is that IK-switch regrasps using exactly the same grasp as before the operation. This has two advantages. First, one avoids complexity explosion when, e.g., computing and using the grasp-placement graph when the number of grasp classes increases while staying expressive enough to solve difficult tasks, as we illustrate in our experiments in Section 5.3. Second, this strategy can be used even when one does not have any other information on the grasp set of the object (e.g., grasp database, grasp classes) since one (given) single bimanual grasp is used throughout the whole manipulation.

Note that in step 1. of IK-switch, as one manipulator releases its grasp, contact with the environment is needed for the object to stay in static equilibrium, i.e., to ensure that the object does not slip out of the other grasping arm. This has been taken into account in the algorithm, see Section 5.2.2.

5.2 Algorithm

Based on the previous analysis, we propose here a planner that can address complex closed-chain manipulation tasks. The planner is built upon the classical BiRRT planner (LaValle and Kuffner 2000) and comprises two planning stages. In the first stage, it plans a *global* path for the closed-chain system. This global path consists of 1. segments of component-level closed-chain motions and 2. IK-switch regrasping requests at the ends of the segments. In the second stage, the planner completes the global path by planning IK-switch moves. Delaying IK-switch planning to the second stage helps improve efficiency of the planner sharply since most regrasping requests, which are stored in the planning tree vertices, will not appear in the final global path connecting c_{start} and c_{goal} .

Note that planning IK-switch might fail when no feasible regrasping moves¹ can be found. Such failure can be handled efficiently by integrating an efficient data structure proposed in Xian, Lertkultanon, and Pham (2017) into the planner such that when IK-switch planning (stage two) fails, the planner can continue with stage-one planning while retaining information already obtained in existing tree vertices.

¹A regrasping move consists of 1. the robots moving the object from the current object configuration T_{cur} to some placement T_{place} ; 2. the actual regrasping operations (transit paths); 3. the robots moving the object back to T_{cur} .

5.2.1 Stage 1: Planning Global Path

Given a closed-chain query determined by a start composite configuration $\mathbf{c}_{\text{start}}$ and a goal object transformation $\mathbf{T}_{\text{goal}}$, we first choose one goal composite configuration $\mathbf{c}_{\text{goal}}$ such that $\pi_p(\mathbf{c}_{\text{goal}}) = \mathbf{T}_{\text{goal}}$. This can be done either at random or according to some certain heuristics. The global path planner is listed in Algorithm 3.

Algorithm 3: Global Path Planner

Input: $\mathbf{c}_{\text{start}}$, $\mathbf{c}_{\text{goal}}$, and R_{max}

Output: A valid closed-chain path (possibly with regrasping) connecting $\mathbf{c}_{\text{start}}$ and $\mathbf{c}_{\text{goal}}$ or Failure

```

1  $\mathcal{T}_a \leftarrow \text{Tree}(); \mathcal{T}_b \leftarrow \text{Tree}()$ 
2  $\text{InitializeTree}(\mathcal{T}_a, \mathbf{c}_{\text{start}})$ 
3  $\text{InitializeTree}(\mathcal{T}_b, \mathbf{c}_{\text{goal}})$ 
4 for iteration = 1 to  $N_{\text{maxiter}}$  do
5    $\mathbf{T}_{\text{rand}} \leftarrow \text{SampleSE3}()$ 
6   if  $\text{ExtendTree}(\mathcal{T}_a, \mathbf{T}_{\text{rand}}, R_{\text{max}})$  then
7     if  $\text{Connect}(\mathcal{T}_a, \mathcal{T}_b, R_{\text{max}})$  then
8        $P \leftarrow \text{ExtractPath}(\mathcal{T}_a, \mathcal{T}_b)$ 
9       if  $\text{PlanIKSwitch}(\mathcal{T}_a, \mathcal{T}_b, P)$  then
10        return  $P$ 
11      else
12         $\text{Reorganize}(\mathcal{T}_a, \mathcal{T}_b)$ 
13      end if
14    end if
15  end if
16   $\text{Swap}(\mathcal{T}_a, \mathcal{T}_b)$ 
17 end for
18 return Failure

```

The planner initializes two trees rooted at $\mathbf{c}_{\text{start}}$ and $\mathbf{c}_{\text{goal}}$. It also accepts as its input R_{max} , which is the maximum number of allowed regrasping operations². The function ExtendTree is listed in Algorithm 4. Brief descriptions of some key functions are given below

- $\text{SampleSE3}()$: A transformation matrix (an element of $SE(3)$) is generated by separately sampling rotational and translational parts. A rotation matrix is uniformly sampled from $SO(3)$ via the method proposed in Kuffner (2004). A translation vector is sampled uniformly from the user defined range.

²One regrasping operation (i.e., IK-switch move) is a single jump from one single-transfer connected set to another. Therefore, it possibly contains multiple robots changing their grasps at a single object configuration,

Algorithm 4: ExtendTree

Input: $\mathcal{T}$, T , and $R_{\max}$ **Output:** True or False

```
1  $\mathcal{V}_{\text{near}} \leftarrow \text{NearestNeighbor}(\mathcal{T}, T, R_{\max})$ 
2  $T_{\text{new}} \leftarrow \text{Threshold}(\mathcal{V}_{\text{near}}, T)$ 
3  $\sigma_{\text{new}} \leftarrow \text{InterpolateSE3Path}(\mathcal{V}_{\text{near}}, T_{\text{new}})$ 
4 if  $\sigma_{\text{new}}$  is not None and collision-free then
5    $\text{result} \leftarrow \text{Track}(\mathcal{V}_{\text{near}}.\text{config}, \sigma_{\text{new}})$ 
6    $P_{\text{new}} \leftarrow \text{result.path}$ 
7    $\mathcal{V}_{\text{new}} \leftarrow \text{Vertex}(P_{\text{new}}(1), P_{\text{new}})$ 
8   if  $\text{result.status} == \text{Reached}$  then
9      $\mathcal{V}_{\text{new}}.\text{needRegrasp} \leftarrow \text{False}$ 
10     $\text{AddVertex}(\mathcal{T}, \mathcal{V}_{\text{near}}, \mathcal{V}_{\text{new}})$ 
11    return True
12  elif  $\text{result.status} == \text{NeedRegrasp}$  and  $\mathcal{V}_{\text{near}}.\text{regraspCount} < R_{\max}$ 
13    then
14       $\mathcal{V}_{\text{new}}.\text{needRegrasp} \leftarrow \text{True}$ 
15       $\mathcal{V}_{\text{new}}.\text{regraspCount} \leftarrow \mathcal{V}_{\text{near}}.\text{regraspCount} + 1$ 
16       $\mathcal{V}_{\text{new}}.\text{config} \leftarrow \text{result.regraspConfig}$ 
17       $\text{AddVertex}(\mathcal{T}, \mathcal{V}_{\text{near}}, \mathcal{V}_{\text{new}})$ 
18      return True
19  end if
20 return False
```

- $\text{Connect}(\mathcal{T}_a, \mathcal{T}_b, R_{\max})$: The procedure is similar to that of `Extendtree`, but without `Threshold` (described below). A connection attempt is successful only if a feasible closed-chain path is generated to connect a pair of vertices from the two tree *and* the number of regrasp requests (`regraspCount`) along the global path connecting c_{start} and c_{goal} does not exceed the maximum $R_{\max}$.
- $\text{ExtractPath}(\mathcal{T}_a, \mathcal{T}_b)$: A global path from c_{start} to c_{goal} is extracted from the two trees by recursively concatenating $\mathcal{V}.\text{path}$. Note that the resulting path P may not yet be a valid path until gaps between closed-chain segments are bridged by IK-switch moves.
- $\text{Reorganize}(\mathcal{T}_a, \mathcal{T}_b)$: This functions re-organizes vertices stored in both trees in case IK-switch planning fails and the planner needs to re-use the two trees for further stage-one planning. The details of this function can be found in Xian, Lertkultanon, and Pham (2017).

- **Threshold($\mathcal{V}, T$):** Given a distance metric w in $SE(3)$ (such as the one introduced in Chapter 3, Equation 3.1) and a user-defined maximum distance W , this function returns an object transformation T_{new} , where
 - if $w(\mathcal{V}.\text{config}.T, T) \leq W$, then $T_{\text{new}} \leftarrow T$;
 - otherwise, T_{new} is an object transformation in the direction from $\mathcal{V}.\text{config}.T$ towards T but within a distance W from $\mathcal{V}.\text{config}.T$.
- **InterpolateSE3Path($\mathcal{V}, T$):** This function generates an $SE(3)$ trajectory connecting $\mathcal{V}.\text{config}.T$ and T , by using the method in Park and Ravani (1997) for the rotational part and using polynomial interpolation for the translational part.
- **Track(c, σ):** Given an object path σ , this function computes a closed-chain path – in the form of a set of bimanual waypoints – such that the robots move the object along σ , beginning at a given composite configuration c . This function is listed in Algorithm 5.

First, it deduces the bimanual grasp g from the given composite configuration c . Then it discretizes the object trajectory σ , using a user-defined time step, to get a list of transformations. While iterating through the list of transformations, it computes the robot configurations at the next step via differential inverse kinematics algorithm (see, e.g., Siciliano et al. 2009) to ensure that the next robot configurations are reachable, i.e., in the same single-transfer connected set. In case the IK computation fails, we check if it is because either robot is going near its joint limits boundary via `IsNearBoundary`. If this is the case, a new robot configuration is computed (for the robot that goes near its configuration space boundary) such that it grasps the object with the same grasp but with an IK solution *further* from its boundary. Suppose the upper and lower joint limits are q_u and q_l , respectively. We compute all IK solutions associated with the grasp g via OpenRAVE (Diankov 2010) IKFast. Then we choose a suitable regrasping configuration as

$$\underset{q}{\operatorname{argmax}}(q_u - q)^\top (q - q_l),$$

which is a heuristic to select a configuration that is *far* from the boundary. The closed-chain path that this function returns will be the path starting from c and going the furthest possible along σ .

After the global planning is successful (Algorithm 3, line 8), the planner proceeds to the second stage to plan for IK-switch moves.

Algorithm 5: Track

Input: c and σ

Output: A data structure “result” storing various information about the task

```
1  $g \leftarrow \text{GetBimanualGrasp}(c)$ 
2  $c_{\text{prev}} \leftarrow c$ 
3  $L \leftarrow \text{List}(); \text{Append}(L, c)$ 
4  $\text{result} \leftarrow$  a new data structure;  $\text{result.status} \leftarrow \text{Failure}$ 
5 for each discretized point  $T$  along  $\sigma$  do
6    $c_{\text{next}} \leftarrow \text{SolveDiffIK}(T, g, c_{\text{prev}})$ 
7   if  $\text{SolveDiffIK}$  succeeds then
8      $\text{Append}(L, c_{\text{next}})$ 
9   elif  $\text{IsNearBoundary}(c_{\text{prev}})$  then
10     $c_{\text{regrasp}} \leftarrow \text{GetRegraspConfig}(c_{\text{prev}})$ 
11    if  $\text{GetRegraspConfig}$  succeeds then
12       $\text{result.regraspConfig} \leftarrow c_{\text{regrasp}}$ 
13       $\text{result.path} \leftarrow \text{ClosedChainPath}(L);$ 
14       $\text{result.status} \leftarrow \text{NeedRegrasp}$ 
15      return result
16    else
17      return result
18    end if
19  end if
20  else
21    return result
22  end if
23   $c_{\text{prev}} \leftarrow c_{\text{next}}$ 
24 end for
25  $\text{result.path} \leftarrow \text{ClosedChainPath}(L);$ 
26  $\text{result.status} \leftarrow \text{Reached}$ 
27 return result
```

5.2.2 Stage 2: Planning IK-Switch Moves

The function `PlanIKSwitch` (Algorithm 3, line 9) handles IK-switch requests to bridges closed-chain segments in the global path. This function accepts the two planning trees, constructed during the first planning stage, and the global path P as inputs. It iterates through all vertices $\mathcal{V}$ along the global path on the two tree at which $\mathcal{V}.\text{needRegrasp}$ is True. At each regrasping vertex, it first computes an object placement configuration T_{place} (see Section 5.2.2). Then it plans a regrasping move that consists of three steps as follows.

1. Both robots move the object (a closed-chain motion) from the current object

configuration T_{cur} to T_{place} .

2. All robots that need regrasping change their grasps.
3. Both robots move the object back to T_{cur} .

Finally, if all steps are successfully planned, the function returns True. Otherwise, it returns False.

Placement Configuration Computation

Given a current object transformation T_{cur} , here we compute a placement configuration T_{place} at which an actual IK-switch can be performed (i.e., a robot changing its grasp). Since there can be one robot grasping the object at any time instant during the move, T_{place} does not have to strictly be a stable placement (as defined in Chapter 3, Definition 2).

We proceed in iterations. In each iteration, we inject some small random perturbation to T_{cur} to get T (except in the first iteration where we do nothing). Then we compute a placement configuration T_{place} that is *close* to T as follows. Considering that the object is to stay in static equilibrium *with the help of contact forces provided by a grasping robot*, we explore all types of contact between the convex hull $\mathcal{H}$ of the object and the support surface: face-face, edge-face, and vertex-face.

Face-face contact: The face f of $\mathcal{H}$ that is the closest to the support surface is found. Then a small rotation is added to the object so as to align the face f with the support surface. Then the object is translated towards the support surface perpendicularly until the face f touches the surface.

Edge-face contact: The edge e of $\mathcal{H}$ that is the closest to the support surface is found. Then a small rotation is added to the object such that e is parallel to the support surface. Then the object is translated towards the support surface perpendicularly until it touches the surface.

Vertex-face contact: The object is translated towards the support surface perpendicularly until it touches the surface.

Note that the small rotation added in the above step (so as to align the object with the support surface) is not unique and can therefore be assigned either randomly or subject to some heuristics or criteria. The object configuration obtained once the

object touches the surface serves as T_{place} . Then we proceed to check if T_{place} , with the grasp by a robot, is in static equilibrium.

Static equilibrium of T_{place} can be checked in the same way as when checking grasp stability (Chapter 3). In short, one formulate the problem as a feasibility problem of contact forces and check if the available contact points (induced by robot-object contact and object-environment contact) can produce enough forces to resist the gravito-inertial wrench (see Chapter 3, Section 3.2.3 for more details).

5.2.3 Properties of the Planner

The complexity of the planning problem depends on various properties of the composite configuration space, including, but not limited to, dimension, expansiveness (Hsu, Latombe, and Motwani 1997), and ϵ -goodness (Kavraki et al. 1995).

At the local and component levels, our planner relies on variants of the classical RRT LaValle 1998. Therefore, it inherits, at these levels, many properties of RRT such as Voronoi bias. The current implementation as presented earlier in this Chapter, however, does not guarantee probabilistic completeness as it draws samples in the object configuration space while growing trees in the composite configuration space. Other probabilistically complete variants can be deployed at the cost of increasing complexity. At the global level, the completeness guarantee might require, for example, considering grasp classes and enumerating all possible IK solutions during IK-switch moves.

5.3 Experiments

The capability and effectiveness of the proposed planner are illustrated in two difficult bimanual tasks. The implementation is in Python and is openly available at <https://github.com/quangounet/bimanual>.

We used OpenRAVE (Diankov 2010) as a simulation environment. All simulations were run on a 4 GHz desktop. The bimanual system consisted of two industrial manipulators Denso VS-060. Each one was equipped with an ATI Gamma Force-Torque sensor and a 2-finger Robotiq gripper 85. For the details of the controller used in this experiment, see Appendix A.

Task 1, as shown in Figure 5.4, requires the two robots to transport a heavy L-shaped object from a start composite configuration to a designated goal object pose. In Task 2, as shown in Figure 5.5, the robots have to flip the chair frame. For each task, one goal composite configuration is selected such that it is *essentially mutually*

Figure 5.4: Snapshots of the robot motion completing Task 1. (a) The start composite configuration. (b) $\rightarrow$ (c) the robots move the object to seek support from the environment. (d) and (e) the robots perform regrasping motions. (h) The goal composite configuration.

disconnected from the start composite configuration³. The maximum number of regrasping allowed $R_{\max}$ for Task 1 and Task 2 is 1 and 3, respectively. We repeated simulations 50 times for each task. Results are summarized in Table 5.1.

Table 5.1: Simulation results for Task 1 (see Figure 5.4) and Task 2 (see Figure 5.5)

	$R_{\max}$	success rate	computation time (s.)	
			Global	IK-switch
Task 1	1	0.953	0.36	2.43
Task 2	3	0.998	3.66	12.15

In both tasks, the planner was able to find a feasible motion to complete the task within a reasonable amount of time. One can see that the IK-switch planning took the majority of the computation time. This is because the second-stage planning involves planning of multiple closed-chain and transit paths as well as static equilibrium checking.

³In these experiments, the start and goal composite configurations are considered essentially mutually disconnected when a closed-chain planner (component-level) does not yield any closed-chain trajectory connecting them within half an hour.

Conclusion

In this Chapter, we have presented a framework for planning closed-chain motions for bimanual manipulation tasks. While most existing works address closed-chain motion planning problems in local and component levels, we proceed further and address the problem in *global* level. The presented planner is able to plan motions across different single-transfer connected sets as it *bridges* configurations from essentially mutually disconnected single-transfer connected sets by the IK-switch moves. We have also presented a regrasping procedure in which the robots seek for the object support from the environment before performing the actual regrasping operations.

Apart from being a standalone closed-chain planner, the presented planner can also serve as a component (i.e., as a local planner) of a bimanual manipulation planner for more complex tasks, such as the one to be presented in Chapter 6.

Figure 5.5: Snapshots of the robot motion completing Task 2. IK-switch moves can be seen in (c), (f), (g), and (j). The video of the robots executing this task can be found at <https://youtu.be/PUkNS3gykC8>.

Chapter 6

Bimanual Manipulation Planning: A Certified-Complete Manipulation Planner

Large or heavy objects are best manipulated using two hands. Humans are good at bimanual manipulation: think of how we can, for example, effortlessly manipulate a large piece of furniture (Fig. 6.1(a)). By contrast, bimanual manipulation is still challenging for robots, mainly because of the closed-chain constraint, which arises whenever two robot hands simultaneously grasp a single rigid object (Fig. 6.1(b)).

In this Chapter, we specifically consider the harder class of problem instances where the manipulated object can be moved *only* when grasped by both hands, which is the case for large and/or heavy objects. We propose a bimanual manipulation planner to solve such problems. The key specificity of the proposed algorithm is that it is **certified-complete**: for a given object and a given environment, we provide a *certificate* that the algorithm will find a solution to *any* bimanual manipulation query in that environment whenever one exists. Moreover, the certificate is constructive: at run-time, it can be used to quickly find a solution to a given query. We showcase the capability of the algorithm, in software and hardware, by solving bimanual queries on a number of large pieces of furniture. An implementation of our algorithm is also openly available at <https://gitlab.com/puttichai/pymanip>.

6.1 The Proposed Approach

We propose the following approach to solving a bimanual manipulation query:

Step 1 Identify the placement classes of T_{start} and T_{goal} as $\mathcal{P}_{\text{start}}$ and $\mathcal{P}_{\text{goal}}$, respectively.

Figure 6.1: (a) A human can effortlessly manipulate a large piece of furniture. (b) Unlike humans, bimanual manipulation is extremely challenging for robots due to, for example, closed-chain constraints.

Step 2 Generate TypeA trajectory, within the placement class $\mathcal{P}_{\text{start}}$, to move the object from T_{start} to some T'_{start} .

Step 3 Generate TypeB trajectory to bring the object from T'_{start} to some T'_{goal} in the placement class $\mathcal{P}_{\text{goal}}$.

Step 4 Generate TypeA trajectory, within the placement class $\mathcal{P}_{\text{goal}}$, to move the object from T'_{goal} to T_{goal} .

A solution to a query will be a sequence of TypeA trajectories, which connect configurations in the same placement class, and TypeB trajectories, which connect configurations from different placement classes.

In the above steps, T'_{start} (respectively T'_{goal}) is an object transformation that can serve as an initial (respectively goal) transformation of the to-be-generated TypeB trajectory. Note also that in some cases, one may need to generate TypeB trajectories to move the object to, and between, some *intermediate* placements since a direct connection between $\mathcal{P}_{\text{start}}$ and $\mathcal{P}_{\text{goal}}$ may not exist or cannot be found. The procedure can be done by repeating **Step 2** and **Step 3** until the goal placement $\mathcal{P}_{\text{goal}}$ is reached.

The completeness of the above approach depends on the completeness of TypeA and TypeB trajectory generation methods. In the remaining of this Section, we give brief overviews of generation of both TypeA and TypeB trajectories, as well as the main algorithm.

6.1.1 Overview of Trajectory Generation

TypeA: To plan TypeA trajectories, we argue that we can consider the set $\mathcal{T}_i \subset SE(2)$ of object configurations, which is the projection via π_p of $\mathcal{P}_i$ (see Section 6.2.1 for more details), instead of examining a placement configuration set $\mathcal{P}_i$, which is a subset of the high-dimensional $\mathcal{C}$.

Given two object configurations T_1 and T_2 in the same connected component of $\mathcal{T}_i$, we first generate an object path σ as if the object could move freely by itself on a support surface. Then we present a procedure to generate a TypeA trajectory that moves the object along σ . We prove that, given a valid object path, such a TypeA trajectory always exists and that our procedure will terminate with a solution in finite time.

TypeB: Since the motions of the system are severely constrained by closed kinematic chains, randomly generating closed-chain queries, where the start and goal configurations are in different placement classes, has slim chances of the queries being solvable. To resolve this issue, we propose a heuristic to generate closed-chain queries in such a way that, by our intuition, does not require a large range of robot motions to solve them.

Now suppose that one has a TypeB trajectory $M^B : [0, 1] \rightarrow \mathcal{C}$ connecting two placement classes $\mathcal{P}_i$ and $\mathcal{P}_j$, i.e., $M^B(0) \in \mathcal{P}_i$ and $M^B(1) \in \mathcal{P}_j$. Observe that provided that the world does not change, whenever one needs to connect configurations $c_1 \in \mathcal{P}_i$ and $c_2 \in \mathcal{P}_j$, one can *reuse* the trajectory M^B by planning two TypeA trajectories, M_1^A and M_2^A , where M_1^A connects c_1 and $M^B(0)$ and M_2^A connects $M^B(1)$ and c_2 . The composition (as defined in Chapter 3) of the three trajectories, i.e., $M = M_1^A * M^B * M_2^A$, then serves as a solution. Since TypeB trajectories can be reused as discuss above, they have to be computed only once and the procedure may as well be offline. This inspires us to introduce a notion of a **certificate**, which is a set of useful TypeB trajectories. Once computed, a solution (if any) to any given bimanual manipulation query can then be constructed from the certificate in the aforementioned manner.

6.1.2 Main Algorithm

First, we generate a certificate $\mathcal{M}$. This step needs to be done only once per problem setting. Given a query $Q = (T_{\text{start}}, T_{\text{goal}})$, we then extract a placement sequence $\mathcal{P}_1 \rightarrow \mathcal{P}_2 \rightarrow \dots \rightarrow \mathcal{P}_n$, where $\mathcal{P}_1 = \mathcal{P}_{\text{start}}$ and $\mathcal{P}_n = \mathcal{P}_{\text{goal}}$, along with their

Figure 6.2: (a) Placement classes and their verified TypeB connectivities. Each $M_i^B, i \in \{1, 2, \dots, 5\}$ is a transfer (TypeB) trajectory connecting two different placement classes. We call the set $\{M_1^B, M_2^B, \dots, M_5^B\}$ a **certificate**. (b) To construct a solution to a query that starts in $\mathcal{P}_1$ and ends in $\mathcal{P}_5$, one simply plans a set of TypeA trajectories, M_1^A, M_2^A, M_3^A , and M_4^A , as shown in dashed red lines, to bridge the TypeB trajectories. A solution trajectory is then $M = M_1^A * M_2^B * M_2^A * M_3^B * M_3^A * M_5^B * M_4^A$.

corresponding transfer trajectories $M_1^B, M_2^B, \dots, M_{n-1}^B$, where M_i^B is a TypeB trajectory connecting the i^{th} placement in the sequence to the next. Next, we generate a TypeA trajectory connecting M_i^B and M_{i+1}^B for every i . Finally, a solution to the query is constructed by concatenating all the trajectory (using the composition operation). Figure 6.2 illustrates the proposed algorithm.

6.2 Algorithm

As we are now dealing with a bimanual manipulation system, we will slightly modify the definition of *feasible* configurations, previously given in Chapter 3.

Definition 15. A composite configuration $c = (q_1, q_2, T)$ is said to be **non-singular**

if the Jacobians of the two robots, $J_{\mathcal{R}_1}$ and $J_{\mathcal{R}_2}$, have maximal rank. Otherwise, the configuration is said to be **singular**.

Definition 16. A composite configuration is said to be **feasible** if it is collision-free and non-singular and at least one of the following holds:

1. The robot is grasping the object with a valid grasp;
2. The object is at a stable placement.

6.2.1 Generating Trajectories within a Placement Class

In this Section, we investigate the *existence* of manipulation paths connecting two composite configurations in the same placement class (TypeA). The goal of this Section is to assert that given a path $\sigma : [0, 1] \rightarrow SE(2)$ of the object moving from one placement to another in the same placement class, there exists a finite-length manipulation path associated with σ . In other words, the projection via π_p of the manipulation path is σ .

It is worth noting that the proof of existence relies on the Heine-Borel theorem of finite covers, which is non-constructive. We further propose an algorithm that, given an object path σ together with a certain set of assumptions, will return a TypeA manipulation path associated with σ in finite time.

Existence of TypeA Paths

Extending from Definition 13 in Chapter 3, we define a single-transfer connected *component* as follows.

Definition 17. A **single-transfer connected component** $\mathcal{F}$ is a single-transfer connected set such that for any point $c \in \partial\mathcal{F}$, every neighborhood of c contains both points that are single-transfer connected and not single-transfer connected with c .

Let $\mathcal{F}$ be the collection of all single-transfer connected components (see Chapter 3, Definition 13) in $\mathcal{G} \cap \mathcal{P}_i$ – the set of composite configurations where the object is in placement class i and the robots are also grasping the object. Since $\mathcal{G} \cap \mathcal{P}_i$ contains no singular configurations, any $c \in \mathcal{G} \cap \mathcal{P}_i$ must be in some single-transfer connected component. In other words, $\mathcal{G} \cap \mathcal{P}_i = \bigcup_{\mathcal{F} \in \mathcal{F}} \mathcal{F}$. Define $\mathcal{T}_i$ by $\mathcal{T}_i = \pi_p(\mathcal{G} \cap \mathcal{P}_i)$. We have the following proposition.

Proposition 1. *Let $\sigma : [0, 1] \rightarrow SE(2)$ be an object path lying in a connected component of $\mathcal{T}_i$. Then σ is a projection via π_p of some finite-length manipulation path.*

Proof. First note that $\mathcal{T}_i = \bigcup_{\mathcal{F} \in \mathcal{F}} \pi_p(\text{int}(\mathcal{F}))$. Let each projection $\pi_p(\text{int}(\mathcal{F}))$ be denoted by $\mathcal{E}$ and $\mathcal{E}$ the collection of such sets. Since π_p is an open map, each $\mathcal{E}$ is open. Therefore, $\mathcal{E}$ is an open covering of $\mathcal{T}_i$.

Since σ lies entirely in some connected component of $\mathcal{T}_i$, there exists a subcollection $\mathcal{E}'$ of $\mathcal{E}$ that covers σ . Let $\mathcal{I}$ be the collection of open intervals where each interval, $\mathcal{I}$, corresponds to a domain of the path σ such that the path segment $\sigma(\mathcal{I})$ lies entirely in some open set $\mathcal{E}$.

Now we have that $\mathcal{I}$ is an open covering of $[0, 1]$. Since $[0, 1]$ is compact Munkres 2000, there exists a finite subcollection of $\mathcal{I}$ that also covers $[0, 1]$. This means that the path σ consists of a finite number of segments where each segment lies entirely in an open set $\mathcal{E}$ and hence is a projection of a transfer path. Therefore, we can conclude that the path σ is a projection of a finite-length manipulation path. ■

However, since the proof of compactness of the interval $[0, 1]$ is not constructive (Cederquist and Negri 1996), the above proposition does not give us a way to construct a finite-length manipulation path associated to a given object path σ . Note also that since the proof of reduction property¹ given in Alami, Laumond, and Siméon (1994) also relied on the Heine-Borel covering theorem, it also does not provide a practical way to construct a manipulation path.

To explicitly construct an algorithm that, given an object path σ , computes in finite time an associated finite-length manipulation path, we need a set of additional assumptions. The idea behind the construction of the algorithm is that from the uncountable collection $\mathcal{E}$, we need to be able to extract from $\mathcal{E}$ a *countable* (possibly infinite) subcollection that still covers the given path σ . Then from the countable subcollection, we can then iterate through combinations of its members until we find one that covers σ . The Heine-Borel covering theorem helps guarantee that these iterations will eventually terminate in finite time.

Before we proceed to state assumptions, we present the following result. Define the set $\mathcal{F}(\mathbf{g})$ as the union of all element $\mathcal{F}$ of $\mathcal{F}$ where the grasp associated with any composite configuration $\mathbf{c} \in \mathcal{F}(\mathbf{g})$ is specified by the bimanual grasp parameter

¹Reduction property states that two configurations in the same connected component of $\mathcal{G} \cap \mathcal{P}$ are connected by some manipulation path.

vector $\mathbf{g}^2$. Since there may exist multiple IK solutions associating with one grasp, we may categorize the set $\mathcal{F}(\mathbf{g})$ further into a number of subsets according to classes of the associated IK solution (Burdick 1989; Gharbi, Cortés, and Siméon 2009). We write $\mathcal{F}(\mathbf{g}, k), k \in K$ to refer to the set $\mathcal{F}$ with a specific grasp $\mathbf{g}$ such that any $\mathbf{c} \in \mathcal{F}(\mathbf{g}, k)$ has the IK solution in the same class as other configurations. Note the according to Burdick (1989), the index set K is bounded.

Consider a set $\mathcal{E}(\mathbf{g}, k)$, defined as the projection via π_p of $\mathcal{F}(\mathbf{g}, k)$.

Lemma 1. *An object path $\sigma : [0, 1] \rightarrow SE(2)$ lies entirely in a connected component of $\mathcal{E}(\mathbf{g}, k)$ if and only if it is a projection of a transfer path.*

Proof. The result follows from the definition of a single-transfer connected component. ■

Assumptions

Now we present a set of assumptions as follows.

Assumption 1. *For any object path $\sigma : [0, 1] \rightarrow SE(2)$. The intersection between σ and the set $\mathcal{E}(\mathbf{g}, k)$, for any grasp $\mathbf{g}$ and IK class index k , consists of finitely many path segments and the domain of the path parameter for each segment is computable.*

In usual manipulation planning settings, environments are relatively controlled such that they should not contain physical obstacles of extremely odd geometries that would eventually result in the set $\mathcal{E}(\mathbf{g}, k)$ being divided into infinitely many connected components. Furthermore, the robot singularity set is not likely to divide the feasible configuration space into infinitely many connected components as well. This is true, for example, for a class of *generic* manipulators whose singularity sets consist of finite *smooth manifolds* (Pai and Leu 1992). However, the above assumption is still necessary to ensure that each connected component of $\mathcal{E}(\mathbf{g}, k)$ is well-behaved, in the sense that a finite number of components would not result in infinitely many segments.

The second assumption is stated as follows.

Assumption 2. *Given an object path σ contained in a connected component of $\mathcal{E}(\mathbf{g})$, where $\mathbf{g} = [\mathbf{g}_1^\top \mathbf{g}_2^\top]^\top$, $\mathbf{g}_1 = [l_1 \ a_1 \ b_1 \ \delta_1]^\top$, and $\mathbf{g}_2 = [l_2 \ a_2 \ b_2 \ \delta_2]^\top$. There exists a lower bound $\epsilon > 0$, which may depend on σ , such that all $\mathcal{E}(\mathbf{g}')$ also contain σ ,*

²A bimanual grasp parameter vector may be represented as $\mathbf{g} = [\mathbf{g}_1^\top \mathbf{g}_2^\top]^\top$, where $\mathbf{g}_1$ and $\mathbf{g}_2$ are unimanual grasp parameter vectors for the two robots, as defined in Chapter 3.

where $\mathbf{g}' = [\mathbf{g}'_1{}^\top \mathbf{g}'_2{}^\top]^\top$, $\mathbf{g}'_1 = [l_1 a_1 b_1 (\delta_1 + \Delta_1)]^\top$, $\mathbf{g}'_2 = [l_2 a_2 b_2 (\delta_2 + \Delta_2)]^\top$, and $0 < \Delta_1, \Delta_2 \leq \epsilon$.

This means that if the robots can grasp the object with the bimanual grasp $\mathbf{g}$ and then trace the object path σ . The robots can also grasp the object and trace the same object path with some *nearby grasps*.

Now suppose, without loss of generality, that the domain of the grasp parameter δ is normalized to $(0, 1)$ for each grasp class. Next, we define $\mathcal{B}(d)$, with $d = (d_1, d_2) \in (0, 1) \times (0, 1)$, as the collection of all $\mathcal{E}(\mathbf{g}, k)$ such that the grasp parameters $\delta_1 = d_1$ and $\delta_2 = d_2$. Consider the set $\mathcal{A}$ defined by

$$\mathcal{A} = \{\mathcal{B}(h_1), \mathcal{B}(h_2), \mathcal{B}(h_3), \dots\}, \quad (6.1)$$

where $\{h_1, h_2, h_3, \dots\}$ is a two-dimensional low-discrepancy sequence, such as the sequence introduced in Lindemann and LaValle (2003). This set $\mathcal{A}$ is then basically an enumeration of $\mathcal{B}(d)$ at values d from such a sequence. Thanks to Assumption 2, we have that there exists an integer N such that the subset $\mathcal{A}'$ of the first N terms of $\mathcal{A}$ covers σ .

Assumption 3. *The connectivity of a set $\mathcal{T}_i$ for all i can be determined empirically, e.g., by discretization.*

The above discretization can be easily done on the three parameters (x, y, θ) parameterizing the placement. Ranges of the parameters x and y are determined by the user while θ ranges from 0 to 2π . After obtaining the set of discretized coordinates, one tests at each point if the object is collision-free and graspable by the robots. Figure 6.3(a) shows the scene in which we tested the connectivity of $\mathcal{T}_i$. The set $\mathcal{T}_i$ is visualized in Figure 6.3(b) by being superimposed into the scene.

The idea here is that once the connectivity of the set $\mathcal{T}_i$ is determined, we can treat different connected components of $\mathcal{T}_i$ (if any) as different placement classes when computing a certificate. Therefore, we shall suppose in the sequel that each $\mathcal{T}_i$ consists of one connected component.

Algorithm

Consider next the algorithm listed in Algorithm 6. To compute a TypeA motion to move the object along a path segment $\sigma(s), s \in [t, t']$ (in ComputeCCMotion line 7), one starts with the initial IK solution of the robot grasping the object at

Figure 6.3: (a) The chair at a transformation in the placement class of interest. (b) The set $\mathcal{T}_i$ plotted out in 3D, shown in green. Each 3D point represents (x, y, θ) at which the object can be grasp by the robots. The vertical axis is θ , varying from 0 to 2π . One can see from the figure that $\mathcal{T}_i$ is indeed connected.

$\sigma(t)$. Note that the grasp as well as the IK solution can be determined uniquely from the element A . Since $[t, t'] \subset (a, b)$, it is possible for one to use a differential IK algorithm (Siciliano et al. 2009) to solve for remaining IK solutions along the path (according to Lemma 1).

Based on the three assumptions presented in the previous Section, we have the following proposition.

Proposition 2. *Given an object path σ lying entirely in $\mathcal{T}_i$, Algorithm 6, with a complete motion planner deployed as a local planner, will terminate in finite time with a finite-length manipulation trajectory whose projection via π_p is σ .*

Proof. Consider first the function Planner in Algorithm 6. Since the set $\mathcal{A}$ provides a countable open covers of the path σ , and since the closed interval $[0, 1]$ of the path parameter is compact, there exists a finite subcover of σ . Therefore, only a finite number of iterations is required before the while loop (in Planner line 2) terminates. In each iteration of the while loop, one call to the function CheckCover is made. From Assumption 1 of finite intersections with σ , one would require finite time to verify intersections of each element A of $\mathcal{A}'$ with the path σ . Therefore, CheckCover always terminates in finite time.

Since $\mathcal{A}'$ is a finite subcover of the path σ , the while loop in ComputeCCMotion

will eventually terminate. This concludes the proof. ■

Note here that instead of a complete motion planner, a *probabilistically* complete motion planner can be used in Algorithm 6, ComputeCCMotion, line 7. In this case, the overall planner will be *probabilistically* certified-complete (see Section 6.4 for a more detailed discussion).

Algorithm 6: In-placement manipulation planner

Input: object path σ

Output: TypeA trajectory

```

1  $\mathcal{A}' \leftarrow \{\}, j \leftarrow 1$ 
2 while True do
3   Append the  $j^{\text{th}}$  element of  $\mathcal{A}$  to  $\mathcal{A}'$ ,  $j++$ 
4   covered  $\leftarrow$  CheckCover( $\sigma, \mathcal{A}'$ )
5   if covered then
6     break
7   end if
8 end while
9 return ComputeCCMotion( $\sigma, \mathcal{A}'$ )
```

6.2.2 Generating Trajectories between Placement Classes

It follows from Proposition 1 that if we have one TypeB trajectory that starts in some placement class $\mathcal{P}_i$ and ends in some other placement class $\mathcal{P}_j$, any pair of composite configurations in $\mathcal{P}_i \cup \mathcal{P}_j$ are also manipulation path-connected. In the case when there are only two placement classes available, any TypeB path between the two placement classes then guarantees the existence of a solution to any feasible manipulation query. With n_p placement classes available, one only needs a minimum of $n_p - 1$ TypeB trajectories between different pairs of placement classes in order to guarantee the existence of a solution to any feasible query. Therefore, we define the notion of a *certificate* as follows.

Definition 18. A *certificate* is a set of transfer paths that spans all the placement classes.

One can think of placement classes as nodes in a graph. A certificate is then analogous to a set of edges that contains a spanning tree's edges. Although $n_p - 1$ transfer paths are sufficient to guarantee the existence of solutions to any manipulation query, the more transfer paths one has (between distinct pairs of placement

Algorithm 7: ComputeCCMotion

Input: σ and $\mathcal{A}'$ **Output:** TypeA trajectory

```
1  $t \leftarrow 0$ , reached  $\leftarrow$  False
2 traj  $\leftarrow$  an empty trajectory
3 while not reached do
4   Select an element  $A$  from  $\mathcal{A}'$  that contains  $\sigma(t)$ . Let  $(a, b)$  be the
     domain of  $\sigma$  covered by  $A$  such that  $t \in (a, b)$ 
5   Select an element  $A'$  from  $\mathcal{A}'$  that contains  $\sigma(b)$ . Let  $(a', b')$  be the
     domain of  $\sigma$  covered by  $A'$  such that  $b \in (a', b')$ .
6   Select  $t' \in (a', b)$ 
7   Compute a closed-chain motion for  $\sigma(s), s \in [t, t']$  (and other
     necessary transit motions for regrasping)
8   Append the compute trajectories to traj
9   if  $t' == 1$  then
10    | break
11   end if
12    $t \leftarrow t'$ 
13 end while
14 return traj
```

classes) can contribute to higher quality of solutions since the system may need to visit a fewer number of intermediate placement classes before reaching the desired placement class.

Since the process of computing a certificate needs to be done only once per problem setting, we suppose that this computation can be done offline and the computation time is not a limiting factor. Therefore, one may aim at generating all ${}^{n_p}C_2 = n_p(n_p - 1)/2$ transfer paths connecting all possible different pairs of placement classes.

Given a pair of placement classes $\mathcal{P}_i$ and $\mathcal{P}_j$, we divide the process of generating a TypeB trajectory into two main parts: 1. generating a closed-chain query; and 2. solving a closed-chain query.

Generating a Closed-Chain Query

Randomly sampling two object transformations, one from each placement class, may have a relatively low probability that the resulting closed-chain query is solvable. This is mainly due to the fact that the closed-chain constraint greatly reduces the range of motions of the system. To deal with this issue, we propose a heuris-

tic to help generate closed-chain queries that are likely solvable. The idea behind this is that since the bimanual manipulation system can exhibit a very limited range of motions, queries should be generated such that they intuitively do not require a large range of robot motions to solve them.

Recall that object transformations in any placement class can be parameterized by three parameters (Section 6.2.1). The problem of generating closed-chain queries is then boiled down to how one generates the three parameters for the start and goal transformations. We first define a **manipulation point** (x_m, y_m) on the support surface. The position parameters (x, y) of the transformations to be generated will be assigned to be this point. Doing so greatly simplifies query generation while not drastically reduce the possibilities of the queries generated since the motion range of the system is already very limited. The manipulation point may be assigned to be on the middle line passing between the two robots (the green line in Figure 6.4), roughly speaking, to maximize the reachability of the two robots.

Then the rotation parameter θ can be computed as follows. Let f_j denote the face of $\mathcal{H}$ corresponding to placement class $\mathcal{P}_j$ (the goal placement class). Let $\mathbf{n}_j$ be a normal vector pointing outwards from the face f_j (see the blue arrow in Figure 6.4(a)). The desired value of θ is such that the projection of $\mathbf{n}_j$ onto the support surface is parallel to the line l . The reason behind this is that once the object is arranged as mentioned, the expected closed-chain motion to move the object from placement $\mathcal{P}_i$ to $\mathcal{P}_j$ will be a relatively easy flipping motion, as shown by the blue arrow in Figure 6.4(a). After the start transformation has been computed, the goal transformation can be computed accordingly (Figure 6.4(b)).

Apart from the two transformations T_{start} and T_{goal} , we may also include into the query a grasp $\mathbf{g}$ together with associated IK solutions at T_{start} and/or T_{goal} . Generating a grasp $\mathbf{g}$ is straightforward since it can be sampled from a set of grasps available at both transformations.

Computing associated IK solutions at T_{start} and/or T_{goal} , however, is non-trivial when the query is to be solved via a bidirectional planner. This is because given two composite configurations in the same grasp class, there is no known way to completely determine if they belong to the same connected component of $\mathcal{G}$. Generally, one set of IK solutions of robots grasping the object corresponds to one connected component of $\mathcal{G}$ called self-motion manifold (Burdick 1989). To choose IK solutions of robots grasping the object at both T_{start} and/or T_{goal} , we rely on an ad hoc heuristic since, to our knowledge, there is currently no known generalized way to do so.

Finally, note that this two-stage approach in generating and solving queries may

Figure 6.4: (a) The start transformation of the chair. The chair is placed around the center line (green) such that the normal vector (the dotted arrow) of the face f_j is pointing in the direction of the center line. By arranging the chair as shown, the chair is only needed to be flipped in the direction depicted by the curved arrow. (b) The goal transformation of the chair.

proceed in iterations. If a generated query is not solvable within the given time, one can generate a new query by defining a new manipulation point, e.g., by adding some small perturbation to the point.

Solving a Closed-Chain Query

Our closed-chain motion planner (CCPlanner) is adapted from a bidirectional RRT planner (Lavalle and Kuffner 2001). In particular, we build two trees $\mathcal{T}_a$ and $\mathcal{T}_b$, each one is a data structure storing vertices. A vertex $\mathcal{V}$ keeps information of a composite configuration, its parent on the tree, as well as a closed-chain trajectory connecting itself and its parent. The algorithm for CCPlanner is listed in Algorithm 8.

CCPlanner accepts the start and goal transformations, T_{start} and T_{goal} and IK solutions at T_{start} and T_{goal} as its inputs. It starts by initializing two trees, i.e., creating root vertices storing configuration information. In each planning iteration, CCPlanner randomly samples an object transformation matrix T . Then the planner tries to extend a tree towards it. Upon a successful extension the planner tries to connect two trees together. If the connection attempt is successful, the closed-chain path is extracted from the tree and returned. Some key functions in Algorithm 8 and Algorithm 9 are described in details below.

- **SampleSE3()**: A transformation matrix (an element of $SE(3)$) is generated by separately sampling rotational and translational parts. A rotation matrix is uniformly sampled from $SO(3)$ via the method proposed in Kuffner (2004). A translation

Algorithm 8: Closed-chain motion planner (CCPlanner)

Input: $T_{\text{start}}, T_{\text{goal}}, (q_{1,\text{start}}, q_{2,\text{start}})$, and $(q_{1,\text{goal}}, q_{2,\text{goal}})$ **Output:** Closed chain trajectory or None

```
1  $\mathcal{T}_a \leftarrow \text{Tree}(); \mathcal{T}_b \leftarrow \text{Tree}()$ 
2  $\text{InitializeTree}(\mathcal{T}_a, T_{\text{start}}, (q_{1,\text{start}}, q_{2,\text{start}}))$ 
3  $\text{InitializeTree}(\mathcal{T}_b, T_{\text{goal}}, (q_{1,\text{goal}}, q_{2,\text{goal}}))$ 
4 while time is not exhausted do
5    $T \leftarrow \text{SampleSE3}()$ 
6   if  $\text{ExtendTree}(\mathcal{T}_a, T)$  then
7     if  $\text{Connect}(\mathcal{T}_a, \mathcal{T}_b)$  then
8       return  $\text{ExtractTrajectory}(\mathcal{T}_a, \mathcal{T}_b)$ 
9     end if
10  end if
11   $\text{Swap}(\mathcal{T}_a, \mathcal{T}_b)$ 
12 end while
13 return None
```

vector is sampled uniformly from the user defined range.

- **Threshold($\mathcal{V}, T$):** Given a distance metric w in $SE(3)$ (such as the one introduced in Chapter 3, Equation 3.1) and a user-defined maximum distance W , this function returns an object transformation T_{new} , where
 - if $w(\mathcal{V}.\text{config}.T, T) \leq W$, then $T_{\text{new}} \leftarrow T$;
 - otherwise, T_{new} is an object transformation in the direction from $\mathcal{V}.\text{config}.T$ towards T but within a distance W from $\mathcal{V}.\text{config}.T$.
- **InterpolatePath($\mathcal{V}, T$):** This function generates a closed-chain trajectory that starts from the composite configuration $\mathcal{V}.\text{config}$ to bring the object to T . To do so, we generate an $SE(3)$ trajectory first, by using the method in Park and Ravani (1997) for the rotational part and using polynomial interpolation for the translational part. Then the trajectory is discretized into small time steps. IK solutions of the two robots are computed at each time step using the differential IK method (Siciliano et al. 2009).

Apart from being less complicated implementation-wise, this method gives an exact parameterization of the object trajectory. One can then incorporate various types of constraints into the object trajectory by means of time-parameterization to obtain time-optimal trajectory with respect to the constraints (see Pham (2014) for more details on time-optimal path parameterization). Examples of applicable

Algorithm 9: ExtendTree

Input: $\mathcal{T}, T$ **Output:** True or False

```
1 for  $\mathcal{V}_{\text{near}}$  in  $\text{KNN}(\mathcal{T}, T)$  do
2    $T_{\text{new}} \leftarrow \text{Threshold}(\mathcal{V}_{\text{near}}, T)$ 
3    $P_{\text{new}} \leftarrow \text{InterpolatePath}(\mathcal{V}_{\text{near}}, T_{\text{new}})$ 
4   if  $P_{\text{new}}$  is not None and  $P_{\text{new}}$  is collision-free then
5      $\mathcal{V}_{\text{new}} \leftarrow \text{Vertex}()$ 
6      $\mathcal{V}_{\text{new}}.\text{path} \leftarrow P_{\text{new}}$ 
7      $\text{AddVertex}(\mathcal{T}, \mathcal{V}_{\text{near}}, \mathcal{V}_{\text{new}})$ 
8     return True
9   end if
10 end for
11 return False
```

constraints are velocity and acceleration limits for rigid body motions (Nguyen and Pham 2016) and dynamic grasp stability.

Nevertheless, the user can utilize their trajectory generation method of choice. In case an exact parameterization of a closed-chain trajectory is available, one may also use the TOPP method for redundantly-actuated systems (Pham and Stasse 2015) (which is the case for bimanual systems) to enforce the aforementioned constraints along the trajectory.

- $\text{Connect}(\mathcal{T}_a, \mathcal{T}_b)$: After a successful tree extension, the planner will attempt to connect the newly added vertex to the other tree. The details of procedure are mainly similar to `Extend`, except this function does not include `Threshold`.

Note that for this closed-chain planning component, other variants such as the planner presented in Chapter 5 can also be utilized.

6.3 Experiments

The planner and all related functions were implemented in Python. We used OpenRAVE Diankov 2010 as a simulation environment. The robots were two identical 6-DOF industrial manipulators Denso VS-060. Each one was equipped with an ATI Gamma Force-Torque sensor and a 2-finger Robotiq gripper 85. The planning environment is as shown in Figure. 6.3. All simulations were run on a 2.4 GHz desktop.

6.3.1 Computing Certificates

To validate our certificate issuing planner, we ran the planner to compute certificates for a set of objects. All objects, listed in Figure 6.5, were furniture pieces that were relatively large such that they needed two robots grasping in order to move them. For each object, we repeated certificate generation for 50 times. When computing each transfer path, we also had the following additional computations: 1. grasp equilibrium checking 2. closed-chain trajectory shortcutting (200 iterations). Statistics collected from the runs are reported in Table 6.1 in “Computing Certificate” category. Here are a few things we would like to point out:

- The planner may spend up to around 45% of the total time generating and planning unsuccessful queries. This is because there currently exists no definite method to check that the generated IK solutions associated with the start and goal transformations belong to the same connected component (self-motion manifold).
- Due to the large number of grasp classes, solving a bimanual manipulation query via, for example, the three-dimensional extension of the high-level Grasp-Placement Graph (Lertkultanon and Pham 2015) could potentially be rendered infeasible. Consider Object 2 (Figure 6.5(b)), for example, which has 3 placement classes and 51 *unimanual* grasp classes. While the two-dimensional Grasp-Placement Graph has 104 vertices and 2028 edges, its three-dimensional counterpart contains over 3,000 vertices with over 6 million edges.
- For each run, the certificate computation procedure is considered successful if the computed transfer paths span all the placement classes. Although the number of successfully planned TypeB paths varied slightly among different runs, the resulting set of TypeB paths still spanned all placement classes in every run.
- In the current implementation, we check grasp equilibrium by solving a linear program at each discretized time step along a closed-chain trajectory. This approach is, however, time-consuming and restrictive in that it only guarantees static equilibrium³. One possible improvement is by formulating contact constraints in terms of inequalities in path parameters and its derivatives (Caron, Pham, and Nakamura 2015a) by utilizing cone double-description (CDD) method (Fukuda and Prodon 1996). Then one can *time-optimally* parameterize the tra-

³Trajectories with static equilibrium are only guaranteed to be executable at *arbitrarily slow* speed.

jectory such that it moves as fast as possible while respecting all the constraints (see Pham (2014) and references therein for more details).

6.3.2 Solving Bimanual Queries

First, for each object listed in Figure 6.5, we hand-pick two placement classes that do not have any direct connection via a TypeB trajectory (information provided by a certificate) and generated a pair of object transformations T_{start} and T_{goal} from each of the placement classes. Then we repeated solving each query $Q = (T_{\text{start}}, T_{\text{goal}})$ for 50 times. Statistics collected from the runs are reported in Table 6.1 in “Solving Queries” category. One of the main factors that causes variations in the running time is the geometry of each object. Larger objects, for example, have *narrower* free space to navigate on the support surface. Furthermore, with larger objects, it is also more difficult for robots to move around and change grasps.

A solution manipulation path to any of the above queries needs to be at least of length 5. However, it is not the case here since a TypeA trajectory, which connects two TypeB trajectories from a certificate, will always contain regrasp operations. This is because the grasps used in the two TypeB trajectories are always different. Therefore, if a planner with an extension of the high-level Grasp-Placement Graph is used, it will spend a considerable amount of time invalidating manipulation paths of shorter length, hence not practical. A general manipulation planner such as a Random-MMP (Hauser and Ng-Thow-Hing 2011) is not likely to terminate with a solution within a reasonable amount of time as well since it needs to sample correctly a relative long sequence of transit and transfer.

Apart from the simulation results, we also successfully carried out a hardware experiment. We constructed a query for Object 2 (Figure 6.5(b)) such that there is no direct TypeB trajectory connecting the two placement classes. The scene with the start and goal transformations of the object provided by the query are shown in Figure 6.6(a) and Figure 6.6(f), respectively. The computed solution to this query consists of three TypeA trajectories ((a)→(b), (c)→(d), and (e)→(f)) and two TypeB trajectories ((b)→(c) and (d)→(e)).

The controller used in this experiment was similar to the one presented in Xian, Lertkultanon, and Pham (2017) (see Appendix A for more details). The video of the robots executing the motion solving this query (on the real platform) can be found at <https://youtu.be/4DcMwr2xxrQ>.

Figure 6.5: Objects used in our experiments of certificate computation (not to scale).

Figure 6.6: Key object transformations along the computed bimanual manipulation trajectory solving the given query for Object 2 (Figure 6.5(b)).

Table 6.1: Computation Time for Computing Certificates and Solving Queries for the Objects in Figure 6.5.

		Object 1	Object 2	Object 3	Object 4	Object 5	Object 6
Computing Certificates	# placement classes	4	3	6	6	8	6
	# grasp classes	1849	2601	2401	3364	1089	784
	total	avg. (s.)	415.97	281.18	867.71	1374.27	574.54
	computation time	std. (s.)	141.49	98.48	212.54	342.11	120.29
	*planning (s.)		154.36	94.07	604.19	582.54	199.63
	*grasp check (s.)		101.69	76.27	202.90	157.90	154.11
	*shortcutting (s.)		14.51	7.29	32.76	34.83	23.58
	# TypeB trajectories (avg.)		4.00	2.00	11.02	11.98	7.40
	success rate		1.0	1.0	1.0	1.0	1.0
Solving Queries	total	avg. (s.)	103.65	45.08	127.45	109.99	185.08
	computation time	std. (s.)	18.04	14.93	29.93	24.37	31.55
	# TypeB trajectories (avg.)		2.0	2.0	2.0	2.0	2.0

*Row 3 – 5 of “Computing Certificate” category are planning time, grasp checking time, and shortcutting time averaged over successful closed-chain queries.

6.4 Discussion

6.4.1 Completeness Property

The proposed algorithm has a property so-called *certified-completeness*. A computed certificate certifies that the algorithm will find a solution to any bimanual manipulation query, in the environment in which the certificate is computed, whenever one exists. This completeness is, however, up to the completeness of the motion planner deployed as a local planner. To see this, consider the computation of TypeA trajectories according to Algorithm 6. In each iteration of the function `ComputeCCMotion` a segment of the object path σ that lies in a single-transfer connected set is identified. A transfer trajectory that moves the object along the segment of σ can then be computed accordingly by, for example, computing IK solutions for the two robots associated with discretized object configurations along the segment⁴. This closed-chain motion computation will always succeed since the object path segment lies entirely in a single-transfer connected set. However, one also needs to compute transit trajectories to connect consecutive transfer trajectories, i.e., to change the robot grasps. Therefore, the completeness of the overall algorithm will be affected by the completeness of the planner deployed to compute these transit trajectories. When a probabilistically complete motion planner is used, the proposed manipulation planning algorithm will be *probabilistically* certified-complete.

Complexity of Algorithm

We analyze computational complexity of the certificate computation phase and the query phase in terms of the number of motion planning queries that needs to be solved. Consider the certificate computation phase. Let n_p be the number of all placement classes, $S(i, j)$ the event that a closed-chain query for moving the object between placement classes i and j , $i \neq j$, is successfully solved. Suppose there exists a probability p such that $P[S(i, j)] \geq p$, for all $i \neq j$. The value of p may be estimated empirically. Let X be a random variable representing the number of consecutive failure before the first successfully solved closed-chain query. From direction calculation, one has $E[X] \leq 1/p - 1$. Therefore, for any pair of placement classes, the number of closed-chain queries one is expected to solve until encountering the first feasible query is $O(1/p)$. In the worst case scenario, one needs to

⁴Methods such as the differential IK method Siciliano et al. 2009 can be used to ensure continuity of IK solutions.

consider all possible $n_p(n_p - 1)/2$ pairs of placement classes in order to compute a certificate. Therefore, the number of closed-chain queries one needs to solve to compute a certificate is $O(n_p^2/p)$. In the query phase, TypeA trajectories are created via Algorithm 6. Since there are n_p placement classes, the number of TypeA trajectory queries that needs to be solved is $O(n_p)$.

Time complexity of motion planning when solving each open-chain or closed-chain query is affected largely by constraints such as collision avoidance (with both physical obstacles and virtual obstacles such as joint limits). These constraints reduces the configuration space expansiveness in a non-trivial manner and therefore, according to the analysis in Hsu, Latombe, and Motwani (1997), lead to longer running time. In addition, more restrictive constraints could lead to higher number of regrasp operations required in a TypeA trajectory. This, in turn, results in more transit trajectory queries to be solved.

Complexity may be analyzed in greater details by taking into account complexity of each primitive procedure in motion planning such as nearest neighbor search, collision checking, or IK computation as was done in Karaman and Frazzoli (2011). However, these results are theoretical. In practice, there are a number of factors that affect the actual running time of the planning algorithm, for example, the constants omitted in those complexity expressions. Careful implementation can also play a crucial role in reducing running time. As an example, consider transit trajectory generation procedure. When the two n -DOF robots change their grasps, the motion planning problem can be formulated as either one $2n$ -DOF problem or two sequential n -DOF problems. This poses a trade-off between motion planning time and motion execution time and it is therefore up to the user to strike a balance between the two.

6.4.2 Limitations and Future Works

Various improvements can be made to the planning process. For example, currently when computing TypeB trajectories (see Section 6.2.2), the generated closed-chain query will fail when the start and goal configurations live in different single-transfer connected components. To improve success rate of certificate computation, one may deploy a closed-chain planner presented in Xian, Lertkultanon, and Pham (2017), which can plan motions across different connected components, to solve such queries.

Furthermore, the proposed algorithm depends on the accuracy of its inputs, for

example, the 3D model of the object, the relative transformation between the robots, the initial object pose, environment geometry, etc. Any discrepancies between the actual environment and the data are accounted for by the compliant control algorithm used in motion execution. However, as these discrepancies can propagate and be intensified throughout the motion execution process, they may eventually lead to failure of execution or cause damages to the manipulated object. One important future direction is then to investigate how the knowledge of data uncertainty can be incorporated into planning so as to plan more robust trajectories.

Conclusion

We have presented a new bimanual manipulation planner with the property called *certified-completeness*. In particular, for a given object and environment, we have provided a method to compute a certificate that the planner will find a solution to *any* bimanual query whenever one exists. The certificate is also constructive in that it can be used to quickly construct a solution to a given bimanual query. We validated the proposed manipulation planner in software, by computing certificates and solving bimanual queries for a number of different furniture pieces, and hardware, by executing the computed manipulation trajectory on the actual system. To the best of our knowledge, **this is the first time the capability of *bimanually* manipulating (i.e., regrasping) heavy/large objects is demonstrated on an actual bimanual system.**

Chapter 7

Improvement to Motion Planning: Minimum-Switch-Time Constraints

This chapter introduces and discusses benefits of a new type of constraints, called minimum-switch-time constraints, to be incorporated into polynomial trajectory generation. Then an algorithm for planning parabolic trajectories subject to both physical bounds (robot joint velocity and acceleration limits) and the new constraint is presented. This work has been published in **Lertkultanon, Puttichai and Quang-Cuong Pham (2016)**. “Time-optimal parabolic interpolation with velocity, acceleration, and minimum-switch-time constraints”. In: *Advanced Robotics* 30.17–18, pp. 1095–1110. An implementation of parabolic trajectory interpolation routines is openly available at <https://github.com/puttichai/parabint>.

7.1 Trajectory Planning Pipeline

7.1.1 Overview

The planning pipeline that we use is called *plan-and-shortcut*. This pipeline is one of the most robust (Luo and Hauser 2014) and widely used in industrial robotics; it is the default planning pipeline implemented in the robot programming environment OpenRAVE (Diankov 2010), which is used by many research groups and robotic companies worldwide.

This pipeline consists of two main stages: planning and shortcutting. In the first stage, we solve a trajectory planning problem, which asks for a trajectory connecting two given robot configurations $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ in the robot joint space $\mathcal{C} \subset \mathbf{R}^n$, where n is the number of degrees-of-freedom (DOF) of the robot, subject to robot joint velocity and acceleration limits, as well as obstacle avoidance. Here the plan-

ning stage is based on the path-velocity decomposition approach (Kant and Zucker 1986), where the trajectory planning problem is decomposed into path planning and time-parameterization.

In path-planning stage, a path planner, e.g., a Rapidly-exploring Random Tree (RRT) (LaValle 1998), is used to search for a collision-free *path* connecting $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$. The solution path obtained is piecewise linear since it is obtained by concatenating linear path segments together. Note that each linear segment from $\mathbf{x}_a$ to $\mathbf{x}_b$ will be represented by a parameterization $\mathbf{x}(s) = \mathbf{x}_a + s(\mathbf{x}_b - \mathbf{x}_a), s \in [0, 1]$. Next, in time-parameterization stage, a *velocity profile*, which is a time-derivative of the path parameterization function $s : [0, T] \rightarrow [0, 1]$, is computed and assigned to the path. T is the total duration of the velocity profile, and hence of the resulting trajectory. We note here that this time-parameterization problem can alternatively be viewed as a **velocity profile interpolation problem**¹. To preserve the path geometry (avoiding thereby new collisions) at this stage, one must time-parameterize paths of different joints *simultaneously*, i.e., all n paths are time-parameterized through the same parameterization function s . In addition, to avoid any velocity discontinuity that may occur at the junctions of consecutive linear paths, velocities at the beginning and the end of the each path segment must be set to zero. At the end of this stage, one will have a trajectory $\mathbf{x} : [0, T] \rightarrow \mathcal{C}$, with $\mathbf{x}(0) = \mathbf{x}_{\text{start}}$ and $\mathbf{x}(T) = \mathbf{x}_{\text{goal}}$.

In the final stage, a randomized, iterative shortcutting procedure, often called a *smoother*, is applied to smoothen, as well as shorten (in time), the initial trajectory, which often has jerky behavior and long duration due to (i) the nature of a randomized path planner; and (ii) the start-stop behavior at each junction between linear segments. In each shortcutting iteration, two random time instants t_a and t_b , $t_b > t_a$, are selected. Then a *time-optimal* trajectory is interpolated between $(\mathbf{x}(t_a), \dot{\mathbf{x}}(t_a))$ and $(\mathbf{x}(t_b), \dot{\mathbf{x}}(t_b))$. At this stage, different DOFs may be interpolated *independently*, provided that all the resulting interpolants have the same duration. Finally, the original trajectory segment will be replaced by the shortcut if the shortcut has shorter duration than $t_b - t_a$ and is collision-free.

From the pipeline, two interpolation primitives are required, regardless of the presence of the minimum-switch-time constraint:

1. simultaneous interpolation with zero terminal velocities (in the planning stage);
and

¹This perspective will be crucial when solving the problem with the additional minimum-switch-time constraint.

2. independent interpolation with arbitrary terminal velocities subject to the condition that all the interpolants have the same duration (in the shortcutting stage).

The following Sections will briefly recall the algorithm for parabolic trajectory planning *without* the minimum-switch-time constraint. We also make a contribution to the re-interpolation problem without the minimum-switch-time constraint by proposing a new re-interpolation method that achieves a higher success rate than the state-of-the-art method.

7.1.2 Trajectory Planning without Minimum-Switch-Time Constraints

In this section, we first briefly recall the algorithm for parabolic trajectory planning *without* the minimum-switch-time constraint for the single-DOF and multi-DOFs cases. Then we also propose a new method for the multi-DOF with attain a higher success rate than that of the currently available methods.

For convenience, we first give some definitions related to velocity profiles.

Definition 19. A **ramp** is a constant-acceleration velocity profile. A ramp with non-zero acceleration is called a *parabolic ramp* or *P-ramp* while a ramp with zero acceleration is called a *linear ramp* or *L-ramp*.

Definition 20. A *P-ramp* with maximal (respectively minimal) acceleration is denoted by P^+ (respectively P^-). An *L-ramp* with maximal (respectively minimal) acceleration is denoted by L^+ (respectively L^-).

Simultaneous interpolation with zero terminal velocities

Consider a linear path segment connecting two configurations $\mathbf{x}_a$ and $\mathbf{x}_b$, parameterized by a path parameterization function $s : [0, T] \rightarrow [0, 1]$, i.e., $\mathbf{x}(t) = \mathbf{x}_a + s(t)(\mathbf{x}_b - \mathbf{x}_a)$. Recall that at this stage, a valid velocity profile $\mathbf{v} = \dot{\mathbf{x}}$ must be subject to boundary conditions: $\mathbf{v}(0) = \mathbf{v}(T) = 0$. Let $\mathbf{v}_m$ and $\mathbf{a}_m$ be the n -vectors of robot joint velocity and acceleration limits, respectively. From the relation $\mathbf{v}(t) = \dot{s}(t)(\mathbf{x}_b - \mathbf{x}_a)$, we have that the path velocity $\dot{s}$ have to be bounded by

$$\dot{s}_m = \min_i \frac{v_{m,i}}{|\mathbf{x}_{b,i} - \mathbf{x}_{a,i}|}. \quad (7.1)$$

Similarly, from the relation $\mathbf{a}(t) = \ddot{\mathbf{s}}(t)(\mathbf{x}_b - \mathbf{x}_a)$, the path acceleration bound can be computed as

$$\ddot{s}_m = \min_i \frac{a_{m,i}}{|\mathbf{x}_{b,i} - \mathbf{x}_{a,i}|}. \quad (7.2)$$

Now the problem of n -DOF path parameterization has been converted to a single-DOF path parameterization, or *velocity profile interpolation problem*, subject to the velocity bound $\dot{s}_m$ and the acceleration bound $\ddot{s}_m$. The time-optimal velocity profile can be shown to be symmetric and is either of the type P^+P^- or $P^+L^+P^-$. In the P^+P^- case, the switch point can be computed as $t_s = 1/\sqrt{\ddot{s}_m}$. In the $P^+L^+P^-$ case, the first switch point is $t_0 = \dot{s}_m/\ddot{s}_m$ while the switch time between the two switch points is $t_1 = 1/\dot{s}_m - 1/\ddot{s}_m$.

Independent interpolation with arbitrary terminal velocities

The general procedure for this stage can be stated as follows. The details of the steps will be briefly discussed below.

1. Interpolate the time-optimal velocity profile for each DOF *independently*.
2. Determine the *slowest* DOF. Suppose the k^{th} joint trajectory has the longest duration of T seconds.
3. *Re-interpolate* a velocity profile for each of the remaining joint such that all velocity profiles have the same duration T .

Single DOF: Consider a linear path segment connecting two configurations $\mathbf{x}_0$ and $\mathbf{x}_1$. The time-optimal velocity profile for the path can be shown to be one of the following four types: P^+P^- , P^-P^+ , $P^+L^+P^-$, and $P^-L^-P^+$, depending on the terminal velocities v_0 and v_1 , and the velocity and acceleration bounds v_m and a_m ².

Given a set of boundary conditions and bounds, we can *interpolate* the time-optimal velocity profile by exploring all of the above four cases.

Multiple DOFs: Solutions to this step are not unique since there may be multiple ways of *stretching* the duration of a given velocity profile to a given new duration.

One possible way to re-interpolate a velocity profile is proposed in Hauser and Ng-Thow-Hing (2010). Given a duration T and a time-optimal single-DOF velocity profile with some duration less than T , the authors suggested to explore four

²Note that in cases when either of the terminal velocities is at the bound, the time-optimal velocity profile will be L^+P^- or L^-P^+ . However, both cases can be treated as special cases of P^+P^- and P^-P^+ , respectively.

possibilities similar to P^+P^- , P^-P^+ , $P^+L^+P^-$, and $P^-L^-P^+$. In each case, the two P-ramps are constrained to have the same magnitude of accelerations $|a|$. This additional constrain allows them to solve for the acceleration but at the same time considerably deteriorates the performance of the shortcutting method.

To cope with the issue, we propose new re-interpolation methods—with and without considering the minimum-switch-time constraint—that achieve a higher success rate. Our re-interpolation method without the presence of the minimum-switch-time constraint, presented in the remaining part of this Section, was also implemented in OpenRAVE (Diankov 2010) as the current default shortcutting procedure. The method with consideration of the minimum-switch-time constraint will be presented in Section 7.2.2.

New method for independent interpolation with arbitrary terminal velocities

Let T be a given duration to which we want to stretch the duration of the original velocity profile. First, we try to re-interpolate the given velocity profile to have two ramps. If this re-interpolation does not lead to any feasible solution, we try to modify the re-interpolated profile accordingly.

Let a'_0 and a'_1 be the accelerations of the first and last ramps of the new velocity profile, t'_0 the duration of the new first ramp, and $t'_1 = T - t'_0$. Starting from

$$d = \left(v_0 t'_0 + \frac{1}{2} a'_0 t'^2_0 \right) + \left((v_0 + a'_0 t'_0) t'_1 + \frac{1}{2} a'_1 t'^2_1 \right), \quad (7.3)$$

where d is the displacement done by this trajectory and v_0 is the initial velocity. Then we can write a'_0 and a'_1 in terms of t'_0 as

$$a'_0 = A + \frac{B}{t'_0} \quad \text{and} \quad (7.4)$$

$$a'_1 = A - \frac{B}{t'_1}, \quad (7.5)$$

where $A = (v_1 - v_0)/T$, $B = (2d/T) - (v_0 + v_1)$, and v_1 is the final velocity of the profile.

We want to get the new velocity profile that has minimal acceleration. Therefore, we set the minimization objective J to

$$J(t'_0) = a'^2_0(t'_0) + a'^2_1(t'_0). \quad (7.6)$$

To minimize J , we start by calculating feasible ranges of t'_0 according to the acceleration constraint as follows.

1. Acceleration constraint for the first ramp: $-a_m \leq a'_0 \leq a_m$. Substituting the expression of a'_0 from Equation (7.4) into the feasible range of accelerations leads to

$$-a_m - A \leq \frac{B}{t'_0}, \quad (7.7)$$

$$\frac{B}{t'_0} \leq a_m - A. \quad (7.8)$$

Then the feasible ranges of t'_0 can be computed accordingly.

2. Acceleration constraint for the second ramp: $-a_m \leq a'_1 \leq a_m$. Substituting the expression of a'_1 from Equation (7.5) into the feasible range of accelerations leads to

$$-a_m - A \leq \frac{B}{T - t'_0}, \quad (7.9)$$

$$-\frac{B}{T - t'_0} \leq a_m - A. \quad (7.10)$$

As before, the feasible ranges of t'_0 can be computed accordingly.

Each Inequality from (7.7), (7.8), (7.9), and (7.10) will lead to one feasible range of t'_0 . Since t'_0 needs to satisfy all of them, plus the initial feasible range $[0, T]$, we will finally obtain only one single feasible range of t'_0 , which is the common intersection of the above ranges. If the common intersection is empty, the re-interpolation of the given velocity profile is not possible. (Thus, the shortcut is simply discarded.)

Next, we find t'_0 that minimizes $J(t'_0)$ by local extrema, i.e., examining the roots of $dJ(t'_0)/dt'_0 = 0$. Note that by substituting the expressions of a'_0 and a'_1 from Equations (7.4) and (7.5) into Equation (7.6), we have

$$\frac{dJ(t'_0)}{dt'_0} = 2At_0'^4 + (2B - 4AT)t_0'^3 + (3AT^2 - 3BT)t_0'^2 + (AT^3 + 3T^2)t'_0 + (BT^3), \quad (7.11)$$

from which the roots can be solved analytically. The candidates of the minimizer are then the roots, provided that they fall in the feasible range of t'_0 , and the two ends of the feasible range. After obtaining the value t'_0 that minimizes J , we can construct a two-ramp velocity profile accordingly.

Figure 7.1: This figure illustrates the idea of modification of a two-ramp velocity profile that violates the velocity constraint. The excessive area above the velocity limit is colored in red and is denoted by d_{ex} . The area is divided into two parts and then pasted on both sides (shown in green) of the original velocity profile. (Figure is not to scale.)

Since the procedure so far only imposes the acceleration bound s_m into the calculation, we need to check the peak velocity v'_p of the newly re-interpolated velocity profile. If the new velocity profile does not violate the velocity bound, it is a solution to this re-interpolation problem. Otherwise, we proceed by performing the following modification.

Consider the profile before the modification. Let v_p be its peak velocity, t_0 the duration of the first ramp, and t_1 the duration of the second ramp (note also that now $t_0 + t_1 = T$). The idea of this modification is that we cut the excessive area above the velocity limit, denoted by d_{ex} , and paste that on both sides of the velocity profile. We do not divide the excessive area and paste it equally on both sides. Instead, we try to minimize, as is done previously, the sum of the new accelerations squared:

$$\text{minimize } a_0'^2 + a_1'^2, \quad (7.12)$$

where a_0' and a_1' are the acceleration of the first and the second ramps after this modification. This idea is illustrated in Figure 7.1.

Let $\Delta v_0 = v_m - v_0$ and $\Delta v_1 = v_m - v_1$. Let τ_0 , τ_1 , Δt_0 , and Δt_1 be defined as shown in Figure 7.1. Since the two green areas on both sides of the original velocity profile must sum to d_{ex} , we have

$$d_{ex} = \frac{1}{2} \Delta t_0 \Delta v_0 + \frac{1}{2} \Delta t_1 \Delta v_1. \quad (7.13)$$

From the relations $a_0' = \Delta v_0 / (\tau_0 - \Delta t_0)$ and $a_1' = \Delta v_1 / (\tau_1 - \Delta t_1)$, we can write Δt_0 and Δt_1 in terms of a_0' and a_1' . Substituting the expressions of Δt_0 and Δt_1 into

Equation (7.13) gives us

$$\frac{A_2}{a'_0} + \frac{B_2}{a'_1} = C_2, \quad (7.14)$$

where $A_2 = \Delta v_0^2$, $B_2 = \Delta v_1^2$, and $C_2 = \tau_0 \Delta v_0 + \tau_1 \Delta v_1 - 2d_{ex}$. For convenience, let $x = a'_0$ and $y = a'_1$. We can formulate the problem of finding x and y as

$$\begin{aligned} & \underset{x,y}{\text{minimize}} && x^2 + y^2 \\ & \text{subject to} && \frac{A_2}{x} + \frac{B_2}{y} = C_2. \end{aligned} \quad (7.15)$$

From the above problem, we can see that the objective function is a circle, centered at the origin of the $x - y$ plan, while the constraint function is a hyperbola, centered at $(x, y) = (A_2/C_2, B_2/C_2)$. Therefore, the minimizer is the point where both curves touch (while varying the radius of the circle).

Let $p = (x_0, y_0)$ be the point where the two curves touch. Then we must have that the slope of the hyperbola at the point p is perpendicular to the line connecting p with the origin. Solving for the point p from this condition gives us

$$a'_0 = \frac{A_2 + \sqrt[3]{A_2 B_2^2}}{C_2}. \quad (7.16)$$

And we can solve for a'_1 , as well as the duration of each ramp, accordingly. In case any of the computed values violates a constraint, this re-interpolation may be reported as failed.

7.2 Trajectory Planning with Minimum-Switch-Time Constraints

The set of re-interpolation methods with the minimum-switch-time constraint presented in the Section is a more detailed version of that presented in Lertkultanon and Pham 2016.

In the following discussion, let the minimum allowed switch time be $\delta > 0$. For a given set of boundary conditions and bounds, we first interpolate the time-optimal trajectory according to the procedure without considering the minimum-switch-time constraint. If all the *ramps* in the velocity profile are longer than δ , no further modification is required, i.e., the resulting profile is time-optimal subject to velocity, acceleration, *and* minimum-switch-time constraints automatically. Other-

wise, we enumerate all possibilities of minimum-switch-time constraint violation and propose a method to *re-interpolate* the time-optimal velocity profile, which is also subject to the new constraint, for each case.

7.2.1 Simultaneous Interpolation with Zero Terminal Velocities

Here we divide the problem into two cases based on the time-optimal velocity profile computed *without* the minimum-switch-time constraint.

P^+P^- velocity profiles

Suppose that the duration of each ramp³ is $t_s < \delta$. We then construct a new velocity profile $\dot{s}'(t)$ where each of its ramp has the duration $t'_s = \delta$ (so that the velocity profile is time-optimal). The new peak velocity $\dot{s}'_p$ becomes $\dot{s}'_p = 1/\delta$ while the new magnitude of the ramp acceleration is $|\ddot{s}| = 1/\delta^2$. It can be easily verified that the new velocity profile is also respecting the velocity and acceleration constraints.

Before proceeding to the next Section, let us consider the condition in which the original velocity profile is P^+P^- . Consider the relation $1 = \dot{s}_p t_0$, where $\dot{s}$ is the original peak velocity and t_0 is the original duration of the first ramp. We also have that $t_0 = \dot{s}_p / \ddot{s}_m$. Substituting this expression into the above equation leads to $\dot{s}_p^2 = \ddot{s}_m$. Therefore, any P^+P^- velocity profile will necessarily satisfy

$$\ddot{s}_m \leq \dot{s}_m^2. \quad (7.17)$$

$P^+L^+P^-$ velocity profiles

We divide this case into two subcases, depending on the original duration of both P-ramps.

$t_0 < \delta$: In this case, we proceed by stretching both P-ramps such that they attain a new duration of δ while the final (respectively initial) velocity of the first (respectively the last) ramp remains at the maximum. The new acceleration of the first ramp is then computed as $\ddot{s}' = \dot{s}_m / \delta$. By doing so, the middle ramp will be shortened. After this re-interpolation, if the middle ramp is made shorter than δ , we proceed to perform another re-interpolation by using the following procedure for the case $t_0 \geq \delta$.

³Note that the duration of both ramps are equal by construction.

Note also that in case the new middle ramp is shorter than δ , we can assume that the new accelerations of the first and the last ramp are at the (new) acceleration bound. To see why, consider the new acceleration of the first ramp, $\ddot{s}' = \dot{s}_m/\delta$. If the first ramp attain any higher acceleration than $\ddot{s}'$, the velocity profile will reach the maximum velocity $\dot{s}_m$ before the time instant $t = \delta$, thus violating the minimum-switch-time constraint.

$t_0 \geq \delta$: There are two exhaustive possibilities of re-interpolation: (A) the new velocity profile has two ramps; and (B) the new velocity profile has three ramps.

In the first case, the new velocity profile will have two ramps and the peak velocity will be at the maximum. The duration of the new first ramp can then be computed as $t'_0 = 1/\dot{s}_m$. This type of velocity profile is valid as long as it does not violate the minimum-switch-time constraint. Therefore, its validity condition is $\dot{s}_m\delta \leq 1$. Note that this condition will always be satisfied. However, we will select the two-ramp velocity profile as our solution only when it is faster than the fastest of the three-ramp velocity profile. Therefore, we choose to re-interpolate the original velocity profile into a two-ramp profile when $\dot{s}_m\delta \geq 2/3$.

In the latter case, the new velocity profile will be three-ramp. First, we explore the three-ramp velocity profile that is the fastest possible, i.e., all three ramps have duration δ . From direct calculation, the validity condition of this subcase is given by $\dot{s}_m\delta \geq 1/2$. Otherwise, we perform the following re-interpolation. We start by stretching the middle ramp such that it has duration δ while the accelerations of the first and the third ramps remain at the maximum magnitude. Note that we leave the acceleration of the new middle ramp undetermined first. From the condition that the velocity profiles before and after the re-interpolation must cover the same total displacement, $s = 1$, we have the relation between the new duration of the first and the last ramp, t'_0 and t'_2 , as

$$\left(t'_0 + \frac{\delta}{2}\right)^2 + \left(t'_2 + \frac{\delta}{2}\right)^2 = \frac{2}{\ddot{s}_m} + \frac{\delta^2}{2}, \quad (7.18)$$

which is an equation of a circle in the $t'_0 - t'_2$ plane. Applying the acceleration constraint on the acceleration of the middle ramp, we have that the new velocity profile must satisfy $t'_2 - t'_0 \leq \delta$. Also, from the minimum-switch-time constraint, we have that $t'_0 \geq \delta$ and $t'_2 \geq \delta$. By considering the above three inequality, we can determine an arc of the circle, on which the pair (t'_0, t'_2) gives a valid velocity profile. The pairs that gives a time-optimal velocity profile will be at either end of

Figure 7.2: Each region in the plot indicates how a re-interpolation method is chosen given one particular velocity profile. (i) The original velocity profile is P^+P^- . (ii) The original velocity profile is $P^+L^+P^-$ with $t_0 < \delta$. The remaining regions, (iii), (iv), and (v), the velocity profile has three ramps with $t_0 > \delta$.

the arc.

From the conditions for choosing the cases discussed above, we can plot them out into the following diagram, shown in Figure. 7.2. The curved line in the plot is determined from the equation $\ddot{s}_m = \dot{s}_m^2$. The region under the curve (indicated as (i) and colored in red in the plot) is where the original velocity profile is P^+P^- . The green region (ii) is when the original velocity profile is $P^+L^+P^-$ with $t_0 < \delta$. The remaining regions in the graph, i.e., the region between $\dot{s}_m = \ddot{s}_m$ and $\ddot{s}_m = \dot{s}_m^2$, is when the original velocity profile has three ramps with $t_0 > \delta$. If the condition for region (iii) is satisfied, the resulting profile from re-interpolation will be two-ramp. If the condition for region (iv) is satisfied, the resulting profile will be three-ramp and each ramp has duration δ . Otherwise, we follow the procedure described above to get a three-ramp velocity profile.

7.2.2 Independent Interpolation with Arbitrary Terminal Velocities

Single DOF

Consider a linear path segment connecting two configurations x_0 and x_1 . Let the velocity bound be v_m , the acceleration bound a_m , and $d = x_1 - x_0$ the total dis-

placement of the trajectory. All the possibilities of minimum-switch-time constraint violation are listed in Table 7.1. Here we denote the case when the original velocity profile—the one computed without considering the minimum-switch-time constraint—has two (respectively three) ramps as PP (respectively PLP).

	t_0	t_1	t_2
PP1	$< \delta$	$\geq \delta$	-
PP2	$\geq \delta$	$< \delta$	-
PP3	$< \delta$	$< \delta$	-
PLP1	$< \delta$	$\geq \delta$	$\geq \delta$
PLP2	$\geq \delta$	$\geq \delta$	$< \delta$
PLP3	$\geq \delta$	$< \delta$	$\geq \delta$
PLP4	$< \delta$	$< \delta$	$\geq \delta$
PLP5	$\geq \delta$	$< \delta$	$< \delta$
PLP6	$< \delta$	$\geq \delta$	$< \delta$
PLP7	$< \delta$	$< \delta$	$< \delta$

Table 7.1: All possibilities of minimum-switch-time-constraint violation given that the minimum allowed switch time is δ . t_0 and t_1 are the duration of the first and the second ramp of the profile, respectively. For a three-ramp velocity profile, t_2 denotes the duration of the third ramp.

PP1: First, we stretch the duration of the first ramp to δ while constrain the acceleration of the second ramp to be the same. By doing so, the new peak velocity v'_p can be computed as

$$v'_p = \frac{1}{2} \left(k_1 - \text{sgn}(k_1) \sqrt{k_1^2 + 4k_1 v_0 + 4v_1^2 - 8a_1 d} \right), \quad (7.19)$$

where $k_1 = a_1 \delta$ and a_1 is the acceleration of the last (second) ramp. The new acceleration of the first ramp, a'_0 , can be computed as $a'_0 = (v'_p - v_0)/\delta$. Figure 7.3 shows an example of velocity profiles before and after this re-interpolation procedure.

The derivation of the peak velocity in Equation (7.3) involves a division by a_1 . In case a_1 is non-zero, the above equation is always valid since the terms inside the square root in Equation (7.19) is always non-negative (see Appendix B.1). For the case when $a_1 = 0$, which happens when the final velocity v_1 is maximal, i.e., $|v_1| = v_m$, we can still proceed in a similar fashion. That is, we stretch the duration of the first ramp to δ while the last ramp remains at the same acceleration.

The duration of the new second ramp, t'_1 can be written as $t'_1 = (v_1 - v'_p)/a_1$. (The duration t_1 for the case when $a_1 = 0$ can also be easily derived.) In case we have that $t'_1 \geq \delta$, the resulting velocity profile is the solution to the re-interpolation

Figure 7.3: This figure shows the velocity profiles before (some color) and after (some other color) the re-interpolation procedure (PP1). The minimum allowed switch time is $\delta = 0.2$ s.

Figure 7.4: This figure depicts similarity between a corresponding pair of PP1 and PP2 profiles. Swapping the terminal velocities of one profile results in its corresponding profile. Due to this similarity, the re-interpolation procedure for PP1 profiles can be directly adapted to use with PP2 profiles.

problem. Otherwise, we need to perform re-interpolation again. We explore exhaustively the two possibilities of re-interpolation: (A) the new velocity profile has two ramps, both ramps have the duration of δ ; (B) the new velocity profile has one single ramp. In both cases, we can compute explicitly the remaining values (the accelerations, the new peak velocity, etc.) Finally, we choose the profile with the shorter duration as a solution. Note that at least (B) is guaranteed to be valid (see Appendix B.1).

PP2: This case is similar to PP1. A PP2 velocity profile can be seen as the mirror image of its corresponding PP1 profile, as illustrated in Figure 7.4. Therefore, the procedure presented in PP1 can be directly adapted for this case.

PP3: First, we explore two possibilities of re-interpolation: (A) the new velocity profile has two ramps, both ramps have the duration of δ ; (B) the new velocity

Figure 7.5: The original velocity profile is drawn in red. The minimum allowed switch time is $\delta = 0.2$ s.

profile has one single ramp. A solution is chosen as the one that is feasible and has the shorter duration.

However, unlike PP1 and PP2, for PP3, the one-ramp velocity profile is *not* guaranteed to be feasible. There might be case where the above two possibilities are both infeasible. To handle such cases, we shall *flip* the original velocity profile. For example, if the original velocity profile is P^+P^- , then the re-interpolated profile will be of type P^-P^+ . Both ramps of the new velocity profile still saturate the velocity limit. The idea of velocity profile flipping is shown in Figure 7.5.

The new peak velocity, v'_p , can be derived from the following equation:

$$v_p'^2 = \frac{1}{2}(v_0^2 + v_1^2) + a'_0(x_1 - x_0), \quad (7.20)$$

where a'_0 is the acceleration of the new first ramp, provided that the right-hand side of the equation is non-negative. Of the two duration, we select the one that results in a feasible and shorter velocity profile.

In case Equation (7.20) does not lead to any feasible solution or any feasible velocity profile (i.e., v'_p can be computed but the resulting velocity profile still violates the minimum-switch-time constraint), this means that the accelerations of both ramps cannot be maximal simultaneously. This in turn implies that the minimum-switch-time constraint is to be saturated in lieu of the acceleration bound. Then we construct the *flipped* velocity profile following the idea from PP1 and PP2. For example, suppose the *original* duration satisfy $t_0 > t_1$, the re-interpolated profile will then be such that $t'_0 < t'_1$. (Furthermore, since the new velocity profile is a flipped profile, we also have that $a_0 a'_0 < 0$.) To obtain the flipped velocity profile, we follow the procedure for the case PP1. The first ramp of the new velocity profile

will saturate the minimum-switch-time constraint, while the last ramp will saturate the acceleration constraint.

PLP1: There are two possibilities to explore in this case, depending on the re-interpolated velocity profile: (A) the re-interpolated profile has three ramps; and (B) the re-interpolated profile has two ramps. We simply check both options and choose the case that gives a feasible and shorter trajectory.

In PLP1A, we stretch the duration of the first ramp to δ while retaining its final velocity at the bound, i.e., $|v'_p| = v_m$. The acceleration of the new first ramp is then $a'_0 = (v'_p - v_0)/\delta$. Finally, the remain portion of the trajectory is re-interpolated as if it is a PP1 trajectory. In PLP1B, we merge the first two ramps of the original velocity profile together into a single ramp with the same boundary conditions. The third ramp of the original velocity profile remains unchanged. Note that (i) at least the case PLP1B will always be feasible since the duration of the merged ramp will always be greater than δ ; and (ii) the case where the re-interpolated profile has one ramps is not explored since it cannot be time-optimal.

PLP2: Similar to the relation between the cases PP1 and PP2. The case PLP2 can be seen as the mirror image of its corresponding PLP1 profile. Therefore, the procedure presented in PLP1 can be directly adapted for this case.

PLP3: There are also two possibilities of re-interpolation similar to the previous two cases: (A) the re-interpolated profile has three ramps; and (B) the re-interpolated profile has two ramps. As usual, we choose a valid profile with shorter duration.

In PLP3A, we stretch the duration of the middle ramp to δ . The other two ramps are made shorter while staying at the same acceleration. In order to make the resulting velocity profile time-optimal within the class, the middle ramp might not stay at zero acceleration. To compute the new velocity profile, we compute the duration of the new first and last ramps, t'_0 and t'_2 , as follows.

Assume, without loss of generality, that the acceleration of the first ramp of the velocity profile is positive. Consider the relation $d = d_0 + d_1 + d_2$, where $d = x_1 - x_0$, $d_0 = v_0 t'_0 + a_m t'^2_0/2$ is the displacement covered by the first ramp, $d_1 = (v_0 + a_m t'_0 + v_1 + a_m t'_1)\delta/2$ is the displacement covered by the second ramp, and $d_2 = v_1 t'_2 + a_m t'^2_2/2$ is the displacement covered by the last ramp. By manipulating the expression above, we can derive the relationship between the two unknowns, t'_0

and t'_2 , as

$$\left(t'_0 + \left(\frac{\delta}{2} + \frac{v_0}{a_m}\right)\right)^2 + \left(t'_2 + \left(\frac{\delta}{2} + \frac{v_1}{a_m}\right)\right)^2 = \frac{1}{a_m^2} \left(v_0^2 + v_1^2 + 2a_m d + \frac{1}{2}a_m^2 \delta^2\right), \quad (7.21)$$

which, interestingly, is a circle in the t'_0 - t'_2 plane, centered at $(t'_0, t'_2) = (-\frac{\delta}{2} - \frac{v_0}{a_m}, -\frac{\delta}{2} - \frac{v_1}{a_m})$. To find a pair (t'_0, t'_2) that minimizes the total duration of the re-interpolated profile, we proceed as follows.

From the minimum-switch-time constraint, we have

$$\delta \leq t'_0 \quad \text{and} \quad \delta \leq t'_2. \quad (7.22)$$

From the velocity constraint, we have

$$0 \leq t'_0 \leq \frac{v_m - v_0}{a_m} \quad \text{and} \quad 0 \leq t'_2 \leq \frac{v_m - v_1}{a_m}. \quad (7.23)$$

Combining both constraints, the feasible intervals of t'_0 and t'_2 subject to the minimum-switch-time constraint and the velocity bound are

$$\delta \leq t'_0 \leq \frac{v_m - v_0}{a_m} \quad \text{and} \quad \delta \leq t'_2 \leq \frac{v_m - v_1}{a_m}. \quad (7.24)$$

Note that since the first ramp and the last ramp of the original velocity profile are longer than δ , both of the above intervals are non-empty. Applying the acceleration constraint on the new middle ramp, we have

$$\frac{v_0 - v_1}{a_m} - \delta \leq t'_2 - t'_0 \leq \frac{v_0 - v_1}{a_m} + \delta. \quad (7.25)$$

Substituting the feasible range of t'_0 into the above inequalities leads us to

$$\max\left(\delta, \frac{v_0 - v_1}{a_m}\right) \leq t'_2 \leq \frac{v_m - v_1}{a_m}. \quad (7.26)$$

The above feasible intervals of t'_0 and t'_2 lead to an arc of the circle whose points satisfy all the constraints. Note that the arc is also in the first quadrant of the t'_0 - t'_2 plane. Therefore, the point (t'_0, t'_2) that minimizes the duration of the re-interpolated velocity profile will essentially be at either end of the arc. We simply test which point gives the shorter profile and select it as the solution.

In case there is no intersection between the feasible region of t'_0 and t'_2 and the

circle, it means the first and the last ramps cannot saturate the acceleration limit at the same time. From direct calculation, we found that the time-optimal velocity profile in this case (for the class of three-ramp profiles) will be such that all ramps have equal duration of δ . From this knowledge, we can derive the relation between two peak velocities v'_{p0} and v'_{p1} (the velocities at the end of the first and second ramps, respectively) as $v'_{p0} + v'_{p1} = d/\delta - (v_0 + v_1)/2$. Then we can select the values for v'_{p0} and v'_{p1} that satisfy the velocity and acceleration bounds.

Finally, in PLP3B, we proceed by merging two consecutive ramps into one. We choose to merge either the first two ramps or the last two ramps depending on the duration of the resulting velocity profile.

PLP4: This case can also be divided into two subcases: (A) the re-interpolated profile has three ramps; and (B) the re-interpolated profile has two ramps.

For PLP4A, we proceed in the same way as in PLP1A. For PLP4B, we proceed by merging the first two ramps together into one. However, unlike the case PLP1B, PLP4B is not guaranteed to give a valid velocity profile. In case the merged ramp has the duration shorter than δ , we re-interpolate the velocity profile again using the routine for PP1. That is, the resulting profile will be such that its first ramp has the duration of δ , the peak velocity is computed by Equation (7.19), and the last ramp saturates the acceleration bound. From both PLP4A and PLP4B, we finally choose the profile that gives a shorter duration as a solution.

PLP5: This case is similar to PLP4 and therefore, the re-interpolation procedure can be directly adapted from that of PLP4.

PLP6: There are four exhaustive possibilities of modifying the original velocity profile as follows. (A) The duration of the first and the last ramps are stretched to δ . The new middle ramp stays at the velocity bound, hence shortened. (B) Only the duration of the first ramp is stretched to δ . There are two subcases depending on the final velocity of the first ramp, v'_{p0} . In the first subcase, v'_{p0} is set to be at the velocity bound. The remaining portion of the trajectory is re-interpolated as a two-ramp velocity profile by using the routines for PP. In the second subcase, v'_{p0} is calculated from Equation (7.19) and then the remaining portion is re-interpolated accordingly. (C) Only the duration of the last ramp is stretched to δ . There are two subcases similar to (B). (D) The re-interpolated profile has only one ramp.

After exploring all the above possibilities, we choose a valid profile with a shorter duration as a solution. Note that here at least the case (D) will always be valid.

PLP7: First, we explore the following three cases: (A) the re-interpolated velocity profile has three ramps. Since all the ramps in the original velocity profile are shorter than δ , all the ramps in the new velocity profile must have duration δ ; (B) the re-interpolated velocity profile has two ramps; and (C) the re-interpolated velocity profile has one ramp. To re-interpolate a velocity profile following the case PLP7B and PLP7C, we can use the routine for the case PP3. After exploring all the cases above, we choose a valid profile with a shorter duration as a solution.

However, in some special cases, all the three possibilities do not lead to any feasible velocity profile. This happens when the velocity bound is very low and one of the terminal velocity is actually zero. In this case, we can proceed as follows. Let v_p be the peak velocity of the original trajectory. If $|v_p - v_0| > |v_p - v_1|$, then we stretch the duration of the first ramp to δ while constraining its final velocity to be v_p . The remaining portion of the trajectory is re-interpolated to be one ramp. On the other hand, if $|v_p - v_0| < |v_p - v_1|$, we stretch the duration of the last ramp to δ while constraining its initial velocity to be v_p . The remaining portion of the trajectory is re-interpolated to be one ramp, similar to the previous case.

Multiple DOFs

After time-optimally interpolating velocity profiles for different joints independently, the velocity profile of each joint may end up having different duration. However, in order to be a valid shortcut, velocity profiles of all the joints in this shortcut path *must* have the same duration. Let the m^{th} joint, $m \in \{1, 2, \dots, n\}$, have the slowest time-optimal single-DOF velocity profile with duration T . The shortcut therefore cannot have its duration to be less than T . We make this shortcut valid, we need to re-interpolate all $n - 1$ remaining velocity profiles such that they all have duration T .

Given a fixed time duration T and a trajectory (whose duration is less than T), a velocity profile re-interpolation problem has either infinitely many solutions or no solution (Kunz and Stilman 2014). Even in the absence of the minimum-switch-time constraint, deciding if the problem has a solution is difficult (Kröger, Tomiczek, and Wahl 2006). One possible way to cope with the complexity of the re-interpolation problem is to specify beforehand a subset of free parameters such that the remaining

unknowns can be eventually solved analytically. For example, in Hauser and Ng-Thow-Hing (2010) the authors suggested to constrain the acceleration of the first and the last ramps of the re-interpolated velocity profile to have the same magnitude. However, this restrictive constrain led to low re-interpolation success rate, as can be seen from the results in Section 7.3.3.

We propose a re-interpolation that explores both classes of velocity profiles, i.e., two-ramp and three-ramp velocity profiles. The main different to the existing method is that instead of constraining the accelerations, our proposed routine will make some initial guesses on the duration of each ramp of the re-interpolated profile. Furthermore, our routine is able to take into account the minimum-switch-time constraint *between* different DOFs; this is done by requiring that for each possible tuple (i, j, k, l) , if $|t^{sw}(i, j) - t^{sw}(k, l)| > 0$, then

$$\min_{i,j,k,l} |t^{sw}(i, j) - t^{sw}(k, l)| \geq \delta, \quad (7.27)$$

where $t^{sw}(i, k)$ denotes the j^{th} switch point of the i^{th} DOF. The constraint above ensures that *any* pair of switch points will be either at the same time instant or separated by a duration of at least δ .

Suppose single-DOF velocity profiles are $\gamma_i, i = \{1, 2, \dots, n\}$ and γ_m is the slowest profile with duration T . We need to re-interpolate all other velocity profiles, $\gamma_i, i = \{1, 2, \dots, n\}, i \neq m$, such that they all have new duration equal to T . To start with, we divide each ramp of γ_m into equally long time intervals where each interval is the smallest possible but still longer than δ . This gives us $\lfloor t/\delta \rfloor$ segments for a ramp with duration t . In our implementation, the total number of such segments is capped to 50 to avoid long computation time when δ is small or null (i.e. no minimum-switch-time constraint).

Suppose the first ramp of γ_m has duration t . We divide it into $\lfloor t/\delta \rfloor$ segments, where each segment is of duration $\Delta t = t/\lfloor t/\delta \rfloor$, to obtain the set $\{\Delta t, 2\Delta t, \dots, t\}$. We then draw a set of vertical lines (on the $v - t$ plane), each line passing through each point in $\{\Delta t, 2\Delta t, \dots, t\}$. This procedure is repeated for every ramp of γ_m . We call the set of all vertical lines a *grid*. An example of a constructed grid is shown in Figure 7.6. After having constructed the grid, we explore the two classes of velocity profiles as follows.

Two-ramp velocity profiles: In this case, we try to locate the unique switch point (of the to-be re-interpolated profile) at each of grid lines. One the switch point location

Figure 7.6: An example of a grid constructed for a velocity profile (shown in green). The minimum allowed switch time is $\delta = 0.4$ s. Since the first ramp has duration 1.2 s., it is divided into three segments of duration 0.4 s. The second ramp has duration 1.0 s. It is therefore divided into two segments of duration 0.5 s. each.

is determined, the acceleration of each ramp can be analytically determined (see Appendix B.2.1).

Three-ramp velocity profiles: In this case, there are $\binom{N}{2}$ possible ways to choose two locations of switch points out of N grid lines. To reduce computational cost, instead of trying all possibilities, we make further guess on the locations of the switch points. In particular, let t'_0 , t'_1 , and t'_2 be duration of the first, the middle, and the last ramps of the new velocity profile after this re-interpolation. Further guesses are made on the ratios t'_0/T , t'_1/T , and t'_2/T . Choices of these three ratios are *arbitrary*. However, with some trial-and-error testings, we suggest the values $t'_0/T = 1/4$, $t'_1/T = 1/2$, and $t'_2/T = 1/4$, which give fairly high success rate of re-interpolation. Then we examine 4 ways of snapping those switch points to their nearest grid lines. Given the duration of each ramp, we can formulate the problem of finding accelerations, a'_0 , a'_1 , and a'_2 , for all ramps as a feasibility problem, which can be solved analytically (see Appendix B.2.2).

If neither the two- nor the three-ramp attempt succeeds, i.e., solutions of the accelerations do not respect the acceleration bounds or there is no solution available, then the re-interpolation is considered as failed and the corresponding shortcut path is discarded.

7.3 Simulations

We implemented the proposed algorithm in Python. The implementation is openly available at <https://github.com/Puttichai/parabint>.

7.3.1 Comparison with an Optimization-Based Approach

The problem of finding the time-optimal parabolic trajectory subject to velocity, acceleration, and minimum-switch-time constraints can also be formulated as an optimization problem. The general procedure is to explore both possibilities of the resulting velocity profile: two-ramp and three-ramp. For two-ramp profiles, the optimization variables can be written as a vector as $\mathbf{y} = [t'_0 \ t'_1 \ v'_p]^\top$, where v_p is the peak velocity of the profile, i.e., the final velocity of the first ramp. For three-ramp profile, the vector of optimization variables is $\mathbf{z} = [t'_0 \ t'_1 \ t'_2 \ v'_{p0} \ v'_{p1}]^\top$, where v'_{p0} and v'_{p1} are the final velocities of the first and the second ramps, respectively.

In case of two-ramp velocity profiles, the problem can be formulated as

$$\begin{aligned} & \underset{\mathbf{y}}{\text{minimize}} && \mathbf{c}_0^\top \mathbf{y} \\ & \text{subject to} && \mathbf{G}_0 \mathbf{y} \preceq \mathbf{h}_0 \\ & && \frac{1}{2} \mathbf{y}^\top \mathbf{A}_0 \mathbf{y} + \mathbf{b}_0^\top \mathbf{y} - 2d = 0, \end{aligned} \quad (7.28)$$

$$\begin{aligned} \text{where } \mathbf{G}_0 &= \begin{bmatrix} -a_m & -a_m & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -a_m & -a_m & 0 & -1 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 1 & -1 \end{bmatrix}^\top, \mathbf{A}_0 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \\ \mathbf{b}_0 &= \begin{bmatrix} v_0 \\ v_1 \\ 0 \end{bmatrix}, \mathbf{c}_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \text{ and } \mathbf{h}_0 = [v_0 \ -v_0 \ v_1 \ -v_1 \ -\delta \ -\delta \ v_m \ v_m]^\top. \end{aligned}$$

Similarly for the three-ramp velocity profiles, we have

$$\begin{aligned} & \underset{\mathbf{z}}{\text{minimize}} && \mathbf{c}_1^\top \mathbf{z} \\ & \text{subject to} && \mathbf{G}_1 \mathbf{z} \preceq \mathbf{h}_1 \\ & && \frac{1}{2} \mathbf{z}^\top \mathbf{A}_1 \mathbf{z} + \mathbf{b}_1^\top \mathbf{z} - 2d = 0, \end{aligned} \quad (7.29)$$

where

$$\begin{aligned} \mathbf{G}_1 &= \begin{bmatrix} -a_m & -a_m & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -a_m & -a_m & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -a_m & -a_m & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}^\top, \\ \mathbf{h}_1 &= [v_0 \ -v_0 \ 0 \ 0 \ v_1 \ -v_1 \ -\delta \ -\delta \ -\delta \ v_m \ v_m \ v_m \ v_m]^\top, \end{aligned}$$

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, b_1 = \begin{bmatrix} v_0 \\ 0 \\ v_1 \\ 0 \\ 0 \end{bmatrix}, \text{ and } c_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

The inequality constraints are due to velocity and acceleration bounds, and the minimum-switch-time constraint. The equality constraints represent the fact that the trajectory before and after the re-interpolation should cover the same amount of displacement. It should be noted that both Problem (7.28) and (7.29) are non-convex since the constraint matrices A_0 and A_1 are not positive semidefinite, i.e., some of their eigen values are negative. Because of this non-convexity, the problems are difficult and computationally expansive to solve. In addition, a solver might also have a chance to be stuck in a local minimum.

For comparison, we generated 10,000 random problem instances for each case of minimum-switch-time constraint violation, as listed in Table 7.1. Then we solved the problems by using PyIpopt⁴ to solve the optimization problems and by using our proposed routines (Section 7.2). For the PyIpopt solver, we set the tolerance to 10^{-8} and the maximum number of iterations to 2,000. Average computation time, the number of times the optimizer was stuck in a local minimum, and the number of times the optimizer failed were recorded and are reported in Table 7.2. As it can be seen from the table, our proposed method took much less time to compute the same solutions compared to solving via optimization. This emphasizes advantages of solving the problem analytically via the proposed method.

	PP		PLP	
	Optimization	Exact	Optimization	Exact
Computation time (ms.)	6.51	0.0587	14.3	0.257
% stuck	8.5%	—	36.8%	—
% failed	6.8%	—	0.95%	—

Table 7.2: A comparison between an optimization-based and the proposed methods of velocity profile re-interpolation.

Remark. An optimization-based method for interpolating cubic spline trajectories such as that presented in Gasparetto and Zanotto (2008) required solving for a large number of optimization variables. The number of variables also grows with the

⁴PyIpopt is a freely available Python optimization module, which can be found at <https://github.com/xuy/pyipopt>.

number of via-points added between the initial and final robot configurations. However, one can see that even finding time-optimal parabolic interpolation subject to velocity, acceleration, and minimum-switch-time constraints via an optimization-based method, which requires solving for only a few unknown variables, took significantly more time than using our proposed method. This suggests that our method provides a reasonable trade-off between jerk-boundedness and computational efficiency.

7.3.2 Trajectory Planning and Shortcutting with Minimum-Switch-Time Constraint

Here we include the minimum-switch-time constraint into the motion planning pipeline described in Section 7.1. In the trajectory planning stage we use the routines described in Section 7.1.2 to time-parameterize the path taking into account the minimum-switch-time constraint. The parameterization is done through the path parameter s and therefore does not affect the path geometry.

In the shortcutting stage the condition (7.27) also needs to be satisfied. In each shortcutting iteration, we begin by randomly selecting two time instants t_a and t_b . Let t_a^{sw} be the nearest switch point preceding t_a and t_b^{sw} the nearest consecutive switch point of t_b . If $t_a - t_a^{sw} < \delta$, then we re-assign t_a to be t_a^{sw} . Similarly, if $t_b^{sw} - t_b < \delta$, then we re-assign t_b to be t_b^{sw} . Next we continue in a usual manner by first interpolating a velocity profile for each DOF independently and determining the duration T of the slowest DOF. In case $T > t_b - t_a$, the shortcut is discarded right away. Otherwise, we use the routines described in Section 7.1.2 to re-interpolate all the remaining velocity profiles to have duration T . And if the resulting trajectory is collision-free, then we replace the original trajectory segment with the shortcut. Figure 7.7 shows velocity profiles of the original trajectory before shortcutting, shortcut trajectory with the minimum-switch-time constraint, and shortcut trajectory without the constraint. The original trajectory has duration 5.495 s., which is relatively long since it stops at every waypoint. The shortcut trajectory with the minimum-switch-time constraint (in this case $\delta = 0.08$ s.) has duration 1.981 s. while the shortcut trajectory (from the same original trajectory) without the minimum-switch-time constraint has duration 2.280 s. The shortcut trajectory with the minimum-switch-time constraint is shorter and its distribution of switch points is much more sparse. (Red vertical lines in each subfigure indicate time instants where at least one joint changes its acceleration.) In terms of the number of

switch points, when not considering the minimum-switch-time constraint the resulting trajectory has 40 switch points while our method produced only 14 switch points.

Figure 7.7: Velocity profiles of the original trajectory before shortcutting and of trajectories after shortcutting the original trajectory with and without considering the minimum-switch-time constraint. The minimum-switch-time constraint is set to be $\delta = 0.08$ s. Red vertical lines indicate time instants when at least one DOF switches its acceleration (switch points).

7.3.3 Comparison with an Existing Method

In all the simulations, we used the joint velocity and acceleration bounds of Denso VS-060, a 6-DOF industrial manipulator.

In the first case, we made comparisons, in terms of success rate of multi-DOF re-interpolation, between the re-interpolation routine proposed in Hauser and Ng-Thow-Hing (2010) and our method (Section 7.2) with three different minimum-switch-time constraints: $\delta = 0$ s. (no minimum-switch-time constraint), $\delta = 0.008$ s., and $\delta = 0.1$ s. Comparisons were made in two subcases, depending on how boundary conditions (terminal configurations and velocities) were generated.

In the first subcase, we randomly generated 1,000 pairs of boundary conditions, $\{(x_0, \dot{x}_0), (x_1, \dot{x}_1)\}$. In the second subcase, 1,000 pairs of boundary conditions were randomly picked from existing trajectories, as in a shortcutting step. In the latter case, we started with two random pairs of start and goal configurations. For each pair $(x_{\text{start}}, x_{\text{goal}})$, we planned 50 trajectories connecting them using the aforementioned planning procedure (without shortcutting). Then for each of 100 trajectories, we randomly picked 10 pairs of time instants. Configurations and velocities at those time instants served as boundary conditions. The statistics are reported in Table 7.3. One can see that the re-interpolation method proposed in Hauser and Ng-Thow-Hing (2010) had relatively low success rate, due to its restrictive constraints on ramp accelerations.

For the second case, a piecewise linear path was generated for each of two pairs of the start and goal configurations. Then for each of path segments, we simultaneously interpolated velocity profiles with zero terminal velocities, both with and without the minimum-switch-time constraint. Finally we gave 1 s. to both proposed method and the one proposed in Hauser and Ng-Thow-Hing (2010) to shortcut each trajectory. This process was repeated 1,000 times. Initial trajectory duration, average final duration, and numbers of shortcutting iterations executed from each method are reported in Table 7.4.

Despite the fact that the proposed method is more computationally expensive, as can be seen from Table 7.3, when the same amount of time are given to both methods, the proposed method still performs better in terms of resulting trajectory duration after shortcutting.

Remark. From the results presented in Table 7.3, the proposed method is fast enough to be used in a real-time controller to interpolate a trajectory between two given waypoints. Suppose a robot has 6 DOFs. The longest average computation time for independent interpolation with arbitrary terminal velocities is less than 3 ms. Therefore, the proposed interpolation method can be used in a real-time controller with frequency of at least 300 Hz.

Remark. Applicability of parabolic trajectories to real robots has also been verified through our recent experiment. We employed our proposed method to plan and shortcut parabolic trajectories for a pick-and-place manipulation task (Lertkultanon and Pham 2015). The video of our robot executing the motion can be found at <https://youtu.be/tLouwj0wITQ>. One can see that the robot motion is sufficiently smooth to complete the task even without the jerk-boundedness constraint.

	$(\mathbf{x}_0, \dot{\mathbf{x}}_0)$ and $(\mathbf{x}_1, \dot{\mathbf{x}}_1)$ generated randomly		$(\mathbf{x}_0, \dot{\mathbf{x}}_0)$ and $(\mathbf{x}_1, \dot{\mathbf{x}}_1)$ randomly picked from trajectories			
	Success rate	Avg. comp. time (ms.)	Trajectories connecting $\mathbf{x}_{\text{start},0}$ and $\mathbf{x}_{\text{goal},0}$		Trajectories connecting $\mathbf{x}_{\text{start},1}$ and $\mathbf{x}_{\text{goal},1}$	
			Success rate	Avg. comp. time (ms.)	Success rate	Avg. comp. time (ms.)
From Hauser and Ng-Thow-Hing (2010)	36.5%	0.364	61.0%	0.324	51.8%	0.381
$\delta = 0.0$	78.1%	1.394	90.8%	1.512	85.8%	1.606
$\delta = 0.008$	77.0%	1.431	89.8%	1.581	85.2%	1.666
$\delta = 0.1$	75.7%	1.420	75.4%	2.806	74.0%	2.859

Table 7.3: Comparisons of independent interpolation methods with arbitrary terminal velocities (6 DOFs): success rate and computation time.

	A trajectory connecting $\mathbf{x}_{\text{start},0}$ and $\mathbf{x}_{\text{goal},0}$			A trajectory connecting $\mathbf{x}_{\text{start},1}$ and $\mathbf{x}_{\text{goal},1}$		
	Initial duration (s.)	Avg. final duration (s.)	Avg. # iterations	Initial duration (s.)	Avg. final duration (s.)	Avg. # iterations
From Hauser and Ng-Thow-Hing (2010)	2.831	1.802	413.1	7.000	3.002	237.3
$\delta = 0.0$	2.831	1.360	171.71	7.000	2.275	131.6
$\delta = 0.008$	2.831	1.372	178.4	7.000	2.272	127.8
$\delta = 0.1$	2.903	1.710	228.7	7.088	2.235	153.1

Table 7.4: Both the proposed method and the one presented in Hauser and Ng-Thow-Hing (2010) were given 1 s. to shortcut given trajectories. The experiment was repeated 1,000 times for each trajectory.

Conclusion

We have studied the minimum-switch-time constraint, which is a constraint on time intervals between consecutive switch points. Integrating this constraint into trajectory interpolation routines prevents high-frequency acceleration switching, which might hamper robot performance as well as cause motor wear. We have presented an algorithm for planning piecewise linear parabolic trajectories subject to velocity, acceleration, and this minimum-switch-time constraints. Simulation results show that our method is more computationally efficient than an optimization-based method. Furthermore, our method also achieves a higher re-interpolation success rate and produces trajectories with higher quality as compared to an existing method.

Chapter 8

Improvement to Motion Planning: Extension to Admissible Velocity Propagation Algorithm

One way to solve a kinodynamic motion planning problem is to decouple it into two smaller subproblems: path planning and time-parameterization, and address them sequentially. The approach is also called *path-velocity decomposition*. A fundamental requirement of the approach is that the path obtained in the first—path planning—stage must be *time-parameterizable*. To fulfill the requirement, Pham et al. proposed in Pham, Caron, and Nakamura (2013) an algorithm called Admissible Velocity Propagation (AVP), which when integrated into a path planner such as RRT, allows the planner to discover dynamically feasible motions while staying in the configuration space. The mechanism of the proposed AVP algorithm, however, makes the algorithm integrable only with unidirectional path planners. In this Chapter, we present a new bidirectional planner called **AVP-BiRRT**, which is an integration of the propagation method into a bidirectional RRT planner. Such integration is made possible by a newly proposed extension to AVP algorithm called **AVP-Backward**. Since the dynamic equation of a robot is generally not time-reversible, AVP-Backward is not a trivial extension to AVP. This work has been published in Pham, Quang-Cuong, Stéphane Caron, **Puttichai Lertkultanon**, and Yoshihiko Nakamura (2016). “Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots”. In: *The International Journal of Robotics Research* 36.1, pp. 44–67.

We first review the Time-Optimal Path Parameterization algorithm (TOPP), on top of which AVP was built. Then after giving an overview of AVP algorithm, we present the proposed extension, the planner, as well as our experiments.

8.1 Time-Optimal Path Parameterization (TOPP)

Let $\mathbf{q} \in \mathcal{C}$ be an n -dimensional vector representing a robot configuration. Consider a configuration space path P , which can be viewed as an underlying path of some trajectory $\mathbf{q}(s)$, $s \in [0, s_{\text{end}}]$. Assume that $\mathbf{q}(s)$ is C^1 -and piecewise C^2 -continuous. A time-optimal path parameterization problem seeks to find an increasing scalar function $s : [0, T] \rightarrow [0, s_{\text{end}}]$, with minimum T , which governs how the robot traverses P in time while respecting all the given kinodynamic constraints. Here we follow the numerical integration approach to TOPP, originally introduced in Bobrow, Dubowsky, and Gibson (1985) and Shin and McKay (1986b).

Suppose that the robot is to traverse P subject to second-order inequality constraints of the form

$$\mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{f}(\mathbf{q}) \preceq \mathbf{0}_M, \quad (8.1)$$

where $\mathbf{A}(\mathbf{q})$, $\mathbf{B}(\mathbf{q})$, $\mathbf{f}(\mathbf{q})$ are respectively an $M \times n$ matrix, an $n \times M \times n$ tensor, and an M -dimensional vector; and $\mathbf{0}_M$ is an M -dimensional zero vector. Inequality (8.1) is general and represents a wide spectrum of second-order systems and constraints (Pham 2014). Differentiating $\mathbf{q}(s(t))$ with respect to time yields

$$\dot{\mathbf{q}} = \mathbf{q}_s \dot{s} \quad \text{and} \quad \ddot{\mathbf{q}} = \mathbf{q}_{ss} \dot{s}^2 + \mathbf{q}_s \ddot{s}, \quad (8.2)$$

where dots denote differentiation with respect to time t , $\mathbf{q}_s = d\mathbf{q}/ds$, and $\mathbf{q}_{ss} = d^2\mathbf{q}/ds^2$. To validate the constraint (8.1) along the time-parameterized trajectory, substitute the expressions from (8.2) into the constraint to get the constraint in the *canonical form*:

$$\mathbf{a}(s)\ddot{s} + \mathbf{b}(s)\dot{s}^2 + \mathbf{c}(s) \preceq \mathbf{0}_M, \quad (8.3)$$

where

$$\begin{aligned} \mathbf{a} &:= \mathbf{A}(\mathbf{q}(s))\mathbf{q}_s(s), \\ \mathbf{b} &:= \mathbf{A}(\mathbf{q}(s))\mathbf{q}_{ss}(s) + \mathbf{q}_s^\top \mathbf{B}(\mathbf{q}(s))\mathbf{q}_s, \\ \mathbf{c} &:= \mathbf{f}(\mathbf{q}(s)). \end{aligned} \quad (8.4)$$

Inequality (8.1) is now turned into a set of *scalar* inequalities, one from each row i of (8.3):

$$a_i(s)\ddot{s} + b_i(s)\dot{s}^2 + c_i(s) \leq 0. \quad (8.5)$$

Since the system is subject to constraints of at most second order, the path parameterization problem is now reduced to a *two-dimensional* problem of finding a curve in $s - \dot{s}$ plane subject to bounds on $\ddot{s}$ (and $\dot{s}$, if pure velocity bounds are taken into account (Žlajpah 1996)) to be determined by Inequality (8.5). From Pontryagin Maximum Principle, one then has that the time-optimal path parameterization has *bang-bang* structure, i.e., it follows either minimum or maximum accelerations.

From Inequality (8.5), bounds on $\ddot{s}$ can be derived as follows:

- If $a_i(s) > 0$, then one has $\ddot{s} \leq (b_i(s)\dot{s}^2 - c_i(s))/a_i(s)$. This inequality defines an acceleration *upper bound* $\beta_i(s, \dot{s}) = (b_i(s)\dot{s}^2 - c_i(s))/a_i(s)$.
- If $a_i(s) < 0$, then one has $\ddot{s} \geq (b_i(s)\dot{s}^2 - c_i(s))/a_i(s)$. This inequality defines an acceleration *lower bound* $\alpha_i(s, \dot{s}) = (b_i(s)\dot{s}^2 - c_i(s))/a_i(s)$.
- If $a_i(s) = 0$, then s is a zero-inertia point. Optimal accelerations can also be derived. The reader is referred to Pham (2013) for a detailed treatment of this case.

Then for each point $(s, \dot{s})$ in the plane, one can define *the* minimum and maximum accelerations, respectively, as

$$\alpha(s, \dot{s}) := \max_i \alpha_i(s, \dot{s}) \quad \text{and} \quad \beta(s, \dot{s}) := \min_i \beta_i(s, \dot{s}).$$

Therefore, one can conclude that $\mathbf{q}(s(t))$, $t \in [0, T]$ satisfies all the constraints (8.1) *if and only if*

$$\forall t \in [0, T] \quad \alpha(s, \dot{s}) \leq \ddot{s} \leq \beta(s, \dot{s}). \quad (8.6)$$

Notice that if $\alpha(s, \dot{s}) > \beta(s, \dot{s})$, there exists no acceleration $\ddot{s}$ satisfying the above constraint. Such regions in $s - \dot{s}$ plane where $\alpha(s, \dot{s}) > \beta(s, \dot{s})$ are called *inadmissible region* (Shin and McKay 1985). Therefore, any valid velocity profile – a curve in $s - \dot{s}$ plane – has to stay outside of any inadmissible region. In most cases, this is equivalent to staying *below* the maximum velocity curve (MVC) defined by

$$\text{MVC}(s) = \begin{cases} \min\{\dot{s} \mid \alpha(s, \dot{s}) = \beta(s, \dot{s})\} & \text{if } \alpha(s, 0) \leq \beta(s, 0) \\ 0 & \text{otherwise.} \end{cases} \quad (8.7)$$

Remark. In general scenarios, the inadmissible region has one connected component and is the epigraph of the MVC. In those cases, the admissible velocity at a particular s ranges from 0 to $\text{MVC}(s)$. However, in some cases, at some particular s ,

there might exist more than one interval of admissible velocities (Shin and McKay 1985) or the range is $[\dot{s}', \text{MVC}(s)]$ for some $\dot{s}' > 0$. The latter case happens when *no quasi-static velocity profile is feasible*. In any case, the numerical integration procedure, to be briefly given, does not change.

Note that $(s, \dot{s}) \mapsto (\dot{s}, \alpha(s, \dot{s}))$ and $(s, \dot{s}) \mapsto (\dot{s}, \beta(s, \dot{s}))$ can be viewed as two vector fields, α and β respectively, in the $s - \dot{s}$ plane. Starting from any point in the admissible region, one can then *integrate* following the field α to obtain the *minimum-acceleration* velocity profile (or α -profile); or following the field β to obtain the *maximum-acceleration* profile (or β -profile).

Since the time-optimal velocity profile has bang-bang structure, the profile follows the fields β and α alternatively while always stays in the admissible region – below the MVC. The algorithm for computing the time-optimal parameterization of a given path P is then given as follows.

- Step 1** Find all possible $\alpha \rightarrow \beta$ switch points. They are all on the MVC. The detailed description and procedure can be found in Pham (2014).
- Step 2** Compute Limiting Curves (LC) as follows (Slotine and Yang 1989). From each switch point, integrate backward (in the direction of decreasing s) following α and integrate forward (in the direction of increasing s) following β .
- Step 3** Construct the Continuous Limiting Curve (CLC) where $\text{CLC}(s)$ is the lowest LC at s .
- Step 4** Integrate forward from $(0, \dot{s}_{\text{beg}})$ following β and backward from $(s_{\text{end}}, \dot{s}_{\text{end}})$ following α until they intersect each other or the CLC. Note that the terminal path velocities $\dot{s}_{\text{beg}}$ and $\dot{s}_{\text{end}}$ can be computed from the terminal velocities v_{beg} and v_{end} as $\dot{s}_{\text{beg}} = v_{\text{beg}} / \|\mathbf{q}_s(0)\|$ and $\dot{s}_{\text{end}} = v_{\text{end}} / \|\mathbf{q}_s(s_{\text{end}})\|$.
- Step 5** Construct the solution by concatenating together the forward profile from $(0, \dot{s}_{\text{beg}})$, the CLC (if applicable), and the backward profile from $(s_{\text{end}}, \dot{s}_{\text{end}})$.

The proof of validity and correctness of the above procedure can be found in, for example, Pham (2014). Figure 8.1 illustrates the MVC, the CLC, and the other profiles from the above steps.

Figure 8.1: This figure illustrates the Maximum Velocity Curve (MVC), the Concatenated Limiting Curve (CLC) and other α - and β -profiles. The time-optimal velocity profile is a concatenation of the β -profile from $(0, \dot{s}_{beg})$, a portion of the CLC, and the α -profile.

8.2 Admissible Velocity Propagation

8.2.1 Overview

This Section gives a brief overview of the Admissible Velocity Propagation (AVP) algorithm presented in Pham, Caron, and Nakamura (2013). We shall call this algorithm “AVP-Forward” to distinguish it from the proposed extension “AVP-Backward”, which will be presented in the next Section.

AVP-Forward takes as its inputs

- a path P in the configuration space; and
- an initial path velocity interval $[\dot{s}_{beg}^{\min}, \dot{s}_{beg}^{\max}]$.

It returns the interval $[\dot{s}_{end}^{\min}, \dot{s}_{end}^{\max}]$ of *all* path velocities that the system can reach at the end of the path P after traversing P while respecting all the system constraints. The algorithm is given as follows.

Step 1 Compute the Concatenated Limiting Curve (CLC).

Step 2 Determine the maximum final velocity $\dot{s}_{end}^{\max}$.

Step 3 Determine the minimum final velocity $\dot{s}_{end}^{\min}$.

The details of the above three steps can be found in Pham et al. (2016).

8.2.2 Combining AVP with (Unidirectional) Sampling-Based Planners

AVP-Forward can be combined with various sampling-based planners. The example planner presented in this Section integrates AVP-Forward into the traditional RRT planner (LaValle and Kuffner 2000), resulting in a planner called AVP-RRT. The planner is listed in Algorithm 10.

Algorithm 10: AVP-RRT

Input: q_{start} and q_{goal}

Output: A valid trajectory connecting q_{start} and q_{goal} or Failure

```

1  $\mathcal{T} \leftarrow \text{Tree}()$ 
2  $\mathcal{V}_{\text{start}} \leftarrow \text{Vertex}()$ 
3  $\mathcal{V}_{\text{start}}.\text{config} \leftarrow q_{\text{start}}; \mathcal{V}_{\text{start}}.\text{interval} \leftarrow [0, 0]; \mathcal{V}_{\text{start}}.\text{path} \leftarrow \text{Null}$ 
4  $\text{InitializeTree}(\mathcal{T}, \mathcal{V}_{\text{start}})$ 
5 for iteration = 1 to  $N_{\text{maxiter}}$  do
6    $q_{\text{rand}} \leftarrow \text{SampleConfig}()$ 
7   if  $\text{ExtendTree}(\mathcal{T}, q_{\text{rand}})$  then
8     if  $\text{Connect}(\mathcal{T}, q_{\text{goal}})$  then
9       return  $\text{ExtractTrajectory}(\mathcal{T}, q_{\text{goal}})$ 
10    end if
11  end if
12 end for
13 return Failure

```

AVP-RRT proceeds in the same way RRT does. Initially, the algorithm initializes a tree $\mathcal{T}$ and a start vertex $\mathcal{V}_{\text{start}}$. However, apart from storing a configuration and a path *from* itself to its parent on the tree, here a vertex $\mathcal{V}$ also stores “interval”, which is the reachable interval of path velocities at $\mathcal{V}.\text{config}$, i.e., at the end of $\mathcal{V}.\text{path}$.

At each iteration, the algorithm first samples a random configuration q_{rand} . Then it extends the tree $\mathcal{T}$ (see Algorithm 11) towards the newly sampled configuration from the closest vertex, measured under some certain distance metric d , on the tree. The algorithm terminates when one of the following occurs:

1. A new vertex generated by ExtendTree can be connected to the goal configuration (Algorithm 10 line 8). By recursion, AVP guarantees that the path connecting q_{start} and q_{goal} is time-parameterizable.
2. No vertex can be connected to the goal configuration after N_{maxiter} iterations. In this case, AVP-RRT returns Failure.

Algorithm 11: ExtendTree

Input: $\mathcal{T}$ and q_{rand} **Output:** True or False

```
1  $\mathcal{V}_{\text{near}} \leftarrow \text{NearestNeighbor}(\mathcal{T}, q_{\text{rand}})$ 
2  $P_{\text{new}}, q_{\text{new}} \leftarrow \text{InterpolatePath}(\mathcal{V}_{\text{near}}, q_{\text{rand}})$ 
3 if  $P_{\text{new}}$  is collision-free then
4    $[\dot{s}_{\text{min}}, \dot{s}_{\text{max}}] \leftarrow \text{AVP-Forward}(P_{\text{new}}, \mathcal{V}_{\text{near}}.\text{interval})$ 
5   if AVP-Forward succeeds then
6      $\mathcal{V}_{\text{new}} \leftarrow \text{Vertex}()$ 
7      $\mathcal{V}_{\text{new}}.\text{config} \leftarrow q_{\text{new}}; \mathcal{V}_{\text{new}}.\text{interval} \leftarrow [\dot{s}_{\text{min}}, \dot{s}_{\text{max}}]; \mathcal{V}_{\text{new}}.\text{path}$ 
        $\leftarrow P_{\text{new}}$ 
8      $\text{AddVertex}(\mathcal{T}, \mathcal{V}_{\text{near}}, \mathcal{V}_{\text{new}})$ 
9     return True
10  end if
11 end if
12 return False
```

Brief descriptions of other functions in Algorithm 10 and Algorithm 11 are given below.

- **Connect**($\mathcal{T}, q_{\text{goal}}$): This function attempts to connect the most recently added vertex ($\mathcal{V}_{\text{new}}$ from Algorithm 11 line 8) to $\mathcal{T}$ to the goal configuration. The procedure is similar to that of Algorithm 11, starting from line 2. However, in the path feasibility checking procedure, starting from line 3, an extra checking is added to make sure that the goal velocity is included in the propagated velocity interval.
- **ExtractTrajectory**($\mathcal{T}, q_{\text{goal}}$): A path P from q_{start} to q_{goal} is extracted from $\mathcal{T}$ by recursively concatenating $\mathcal{V}.\text{path}$. Then P is time-parameterized by using TOPP, as described in Section 8.1. Thanks to AVP-Forward, P is guaranteed to be time-parameterizable.
- **NearestNeighbor**($\mathcal{T}, q$): This function returns $\mathcal{V}_{\text{near}}$ such that $\mathcal{V}_{\text{near}}.\text{config}$ is the closest, among all vertices on the tree, to q according to the distance metric d .
- **InterpolatePath**($\mathcal{V}, q$): Given a distance metric d and a user-defined maximum extension distance D , this function returns a new configuration q_{new} and a path P_{new} , where
 - if $d(\mathcal{V}.\text{config}, q) \leq D$, then $q_{\text{new}} \leftarrow q$; otherwise, q_{new} is a configuration in the direction from $\mathcal{V}.\text{config}$ towards q but within a distance D from $\mathcal{V}.\text{config}$.

- P_{new} is a continuous path connecting $\mathcal{V}.\text{config}$ and $\mathbf{q}_{\text{new}}$ such that the concatenation of $\mathcal{V}.\text{path}$ and P_{new} is C^1 continuous at $\mathcal{V}.\text{config}$ ¹.
- $\text{AddVertex}(\mathcal{T}, \mathcal{V}_{\text{parent}}, \mathcal{V}_{\text{new}})$: This function adds the vertex $\mathcal{V}_{\text{new}}$ to $\mathcal{T}$ as a child of $\mathcal{V}_{\text{parent}}$.

When integrated into a sampling-based planner, as described above in case of AVP-RRT, AVP-Forward serves as another unit for path feasibility checking in the tree extension routine. It is readily applicable to a unidirectional planner since in this case, velocity intervals need to be propagated in the *forward* direction, starting from the initial vertex $\mathcal{V}_{\text{start}}$. To integrate AVP into a bidirectional planner such as a bidirectional RRT (BiRRT) (LaValle and Kuffner 2000), however, one also needs to be able to propagate velocity intervals *backward* along the paths *towards* $\mathbf{q}_{\text{goal}}$. In the next Section, we describe in detail the proposed extension – AVP-Backward – that propagates a valid *final* velocity interval *backward* along a given path to obtain the valid *initial* velocity interval.

8.3 AVP-Backward: The Proposed Extension

The algorithm presented here is a more detailed version of Lertkultanon and Pham (2014). We have implemented AVP-Backward in C++ as part of TOPP library, which can be found at <https://github.com/quangounet/TOPP>.

8.3.1 Algorithm

AVP-Backward takes as its input

- a path P in the configuration space; and
- a *final* path velocity interval $[\dot{s}_{\text{end}}^{\min}, \dot{s}_{\text{end}}^{\max}]$.

It returns the interval $[\dot{s}_{\text{beg}}^{\min}, \dot{s}_{\text{beg}}^{\max}]$ of all path velocities such that the system can traverse P while respecting all the system constraints and reach a final velocity $\dot{s}_{\text{end}} \in [\dot{s}_{\text{end}}^{\min}, \dot{s}_{\text{end}}^{\max}]$ *if and only if* the system starts traversing P with $\dot{s}_{\text{beg}} \in [\dot{s}_{\text{beg}}^{\min}, \dot{s}_{\text{beg}}^{\max}]$.

Similar to AVP-Forward, the algorithm AVP-Backward comprises the following three steps:

¹If the concatenated path is not C^1 -continuous at $\mathcal{V}.\text{config}$, any valid velocity profile must *stop*, i.e., reaching zero velocity, at the $\mathcal{V}.\text{config}$, as previously discussed in Chapter 7 Section 7.1.

Step 1 Compute the Concatenated Limiting Curve (CLC).

Step 2 Determine the maximum initial velocity $\dot{s}_{\text{beg}}^{\text{max}}$.

Step 3 Determine the minimum initial velocity $\dot{s}_{\text{beg}}^{\text{min}}$.

Now we detail each of the above steps.

Step 1: Computing the Limiting Curve

First, we state a useful Lemma proven in Pham et al. (2016).

Lemma 2. *One and only one of the following is true:*

- *One the the limiting curves reaches $\dot{s} = 0$.*
- *CLC is continuous.*

When a limiting curve hits $\dot{s} = 0$, the path cannot be traverse without violating any constraint. In this case, AVP-Backward terminates with failure. Therefore, in the following, we assume that the CLC is continuous and strictly positive. Since the CLC is bounded by $s = 0$ from the left, $s = s_{\text{end}}$ from the right, the MVC from the above, and $\dot{s} = 0$ from the bottom, there are only four exclusive and exhaustive cases, listed below.

- 1A** The CLC hits the MVC while integrating backward and forward. Let $\dot{s}_{\text{end}}^* = \text{MVC}(s_{\text{end}})$ and go to **Step 2**. Note that the case when there is no $\alpha \rightarrow \beta$ switch point on the MVC is assimilated into this case.
- 1B** The CLC hits the MVC while integrating backward and $s = s_{\text{end}}$ while integrating forward. Let $\dot{s}_{\text{beg}}^* = \text{CLC}(s_{\text{end}})$ and go to **Step 2**.
- 1C** The CLC hits $s = 0$ while integrating backward and the MVC while integrating forward. Let $\dot{s}_{\text{end}}^* = \text{MVC}(s_{\text{end}})$ and go to **Step 2**.
- 1D** The CLC hits $s = 0$ while integrating backward and $s = s_{\text{end}}$ while integrating forward. Let $\dot{s}_{\text{end}}^* = \text{CLC}(s_{\text{end}})$ and go to **Step 2**.

In all of the above cases, $\dot{s}_{\text{end}}^*$ has been defined such that no valid velocity profile can end with a velocity $\dot{s} > \dot{s}_{\text{end}}^*$. Therefore, if $\dot{s}_{\text{end}}^{\text{min}} > \dot{s}_{\text{end}}^*$, AVP-Backward terminates with failure². Otherwise, we now have the *valid* final velocity interval as $[\dot{s}_{\text{end}}^{\text{min}}, \dot{s}_{\text{end}}^{\text{max}*}]$, where $\dot{s}_{\text{end}}^{\text{max}*} = \min(\dot{s}_{\text{end}}^{\text{max}}, \dot{s}_{\text{end}}^*)$.

²It is worth noting that when AVP-Backward terminates with failure, the path P may actually be traversable. However, in that case, the system cannot traverse P and end with a final velocity in the desired range.

Figure 8.2: This figure illustrates **Step 2**. The initial maximum velocity $\dot{s}_{\text{beg}}^{\text{max}}$ is determined by integrating backward from $(s_{\text{end}}, \dot{s}_{\text{end}}^{\text{max}*})$ following α .

Step 2: Determining the Maximum Initial Path Velocity

Definition 21. A velocity $\dot{s}_{\text{beg}}$ is a valid initial velocity if there exists a valid velocity profile that starts at $(0, \dot{s}_{\text{beg}})$ and ends at $(s_{\text{end}}, \dot{s}_{\text{end}})$, for some $\dot{s}_{\text{end}} \in [\dot{s}_{\text{end}}^{\text{min}}, \dot{s}_{\text{end}}^{\text{max}*}]$.

The maximum initial path velocity can be obtained by integrating *backward* from $(s_{\text{end}}, \dot{s}_{\text{end}}^{\text{max}*})$ following α . Let Ψ denote the resulting velocity profile from the backward integration. Since Ψ is continuous and bounded by $s = 0$ from the left, $\dot{s} = 0$ from the bottom, and the MVC or CLC from the top, there are four exclusive and exhaustive cases as follows.

- 2A** Ψ hits $\dot{s} = 0$ (see the Ψ -profile **2A** in Figure 8.2). In this case, the path P is not traversable. AVP-Backward terminates with failure.
- 2B** Ψ hits $s = 0$ (see the Ψ -profile **2B** in Figure 8.2). In this case, one has $\dot{s}_{\text{beg}}^{\text{max}} = \Psi(0)$. This is because if one starts with a velocity $\dot{s} > \Psi(0)$, the reachable minimum final velocity will be greater than $\dot{s}_{\text{end}}^{\text{max}}$, which is therefore invalid. Go to **Step 3**.
- 2C** Ψ hits the CLC. There are two subcases depending on the CLC:
 - 2C-1** The CLC begins at $s = 0$, i.e., we proceed from the case **1C** or **1D**. Then indeed $\text{CLC}(0)$ is our maximum initial velocity. Go to **Step 3**.
 - 2C-2** The CLC hits the MVC, say at $s = s_{\text{hit}} > 0$, while integrating backward from a switch point, i.e., we proceed from the cases **1A** or **1B**. Then go to **2D**.
- 2D** Ψ hits the MVC, say at $s = s_{\text{hit}} > 0$. We mark $\text{MVC}(0)$ as a candidate for $\dot{s}_{\text{beg}}^{\text{max}}$ and test its validity via the function `IsValid`, listed in Algorithm 12:

2D-1 IsValid returns True. Here $MVC(0)$ is the maximum initial velocity.
Go to **Step 3**.

2D-2 IsValid returns False. The path P is not traversable. AVP-Backward terminates with failure. This is because if $IsValid(\dot{s}_{test})$ returns False, then no velocity $\dot{s} < \dot{s}_{test}$ can be valid either.

Algorithm 12: IsValid

Input: $\dot{s}_{test}$

Output: True if and only if $\dot{s}_{test}$ is a valid initial velocity

```

1 Integrate forward from  $(0, \dot{s}_{test})$  following  $\beta$ . Call the resulting profile  $\Phi$ .
2 if  $\Phi$  hits  $\dot{s} = 0$  then
    | /* 3A (see the  $\Phi$ -profile 3A in Figure 8.3). */
    | /* In this case, no valid velocity profile can start from  $\dot{s} \in [0, \dot{s}_{test}]$ . */
3     return False
4 elif  $\Phi$  hits  $s = s_{end}$  at  $\dot{s} = \dot{s}_0 < \dot{s}_{end}^{min}$  then
    | /* 3B (see the  $\Phi$ -profile 3B in Figure 8.3). */
5     return False
6 elif  $\Phi$  hits  $s = s_{end}$  at  $\dot{s} = \dot{s}_0 \in [\dot{s}_{end}^{min}, \dot{s}_{end}^{max}]$  then
    | /* 3C (see the  $\Phi$ -profile 3C in Figure 8.3). */
7     return True
8 elif  $\Phi$  hits  $\Psi$  (see the  $\Phi$ -profile 3D in Figure 8.3) then
    | /* 3D (see the  $\Phi$ -profile 3D in Figure 8.3). */
    | /* In this case,  $\dot{s}_{test}$  is a valid initial velocity since the concatenation of
    |    $\Phi$  and  $\Psi$ , which starts at  $\dot{s} = \dot{s}_{test}$ , ends at  $\dot{s} = \dot{s}_{end}^{max}$ . */
9     return True
10 else
    | /* 3E In this case,  $\Phi$  hits the CLC (see the  $\Phi$ -profile 3E in Figure 8.3). */
    | /* In this case,  $\dot{s}_{test}$  is a valid initial velocity since the concatenation of
    |    $\Phi$ , a portion of the CLC, and  $\Psi$ , which starts at  $\dot{s} = \dot{s}_{test}$ , ends at
    |    $\dot{s} = \dot{s}_{end}^{max}$ . */
11     return True
12 end if

```

Note that the five cases presented in Algorithm 12 are exclusive and exhaustive since the Φ -profile cannot hit the MVC (Pham et al. 2016).

Step 3: Determining the Minimum Initial Path Velocity

When AVP-Backward reaches this step, the maximum initial velocity $\dot{s}_{beg}^{max}$ has been determined. We now note the following two properties:

Figure 8.3: This figure illustrates the function `IsValid`. The function checks the validity of $\dot{s}_{\text{test}}$ by integrating forward from $(0, \dot{s}_{\text{test}})$ following β .

- If $\dot{s}_0$ is a valid initial velocity, i.e., `IsValid`($\dot{s}_0$) returns True, then any velocity $\dot{s} \in [\dot{s}_0, \dot{s}_{\text{beg}}^{\max}]$ is also a valid initial velocity. This can be deduced from Algorithm 12.
- The set of valid initial velocities is an interval (Pham et al. 2016).

Thanks to the above properties, $\dot{s}_{\text{beg}}^{\min}$ can be determined as follows. First, test if $\dot{s} = 0$ is valid. If `IsValid` returns True, one automatically has $\dot{s}_{\text{beg}}^{\min} = 0$. Otherwise, one can search on the interval $(0, \dot{s}_{\text{beg}}^{\max}]$ using the bisection method.

8.3.2 Integration into Bidirectional Planners

In this Section, we describe a planner called AVP-BiRRT, which is an integration of AVP (AVP-Forward and AVP-Backward) into a standard bidirectional RRT (LaValle and Kuffner 2000). The algorithm AVP-BiRRT is listed in Algorithm 13.

AVP-BiRRT grows two trees; $\mathcal{T}_{\text{FW}}$ is rooted at $\mathcal{V}_{\text{start}}$ and $\mathcal{T}_{\text{BW}}$ is rooted at $\mathcal{V}_{\text{goal}}$. Vertices stored in both trees have the same data structure, i.e., each vertex contains a triplet (config, path, interval). However, there are slight differences in the entries of vertices. For $\mathcal{V} \in \mathcal{T}_{\text{FW}}$, $\mathcal{V}.\text{path}$ stores a path *from* the parent of $\mathcal{V}$ to $\mathcal{V}.\text{config}$ and $\mathcal{V}.\text{interval}$ stores the valid *final* velocity interval. For $\mathcal{V} \in \mathcal{T}_{\text{BW}}$, $\mathcal{V}.\text{path}$ stores a path *to* the parent of $\mathcal{V}$ from $\mathcal{V}.\text{config}$ and $\mathcal{V}.\text{interval}$ stores the valid *initial* velocity interval.

At each iteration, a new random configuration $\mathbf{q}_{\text{rand}}$ is sampled. Then depending on the iteration, one of the two trees is extended towards $\mathbf{q}_{\text{rand}}$. Then, similar to AVP-RRT presented in Section 8.2.2, the algorithm terminates when either

1. A new vertex generated by `ExtendTree` can be connected to the other tree

Algorithm 13: AVP-BiRRT

Input: q_{start} and q_{goal} **Output:** A valid trajectory connecting q_{start} and q_{goal} or Failure

```
1  $\mathcal{T}_{\text{FW}} \leftarrow \text{Tree}(); \mathcal{T}_{\text{BW}} \leftarrow \text{Tree}()$ 
2  $\mathcal{V}_{\text{start}} \leftarrow \text{Vertex}(); \mathcal{V}_{\text{goal}} \leftarrow \text{Vertex}()$ 
3  $\mathcal{V}_{\text{start}}.\text{config} \leftarrow q_{\text{start}}; \mathcal{V}_{\text{start}}.\text{interval} \leftarrow [0, 0]; \mathcal{V}_{\text{start}}.\text{path} \leftarrow \text{Null}$ 
4  $\mathcal{V}_{\text{goal}}.\text{config} \leftarrow q_{\text{goal}}; \mathcal{V}_{\text{goal}}.\text{interval} \leftarrow [0, 0]; \mathcal{V}_{\text{goal}}.\text{path} \leftarrow \text{Null}$ 
5  $\text{InitializeTree}(\mathcal{T}_{\text{FW}}, \mathcal{V}_{\text{start}})$ 
6  $\text{InitializeTree}(\mathcal{T}_{\text{BW}}, \mathcal{V}_{\text{goal}})$ 
7  $\text{FW} \leftarrow 1; \text{BW} \leftarrow 0$ 
8 for iteration = 1 to  $N_{\text{maxiter}}$  do
9    $q_{\text{rand}} \leftarrow \text{SampleConfig}()$ 
10  if mod (iteration, 2) == FW then
11    if  $\text{ExtendTree}(\mathcal{T}_{\text{FW}}, q_{\text{rand}}, \text{FW})$  then
12      if  $\text{Connect}(\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}}, \text{FW})$  then
13        return  $\text{ExtractTrajectory}(\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}})$ 
14      end if
15    end if
16  else
17    if  $\text{ExtendTree}(\mathcal{T}_{\text{BW}}, q_{\text{rand}}, \text{BW})$  then
18      if  $\text{Connect}(\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}}, \text{BW})$  then
19        return  $\text{ExtractTrajectory}(\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}})$ 
20      end if
21    end if
22  end if
23 end for
24 return Failure
```

(Algorithm 13 line 11 or line 18). By recursion, AVP guarantees that the path connecting q_{start} and q_{goal} is time-parameterizable.

2. The two trees cannot be connected to each other after N_{maxiter} iterations. In this case, AVP-BiRRT returns Failure.

The functions in Algorithm 13 are similar to the functions in AVP-RRT (Algorithm 10). For clarity, we discuss the functions `ExtendTree` and `Connect` below.

- `ExtendTree( $\mathcal{T}$ ,  $q_{\text{rand}}$ , direction)`: The general pipeline of this function is the same as `ExtendTree` described in Algorithm 11. However, the detailed procedure is slightly different.
 - If direction is FW, the procedure is exactly the same as in Algorithm 11.

- Else direction is BW. The new configuration q_{new} is determined as a configuration within a distance D from $\mathcal{V}_{\text{near.config}}$. Then `InterpolatePath` generates a path P_{new} from q_{new} to $\mathcal{V}_{\text{near.config}}$. If the path is collision-free, `AVP-Backward` is called to obtain the valid *initial* velocity interval.
- `Connect( $\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}}, \text{direction}$ )`: If `direction` is FW, this means the forward tree $\mathcal{T}_{\text{FW}}$ has just been extended. This function then attempts to connect $\mathcal{V}_{\text{new}}$ on $\mathcal{T}_{\text{FW}}$ with the nearest vertex on $\mathcal{T}_{\text{BW}}$. Otherwise, when `direction` is BW, the function attempts to connect the new vertex on $\mathcal{T}_{\text{BW}}$ with the nearest vertex on $\mathcal{T}_{\text{FW}}$.

When a collision-free path is successfully generated between the two trees, one needs to check that the final velocity interval propagated from $\mathcal{V}_{\text{start}}$ and the initial velocity interval propagated from $\mathcal{V}_{\text{goal}}$ have non-empty intersection. If the intersection is non-empty, the overall trajectory is constructed via `ExtractTrajectory`. If the two intervals do not intersect, the path, although connected geometrically, is not time-parameterizable and `Connect` returns `False`.

8.4 Example Application: Dynamic Non-Prehensile Object Transportation

In this Section, we demonstrate the capability of AVP-BiRRT by addressing a kinodynamic motion planning problem where *no quasi-static solution exists*. In the task, a bottle placed on a tray is to be transported through a small opening using a commercially available 6-DOF manipulator. This is, to the best of our knowledge, *the first successful demonstration on a non-custom-built robot* that kinodynamic planning can succeed when quasi-static planning is guaranteed to fail.

8.4.1 Task Description

We consider the non-prehensile transportation of a bottle on a tray attached to the end-effector of an industrial manipulator. The motion may be recognized as “waiter motion”. Unlike prehensile object transportation in which a robot carries an object by grasping, here the transportation relies on only contact and friction forces between the object and the tray. Non-prehensile transportation can, therefore, be faster and more efficient since it skips the time-consuming grasping and ungrasping processes entirely.

The plastic milk bottle used in this experiment had 8 cm. $\times$ 8 cm. base and

its height was 24 cm. It was partially filled with sand³ up to 16 cm. The bottle was placed on a tray – without any fixation device – that was mounted on the end-effector of a 6-DOF serial manipulator (Denso VS-060). The robot had to bring the bottle from an initial configuration towards a goal configurations. The two configurations were separated by a small opening. The scene is as shown in Figure 8.5(a).

For the bottle to remain stationary with respect to the tray throughout the motion, three conditions must be met:

1. Unilaterality: The normal component f_n of the reaction force $\mathbf{f}$ (see Figure C.1) must be non-negative.
2. Non-slippage: The tangential component f_t of the reaction force $\mathbf{f}$ (see Figure C.1) must satisfy $\|\mathbf{f}_t\| \leq \mu f_n$, where μ is the static friction coefficient of the contact between the bottle and the tray. In our experiment set-up, the friction coefficient was set to a high value ($\mu = 1.7$), such that non-slippage condition was never violated before the ZMP condition.
3. ZMP (Vukobratovic, Borovac, and Surdilovic 2001): For the bottle not to tip over, its zero-moment point (ZMP) must lie inside the bottle support area.

The height of the opening was designed so that for the bottle to go through the opening, it must be tilted by at least an angle θ^* . However, when the bottle is tilted by that angle, the projection of the center of mass (COM) of the bottle lies *outside* the support area of the bottle. As the projection of the COM coincides with the ZMP in the quasi-static case, tilting the bottle by the angle θ^* violates the ZMP condition and therefore, the bottle will tip over. One can thus conclude that *no quasi-static motion* can bring the bottle through the opening while respecting all the constraints.

8.4.2 Solution using AVP-BiRRT

To use AVP-BiRRT to plan “waiter motion” subject to the aforementioned kinodynamic constraints, we first reduced the three constraints into the canonical form (Inequality 8.3). The details of the reduction are presented in Appendix C.

To further enhance the performance of AVP-BiRRT, we implemented some additional heuristics as follows.

³Sand was used instead of water or other liquid to avoid spilling, which would affect the dynamics of the system.

Figure 8.4: This figure shows how different opening heights affect the computation time. The red dashed vertical line indicates the critical height below which no quasi-static trajectory is possible.

K nearest neighbors: In `ExtendTree`, the function attempts to connect the sampled configuration q_{rand} with K nearest neighbors, sorted in ascending order of their distances to q_{rand} . In this experiment, we set K to be the number of all vertices on the tree.

Bridge test: As the opening was very small, we considered it as a narrow passage in the configuration space. We made use of the bridge test (Hsu et al. 2003) in order to automatically sample a sizable number of configurations that are close, or even inside, the opening.

To account for the discrepancy between the planned motion and the motion actually executed on the robot, we set the bottle support area, used in checking the ZMP condition, to be 5.5 cm. $\times$ 5.5 cm. (the actual base size was 8 cm. $\times$ 8 cm.), which made the planning problem even harder. Nevertheless, our algorithm was able to find a feasible movement in about 3 hours on a 3.2 GHz desktop with 3.8 GB RAM. This movement was successfully executed on the actual robot (see Figure 8.6). The video of the robot executing the motion can be found at <https://youtu.be/LdZSjNwpJs0>.

Remark. Note that the computation time of 3 hours was for a particularly difficult problem instance: if the opening were only 5 cm. higher, the computation time would be about only 2 minutes (see Figure 8.4).

To see how the height of the opening affects the running time, we varied the opening height and determined the average and standard deviation of computation time required to find a solution. We carried out 30 runs for each of the opening heights from 0.4 m. to 0.365 m., 10 runs for 0.36 m., 3 runs for 0.355 m. and 0.35 m., and 2 runs for 0.345 m. The results are plotted in Figure 8.4. Here we used

the actual size of the bottle base (8 cm. $\times$ 8 cm.) as the boundaries for the ZMP condition, so that the computed motions, while theoretically feasible, might not be actually feasible.

Figure 8.5: (a) Simulation environment. The robot has to bring the bottle to the other side of the wall through the opening while keeping it stationary with respect to the tray. (b) The two trees of AVP-BiRRT plotted in the workspace. The start tree had 125 vertices while the goal tree had 116 vertices. Each green dot corresponds to the position of the bottle COM stored in a tree. Red stars represent the initial and goal positions of the bottle COM. The successful path is highlighted in blue; it had 6 vertices. (c) and (d) The MVC (red), Φ -profile (yellow), Ψ -profile (blue), the final velocity profile (green), and other velocity profiles (black) are plotted in the $s - \dot{s}$ plane. (e) The ZMP of the bottle in the tray reference frame (RF). Note that the ZMP always stayed within the imposed safety borders ± 2.75 cm. (f) The COM of the bottle in the tray RF. The x -coordinate of the COM reached the maximum of 4.03 cm. around the moment when the bottle went through the opening, indicating that the successful trajectory would not be quasi-statically stable.

Figure 8.6: Snapshots of the motion discovered by AVP-BiRRT, taken every 0.5 s. The first rows show the motion in the simulation environment. The last two rows show the motion executed on the actual robot (see also the video at <https://youtu.be/LdZSjNwpJs0>).

Conclusion

We have presented an extension to the AVP algorithm, called AVP-Backward. The extension enables the velocity propagation method to be integrated into bidirectional sampling-based planners. We have implemented AVP-Backward, integrated it into a planner, resulting in AVP-BiRRT. The planner was used to solve a challenging kinodynamic motion planning problem and the discovered motion was successfully executed on the actual robot. To the best of our knowledge, **this is the first successful demonstration on a non-custom-built robot that kinodynamic planning can succeed when quasi-static planning is guaranteed to fail.**

Chapter 9

Conclusion

9.1 Summary

In this thesis, we have presented novel unimanual and bimanual manipulation planners addressing complex manipulation tasks requiring regrasping operations. In addition, we have also presented two improvements to motion planning.

Unimanual Manipulation Planning

When solving a manipulation problem with regrasping operations, information on how the object can be picked up (i.e., the grasp set), how the object can be placed down (i.e., the placement set), and interrelation between them plays a critical role in guiding a manipulation planner to explore the high-dimensional composite configuration space systematically. We have presented the high-level Grasp-Placement Graph, which captures the potential structure of the composite configuration space, while not requiring heavy pre-processing and being specific to only the gripper and the manipulated object. Then we presented a planner that explores the space based on the guidance given by the graph. The capabilities of the proposed planner was illustrated through software and hardware experiments.

Bimanual Manipulation Planning

While unimanual manipulation planning is a relatively established field of research with a number of working demonstrations, the bimanual manipulation problem when the object is large/heavy such that it can only be moved when grasped by both hands has received scarcely any attention.

We first addressed a more specific when a bimanual grasp remains the same throughout the motion. We have presented a new concept of essentially mutu-

ally disconnectedness of closed-chain configurations that gives a new perspective of characterizing the feasible configuration space, which is essentially a set of manifolds. We presented a new closed-chain planner that, thanks to the new move called IK-switch, is able to plan motions across different essentially mutually disconnected components of closed-chain configurations. Moreover, this planner can also be integrated into any bimanual manipulation planner as a component to plan challenging closed-chain motions.

We then moved on to a more general setting where multiple (supposedly infinite) bimanual grasps are available and the object needs both hands grasping in order to be moved. We presented a certified-complete bimanual manipulation planner that, when a certificate is available, guarantees to find a solution to any bimanual manipulation query whenever one exists. To the best of our knowledge, this is the first planner that exhibits such a property, and the presented hardware experiment is also the first ever to illustrate such regrasping capability that solves a complex bimanual regrasping problem, as shown in Figure 1.2, on an actual bimanual system.

Improvements to Motion Planning

Motion planning is a crucial component in a manipulation planner. Fast manipulation motions largely benefit from fast trajectories, subject to various constraints, generated by a motion planner. We have presented two improvements towards generation of safer and faster motions.

First, we noticed that existing methods for generating second-order trajectories produced a large number of switch points – the time instants when the acceleration of any joint changes. These switch points, which are sometimes also concentrated in a short time interval, can potentially result in poor tracking and cause motor wear. Therefore, we presented a new constraint called minimum-switch-time constraint that essentially controls concentration of switch points. Then we presented an algorithm that generates time-optimal second-order trajectories subject to not only velocity and acceleration bounds but also the new minimum-switch-time constraint.

From our experiments, we notice that there is a trade-off between the value of minimum allowed switch time and the trajectory duration. Longer minimum switch time may lead to longer trajectory duration. Further investigation is needed in order to systematically determine suitable minimum switch time for various applications.

Second, when generating trajectories following the path-velocity decomposition

approach, the generated path in the first stage might not always be time-parameterizable. To alleviate this issue, Admissible Velocity Propagation (AVP) algorithm can be used. When integrated into a path planner, AVP algorithm enables the planner to discover dynamic ranges of motions. However, the original mechanism of AVP algorithm only makes possible the integration into unidirectional path planners. In this thesis, we have presented a bidirectional planner called AVP-BiRRT, which is a combination of the traditional bidirectional RRT and the AVP algorithm. This integration is rendered possible by our newly proposed extension to AVP—AVP-Backward. The capability of the integrated planner was demonstrated via a hardware experiment that, to the best of our knowledge, is the first demonstration on a non-custom built robot of truly dynamic motions in a scenario where no quasi-static motion exists.

Further work is needed in order to seamlessly integrate AVP-BiRRT into a manipulation planner as a motion planning component. For example, constraints on gripper contact force need to be formulated as TOPP constraints (Inequality (8.1)) such that a transfer path can be time-parameterized to get a fast trajectory while the robot securely grasps the object.

9.2 Future Works

Looking forward, there are, still, a number of obstacles lying ahead along the road towards complete automation of manipulation tasks. In this Section, we point out some interesting and important directions for future research.

Additional Flexibility of Manipulation Actions

So far, we only allow certain manipulation actions to be planned and performed. For example, the only allowed interaction between a robotic system and the manipulated object is grasping. Although grasping alone has already enabled a robotic system to solve a wide range of complex manipulation tasks, being the only available interaction sets a certain limitation to manipulation solutions that can be attained. Allowing other actions such as *pushing* or *dragging* a movable object can potentially expand the set of possible solutions and lead to discovery of better solutions (e.g., in terms of manipulation path length or overall execution time). These additional actions can also help solve problems where grasping alone is insufficient. As an example, consider when a robotic system needs to manipulate a chair seat laying flat on a table. For such a movable object, a robot could only grasp it from one of its sides. Such grasps, however, would not be possible unless the robot moves the piece

towards an edge of the table first. Existing works that incorporate multiple actions such as those presented in Hauser and Ng-Thow-Hing (2011) and Barry et al. (2013) are still unable to fully address the aforementioned issue.

Apart from allowing additional types of robot-object interaction, one may also consider incorporating environment aided-placements into manipulation planning. For example, a movable object may be allowed to *lean* against some obstacles in the scene. These additional placements can potentially lead to shorter manipulation sequences.

Dealing with Uncertainty and Failures

In this thesis, we implicitly assume that all inputs to manipulation planning algorithms such as environment models, robot kinematics, and initial object poses are precise. However, there are always uncertainty in those knowledge; simulation models might be inaccurate, the vision system can only provide data with some limited accuracy, etc. These uncertainties can propagate and even be intensified via various interactions during manipulation motions. Execution of such erroneous plans can eventually cause damages to the manipulated object or even to the robotic system itself. Therefore, one important research direction is then to investigate how knowledge of data uncertainty can be incorporated into planning and even motion execution to produce sufficiently robust manipulation motions.

Unforeseen events happening during motion execution may also cause the executed motion to deviate from the plan or interrupt the execution entirely. Thus, it is also desirable that a robotic system is capable of recovery from such failures.

Appendix A

Closed-Chain Trajectory Execution and Control

Unlike open-chain motions, in closed-chain motions, the robots need to satisfy the closure constraint at all time instant. The two robots interact with each other through the common grasped object. When executing such closed-chain motions on position-controlled robots, whose joints are extremely stiff, even small position errors, such as errors in the relative pose of the two robots, can cause high stress on the grasped object and can eventually damage it. We describe here our control method that we used when executing bimanual closed-chain motions on the actual system, which consists in two *position-controlled* robots, each one equipped with a force-torque (F/T) sensor at its end-effector.

Our control method uses position-based force control (Seraji 1994; Suárez-Ruiz and Pham 2016). Robot trajectories are pre-computed as sets of waypoints. The robots themselves are controlled in a leader-follower fashion. The leader is solely position-controlled, i.e., rigidly follows the given waypoints, while the follower executes trajectories with some added compliance.

Consider the follower robot. Let Δt be the robot controller sampling time (and hence the discretization time step for trajectories), k the discrete time index, $\mathbf{q}$ the command joint values to be given to the controller, $\mathbf{q}_{\text{ideal}}$ the joint values computed by a motion planner, and $\mathbf{q}_{\text{comp}}$ the compensation added to the ideal values for compliance. At time instant k , we read the feedback $\mathbf{f}_{\text{real}}$ F/T sensor and compute the end-effector compensation values $\mathbf{x}_{\text{comp}}$ as

$$\mathbf{x}_{\text{comp}}[k] = \mathbf{K}_p \mathbf{f}_{\text{err}}[k] + \mathbf{K}_v \dot{\mathbf{f}}_{\text{err}}[k], \quad (\text{A.1})$$

where $\mathbf{f}_{\text{err}} = \mathbf{f}_{\text{ideal}} - \mathbf{f}_{\text{real}}$ is the difference between the ideal contact force to be maintained, $\mathbf{f}_{\text{ideal}}$, and the actual measure, $\dot{\mathbf{f}}_{\text{err}}[k] = (\mathbf{f}_{\text{err}}[k] - \mathbf{f}_{\text{err}}[k-1])/\Delta t$, and $\mathbf{K}_p$ and $\mathbf{K}_v$ are control gain matrices. Then the compensation joint values are com-

puted by

$$\mathbf{q}_{\text{comp}}[0] = \mathbf{0}, \quad (\text{A.2})$$

$$\mathbf{q}_{\text{comp}}[k] = \mathbf{q}_{\text{comp}}[k-1] + \mathbf{J}^{-1} \mathbf{x}_{\text{comp}}[k], \quad (\text{A.3})$$

where $\mathbf{J}$ is the Jacobian matrix and a pseudo inverse may be used instead in case of non-square Jacobian. Finally, the command joint values at time k are given by

$$\mathbf{q}[k] = \mathbf{q}_{\text{ideal}}[k] + \mathbf{q}_{\text{comp}}[k]. \quad (\text{A.4})$$

The above per-cycle computation is relatively not computational expensive and can fit into a controller cycle of, at least, as fast as 8 ms., as in our successfully executed hardware experiments shown earlier in Chapter 5 and Chapter 6.

Appendix B

Appendix to Minimum-Switch-Time Constraints

B.1 Proof of Time-Optimality

We present in this Section detailed proofs for the cases PP1 and PLP1. Proofs for the other cases are not given: proofs for the cases PP2 and PLP2 can be easily adapted from PP1 and PLP1, respectively; for the remaining cases, we have already explored all possibilities of re-interpolation.

Consider first the case PP1. Let T_{PP1} be the total duration of the newly re-interpolated velocity profile and t'_0 the duration of the new first ramp. Thus, $T_{PP1} = t'_0 + (v_1 - v'_p)/a_1$, where v_1 and a_1 are the final velocity and the acceleration of the second ramp, and v'_p is the new peak velocity. The following proposition holds.

Proposition 3. T_{PP1} is an increasing function in t'_0 .

Proof. We first consider the case P^+P^- . Substituting the expression of v'_p from (7.19) and $a_1 = -a_m$ into the expression of T_{PP1} yields

$$\begin{aligned} T_{PP1} &= t'_0 + \frac{1}{2a_m} \left(k_1 + \sqrt{k_1^2 + 4k_1v_0 + 4v_1^2 + 8a_md} \right) - \frac{v_1}{a_m} \\ T_{PP1} &= \frac{1}{2} \left(t'_0 - \frac{2v_1}{a_m} \right) + \frac{1}{2} \sqrt{\Delta}, \end{aligned}$$

where $\Delta = (t'_0 - 2v_0/a_m)^2 + 4(v_1^2 - v_0^2 + 2a_md)/a_m^2$. The first derivative of T_{PP1} with respect to t'_0 is then given by

$$\frac{d}{dt'_0} T_{PP1} = \frac{1}{2} + \frac{1}{2} \frac{t'_0 - \frac{2v_0}{a_m}}{\sqrt{\left(t'_0 - \frac{2v_0}{a_m}\right)^2 + \frac{4}{a_m^2} (v_1^2 - v_0^2 + 2a_md)}}. \quad (\text{B.1})$$

If $t'_0 > 2v_0/a_m$, it follows directly that $dT_{PP1}/dt'_0 > 0$. Otherwise, we examine the second term in the square root.

Before proceeding further, notice that for all P^+P^-1 -velocity profile, we always have that $v_0 > v_1$. Assume, for contradiction, that a velocity profile falls in the case P^+P^-1 with $v_0 < v_1$. Since $|a_0| = |a_1| = a_m$, we then have that $(v_p - v_0)/a_m > (v_p - v_1)/a_m$, which results in $t_0 > t_1$. This contradicts with the assumption that the trajectory is in the case P^+P^-1 .

Now consider when $v_0 > v_1$. The time-optimal velocity profile connecting v_0 and v_1 subject to velocity and acceleration bounds has one ramp if and only if the total displacement is $d^* = (v_0^2 - v_1^2)/(2a_m)$. Therefore, for any P^+P^- -velocity profile, the total displacement must be greater than d^* , i.e.,

$$v_1^2 - v_0^2 + 2a_md > 0 \quad (\text{B.2})$$

By applying (B.2) to (B.1), we can see that dT_{PP1}/dt'_0 is always positive. This completes the proof for the case P^+P^- . For the case P^-P^+ , the same argument, which leads to $dT_{PP1}/dt'_0 > 0$, can be made. This completes the proof for the case PP1. ■

Therefore, following Proposition 3, to time-optimally re-interpolate a velocity profile subject to the minimum-switch-time constraint δ , the resulting duration of the first ramp must be the shortest possible, $t'_0 = \delta$.

Next consider the case PLP1. Let t'_0 be the duration of the first ramp of the re-interpolated velocity profile. We claim that in order for the re-interpolated velocity profile to be time-optimal as well as satisfying all the constraints, t'_0 must be only in $[t_0^A, t_0^B]$, where t_0^A is the duration of the new first ramp in case the velocity profile is re-interpolated according to the case PLP1A, i.e., $t_0^A = \delta$, and t_0^B is the duration of the new first ramp in case the velocity profile is re-interpolated according to the case PLP1B. Figure B.1 shows velocity profiles of the original PLP1-velocity profile as well as resulting velocity profiles from PLP1A and PLP1B.

The re-interpolation procedure of a PLP1-velocity profile can be thought of as first re-interpolating the first ramp with a duration $t'_0 \in [t_0^A, t_0^B]$ and then re-interpolating the rest of the profile using routines for PP1. Let $d'_0 = (v'_p - v_0)t'_0/2$ be the displacement covered by the first ramp of the newly re-interpolated velocity profile. The displacement covered by the remaining part of the velocity profile is then $d_{\text{rem}} = d - d'_0$. Let the total duration of the new velocity profile be T_{PLP1} . The following proposition holds.

Proposition 4. T_{PLP1} , as a function of t'_0 , attains its minimum only when $t'_0 = t_0^A$ or $t'_0 = t_0^B$.

Figure B.1: The original PLP1-velocity profile that violates the minimum-switch-time constraint $\delta = 0.2$ is shown in red. The green and blue velocity profiles result from re-interpolating according to PLP1A and PLP1B respectively. The time instants t_0^A and t_0^B are marked with a dashed green line and a dotted blue line respectively. In this particular example, re-interpolation with PLP1A gives a shorter velocity profile.

Proof. First, consider the case when $v_p' = v_m$. The total duration T_{PLP1} can be expressed as

$$T_{\text{PLP1}} = t_0' + T_{\text{PP1}}$$

$$T_{\text{PLP1}} = t_0' + \frac{1}{2} \left(\delta - \frac{2v_1}{a_m} \right) + \frac{1}{2} \sqrt{\left(\delta - \frac{2v_p'}{a_m} \right)^2 + \frac{4}{a_m^2} (v_1^2 - v_p'^2 + 2a_m d_{\text{rem}})}.$$

From the expression of T_{PLP1} , we can compute $d^2 T_{\text{PLP1}} / dt_0'^2$, which is always negative. This implies that the minimum of T_{PLP1} can only occur when either $t_0' = t_0^A$ or $t_0' = t_0^B$.

For the case when $v_p' = -v_m$, the similar argument, which leads to $d^2 T_{\text{PLP1}} / dt_0'^2$, can be made. This completes the proof for the case PLP1. $\blacksquare$

Therefore, following Proposition 4, we can conclude that the time-optimal velocity profile subject to the minimum-switch-time constraint δ must be obtained from either PLP1A or PLP1B.

B.2 Velocity Profile Re-Interpolation with Fixed Total Duration

In this Section, we present a procedure to analytically interpolate a velocity profile given the duration of each ramp.

B.2.1 Velocity Profile Re-Interpolation into Two-Ramp

Let T be the given total duration and τ_0 the given switch point. Since the total displacement must remain the same after re-interpolation, we have

$$d = \left(v_0 \tau_0 + \frac{1}{2} a'_0 \tau_0^2 \right) + \left((v_0 + a'_0 \tau_0) \tau_1 + \frac{1}{2} a'_1 \tau_1^2 \right), \quad (\text{B.3})$$

where a'_0 and a'_1 are accelerations of the first and the last ramp of the new velocity profile respectively, and $\tau_1 = T - \tau_0$. We also have a relationship between v_0 and v_1 as

$$v_1 = v_0 + a'_0 \tau_0 + a'_1 \tau_1. \quad (\text{B.4})$$

We can rewrite Equations (B.3) and (B.4) as a system of linear equations in a'_0 and a'_1 as

$$\begin{bmatrix} \frac{1}{2} \tau_0^2 + \tau_0 \tau_1 & \frac{1}{2} \tau_1^2 \\ \tau_0 & \tau_1 \end{bmatrix} \begin{bmatrix} a'_0 \\ a'_1 \end{bmatrix} = \begin{bmatrix} d - v_0 T \\ v_1 - v_0 \end{bmatrix}. \quad (\text{B.5})$$

We can then solve for a'_0 and a'_1 . Since the determinant of the square matrix is $\tau_0 \tau_1 (\tau_0 + \tau_1) / 2 \neq 0$, the system (B.5) is always solvable.

B.2.2 Velocity Profile Re-Interpolation into Three-Ramp

Let T be the given total duration; τ_0, τ_1, τ_2 the desired duration of the first, the middle and the last ramps of the new velocity profile, respectively. Since the total displacement must remain the same after re-interpolation, we have

$$d = \left(v_0 \tau_0 + \frac{1}{2} a'_0 \tau_0^2 \right) + \left((v_0 + a'_0 \tau_0) \tau_1 + \frac{1}{2} a'_1 \tau_1^2 \right) + \left((v_0 + a'_0 \tau_0 + a'_1 \tau_1) \tau_2 + \frac{1}{2} a'_2 \tau_2^2 \right), \quad (\text{B.6})$$

where v_0 is the initial velocity; v'_{p0} and v'_{p1} are the first and the second peak velocities respectively; a'_0, a'_1 , and a'_2 are accelerations of the first, middle, and last ramps of the new velocity profiles respectively. Let $\alpha = \tau_0^2/2 + \tau_0 \tau_1 + \tau_0 \tau_2$, $\beta = \tau_1^2/2 + \tau_1 \tau_2$, and $\sigma = \tau_2^2/2$. Then Equation (B.6) can be more compactly rewritten as

$$\alpha a'_0 + \beta a'_1 + \sigma a'_2 = d - v_0 T. \quad (\text{B.7})$$

Furthermore, we can derive a relationship between v_0 and v_1 , the final velocity, as

$$v_1 = v_0 + \tau_0 a'_0 + \tau_1 a'_1 + \tau_2 a'_2. \quad (\text{B.8})$$

We can write Equations (B.7) and (B.8) as a system of linear equations in $\mathbf{a} = [a'_0, a'_1, a'_2]^T$ as

$$\mathbf{A}\mathbf{a} = \mathbf{b}, \quad (\text{B.9})$$

where $\mathbf{A} = \begin{bmatrix} \alpha & \beta & \sigma \\ \tau_0 & \tau_1 & \tau_2 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} d - v_0 T \\ v_1 - v_0 \end{bmatrix}$.

The system (B.9) is underdetermined. Therefore, a general solution, $\mathbf{a}_g$, to the system can be expressed as $\mathbf{a}_g = \mathbf{a}_p + \mathbf{a}_h$, where $\mathbf{a}_p$ is a particular solution and $\mathbf{a}_h$ is a non-trivial element of the null space of $\mathbf{A}$, i.e., satisfying $\mathbf{A}\mathbf{a}_h = \mathbf{0}$. Here we take the minimum-norm solution to (B.9) as a particular solution, i.e., $\mathbf{a}_p = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b}$. For a homogeneous solution, we perform Gaussian elimination on $\mathbf{A}$ to obtain

$$\mathbf{a}_h = k \begin{bmatrix} (\beta\tau_2 - \sigma\tau_1)/(\alpha\tau_1 - \beta\tau_0) \\ (\alpha\tau_2 - \sigma\tau_0)/(\alpha\tau_1 - \beta\tau_0) \\ 1 \end{bmatrix},$$

where $k \in \mathbf{R}$.

Next consider the velocity bound v_m . The peak velocities of the re-interpolated velocity profiles are $v'_{p0} = v_0 + a'_0\tau_0$ and $v'_{p1} = v_1 - a'_2\tau_2$. Therefore, we get two sets of constraints on the peak velocities, in terms of the accelerations, as

$$\begin{aligned} (-v_m - v_0)/\tau_0 &\leq a'_0 \leq (v_m - v_0)/\tau_0 \quad \text{and} \\ (-v_m + v_1)/\tau_0 &\leq a'_2 \leq (v_m + v_1)/\tau_0. \end{aligned}$$

Combining velocity and acceleration bounds together yields constraints on accelerations as

$$\mathbf{l} \preceq \mathbf{a}_p + k\mathbf{a}_h \preceq \mathbf{u}, \quad (\text{B.10})$$

where

$$\mathbf{l} = \begin{bmatrix} \min\left(\frac{v_m + v_0}{\tau_0}, a_m\right) \\ -a_m \\ \min\left(\frac{v_m - v_1}{\tau_0}, a_m\right) \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} \min\left(\frac{v_m - v_0}{\tau_0}, a_m\right) \\ a_m \\ \min\left(\frac{v_m + v_1}{\tau_0}, a_m\right) \end{bmatrix}.$$

Finally, we can solve for the feasible interval k from (B.10). If there is no such k satisfying (B.10), then no feasible solution for acceleration exists. Otherwise, any k in the interval can be used to construct a feasible solution for accelerations.

Appendix C

Dynamic Non-Prehensile Object Transportation: Reduction of Constraints into the Canonical Form

In this Chapter, we show how to reduce the constraints in the dynamic non-prehensile object transportation problem into the canonical form (Inequality 8.3).

Suppose the system consists of an n -DOF manipulator with a tray mounted on its end-effector, and a rectangular prism bottle of dimension $2d_x \times 2d_y \times 2d_z$ placed on the tray. First, we start by establishing a connection between the path parameter-related quantities and the bottle COM-related quantities. Let $\mathbf{p}$ be the position of the COM of the bottle in the global frame and $\mathbf{q}$ a robot configuration. Here $\mathbf{p}$ and $\mathbf{q}$ are related through the forward kinematics equation $\mathbf{p} = \mathbf{r}(\mathbf{q})$. Time-differentiating $\mathbf{p}$ once and twice yield the expressions of the velocity and acceleration of the COM as

$$\dot{\mathbf{p}} = \mathbf{r}_q \dot{\mathbf{q}} \quad \text{and} \quad \ddot{\mathbf{p}} = \mathbf{r}_q \ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{r}_{qq} \dot{\mathbf{q}}, \quad (\text{C.1})$$

where $\mathbf{r}_q \in \mathbf{R}^{3 \times n}$ and $\mathbf{r}_{qq} \in \mathbf{R}^{3 \times n \times n}$ are the Jacobian matrix and the Hessian tensor of the function $\mathbf{r}$ with respect to $\mathbf{q}$, respectively. Utilizing the relations in Equation 8.2, one obtains

$$\dot{\mathbf{p}} = \mathbf{p}_s \dot{s} \quad \text{and} \quad (\text{C.2})$$

$$\ddot{\mathbf{p}} = \mathbf{p}_s \ddot{s} + \mathbf{p}_{ss} \dot{s}^2, \quad (\text{C.3})$$

where $\mathbf{p}_s = \mathbf{r}_q \mathbf{q}_s$ and $\mathbf{p}_{ss} = \mathbf{r}_q \mathbf{q}_{ss} + \mathbf{q}_s^\top \mathbf{r}_{qq} \mathbf{q}_s$.

Figure C.1: The free body diagram of the bottle. x_b , y_b , and z_b are the three axes of the bottle reference frame. f_t and $f_n z_b$ are the tangential and normal components of the reaction force, respectively. p is the position of the bottle COM and d is the position of the ZMP, both described in the global frame. k points from the projection of p_b onto the bottle base along z_b to the ZMP.

C.1 Unilaterality Constraint

Consider the free body diagram of the bottle in Figure C.1. From the Newton's second law of motion, one has

$$m\ddot{p} = mg + f_t + f_n z_b \quad (\text{C.4})$$

where f_t is the tangential component of the reaction force f , f_n is the magnitude of the normal component of f , g is the gravitational acceleration, and m is the bottle mass. In order for the bottle not to lose its contact with the tray, f_n must always be non-negative. From Equation (C.4), one can derive an expression of f_n as

$$f_n = m z_b^\top (\ddot{p} - g), \quad (\text{C.5})$$

since $z_b^\top f_t = 0$. By using Equation (C.3), the magnitude f_n can be written in terms of the path parameter s and its time-derivatives as

$$\begin{aligned} f_n &= m z_b^\top (p_s \ddot{s} + p_{ss} \dot{s}^2) - m z_b^\top g \\ f_n &= f_{ns} \ddot{s} + f_{nss} \dot{s}^2 + f_{n0}, \end{aligned} \quad (\text{C.6})$$

Figure C.2: Friction cone. The cone axis coincides with the normal component $f_n \mathbf{z}_b$. The aperture of the cone is $2 \tan^{-1} \mu$, where μ is the static friction coefficient.

where $f_{ns} = m \mathbf{z}_b^\top \mathbf{p}_s$, $f_{nss} = m \mathbf{z}_b^\top \mathbf{p}_{ss}$, and $f_{n0} = -m \mathbf{z}_b^\top \mathbf{g}$. Therefore, the unilaterality constraint can be expressed in the canonical form as

$$-f_{ns}\ddot{s} - f_{nss}\dot{s}^2 - f_{n0} \leq 0. \quad (\text{C.7})$$

C.2 Non-Slippage Constraint

Consider the tangential reaction force, i.e. friction force, $\mathbf{f}_t = m\ddot{\mathbf{p}} - m\mathbf{g} - f_n \mathbf{z}_b$. Substituting f_n from Equation (C.6) and $\ddot{\mathbf{p}}$ from Equation (C.3) into the above equation gives

$$\begin{aligned} \mathbf{f}_t &= m(\ddot{s}\mathbf{p}_s + \dot{s}^2\mathbf{p}_{ss}) - m\mathbf{g} - (f_{ns}\ddot{s} + f_{nss}\dot{s}^2 + f_{n0})\mathbf{z}_b \\ \mathbf{f}_t &= \mathbf{f}_{ts}\ddot{s} + \mathbf{f}_{tss}\dot{s}^2 + \mathbf{f}_{t0}, \end{aligned} \quad (\text{C.8})$$

where $\mathbf{f}_{ts} = m\mathbf{p}_s - f_{ns}\mathbf{z}_b$, $\mathbf{f}_{tss} = m\mathbf{p}_{ss} - f_{nss}\mathbf{z}_b$, and $\mathbf{f}_{t0} = -m\mathbf{g} - f_{n0}\mathbf{z}_b$.

For the bottle to stay on the tray without slipping, $\mathbf{f}_t$ must satisfy

$$\|\mathbf{f}_t\| \leq \mu f_n, \quad (\text{C.9})$$

where μ is the static friction coefficient of the contact. The above inequality can be interpreted as constraining the reaction force $\mathbf{f}_t + f_n \mathbf{z}_b$ to be inside of the friction cone, as depicted in Figure C.2.

However, the constraint in Inequality (C.9) cannot be directly reduced into the canonical form – which is linear in $\ddot{s}$ and $\dot{s}^2$ – due to non-linearity induced by the Euclidean norm. To resolve this problem, we will use a convex approximation of the

Figure C.3: Square-based friction pyramid (blue). Each side of the pyramid base has the length of $\sqrt{2}\mu f_n$.

friction cone, i.e., a friction pyramid (see, e.g., Shimoga 1996). A friction pyramid is a polygonal-based pyramid inscribed in a friction cone. Here we use a square-based pyramid, as shown in Figure C.3.

By considering the friction pyramid, the friction constraint can then be transformed into the following four inequalities:

$$\mathbf{x}_b^\top \mathbf{f}_t \leq \frac{\mu f_n}{\sqrt{2}}, \quad -\mathbf{x}_b^\top \mathbf{f}_t \leq \frac{\mu f_n}{\sqrt{2}}, \quad \mathbf{y}_b^\top \mathbf{f}_t \leq \frac{\mu f_n}{\sqrt{2}}, \quad \text{and} \quad -\mathbf{y}_b^\top \mathbf{f}_t \leq \frac{\mu f_n}{\sqrt{2}}. \quad (\text{C.10})$$

Plugging Equation (C.6) and (C.8) into the above inequalities, one obtains

$$\left(\mathbf{x}_b^\top \mathbf{f}_{ts} - \frac{\mu}{\sqrt{2}} f_{ns} \right) \ddot{s} + \left(\mathbf{x}_b^\top \mathbf{f}_{tss} - \frac{\mu}{\sqrt{2}} f_{nss} \right) \dot{s}^2 + \left(\mathbf{x}_b^\top \mathbf{f}_{t0} - \frac{\mu}{\sqrt{2}} f_{n0} \right) \leq 0, \quad (\text{C.11})$$

$$\left(-\mathbf{x}_b^\top \mathbf{f}_{ts} - \frac{\mu}{\sqrt{2}} f_{ns} \right) \ddot{s} + \left(-\mathbf{x}_b^\top \mathbf{f}_{tss} - \frac{\mu}{\sqrt{2}} f_{nss} \right) \dot{s}^2 + \left(-\mathbf{x}_b^\top \mathbf{f}_{t0} - \frac{\mu}{\sqrt{2}} f_{n0} \right) \leq 0, \quad (\text{C.12})$$

$$\left(\mathbf{y}_b^\top \mathbf{f}_{ts} - \frac{\mu}{\sqrt{2}} f_{ns} \right) \ddot{s} + \left(\mathbf{y}_b^\top \mathbf{f}_{tss} - \frac{\mu}{\sqrt{2}} f_{nss} \right) \dot{s}^2 + \left(\mathbf{y}_b^\top \mathbf{f}_{t0} - \frac{\mu}{\sqrt{2}} f_{n0} \right) \leq 0, \quad (\text{C.13})$$

$$\left(-\mathbf{y}_b^\top \mathbf{f}_{ts} - \frac{\mu}{\sqrt{2}} f_{ns} \right) \ddot{s} + \left(-\mathbf{y}_b^\top \mathbf{f}_{tss} - \frac{\mu}{\sqrt{2}} f_{nss} \right) \dot{s}^2 + \left(-\mathbf{y}_b^\top \mathbf{f}_{t0} - \frac{\mu}{\sqrt{2}} f_{n0} \right) \leq 0, \quad (\text{C.14})$$

which are all in the canonical form.

Remark. Summing together all inequalities in (C.10) gives $0 \leq \sqrt{2}\mu f_n$. Therefore, whenever the friction constraints are satisfied, the unilaterality constraint (C.7)

is automatically satisfied, hence theoretically redundant. However, in some cases when safety is emphasized, one may ensure that the bottle will never leave the tray by modifying the unilaterality constraint to be

$$\begin{aligned} \varepsilon &\leq f_n \\ -f_{ns}\ddot{s} - f_{nss}\dot{s}^2 + (\varepsilon - f_{n0}) &\leq 0. \end{aligned} \quad (\text{C.15})$$

where $\varepsilon > 0$ is a small, strictly positive number.

C.3 ZMP Constraint

In order for the bottle not to tip over, the zero-moment point (ZMP) of the bottle must lie in the support area of the bottle at all time instants.

Consider the total moment acting on the bottle about some point $\mathbf{d}$ lying on the contact surface,

$$\mathbf{M}_{\text{total}} = \mathbf{l} \times m(\mathbf{g} - \ddot{\mathbf{p}}) + \mathbf{M}, \quad (\text{C.16})$$

where $\mathbf{M}$ is the time derivative of the angular momentum of the bottle, $\mathbf{h} = \mathbf{p} - \mathbf{d}$, i.e., the vector pointing from $\mathbf{d}$ to the bottle COM. The point $\mathbf{d}$ is the ZMP of the bottle if the total moment $\mathbf{M}_{\text{total}}$ about $\mathbf{d}$ has no tangential component (Vukobratovic, Borovac, and Surdilovic 2001), i.e.,

$$\mathbf{z}_b \times \mathbf{M}_{\text{total}} = \mathbf{0}_3, \quad (\text{C.17})$$

where $\mathbf{0}_3 \in \mathbf{R}^3$ is a zero vector. By substituting the expression of $\mathbf{M}$ (Equation (C.16)) into the above equation, one gets

$$\mathbf{z}_b \times (\mathbf{l} \times m(\mathbf{g} - \ddot{\mathbf{p}})) + \mathbf{z}_b \times \mathbf{M} = \mathbf{0}_3.$$

Applying the identity $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a}^\top \mathbf{c}) \times \mathbf{b} - (\mathbf{a}^\top \mathbf{b}) \times \mathbf{c}$ to the first term of the left-hand side of the above expression, one obtains

$$m\mathbf{z}_b^\top (\mathbf{g} - \ddot{\mathbf{p}})\mathbf{l} - m(\mathbf{z}_b^\top \mathbf{l})(\mathbf{g} - \ddot{\mathbf{p}}) + \mathbf{z}_b \times \mathbf{M} = \mathbf{0}_3.$$

Note that $\mathbf{z}_b^\top \mathbf{l} = d_z$, i.e., half of the height of the bottle, and $f_n = m\mathbf{z}_b^\top (\ddot{\mathbf{p}} - \mathbf{g})$

Figure C.4: Support area of the bottle. The vector $\mathbf{k}$ points from the projection of the bottle COM to the ZMP.

(Equation (C.6)). Therefore, from the above equation, $\mathbf{l}$ can be expressed as

$$\mathbf{l} = \frac{1}{f_n} (m d_z (\ddot{\mathbf{p}} - \mathbf{g}) + (\mathbf{z}_b \times \mathbf{M})). \quad (\text{C.18})$$

Now define $\mathbf{k}$ as the vector pointing from the projection of the COM on the bottle base to the ZMP, i.e., $\mathbf{k} = d_z \mathbf{z}_b - \mathbf{l}$. Using Equation (C.18), the expression of $\mathbf{k}$ becomes

$$\mathbf{k} = \frac{1}{f_n} (d_z f_n \mathbf{z}_b - m d_z (\ddot{\mathbf{p}} - \mathbf{g}) - (\mathbf{z}_b \times \mathbf{M})). \quad (\text{C.19})$$

This vector $\mathbf{k}$ can be used as an indicator to tell whether the ZMP lies inside the support area of the bottle (see Figure C.4). More precisely, the ZMP is inside the support area if and only if

$$-d_x \leq \mathbf{z}_b^\top \mathbf{k} \leq d_x \quad \text{and} \quad -d_y \leq \mathbf{z}_b^\top \mathbf{k} \leq d_y. \quad (\text{C.20})$$

To transform the above constraints into the canonical form, one needs to write $\mathbf{k}$ (Equation (C.19)) in terms of $\ddot{\mathbf{s}}$ and $\dot{\mathbf{s}}^2$.

Consider the time derivative of the angular momentum of the bottle, $\mathbf{M}$. Let $\mathbf{I} \in \mathbf{R}^{3 \times 3}$ be the inertia matrix of the bottle with respect to the inertial frame, $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ the angular velocity and acceleration of the bottle, respectively. Then $\mathbf{M}$ can be written as (Spong and Vidyasagar 2008)

$$\mathbf{M} = \mathbf{I} \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I} \boldsymbol{\omega}. \quad (\text{C.21})$$

The angular velocity and acceleration of the bottle are related to the robot joint velocity and acceleration via

$$\boldsymbol{\omega} = \mathbf{r}^\omega \dot{\mathbf{q}} \quad \text{and} \quad \dot{\boldsymbol{\omega}} = \mathbf{r}^\omega \ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{r}_q^\omega \dot{\mathbf{q}}, \quad (\text{C.22})$$

where $\mathbf{r}^\omega \in \mathbf{R}^{3 \times n}$ is the angular velocity Jacobian matrix and $\mathbf{r}_q^\omega$ is the Jacobian tensor. Substituting $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ from the above equation into Equation (C.21), with some algebraic manipulation, yields

$$\mathbf{M} = \mathbf{M}_s \ddot{s} + \mathbf{M}_{ss} \dot{s}^2, \quad (\text{C.23})$$

where $\mathbf{M}_s = \mathbf{I} \mathbf{r}^\omega \mathbf{q}_s$ and $\mathbf{M}_{ss} = \mathbf{I}(\mathbf{r}^\omega \mathbf{q}_{ss} + \mathbf{q}_s^\top \mathbf{r}_q^\omega \mathbf{q}_s) + \mathbf{r}^\omega \mathbf{q}_s \times \mathbf{I} \mathbf{r}^\omega \mathbf{q}_s$.

Utilizing Equations (C.3), (C.6), and (C.23), a new expression of $\mathbf{k}$ can be derived from Equation (C.19) as

$$\mathbf{k} = \frac{\mathbf{k}_s \ddot{s} + \mathbf{k}_{ss} \dot{s}^2 + \mathbf{k}_0}{f_{ns} \ddot{s} + f_{nss} \dot{s}^2 + f_{n0}}, \quad (\text{C.24})$$

where $\mathbf{k}_s = d_z f_{ns} \mathbf{z}_b - m d_z \mathbf{p}_s - \mathbf{z}_b \times \mathbf{M}_s$, $\mathbf{k}_{ss} = d_z f_{nss} \mathbf{z}_b - m d_z \mathbf{p}_{ss} - \mathbf{z}_b \times \mathbf{M}_{ss}$, and $\mathbf{k}_0 = d_z f_{n0} \mathbf{z}_b + m d_z \mathbf{g}$.

Finally, by plugging the above expression of $\mathbf{k}$ into the constraints (C.20), one obtains 4 ZMP constraints in the canonical form as

$$(\mathbf{x}_b^\top \mathbf{k}_s - d_x f_{ns}) \ddot{s} + (\mathbf{x}_b^\top \mathbf{k}_{ss} - d_x f_{nss}) \dot{s}^2 + (\mathbf{x}_b^\top \mathbf{k}_0 - d_x f_{n0}) \leq 0, \quad (\text{C.25})$$

$$(-\mathbf{x}_b^\top \mathbf{k}_s - d_x f_{ns}) \ddot{s} + (-\mathbf{x}_b^\top \mathbf{k}_{ss} - d_x f_{nss}) \dot{s}^2 + (-\mathbf{x}_b^\top \mathbf{k}_0 - d_x f_{n0}) \leq 0, \quad (\text{C.26})$$

$$(\mathbf{y}_b^\top \mathbf{k}_s - d_x f_{ns}) \ddot{s} + (\mathbf{y}_b^\top \mathbf{k}_{ss} - d_x f_{nss}) \dot{s}^2 + (\mathbf{y}_b^\top \mathbf{k}_0 - d_x f_{n0}) \leq 0, \quad (\text{C.27})$$

$$(-\mathbf{y}_b^\top \mathbf{k}_s - d_x f_{ns}) \ddot{s} + (-\mathbf{y}_b^\top \mathbf{k}_{ss} - d_x f_{nss}) \dot{s}^2 + (-\mathbf{y}_b^\top \mathbf{k}_0 - d_x f_{n0}) \leq 0. \quad (\text{C.28})$$

Remark. In case the support area of bottle takes a general polygonal shape, the ZMP constraint can also be derived in a similar manner as follows.

Let $\mathbf{p}'$ be the projection of the bottle COM onto the support area. Suppose the polygonal support area has N edges and $\mathbf{n}_i$ is a unit vector pointing from $\mathbf{p}'$ towards the i^{th} edge perpendicularly. Let d_i be the distance from the point $\mathbf{p}'$ to the i^{th} edge. The ZMP constraint can then be expressed as a set of inequalities

$$\mathbf{n}_i^\top \mathbf{k} \leq d_i, \quad i \in \{1, 2, \dots, N\}. \quad (\text{C.29})$$

Bibliography

- Ahn, Kitak, Wan Kyun Chung, and Youngil Youm (2004). “Arbitrary states polynomial-like trajectory (ASPOT) generation”. In: *Industrial Electronics Society (IECON), 2004. 30th Annual Conference of IEEE*. Vol. 1, pp. 123–128.
- Alami, Rachid, Jean-Paul Laumond, and Thierry Siméon (1994). “Two manipulation planning algorithms”. In: *WAFR Proceedings of the workshop on Algorithmic foundations of robotics*, pp. 109–125.
- Amato, Nancy M, O Burchan Bayazit, Lucia K Dale, Christopher Jones, and Daniel Vallejo (1998). “Choosing good distance metrics and local planners for probabilistic roadmap methods”. In: *Robotics and Automation, 1998. Proceedings. ICRA '98. IEEE International Conference on*. Vol. 1, pp. 630–637.
- Au, Wesley, Chao Chen, and Hoam Chung (2012). “Path planning of planar parallel mmechanism using global workspace road maps”. In: *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 4, pp. 1269–1276.
- Au, Wesley, Hoam Chung, and Chao Chen (2014). “Generation of the global workspace roadmap of the 3-RPR using rotary disk search”. In: *Mechanism and Machine Theory* 78, pp. 248–262.
- Balaguer, Benjamin and Stefano Carpin (2012). “Bimanual regrasping from unimanual machine learning”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3264–3270.
- Barry, Jennifer, Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez (2013). “Manipulation with multiple action types”. In: *Experimental Robotics*. Vol. 88. Springer Tracts in Advanced Robotics. Springer International Publishing, pp. 531–545.
- Bersch, Christian, Benjamin Pitzer, and Sören Kammel (2011). “Bimanual robotic cloth manipulation for laundry folding”. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 1413–1419.

- Bicchi, Antonio (1995). “On the closure properties of robotic grasping”. In: *The International Journal of Robotics Research* 14.4, pp. 319–334.
- Bobrow, James E, Steven Dubowsky, and J S Gibson (1985). “Time-optimal control of robotic manipulators along specified paths”. In: *The International Journal of Robotics Research* 4.3, pp. 3–17.
- Bretl, Timothy and Sanjay Lall (2008). “Testing static equilibrium for legged robots”. In: *IEEE Transactions on Robotics* 24.4, pp. 794–807.
- Broquère, Xavier, Daniel Sidobre, and Ignacio Herrera-Aguilar (2008). “Soft motion trajectory planner for service manipulator robot”. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 2808–2813.
- Burdick, Joel W (1989). “On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds”. In: *Advanced Robotics: 1989*. Springer Berlin Heidelberg. Chap. 3.
- Cambon, Stéphane, Rachid Alami, and Fabien Gravot (2009). “A hybrid approach to intricate motion, manipulation and task planning”. In: *The International Journal of Robotics Research* 28.1, pp. 104–126.
- Caron, Stéphane, Quang-Cuong Pham, and Yoshihiko Nakamura (2015a). “Leveraging cone double description for multi-contact Stability of humanoid robots with applications to statics and dynamics”. In: *Proceedings of Robotics: Science and Systems*.
- Caron, Stéphane, Quang-Cuong Pham, and Yoshihiko Nakamura (2015b). “Stability of surface contacts for humanoid robots: Closed-form formulae of the Contact Wrench Cone for rectangular support areas”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 5107–5112.
- Caron, Stéphane, Quang-Cuong Pham, and Yoshihiko Nakamura (2017). “Completeness of randomized kinodynamic planners with state-based steering”. In: *Robotics and Autonomous Systems* 89, pp. 85–94.
- Cederquist, Jan and Sara Negri (1996). “A constructive proof of the Heine-Borel covering theorem for formal reals”. In: *Types for Proofs and Programs: International Workshop, TYPES '95 Torino, Italy, June 5–8, 1995 Selected Papers*. Ed. by Berardi, Stefano and Mario Coppo. Vol. 1158. Lecture Notes in Computer Science (LNCS). Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 62–75. ISBN: 978-3-540-70722-6. DOI: 10.1007/3-540-61780-9_62. URL: http://dx.doi.org/10.1007/3-540-61780-9_62.
- CGAL: Computational Geometry Algorithms Library. URL: <http://www.cgal.org>.

- Chan, T M (1996). “Optimal output-sensitive convex hull algorithms in two and three dimensions”. In: *Discrete & Computational Geometry* 16.4, pp. 361–368.
- Chazelle, Bernard (1993). “An optimal convex hull algorithm in any fixed dimension”. In: *Discrete & Computational Geometry* 10.4, pp. 377–409.
- Cho, Kyoungrae, Munsang Kim, and Jae-Bok Song (2003). “Complete and rapid re-grasp planning with look-up table”. In: *Journal of Intelligent and Robotic Systems* 36.4, pp. 371–387.
- Cohen, Benjamin, Sachin Chitta, and Maxim Likhachev (2013). “Single- and dual-arm motion planning with heuristic search”. In: *The International Journal of Robotics Research* 33.2, pp. 305–320.
- Cortés, Juan, Thierry Siméon, and Jean-Paul Laumond (2002). “A random loop generator for planning the motions of closed kinematic chains using PRM methods”. In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*.
- Diankov, Rosen (2010). “Automated Construction of Robotic Manipulation Programs”. Available at http://www.programmingvision.com/rosen_diankov_thesis.pdf. PhD thesis. Carnegie Mellon University, Robotics Institute.
- Diankov, Rosen, Nathan Ratliff, Dave Ferguson, Siddhartha S Srinivasa, and James J Kuffner (2008). “BiSpace planning: concurrent multi-space exploration”. In: *Proceedings of Robotics: Science and Systems IV*. Vol. 63, pp. 159–166.
- Donald, Bruce, Patrick Xavier, John Canny, and John Reif (1993). “Kinodynamic motion planning”. In: *Journal of the ACM (JACM)* 40, pp. 1048–1066.
- Edsinger, Aaron and Charles C Kemp (2008). “Two Arms Are Better Than One: A Behavior Based Control System for Assistive Bimanual Manipulation”. In: *Recent Progress in Robotics: Viable Robotic Service to Human*. Vol. 370. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, pp. 345–355.
- Erkorkmaz, Kaan and Yusuf Altintas (2001). “High speed CNC system design. Part I: jerk limited trajectory generation and quintic spline interpolation”. In: *International Journal of Machine Tools and Manufacture* 41.9, pp. 1323–1345.
- Ezair, Ben, Tamir Tassa, and Zvi Shiller (2014). “Planning High Order Trajectories with General Initial and Final Conditions and Asymmetric Bounds”. In: *The International Journal of Robotics Research* 33.6, pp. 898–916.
- Fukuda, Komei and Alain Prodon (1996). “Double description method revisited”. In: *Combinatorics and Computer Science*. Vol. 1120. Lecture Notes in Computer Science (LNCS). Springer, Berlin, Heidelberg, pp. 91–111.

- Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie Pack Kaelbling (2015). “FFRob: an efficient heuristic for task and motion planning”. In: *Algorithmic Foundations of Robotics XI*. Vol. 107. Springer International Publishing, pp. 179–195.
- Gasparetto, Alessandro and Vanni Zanutto (2008). “A technique for time-jerk optimal planning of robot trajectories”. In: *Robotics and Computer-Integrated Manufacturing* 24.3, pp. 415–426.
- Geraerts, R and Mark H Overmars (2007). “Creating high-quality paths for motion planning”. In: *The International Journal of Robotics Research* 26.8, pp. 845–863.
- Gharbi, M, Juan Cortés, and Thierry Siméon (2008). “A sampling-based path planner for dual-arm manipulation”. In: *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, pp. 383–388.
- Gharbi, Mokhtar, Juan Cortés, and Thierry Siméon (2009). “Roadmap Composition for Multi-Arm Systems Path Planning”. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 2471–2476.
- Graetz, Georg and Guy Michaels (2015). “Robots at Work”. In: *CEPR Discussion Paper Series*, pp. 1–55. URL: <https://ssrn.com/abstract=2575781>.
- Han, Li and Nancy M Amato (2000). “A kinematic-based probabilistic roadmap method for closed chain systems”. In: *International Workshop on the Algorithmic Foundations of Robotics*, pp. 233–246.
- Harada, Kensuke, Tokuo Tsuji, and Jean-Paul Laumond (2014). “A manipulation motion planner for dual-arm industrial manipulators”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 928–934.
- Harada, Kensuke, Kris Hauser, Timothy Bretl, and Jean-Claude Latombe (2006). “Natural motion generation for humanoid robots”. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 833–839.
- Haschke, Robert, Erik Weitnauer, and Helge Ritter (2008). “On-line planning of time-optimal, jerk-limited trajectories”. In: *Intelligent Robots and Systems (IROS), 2008 IEEE/RSJ International Conference on*, pp. 3248–3253.
- Hauser, Kris (2014). “Fast interpolation and time-optimization with contact”. In: *The International Journal of Robotics Research* 33.9, pp. 1231–1250.
- Hauser, Kris and Jean-Claude Latombe (2010). “Multi-modal motion planning in non-expansive spaces”. In: *The International Journal of Robotics Research* 29.7, pp. 897–915.
- Hauser, Kris and Victor Ng-Thow-Hing (2010). “Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts”. In: *Robotics and Au-*

- tation, 2010. *Proceedings. ICRA '10. IEEE International Conference on*, pp. 2493–2498.
- (2011). “Randomized multi-modal motion planning for a humanoid robot manipulation task”. In: *The International Journal of Robotics Research* 30.6, pp. 678–698.
- Hsu, David, Jean-Claude Latombe, and Rajeev Motwani (1997). “Path planning in expansive configuration spaces”. In: *Robotics and Automation, 1997. Proceedings. ICRA '97. IEEE International Conference on*. Vol. 3, pp. 2719–2726.
- Hsu, David, Robert Kindel, Jean-Claude Latombe, and Stephen Rock (2002). “Randomized kinodynamic motion planning with moving obstacles”. In: *The International Journal of Robotics Research* 21.3, pp. 233–255.
- Hsu, David, Tingting Jiang, John Reif, and Zheng Sun (2003). “The bridge test for sampling narrow passages with probabilistic roadmap planners”. In: *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*.
- Huang, Q, K Tanie, and S Sugano (1999). “Stability compensation of a mobile manipulator by manipulator motion: Feasibility and planning”. In: *Advanced Robotics* 6.8, pp. 25–40.
- Huebner, Kai, Steffen Ruthotto, and Danica Kragic (2008). “Minimum volume bounding box decomposition for shape approximation in robot grasping”. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 1628–1633.
- Hwang, Myun Joong, Doo Yong Lee, and Seong Youb Chung (2007). “Motion planning of bimanual robot for assembly”. In: *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pp. 240–245.
- Innocenti, C and V Parenti-Castelli (1998). “Singularity-free evolution from one configuration to another in serial and fully-parallel manipulators”. In: *Journal of Mechanical Design* 120.1, pp. 73–79.
- International Federation of Robotics (2016). *Executive Summary World Robotics 2016 Industrial Robots*. URL: https://ifr.org/img/uploads/Executive_Summary_WR_Industrial_Robots_20161.pdf.
- Jaillet, Léonard and Josep M Porta (2013). “Path planning under kinematic constraints by rapidly exploring manifolds”. In: *IEEE Transactions on Robotics* 29, pp. 105–117.
- Kaelbling, Leslie Pack and Tomás Lozano-Pérez (2010). “Hierarchical planning in the now”. In: *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pp. 33–42.

- Kalakrishnan, Mrinal, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal (2011). "STOMP: Stochastic trajectory optimization for motion planning". In: *Robotics and Automation, 2011. Proceedings. ICRA '11. IEEE International Conference on*, pp. 4569–4574.
- Kant, Kamal and Steven W Zucker (1986). "Toward efficient trajectory planning: the path-velocity decomposition". In: *The International Journal of Robotics Research* 5.3, pp. 72–89.
- Karaman, Sertac and Emilio Frazzoli (2011). "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7, pp. 846–894.
- Kavraki, Lydia E, Jean-Claude Latombe, Rajeev Motwani, and Prabhakar Raghavan (1995). "Randomized query processing in robot path planning". In: *STOC '95 Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pp. 353–362.
- Kavraki, Lydia E, Petr Svestka, Jean-Claude Latombe, and Mark H Overmars (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4, pp. 566–580.
- Koga, Yoshihito and Jean-Claude Latombe (1992). "Experiments in dual-arm manipulation planning". In: *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pp. 2238–2245.
- Koga, Yoshihito, Koichi Kondo, James J Kuffner, and Jean-Claude Latombe (1994). "Planning motions with intentions". In: *SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 395–408.
- Kröger, Torsten, Adam Tomiczek, and Friedrich M Wahl (2006). "Towards on-line trajectory computation". In: *Intelligent Robots and Systems (IROS), 2006 IEEE/RSJ International Conference on*, pp. 736–741.
- Kröger, Torsten and Friedrich M Wahl (2010). "Online Trajectory Generation: Basic Concepts for Instantaneous Reactions to Unforeseen Events". In: *IEEE Transactions on Robotics* 26.1, pp. 94–111.
- Krontiris, Athanasios and Kostas E Bekris (2015). "Dealing with difficult instances of object rearrangement". In: *Proceedings of Robotics: Science and Systems*.
- Kuffner, James J (2004). "Effective sampling and distance metrics for 3D rigid body path planning". In: *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. Vol. 4, pp. 3993–3998.

- Kuffner, James J, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue (2002). “Dynamically-stable motion planning for humanoid robots”. In: *Autonomous Robots* 12.1, pp. 105–118.
- Kunz, Tobias and Mike Stilman (2014). “Probabilistically Complete Kinodynamic Planning for Robot Manipulators with Acceleration Limits”. In: *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*, pp. 3713–3719.
- Kyriakopoulos, Konstantinos J and George N Saridis (1988). “Minimum jerk path generation”. In: *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. Vol. 1, pp. 364–369.
- Lamiriaux, Florent and Joseph Mirabel (2015). *HPP: a new software framework for manipulation planning*. URL: <https://hal.archives-ouvertes.fr/hal-01138118>.
- LaValle, Steven M (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. Iowa State University.
- (2006). *Planning algorithms*. Cambridge University Press.
- LaValle, Steven M and James J Kuffner (2000). *Rapidly-exploring random trees: Progress and prospects*. Algorithmic and Computational Robotics: New Directions.
- (2001). “Randomized kinodynamic planning”. In: *The International Journal of Robotics Research* 20.5, pp. 378–400.
- Lavalle, Steven M and James J Kuffner (2001). “Rapidly-exploring random trees: Progress and prospects”. In: *Algorithmic and Computational Robotics: New Directions*. Ed. by Donald, B R, Kevin M Lynch, and Daniela Rus. A. K. Peters, pp. 293–308.
- Lee, Gilwoo, Tomás Lozano-Pérez, and Leslie Pack Kaelbling (2015). “Hierarchical planning for multi-contact non-prehensile manipulation”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 264–271.
- Lertkultanon, Puttichai** and Quang-Cuong Pham (2014). “Dynamic non-prehensile object transportation”. In: *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*, pp. 1392–1397.
- (2015). “A single-query manipulation planner”. In: *IEEE Robotics and Automation Letters* 1.1, pp. 198–205. DOI: 10.1109/LRA.2015.2513731.
- (2016). “Time-optimal parabolic interpolation with velocity, acceleration, and minimum-switch-time constraints”. In: *Advanced Robotics* 30.17–18, pp. 1095–1110.
- (2017). “A certified-complete bimanual manipulation planner”. In: *IEEE Transactions on Automation Science and Engineering*. Conditionally Accepted.

- Li, Tsai-Yen and Jean-Claude Latombe (1997). “On-line manipulation planning for two robot arms in a dynamic environment”. In: *The International Journal of Robotics Research* 16.2, pp. 144–167.
- Lindemann, Stephen R and Steven M LaValle (2003). “Incremental low-discrepancy lattice methods for motion planning”. In: *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, pp. 2920–2927.
- Liu, Honghai and Jian Dai (2003). “An approach to carton-folding trajectory planning using dual robotic fingers”. In: *Robotics and Autonomous Systems* 42.1, pp. 47–63.
- Liu, Steven (2002). “An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators”. In: *Advanced Motion Control, 2002. 7th International Workshop on*, pp. 365–370.
- Lozano-Pérez, Tomás, Joseph L Jones, Emmanuel Mazer, Patrick A O'Donnell, W Eric L Grimson, Pierre Tournassoud, and Alain Lanusse (1987). “HaHand: a robot system that recognizes, plans, and manipulates”. In: *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, pp. 843–849.
- Luo, Jingru and Kris Hauser (2014). “An empirical study of optimal motion planning”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 1761–1768.
- Macfarlane, Sonja and Elizabeth A Croft (2003). “Jerk-bounded manipulator trajectory planning: design for real-time applications”. In: *IEEE Transactions on Robotics and Automation* 19, pp. 42–52.
- Maitin-Shepard, Jeremy, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel (2010). “Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2308–2315.
- McMahon, Troy, Shawna Thomas, and Nancy M Amato (2014). “Sampling Based Motion Planning with Reachable Volumes: Application to Manipulators and Closed Chain Systems”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 3705–3712.
- Miller, Andrew T, Steffen Knoop, Henrik I Christensen, and Peter K Allen (2003). “Automatic grasp planning using shape primitives”. In: *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, pp. 1824–1829.
- Mirabel, Joseph, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylène Campana, Nicolas Mansard, and Florent Lamiraux (2016). “HPP: A new software

- for constrained motion planning”. In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 383–389.
- Munkres, James Raymond (2000). *Topology*. 2nd ed. Prentice Hall.
- Nguyen, Huy and Quang-Cuong Pham (2016). “Time-optimal path parameterization of rigid-body motions: applications to spacecraft reorientation”. In: *Journal of Guidance, Control, and Dynamics* 39.7, pp. 1667–1671.
- Nielsen, Christian L and Lydia E Kavraki (2000). “A two level fuzzy PRM for manipulation planning”. In: *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*. Vol. 3, pp. 1716–1721.
- Ota, Jun (2004). “Rearrangement of multiple movable objects”. In: *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, pp. 1962–1967.
- Pai, Dinesh K and Ming C Leu (1992). “Genericity and singularities of robot manipulators”. In: *IEEE Transactions on Robotics and Automation* 8.5, pp. 545–559.
- Park, Chonhyon, Jia Pan, and Dinesh Manocha (2012). “ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments”. In: *International Conference on Automated Planning and Scheduling*, pp. 207–215.
- Park, Frank Chongwoo (1995). “Distance metrics on the rigid-body motions with applications to mechanism design”. In: *Journal of Mechanical Design* 117.1, pp. 48–54.
- Park, Frank Chongwoo and Bahram Ravani (1997). “Smooth invariant interpolation of rotations”. In: *ACM Transactions on Graphics* 16.3, pp. 277–295.
- Pfeiffer, Friedrich and Rainer Johanni (1987). “A concept for manipulator trajectory planning”. In: *IEEE Journal of Robotics and Automation* 3.2, pp. 115–123.
- Pflueger, Max and Gaurav S Sukhatme (2015). “Multi-step planning for robotic manipulation”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 2496–2501.
- Pham, Quang-Cuong (2013). “Characterizing and Addressing Dynamic Singularities in the Time-Optimal Path Parameterization Algorithm”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 2357–2363.
- (2014). “A general, fast, and robust implementation of the time-optimal path parameterization algorithm”. In: *IEEE Transactions on Robotics* 30, pp. 1533–1540. doi: 10.1109/TR0.2014.2351113.
- Pham, Quang-Cuong, Stéphane Caron, and Yoshihiko Nakamura (2013). “Kinodynamic Planning in the Configuration Space via Admissible Velocity Propagation”. In: *Proceedings of Robotics: Science and Systems*.

- Pham, Quang-Cuong and Olivier Stasse (2015). “Time-optimal path parameterization for redundantly actuated robots: a numerical integration approach”. In: *IEEE/ASME Transactions on Mechatronics* 20.6, pp. 3257–3263.
- Pham, Quang-Cuong, Stéphane Caron, **Puttichai Lertkultanon**, and Yoshihiko Nakamura (2016). “Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots”. In: *The International Journal of Robotics Research* 36.1, pp. 44–67.
- Piazzi, Aurelio and Antonio Visioli (2000). “Global minimum-jerk trajectory planning for robot manipulators”. In: *IEEE Transactions on Industrial Electronics* 47.1, pp. 140–149.
- Porta, Josep M, Léonard Jaillet, and Oriol Bohigas (2012). “Randomized path planning on manifolds based on higher-dimensional continuation”. In: *The International Journal of Robotics Research* 31.2, pp. 201–215.
- Posa, Michael and Russ Tedrake (2013). “Direct trajectory optimization of rigid body dynamical systems through contact”. In: *Algorithmic Foundations of Robotics X*. Ed. by Frazzoli, Emilio, Tomás Lozano-Pérez, Nicholas Roy, and Daniela Rus. Vol. 86. Springer Tracts in Advanced Robotics. Springer, Berlin, Heidelberg, pp. 527–542.
- Prete, Andrea Del, Steve Tonneau, and Nicolas Mansard (2016). “Fast algorithms to test robust static equilibrium for legged robots”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 1601–1607.
- Primrose, E J F (1986). “On the input-output equation of the general 7R-mechanism”. In: *Mechanism and Machine Theory* 21.6, pp. 509–510.
- Röhrdanz, F and Friedrich M Wahl (1997). “Generating and evaluating regrasp operations”. In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 3, pp. 2013–2018.
- Saut, Jean-Philippe, Mokhtar Gharbi, Juan Cortés, Daniel Sidobre, and Thierry Siméon (2010). “Planning pick-and-place tasks with two-hand regrasping”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*.
- Seraji, Homayoun (1994). “Adaptive admittance control: an approach to explicit force control in compliant motion”. In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 2705–2712.
- Shauri, Ruhizan Liza Ahmad and Kenzo Nonami (2011). “Assembly manipulation of small objects by dual-arm manipulator”. In: *Assembly Automation* 31.3, pp. 263–274.

- Shimoga, Karun B (1996). “Robot grasp synthesis algorithms: A survey”. In: *The International Journal of Robotics Research* 15.3, pp. 230–266.
- Shin, Kang G and Neil D McKay (1985). “Minimum-time control of robotic manipulators with geometric path constraints”. In: *IEEE Transactions on Automatic Control* 30.6, pp. 531–541.
- (1986a). “A dynamic programming approach to trajectory planning of robotic manipulators”. In: *IEEE Transactions on Automatic Control* 31.6, pp. 491–500.
 - (1986b). “Minimum-time trajectory planning for industrial robots with general torque constraints”. In: *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*. Vol. 3.
- Shkolnik, Alexander, Metthew Walter, and Russ Tedrake (2009). “Reachability-guided sampling for planning under differential constraints”. In: *Robotics and Automation, 2009. Proceedings. ICRA '09. IEEE International Conference on*, pp. 2859–2865.
- Siciliano, Bruno, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo (2009). “Robotics: Modelling, Planning and Control”. In: 1st ed. Advanced Textbooks in Control and Signal Processing. Springer London. Chap. Differential Kinematics and Statics, pp. 105–160.
- Siméon, Thierry, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani (2004). “Manipulation planning with probabilistic roadmaps”. In: *The International Journal of Robotics Research* 23.7–8, pp. 729–746.
- Simon, Dan and Can Isik (1993). “A trigonometric trajectory generator for robotic arms”. In: *International Journal of Control* 57.3, pp. 505–517.
- Slotine, Jean-Jacques E and Hyun S Yang (1989). “Improving the efficiency of time-optimal path-following algorithms”. In: *IEEE Transactions on Robotics and Automation* 5.1, pp. 118–124.
- Spong, Mark W and Mathukumalli Vidyasagar (2008). *Robot Dynamics and Control*. John Wiley & Sons.
- Stilman, Mike, Jan-Ullrich Schamburek, James J Kuffner, and Tamim Asfour (2007). “Manipulation planning among movable obstacles”. In: *Robotics and Automation, 2007 IEEE International Conference on*, pp. 3327–3332.
- Stoeter, Sascha A, Stephen Voss, Nikolaos P Papanikolopoulos, and Heiko Mosemann (1999). “Planning of regrasp operations”. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 1.
- Suárez-Ruiz, Francisco and Quang-Cuong Pham (2016). “A framework for fine robotic assembly”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 421–426.

- Suleiman, Wael, Fumio Kanehiro, Eiichi Yoshida, Jean-Paul Laumond, and André Monin (2010). "Time parameterization of humanoid-robot paths". In: *IEEE Transactions on Robotics* 26.3, pp. 458–468.
- Surdilovic, D, Y Yakut, T-M Nguyen, X B Pham, A Vick, and R Martin-Martin (2010). "Compliance control with dual-arm humanoid robots: Design, planning and programming". In: *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pp. 275–281.
- Tang, Xinyu, Shawna Thomas, Phillip Coleman, and Nancy M Amato (2010). "Reachable distance space: efficient sampling-based planning for spspatial constrained systems". In: *The International Journal of Robotics Research* 29.7, pp. 916–934.
- Thomas, Federico (1993). "The self-motion manifolds of the N-bar mechanism". In: *Computational Kinematics*. Ed. by Angeles, Jorge, Günter Hommel, and Peter Kovács. Vol. 28. Springer Netherlands, pp. 95–106.
- Thompson, Stuart E and Rajnikant V Patel (1987). "Formulation of joint trajectories for industrial robots using B-splines". In: *IEEE Transactions on Industrial Electronics* IE-34.2, pp. 192–199.
- Tournassoud, Pierre, Tomás Lozano-Pérez, and Emmanuel Mazer (1987). "Regrasping". In: *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*. Vol. 4, pp. 1924–1928.
- Trinkle, J C and R James Milgram (2002). "Complete path planning for closed kinematic chains with spherical joints". In: *The International Journal of Robotics Research* 21.9, pp. 773–789.
- Vahrenkamp, Nikolaus, Dmitry Berenson, Tamim Asfour, James J Kuffner, and Rüdiger Dillmann (2009a). "Humanoid motion planning for dual-arm manipulation and re-grasping task". In: *Intelligent Robots and Systems (IROS), 2009 IEEE/RSJ International Conference on*, pp. 2464–2470.
- Vahrenkamp, Nikolaus, Christian Böge, Kai Welke, Tamim Asfour, Jürgen Walter, and Rüdiger Dillmann (2009b). "Visual servoing for dual arm motions on a humanoid robot". In: *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*.
- Vahrenkamp, Nikolaus, Martin Do, Tamim Asfour, and Rüdiger Dillmann (2010). "Integrated Grasp and Motion Planning". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2883–2888.
- van den Berg, Jur, Stephen Miller, Ken Goldberg, and Pieter Abbeel (2011). "Gravity-based robotic cloth folding". In: *Algorithmic Foundations of Robotics IX*. Ed. by

- Hsu, David, Volkan Isler, Jean-Claude Latombe, and Ming C Lin. Vol. 68. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, pp. 409–424.
- Verscheure, Diederik, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl (2009). “Time-optimal path tracking for robots: A convex optimization approach”. In: *IEEE Transactions on Automatic Control* 54.10, pp. 2318–2327.
- Visser, Matt (2004). “Jerk, snap and the cosmological equation of state”. In: *Classical and Quantum Gravity* 21.11, pp. 2603–2615.
- Vukobratovic, M, B Borovac, and D Surdilovic (2001). “Zero-moment point–proper interpretation and new applications”. In: *Humanoid Robots (Humanoids), 2001 IEEE-RAS International Conference on*.
- Wan, Weiwei and Kensuke Harada (2015a). “Divide-and-conquer manipulation planning by lazily searching a weighted two-layer manipulation graph”. In: *Advanced Robotics and its Social Impacts (ARSO), 2015 IEEE International Workshop on*, pp. 1–8.
- (2015b). “Reorientating Objects with a Gripping Hand and a Table Surface”. In: *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pp. 101–106.
- (2016). “Developing and comparing single-arm and dual-arm regrasp”. In: *IEEE Robotics and Automation Letters* 1.1, pp. 234–250.
- Wan, Weiwei, Matthew M Mason, Rui Fukui, and Yasuo Kuniyoshi (2015). “Improving regrasp algorithms to analyze the utility of work surfaces in a workcell”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 4326–4333.
- Wilfong, Gordon Thomas (1988). “Motion planning in the presence of movable obstacles”. In: *SCG’88 Proceedings of the fourth annual symposium on Computational geometry*, pp. 279–288.
- Xian, Zhou, **Puttichai Lertkultanon**, and Quang-Cuong Pham (2017). “Closed-chain manipulation of large objects by multi-arm robotic systems”. In: *IEEE Robotics and Automation Letters* 2.4, pp. 1832–1839. doi: 10.1109/LRA.2017.2708134.
- Yakey, Jeffery H, Steven M LaValle, and Lydia E Kavraki (2001). “Randomized path planning for linkages with closed kinematic chains”. In: *IEEE Transactions on Robotics and Automation* 17.6, pp. 951–958.
- Yamanobe, Natsuki and Kazuyuki Nagata (2010). “Grasp planning for everyday objects based on primitive shape representation for parallel jaw grippers”. In: *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pp. 1565–1570.

- Zacharias, Franziska, Daniel Leidner, Florian Schmidt, Christoph Borst, and Gerd Hirzinger (2010). “Exploiting structure in two-armed manipulation tasks for humanoid robots”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 5446–5452.
- Zheng, Yuan F and Ming Z Chen (1994). “Trajectory planning for two manipulators to deform flexible beams”. In: *Robotics and Autonomous Systems* 12.1–2, pp. 55–67.
- Žlajpah, Leon (1996). “On time optimal path control of manipulators with bounded joint velocities and torques”. In: *Robotics and Automation, 1996. Proceedings. ICRA '96, IEEE International Conference on*. Vol. 2, pp. 1572–1577.
- Zucker, Matt, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa (2013). “CHOMP: Covariant Hamiltonian optimization for motion planning”. In: *The International Journal of Robotics Research* 32.9–10, pp. 1164–1193.

Publications from this Thesis

Submitted to Journals

- **Lertkultanon, Puttichai** and Quang-Cuong Pham (2017). “A certified-complete bimanual manipulation planner”. In: *IEEE Transactions on Automation Science and Engineering*. Conditionally Accepted

Journals

- Xian, Zhou, **Puttichai Lertkultanon**, and Quang-Cuong Pham (2017). “Closed-chain manipulation of large objects by multi-arm robotic systems”. In: *IEEE Robotics and Automation Letters* 2.4, pp. 1832–1839. DOI: 10.1109/LRA.2017.2708134
- **Lertkultanon, Puttichai** and Quang-Cuong Pham (2016). “Time-optimal parabolic interpolation with velocity, acceleration, and minimum-switch-time constraints”. In: *Advanced Robotics* 30.17–18, pp. 1095–1110
- Pham, Quang-Cuong, Stéphane Caron, **Puttichai Lertkultanon**, and Yoshihiko Nakamura (2016). “Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots”. In: *The International Journal of Robotics Research* 36.1, pp. 44–67
- **Lertkultanon, Puttichai** and Quang-Cuong Pham (2015). “A single-query manipulation planner”. In: *IEEE Robotics and Automation Letters* 1.1, pp. 198–205. DOI: 10.1109/LRA.2015.2513731

International Conferences

- **Lertkultanon, Puttichai** and Quang-Cuong Pham (2014). “Dynamic non-prehensile object transportation”. In: *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*, pp. 1392–1397