

Leveraging Mechanics for Multi-step Robotic Manipulation Planning

by

Rachel Holladay

B.S., Carnegie Mellon University (2017)

M.S., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

© 2024 Rachel Holladay. This work is licensed under a [CC BY-NC-ND 4.0](#) license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Rachel Holladay
Department of Electrical Engineering and Computer Science
August 21, 2024

Certified by: Tomás Lozano-Pérez
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by: Alberto Rodriguez
Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: Leslie A. Kolodziejcki
Professor of Electrical Engineering and Computer Science
Chair, Department Committee for Graduate Students

Leveraging Mechanics for Multi-step Robotic Manipulation Planning

by

Rachel Holladay

Submitted to the Department of Electrical Engineering and Computer Science
on August 21, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

This thesis focuses on enabling robots to robustly perform complex, multi-step manipulation tasks, like chopping vegetables or wielding a wrench. Completing such tasks requires a robot to plan and execute long sequences of actions, where each action involves many connected, discrete and continuous choices that are critically impacted by constraints relating to force, motion and contact. To tackle this, this thesis contributes models and algorithms that exploit the physics and geometry of the world in order to address the dual challenges of long-horizon decision-making and acting under uncertainty. We apply this in the context of three domains: in-hand manipulation, forceful manipulation and briefly-dynamic manipulation.

First, to reorient a grasped object, we develop a sampling-based motion planner to generate sequences of pushes that slide the object in-hand. We derive an abstraction for pushing to enable the planner to reason about frictional constraints. Second, we focus on forceful manipulation tasks, such as opening a childproof medicine bottle or twisting a nut on a bolt, where the robot’s planning choices are impacted by the need to exert force. We define constraints that explicitly consider torque and frictional limits and integrate these into an existing task and motion planning framework. We leverage cost-sensitive planning to enable the robot to generate plans that are robust to uncertainty in the physical parameters. Finally, we frame planning with dynamic actions, like shoveling or toppling, as requiring the robot to reason about both action uncertainty and potential dead ends. We learn a simple action model and formulate a sample-based manipulation planner that guards against dead ends in the face of uncertainty. Throughout this thesis, we validate the practical applicability of our model-based approaches by evaluating them on real robots.

Thesis supervisor: Tomás Lozano-Pérez

Title: Professor of Computer Science and Engineering

Thesis supervisor: Alberto Rodriguez

Title: Associate Professor of Mechanical Engineering

Acknowledgements

It has been my absolute honor and joy to have Alberto Rodriguez and Tomás Lozano-Pérez as my doctoral advisors. You two are the manipulation and mentoring dream team, who have reshaped every aspect of the way I think about and conduct research. I am lucky to have been a very happy graduate student and that is, in part, a product of having such wonderful advisors. I aspire to live up to the example you both set.

I am grateful to have had the support and advice of Professor Leslie Kaelbling, who has served as a role model, a mentor, a thesis committee member and, in too many instances to count, a real-life *deus ex machina*. I would like to thank Professor Dmitry Berenson for serving on my thesis committee and providing a wealth of valuable feedback. Thank you Professor Russ Tedrake for always keeping an open door and kind ear for me and teaching me so much about teaching.

My PhD was indelibly shaped by the gift of being a member of both the MCube (Manipulation and Mechanisms at MIT) Lab and the LIS (Learning and Intelligent Systems) Group. From the MCube Lab, thank you to Anthony, Antonia, Daolin, Elliott, Eric, Francois, Ian, Maria, Neel, Nikhil, Nima, Orion, Peter, Rebecca, Sangwoon, Siyuan and Yen-Chen. From the LIS Group, thank you to Aidan, Aditya, Ari, Beomjoon, Caelan, Caris, Clement, Ferran, Gustavo, Jiayuan, Jorge, Nishanth, Rohan, Sahit, Tom, Willie, Will, Yilun, Yoon, Zelda, Zi and Xiaolin. I've had the joy of having *many* labmates!

This PhD would not have been possible without the patient mentorship of both Caelan and Nikhil. I could not have asked for better teachers in task and motion planning and mechanics (two of the backbones of this thesis!) and I am honored to be your disciple and academic sister. I would also like to give a special thanks to Ian and Elliott for helping me with many fabrication tasks throughout my PhD and always taking the time to teach me. I am very grateful to the students who trusted me to serve as their mentor: Anika, Ashwin, Dani, Greg, Rachel and Tarang. Each of you brought your own unique and fun spin on manipulation (Dynamics! Grasping! Hinges! BOP the FROG! Table Setting! Dicing!) and taught me so much - thank you. Mentoring was a highlight of my PhD. Thank you to Lisa Mayer and Teresa Cataldo for your kindness and enthusiastic facilitation of research and fun.

Thank you to TIG and the EECS Graduate Office for their help throughout the years.

I am very grateful to my friends who took this doctoral journey with me and provided support, laughter, comfort and joy, particularly Lilly, Nili and Vibhaa. To, Mike, Marlyse and Liane, you all were my rock throughout graduate school and I could not have gotten through this without you. Fab 4! To Jiatao, I am supremely lucky to have you in my life and I am excited for our many adventures to come - 我爱你.

Most importantly, I am indebted to my family for their love, support and guidance. My three siblings, Ben, Annie and Sam, are my biggest champions and have always believed that I could succeed (even when I argued the opposite). I aim to be as awesome as they are. My parents, Wendy and Ken, have provided me with every privilege they could, although being their daughter is undoubtedly the greatest privilege. My mother was my very first technical mentor and continues to instill a strong drive within me. It is my joy to receive a doctorate from the same school as my dad, forty-nine years later. I hope I have made my parents proud. This thesis is dedicated to them.

Funding

This work was supported by the National Robotics Initiative IIS-1637753, NSF Grant 2214177, the Boston Dynamics Artificial Intelligence Institute, the MIT Merrill Lynch Fellowship and the NSF Graduate Research Fellowship.

Contents

1	<i>Introduction</i>	17
2	<i>Planar In-Hand Manipulation via Motion Cones</i>	21
	2.1 <i>Introduction</i>	21
	2.2 <i>Related Work</i>	25
	2.3 <i>Mechanics of Planar Pushing</i>	26
	2.4 <i>Motion Cones for Planar Pushing</i>	31
	2.5 <i>Computation and Experimental Validation of Motion Cone</i>	36
	2.6 <i>Planning In-Hand Manipulations via Motion Cones</i>	40
	2.7 <i>Regrasp Examples and Experimental Results</i>	43
	2.8 <i>Robust In-Hand Manipulations via Motion Cones</i>	46
	2.9 <i>Discussion</i>	51
3	<i>Robust Planning for Multi-stage Forceful Manipulation</i>	53
	3.1 <i>Introduction</i>	53
	3.2 <i>Related Work</i>	55
	3.3 <i>Problem Domain</i>	58
	3.4 <i>Approach</i>	60
	3.5 <i>Forceful Kinematic Chain</i>	62
	3.6 <i>PDDLStream</i>	66
	3.7 <i>Incorporating Force into Planning</i>	72
	3.8 <i>Empirical Evaluation</i>	77
	3.9 <i>Discussion</i>	83

4	<i>Briefly-Dynamic Manipulation with Uncertainty and Dead Ends</i>	85
4.1	<i>Introduction</i>	85
4.2	<i>Related Work</i>	87
4.3	<i>Problem Domain</i>	89
4.4	<i>GUARD: Algorithmic Overview</i>	92
4.5	<i>GUARD: Action Regions</i>	93
4.6	<i>GUARD: Danger Zones</i>	95
4.7	<i>Theoretical Guarantees</i>	97
4.8	<i>GUARD: Search</i>	98
4.9	<i>Experiments</i>	100
4.10	<i>Discussion</i>	102
5	<i>Conclusion</i>	105
5.1	<i>Future Directions</i>	106
5.2	<i>Concluding Remarks</i>	107
	<i>Bibliography</i>	109
	<i>Appendix</i>	119
A	<i>Implementation Details</i>	119
B	<i>Further Domain Specification</i>	120
	<i>Index</i>	127

List of Figures and Tables

- 1.1 The goal of this thesis is to enable robots to complete a wide range of multi-step manipulation tasks. 18
- 1.2 Complex manipulation tasks, like opening a push-and-twist childproof medicine bottle, require robots to reason over many physically-complex actions. Here, the robot wants to use a tool (outlined in black) but because the tool is surrounded by other objects, the robot can only reach the top of the tool. The robot then leverages the environment to create a more secure grasp, thus allowing the robot to use the tool to exert the necessary force to uncap the bottle. 19
- 2.1 (top) Example *friction cone* and *motion cone* of an object moving in the vertical plane. The pusher can move the object along any direction $[V_x, V_z, \omega_y]$ inside the motion cone. (bottom) A plan via motion cones. Motion cones capture local reachability. A path in the tree of motion cones generates a pushing strategy to move an object. 21
- 2.2 Manipulating a T-shaped object in a parallel-jaw grasp by pushing it against features in the environment. The manipulation is shown from a side view. 22
- 2.3 (left) An object held in a parallel-jaw gripper is pushed against a fixed feature in the environment. (right) The planner models this interaction as if the object is held by a fixed gripper and pushed by a moving environment, i.e. a pusher. 24
- 2.4 Pushing an object (a) on a horizontal surface, (b) on an inclined surface, (c) in a grasp in the gravity plane, and (d) in a grasp in a tilted plane 25
- 2.5 (Left) Ellipsoidal approximation of the limit surface at the finger contact. For a wrench w_s on the boundary of the limits surface, the twist at the contact v_s is normal to the limit surface. (Right) The wrench w_s and the corresponding object motion is shown in the manipulation scene. 29
- 2.6 (Left) The line pusher contact in this example is modelled with two point contacts. The pusher contact frames and the object frame are drawn in the figure. (Right) The generalized friction cone of the pusher is the object frame representation of the set of forces the two point contacts can offer collectively. 30

- 2.7 To make a push inside the gravity-free motion cone also stable in a scenario with gravity, the unit grasp wrench can be scaled such that the net pusher wrench required for the desired push falls inside/on the generalized friction cone of the pusher. 33
- 2.8 Graphical illustration of the motion cone construction procedure applied to two different planar pushing systems: (top) pushing an object on a horizontal plane with a point pusher and (bottom) pushing an object in a grasp in the gravity plane. From left to right, the construction steps portray modelling of the forces involved in the task (Force Modelling), solving for the set of force-balance solutions in wrench-space (Force Resolution), and finally computing the set of object twists corresponding to the force-balancing solutions (Motion Resolution). This set of object twists is the motion cone. 35
- 2.9 The analytically computed wrench cone (\mathbf{W}_s), motion cone (\mathbf{V}_{obj}), and polyhedral approximation to the motion cone ($\bar{\mathbf{V}}_{\text{obj}}$) for the configuration in Fig.2.3 and 45 N grasping force. Note that the surfaces defining the motion cone (\mathbf{V}_{obj}) are curved. 37
- 2.10 Experimental setup used for the data collection for motion cone validation as well as for the regrasp examples in Sec. 2.7 and Sec. 2.8. 40
- 2.11 Since the sampled grasp configuration is outside the motion cone, it projected on the motion cone. The projected configuration is selected to grow the planning tree. 43
- 2.12 Physical properties of the experimental objects 43
- 2.13 Planning times (sec.) for approaches using motion cone, stable check [Chavan-Dafle and Rodriguez \(2018\)](#) and MNCP [Chavan-Dafle and Rodriguez \(2020\)](#) for unit-step propagation 44
- 2.14 Simulation and experimental run for a pushing strategy to regrasp the aluminum object with low friction pushers. In the simulation figure (top), the finger and pusher contacts are shown in green and magenta color respectively. 44
- 2.15 Simulation and experimental run for a pushing strategy to regrasp the aluminum object with high friction pushers. 45
- 2.16 A pushing strategy for $[X, Z, \theta_\gamma]$ regrasp. In simulation, the direction of gravity remains constant in the pusher frame, because in real experiments, the pushers are fixed features in the environment. 45
- 2.17 Simulation and experimental run to manipulate a T-shaped object. Snapshots of the experimental run are shown in Fig.2.2 46

- 2.18 The motion cone shifts towards the gravity-free motion cone as the friction at the gripper increases due to the increased grip force. The intersection of the gravity-free motion cone and the motion cone is shown by the dotted region for the two grip forces. The motions inside the motion cone that are outside the gravity-free motion cone are feasible only by exploiting the gravitational force on the object (in Z direction). For higher friction at the gripper than expected, these motions may no longer be feasible. One of such motions that is shown in red falls outside the motion cone when the grip force is increased. All the motions inside the intersection however, will always be feasible for the higher than expected friction at the gripper. 48
- 2.19 Simulated motion of the object and snapshots of the experimental run for a pushing strategy to offset the object in the grasp using low coefficient pushers. 49
- 2.20 Simulated motion of the object and snapshots of the experimental run for a general regrasp in $[X, Z, \theta_y]$. 50
- 2.21 Simulated motion of the object and snapshots of the experimental run for the T-shaped object. 50
- 2.22 Displacement of the object with respect to the environment for the regrasp action shown in Fig.2.19. As expected, during pushing, the object sticks to the environment and moves by a negligible amount as the fingers slide on it. 51
- 2.23 Displacement of the object with respect to the environment for the regrasp action shown in Fig.2.20. 51
- 3.1 Opening a childproof bottle involves executing a downward-push and twist on the cap, while fixturing the bottle. Our system can reason over a combinatorial number of strategies to accomplish this forceful manipulation task, including push-twisting with various parts of its end effector, push-twisting with a tool (in blue), fixturing with a vise (in grey), fixturing against the table, or fixturing against a high-friction rubber mat (in red). 54
- 3.2 (a) Opening a childproof bottle involves executing a push-twist on the cap, while fixturing the bottle. (b) Twisting a nut requires exerting a torque about the nut, while fixturing the bolt. (c) To cut, the robot first press down vertically and then slices horizontally. The object being cut must be fixtured. 58
- 3.3 To twist a nut on the bolt, the robot can use either its fingers or a spanner (in blue). While twisting, the robot must fixture the beam that the bolt is attached to. Here we show two fixturing strategies: using another robot to grasp the beam and weighing down the beam with a large mass (in green). 59
- 3.4 The robot uses a knife to cut an object, while fixturing the object. Here a vise is used to fixture a cucumber (left) and a banana (right) while they are being cut with a knife (blue). 59

- 3.5 The goal of the robot is to cut the vegetable, using the knife (in blue). The vegetable must be fixtured, which can be achieved using the vise. However, the robot cannot secure the vegetable in the vise because a red block is preventing a collision-free placement. Our system constructs a plan where the robot first picks up the red block and places it on the table, out of the way. The robot can then pick up the vegetable and fixture it in the vise. Next, the robot grasps the knife and uses it to cut the fixtured vegetable. 60
- 3.6 Along each joint of the forceful kinematic chain, we first project the expected wrench into the subspace defined by each joint and then verify if the joint is stable under that wrench. The figure illustrate the wrench limits for each joint: For circular patch contacts, we check the friction force against a limit surface ellipsoidal model and for each robot joint we check against the 1D torque limits. 63
- 3.7 Here we show two possible grasps on the knife as it cuts a vegetable. For each grasp we visualize the corresponding limit surface with the propagated task wrench. The top grasp is not stable, as the wrench lies outside the boundary of the limit surface. In contrast, the bottom grasp, which leverages kinematics to resist the large torque, is stable. 65
- 3.8 Algorithmic Flow of PDDLStream. The samplers are used to certify static facts. These facts, together with the domain, initial facts and goal, serve as input to a PDDL planner, which searches for a plan. If a plan cannot be found, the algorithm generates more certified facts via the samplers. 68
- 3.9 Pick-and-Place Example. A set of facts characterize the state. Here the robot starts at some configuration \mathbf{q} and a graspable pink block \mathbf{o} starts at some pose \mathbf{p} on the surface region \mathbf{r} . 69
- 3.10 Pick-and-Place Example. Each sampler takes as input some values such that those values satisfy some constraint. The samplers (highlighted in green), output either new parameter values that are certified to satisfy some constraint or simply a certification that the inputs satisfy a constraint. Samplers can be conditioned upon each other such that the output of one sampler is the input to another, as shown by the dotted line. 70
- 3.11 Pick-and-Place Example. Expanded view of the PDDLStream search procedure. The state is composed of facts from the initial set of facts (in blue) and the set of certified static facts generated by the samplers (in green). An action is feasible if all of the facts of the preconditions are met. The state is then updated with the resulting effects (in brown). In this example, the first action is to `move` from configuration \mathbf{q}_1 to \mathbf{q}_2 . Having taken this action, the next is to pick up an object \mathbf{o} at pose \mathbf{p}_1 using grasp \mathbf{g}_2 . The result of the search would be the sequence of ground operators, i.e. `[move_free(\mathbf{q}_1 , \mathbf{q}_2 , \mathbf{t}_4), pick(\mathbf{o} , \mathbf{p}_1 , \mathbf{g}_2 , \mathbf{q}_2 , \mathbf{t}_2)]`. 72
- 3.12 By sampling over the friction coefficient, μ , radius of contact r and task wrench, we can assess if a contact is stable in face of uncertainty in those parameters. 77

- 3.13 For each setting, we provide the number of steps for each strategy and the average planning time in seconds (and standard error) over five runs. *: Utilized a higher friction coefficient μ to increase feasibility **: Invalidated shorter strategies to force to planner to find these longer strategies. 78
- 3.14 In opening the childproof bottle, the robot can fixture against the low-friction table (▣), a medium-friction mat (▢) or a high-friction mat (▤). For each cost threshold we run the planner ten times, noting which surface is used. As the cost threshold decreases, the robot is forced to more frequently use higher friction surfaces that are more robust to uncertainty. 79
- 3.15 We evaluate how decreasing the cost threshold impacts what fixturing surface the planner uses in the childproof bottled domain (Fig.3.14). As the cost threshold decreases, the planning time increases. 79
- 3.16 In the nut-twisting domain we consider the trade-off between the grasp cost and the fixturing cost. On the left, at each weight value, we randomly sample, 100 times, the pose of the weight along the beam and the grasp on the weight. Since, at the extremes, some costs evaluate to infinity, we plot the median and a 95% confidence interval. We then demonstrate how the trade off impacts the choices made by the planner by considering an environment in which there are three possible masses, as shown in the center. The robot can fixture using the 2.6kg mass (▣), the 3.5kg mass (▢) or the 4.4kg mass (▤). Without accounting for robustness, the robot chooses any of the masses. When planning robustly, the robot more often picks the medium weight, which balances the trade-off in costs. In both cases we run the planner ten times, noting which weight is used. 80
- 3.17 In the vegetable cutting domain, we show how robust planning leads the planner to select grasps that are closer to the blade of the knife, because doing so creates a smaller torque that the grasp needs to resist. For each cost threshold, the planner is run 10 times and the grasping offset is plotted. An offset of 0 corresponds to a grasp at the butt of the knife. 81
- 3.18 We evaluating slicing success for three grasps (from top to bottom: `side_grasp`, `top_grasp_close` and `top_grasp_far`) across three foods. For the cucumbers and bananas with peel we perform 15 iterations, for the banana without the peel we perform 10. We classify each interaction as a success / partial success / failure. 83
- 4.1 The robot gives a quick shove in order to push the green block across the table. As shown, there are many possible outcomes as a result of this action, including the object toppling over or teetering on the edge of the table. In this work we focus on enabling robots to leverage dynamic actions like this while accounting for uncertainty and irrecoverable outcomes, or dead ends. 86

- 4.2 The lefthand side show an example environment viewed from the side. The environment includes with a robot, surface (the grey table top), a target object (the blue rectangular prism), an obstacle (in black), a glass vase serving as a non-interactable (in purple) and a goal region (highlighted in green). The center shows this same environment from an overhead view. The righthand side shows the factored 2D representation of our 3D environments. Each of the colors correspond to the lefthand side, but the robot is omitted. This shows one mode, for a fixed choice of target object orientation and face. 90
- 4.3 We consider three experimental domains: `corner` (left), `slippery_slope` (middle) and `glasswall` (right). The target object, at its starting configuration, is blue, the goal is the highlighted green region, obstacles are black and non-interactables are purple. 91
- 4.4 (Left) In the `slippery_slope` environment, for one mode, we overlay the `FeasibleArea` for four actions. (Right) The result of the shattering algorithm is to compute all possible intersections of the subsets of `FeasibleAreas` such that the space is partitioned into regions where each region has a specified set of actions, thus defining the `ActionRegions`. Given this representation, we can query a specific point in configuration space (shown as the black dot) to retrieve the set of feasible actions. 94
- 4.5 `DangerZones` in the `slippery_slope` domain. Here we show the zones across six modes, capturing the three object faces and two orientations. The obstacles (and configuration space obstacles) are given in black (and grey). We color z_0 in dark red and show z_1 and z_2 in brighter shades. 96
- 1 Across varying stiffnesses with Cartesian impedance control, we plot the experimental relation between the offset in the commanded offset in the z direction and the exerted force in z , as measured by an external force-torque sensor. For each stiffness we plot all five experimental runs, bolding the average. The result shows that the relation between the offset and force exerted is nearly linear. 119

1

Introduction

The goal of this thesis is to enable robots to robustly perform multi-step, contact-rich manipulation tasks in everyday environments. We consider tasks such as cooking dinner at home, packing supplies in hospitals and cleaning up messy classrooms (Fig.1.1).

Completing these types of tasks requires a robot to execute *long sequences of actions*, where each action involves many connected, discrete and continuous choices that are critically impacted by *physical constraints*. For example, if a stack of chairs is too heavy for a robot to carry, can the robot detect this and instead choose to push the stack, while also deciding how to push the stack? In preparing stir fry, can the robot choose between different utensils, using various tools to chop vegetables and mix ingredients and then even clean up afterwards? Furthermore, for robots to operate in the real world, it is crucial that they be able to cope with *partial or uncertain information*. While the robot may not know the exact shape or weight of every ingredient, we still want it to be able to manipulate each with ease.

These three issues — **long-horizon decision-making, the mechanics of contact-rich interaction and uncertainty** — are at the core of robot manipulation tasks. These issues interact in critical ways, as illustrated by Fig.1.2, where the robot must generate a sequence of actions to open a childproof medicine bottle.

To exert the significant amount of downward force and torque needed to open the push-and-twist bottle, the robot could use its fingers to grip the cap. In this particular case, however, the blue tool allows the robot to more robustly apply the desired force. Unfortunately, cluttered environment prevents the robot from directly and securely grasping the tool, since only the very top of the tool is reachable. While the robot could rearrange the scene, carefully moving the other tools out of the way, the robot instead grasps the top of the tool and leverages its environment to achieve a more secure grasp. The robot then uses the tool to exert the desired push-and-twist and, finally, uncap the bottle.

In addressing these core challenges of manipulation, this example highlights the importance of enabling the robot to reason over what choices are



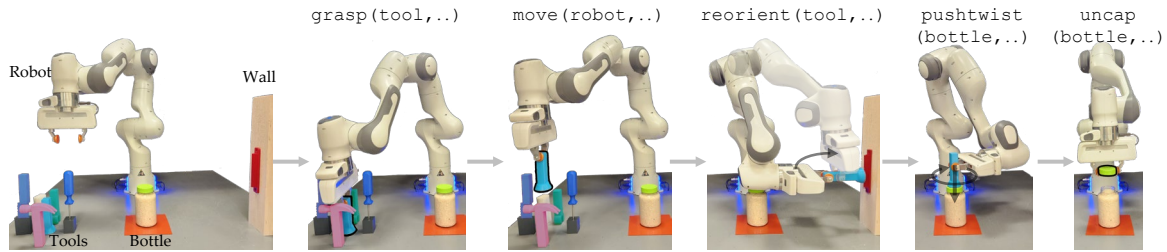
Figure 1.1: The goal of this thesis is to enable robots to complete a wide range of multi-step manipulation tasks.

suitable given the constraints of the task and the environment. In particular, the need to make robust choices, exert substantial force and operate in clutter impacted our robot’s decision-making, both with respect to the sequence of actions and the parameters of those actions. And, if the environment didn’t have a wall for the robot to utilize in changing its grasp on the tool, we would still want our robot to be able to complete the task, by, for instance, using the aforementioned rearrangement strategy.

We believe that this type of reasoning – where the robot is empowered to search over a combinatorial number of strategies and choices — is crucial to enabling a robot’s behavior to generalize across a diverse range of tasks and environments. This thesis adopts a *model-based approach*, where we develop planning frameworks and algorithms that leverage composable models of the world to sequence complex robotic behavior. We contribute models that draw from tools in classical mechanics, including kinematics, statics and dynamics, and those learned from data. We consider models that account for uncertainty, specifically focusing on uncertainty in the physical parameters that govern the mechanics models and uncertainty in the dynamics models of the action.

Summarizing, this thesis **contributes models and algorithms that exploit the physics and geometry of the world in order to tackle the dual challenges of long-horizon decision-making and acting under uncertainty, in the context of robotic manipulation.** We demonstrate this in three domains: in-hand manipulation, forceful manipulation and briefly dynamic manipulation:

In-Hand Manipulation We focus on in-hand manipulation as strategy for the robot to change its grasp on, or regrasp, an object. Specifically, this thesis considers reorienting a grasped object in-hand by repeatedly pushing against features in the environment. Critical to enabling this behavior is capturing how a grasped object can move when pushed, accounting for the



underactuation of friction and the effect of gravity. We contribute a model for frictional pushing in the gravitational plane that extends the notion and construction of the motion cone (Mason, 1986). This model defines the set of motions that it is possible to produce when pushing an object. We leverage in a sample-based motion planning algorithm to generate sequences of pushes to manipulate the object in-hand. We additionally show how slight modifications to the planner enable it to generate plans that are robust to uncertainty in the physical parameters. (Chapter 2)

Forceful Manipulation We define forceful manipulation tasks as those where the ability to generate and transmit the necessary force to objects is an active limiting factor that constrains the action selection, both with respect to the sequence of actions and the parameters of those actions. In this thesis, we use opening a childproof medicine bottle, twisting a nut on a bolt and chopping vegetables as illustrative forceful manipulation tasks. Accomplishing these tasks requires the robot to make a wide range of discrete and continuous choices that are constrained by both motion and, critically, force-related constraints. To model the latter, we contribute the forceful kinematic chain constraint, which captures the system’s ability to stably exert the desired force. We frame forceful manipulation as a task and motion planning (TAMP) problem and incorporate this constraint into an existing state-of-the-art framework (Garrett et al., 2020a). Additionally, by quantifying the probability of task success as an action cost and using cost-sensitive planning, we enable the robot to generate plans that are robust to uncertainty in physical parameters. (Chapter 3)

Briefly-Dynamic Manipulation Finally, we consider generating sequences of actions that have brief periods of dynamics, referred as briefly-dynamic manipulation (Mason, 2001). We focus on dynamic actions as an illustrative instance of manipulation that has a high degree of action uncertainty that must be reasoned over, particularly because some actions may have outcomes that are irrecoverable, making it impossible to achieve the goal. We learn a simplified model of the action dynamics and contribute a search algorithm GUARD (Guiding Uncertainty Accounting for Risk and Dynamics) for iteratively planning and executing actions. Critically, our algorithm computes where actions may be irrecoverable and steers the search away from these areas, enabling the robot to act safely in the face of uncertainty. (Chapter 4)

Figure 1.2: Complex manipulation tasks, like opening a push-and-twist childproof medicine bottle, require robots to reason over many physically-complex actions. Here, the robot wants to use a tool (outlined in black) but because the tool is surrounded by other objects, the robot can only reach the top of the tool. The robot then leverages the environment to create a more secure grasp, thus allowing the robot to use the tool to exert the necessary force to uncap the bottle.

In each domain we model the mechanics of the task that serve as constraints that are integrated into a multi-step planner. We contribute methods to generate plans that are robust to various sources of uncertainty. Our model-based approach, both in representing the mechanics and in planning, enable our algorithms and frameworks to generalize to new environments and to solve long-horizon tasks. Additionally, throughout this thesis we ground the research in real robot experiments, which are fundamental to verifying the contributed methods.

2

Planar In-Hand Manipulation via Motion Cones

2.1 Introduction

A *motion cone* is the set of feasible motions that a rigid body can follow under the action of a frictional push. We can think of it as a representation of the underactuation inherent to frictional contacts. Since contacts can only push, and since friction is limited, a contact can move an object only along a limited set of rays. The concept was introduced by Mason (1986) for a point contact in the context of a planar horizontal pushing task.

Motion cones are a practical geometric representation that abstracts the algebra involved in simulating frictional contact dynamics, and provides direct bounds on the set of feasible object motions. A contact force on the inside (or boundary) of the friction cone produces sticking (or slipping) behavior, and leads to motion rays on the inside (or boundary) of the motion cone. Lynch and Mason (1996) generalized the construction of motion cones to line contacts in a horizontal plane. They used them to plan stable pushing trajectories without having to deal explicitly with the complexities of the complementarity formulations of contact dynamics (Stewart and Trinkle, 1996; Posa et al., 2014; Chavan-Dafle and Rodriguez, 2020). Since then, motion cones have been the basis of several efficient planning and control strategies for planar manipulations on a horizontal support surface (Erdmann, 1998; Dogar and Srinivasa, 2010; Hogan and Rodriguez, 2016; Zhou and Mason, 2017).

This chapter studies the construction of motion cones for a broader set of planar pushing tasks, moving beyond the horizontal plane, and including the effect of gravity. In particular, we highlight the case of prehensile manipulation in the vertical plane where gravity—or other possible external forces—alter the dynamics of the contact interactions between an external pusher and a grasped object. In this chapter we show that motion cones are an effective representation to capture the coupled mechanics between gripper-object-environment, which is critical for efficient simulation, planning, and control.

We present four main contributions:

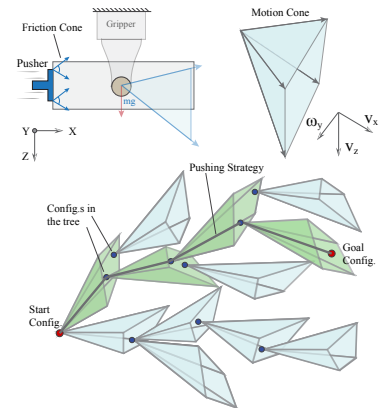


Figure 2.1: (top) Example *friction cone* and *motion cone* of an object moving in the vertical plane. The pusher can move the object along any direction $[V_x, V_z, \omega_y]$ inside the motion cone. (bottom) A plan via motion cones. Motion cones capture local reachability. A path in the tree of motion cones generates a pushing strategy to move an object.

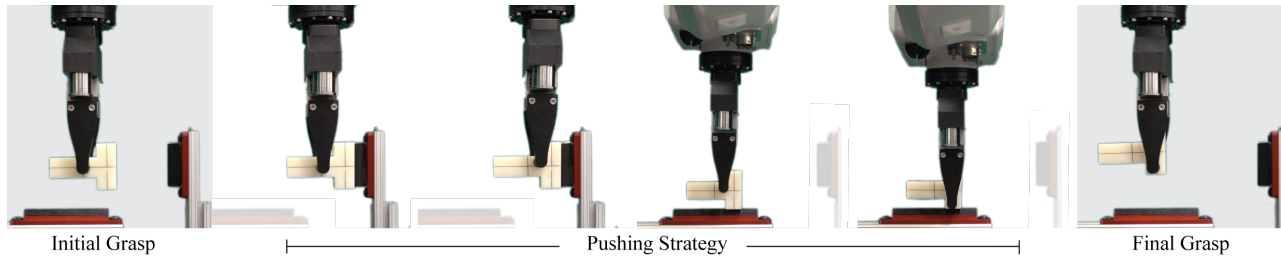


Figure 2.2: Manipulating a T-shaped object in a parallel-jaw grasp by pushing it against features in the environment. The manipulation is shown from a side view.

- **Mechanics** of motion cones for planar tasks in the gravity plane. We show that the motion cone is defined by a set of low-curvature surfaces, intersecting at a point and pairwise in lines. We propose a polyhedral approximation to the motion cone for efficient computation.
- **Experimental validation** of the boundaries of the motion cones in a planar prehensile pushing task. We record the object motion and the stick/slip of the pusher for thousands of prehensile-pushes in three different grasp configurations. The motion cones characterize the feasible object motions with 80%+ average accuracy. These numbers are considerably larger when taking conservative approximations of the coefficient of friction.
- **Application** of motion cones in a sampling-based planning framework for planar in-hand manipulation using prehensile pushes (see Fig.2.1). This yields significant speed improvements with respect to prior work (Chavan-Dafle and Rodriguez, 2020, 2018). Fig.2.2 shows an example of a prehensile pushing trajectory to change the grasp on a T-shaped object. The trajectory is a sequence of continuous stable pushes, during which the object sticks to an external pusher. The proposed planning algorithm obtains these trajectories consistently in less than a second.
- **Robustness** of prehensile pushes to variations in friction coefficients and gripping force. We show that conservative approximations motion cones, and a careful selection of the orientation of the task with respect to gravity, yield increased reliability, and, in some cases, invariance.

The generalization of motion cones to interactions with external forces such as gravity opens a door for efficient and robust planning of in-hand manipulation that respects—and exploits—the basic principles of frictional rigid-body contact interactions: Newton’s second law, Coulomb’s friction law, the principle of Maximal Dissipation, and the rigidity of rigid-bodies.

2.1.1 Structure of the Chapter

In this chapter we describe the mechanics and algorithms involved in using motion cones. When possible, we use both the languages of algebra (for implementation) and geometry (for intuition) to describe them.

The chapter can be divided in four main parts: i) Mechanics of planar pushing; ii) Construction, computation, and validation of motion cones; iii) Using motion cones to plan in-hand manipulation; iv) Different mechanisms to increase the reliability of motion cones.

The mechanics of planar pushing. This is the basis of the mechanics analysis in the chapter and is reviewed in detail in Sec. 2.3. We show how the motion of a pushed object is governed by three key laws (described in three sub-sections): 1) The frictional interaction at the contact between an object and a supporting plane, the *limit surface*. 2) The frictional interaction at the contact between the object and the pusher. When the contact is a point, Coulomb’s law defines a friction cone. When the contact is a line or more complex, it defines a *generalized friction cone*. 3) The force balance imposed by Newton’s second law. These three constraints define the set of possible force equilibria between the gripper, object, and environment.

Construction, computation, and validation of motion cones. In Sec. 2.4 we show how a particular choice of pusher motion, alongside the *principle of maximal dissipation*, i.e., relation between the friction and velocity of a sliding object, determines the motion of the pushed object. The set of object motions for all possible pusher motions defines the *motion cone*. Sec. 2.4 describes how to compute motion cones for the cases of an object pushed in the horizontal plane (review) and in the vertical plane with gravity (contribution). In Sec. 2.5 we provide an analytic expression for the motion cone as well as an approximation in the form of a polyhedral motion cone, which will be key to using them in a fast planning framework. At the end of this section we detail efforts to verify experimentally the validity of motion cones for three different task configurations, i.e., three different combinations of grasps and prehensile pushes.

Motion cones for planning in-hand manipulation. In Sec. 2.6 we describe how to use motion cones to manipulate grasped objects by pushing them against contacts in the environment. Motion cones directly describe the set of directions along which a grasped object can be displaced for a given external contact. In practice, we compute motion cones in real-time and use them to efficiently sample feasible regrasps in an RRT*-based planning framework. In Sec. 2.7 we show different examples of planned and executed planar in-hand regrasps.

Robustness of motion cones. Finally, we explore different ways to improve the reliability of motion cones. Sec. 2.8 shows that either a conservative approximation of motion cones, or a careful selection of the orientation of the task with respect to gravity, are sufficient to yield mechanical robustness with

respect to variations in the available friction at the contacts and the gripping force.

We provide a video summarizing our approach at <https://youtu.be/tVDO8QMuYhc>.

2.1.2 Assumptions of the Approach

This chapter presents a model-based approach to planar in-hand manipulation of grasped objects. The approach relies on the following assumptions and representations:

- **Objects:** three dimensional objects grasped along parallel faces. The manipulation plane is parallel to the grasped faces. The outline of the object in the manipulation plane can be non-convex. We assume we know the geometry and mass of the object.
- **Gripper:** a parallel-jaw gripper with force control. The gripping force remains constant during the manipulation. The friction coefficient between the gripper-fingers and the objects is known, with some bounded uncertainty.
- **Environment:** A rigid environment with geometric features (e.g. flat faces, edges, corners) to push against. These provide affordances of controlled patch, line or point contacts. We know the location and geometry of the features with precision and their friction coefficient with some bounded uncertainty.
- **Regrasp Examples:** planar regrasps are in the plane orthogonal to the axis between the fingers of the parallel jaw-grasp. The configuration space, i.e., the space of object grasps, is three-dimensional; the object can translate and rotate in the manipulation plane. The desired regrasp is specified by the initial and final object pose in the grasp, as illustrated in Fig.2.2.

The planning framework proposed in this work uses pushing dynamic, captured in the form of motion cones, to generate a sequence of external pushes that moves the object from the initial to the final grasp. In our implementation, the external pushes are executed by a robot forcing the grasped object against fixed features in the environment. These pushes could also abstract the interactions with a second robot arm or with extra fingers in a multi-finger gripper.

Fig.2.3 shows the schematic from the planner’s perspective: in the planning phase, we imagine the alternate perspective where the parallel-jaw gripper is fixed in the world and the environment is a “virtual” pusher with full 3DOF mobility in the manipulation plane pushing the grasped object. Planning external pushes is then equivalent to planning the motion of the virtual pusher to achieve the desired regrasp. The “virtual” pusher motion is then rendered in the real world by instead moving the gripper.

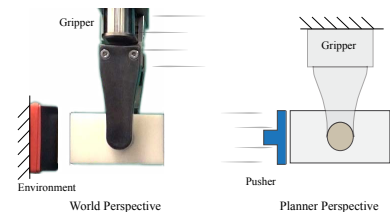


Figure 2.3: (left) An object held in a parallel-jaw gripper is pushed against a fixed feature in the environment. (right) The planner models this interaction as if the object is held by a fixed gripper and pushed by a moving environment, i.e. a pusher.

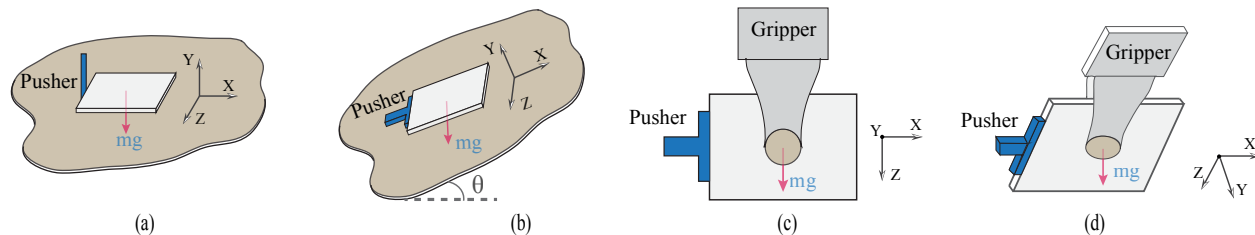


Figure 2.4: Pushing an object (a) on a horizontal surface, (b) on an inclined surface, (c) in a grasp in the gravity plane, and (d) in a grasp in a tilted plane

2.2 Related Work

Planning and control through contact is a central topic in robotic manipulation research. A rigid-body contact is modeled as a series of constraints on the possible motions and forces at contact. A particular contact mode, either sticking or sliding, invokes a specific set of constraints on the motions and forces at contact. Stewart and Trinkle (1996) and Trinkle et al. (1997) show that a general rigid body dynamics problem with hybrid sticking/slipping dynamics at contact interactions can be modeled as a complementarity problem.

Recent work on trajectory optimization and manipulation planning show that it is possible to reason about hybrid stick/slip contact modes and plan trajectories through continuous contact with complementarity constraints (Posa et al., 2014; Chavan-Dafle and Rodriguez, 2020). An alternative approach is to replace the hard complementarity constraints by optimization-friendlier soft-constraints, which yields a faster planning and control framework (Kumar et al., 2014; Todorov et al., 2012). These methods, though broad in their scope of application, often have to compromise between computational efficiency and the realism of contact dynamics (Kolbert et al., 2017).

In contrast, a pivotal theme in the non-prehensile manipulation literature is to identify application-specific and compact representations of contact constraints and the mechanics of the task. Goyal (1989) introduced the concept of a *limit surface*, a compact mapping from the friction wrench between an object and its support surface and the sliding twist at contact. Mason (1986) studied the mechanics of pushing and proposed the concept of the *motion cone*. These two fundamental geometric constructions provide direct force-motion mappings for contact interactions and have facilitated efficient planning and control techniques in non-prehensile manipulation (Lynch et al., 1992; Lynch and Mason, 1996; Dogar and Srinivasa, 2010, 2011; Hogan and Rodriguez, 2016; Zhou and Mason, 2017).

Lynch and Mason (1996) extend the idea of motion cones to a line pusher on a horizontal surface and show its application for planning robust pushing strategies for which the pusher contact sticks to the object. Dogar and Srinivasa (2011) demonstrate the application of motion cones in the push-grasping framework to plan pushes that capture an object before grasping it. Hogan and Rodriguez (2016) and Zhou and Mason (2017) exploit the direct action-effect mapping in motion cones to develop efficient planning and control

techniques for pushing an object on a horizontal plane.

Some recent work focuses on other types of manipulation primitives. [Shi et al. \(2015\)](#) demonstrates dynamic in-hand manipulation planning in a parallel-jaw grasp by exploiting inertial forces. With a pre-defined contact mode sequence at the fingers and a limit surface approximation for the force-motion interaction at the fingers, they derive a control law that can move the object to the goal grasp through a sequence of fast accelerations and decelerations. Similar approaches are explored for planning and controlling in-hand manipulations by actively using gravity ([Viña B et al., 2016](#)) or dynamic motions ([Holladay et al., 2015](#); [Sintov and Shapiro, 2016](#); [Hou, 2017](#)).

[Sundaralingam and Hermans \(2017\)](#) propose a purely-kinematic approach for in-hand manipulation based on trajectory optimization with a multi-finger gripper. They assume that the fingers on the object do not slip and impose soft constraints that encourage the minimization of finger slip. By assuming all finger contacts to be sticking, they bypass the need for modelling the dynamics of contacts and obtain fast kinematic plans.

Recent work presented a sampling-based planning framework for in-hand manipulations with prehensile pushes, where the pusher contact is forced to stick to the object using the mechanics of the task ([Chavan-Dafle and Rodriguez, 2018](#)). The plans are discrete sequences of continuous pushes that respect friction, contact, and rigid-body constraints.

These promising results on in-hand manipulation, which are limited to certain types of manipulation primitives, motivate us to extend the concept of motion cones to more general pushing tasks. Motion cones provide direct bounds on the set of possible object motions and can be used in sampling-based frameworks to guide sampling, or as a set of direct constraints in a trajectory optimization framework.

2.3 Mechanics of Planar Pushing

Fig.2.4 shows four different cases of planar pushing. In case (a), the pusher force is the only external force on the object in the manipulation plane. However, in the other cases, a component of gravity is also present. The concept of motion cones, as originally studied in [Mason \(1986\)](#) and [Lynch and Mason \(1996\)](#), is limited to case (a). The extension of the mechanics of motion cones that we present in this work is valid for all cases in Fig.2.4.

In this section we discuss the mechanics of pushing an object in a plane. First, we will review the fundamental concepts, namely, the *limit surface* ([Goyal, 1989](#)), and the *generalized friction cone* ([Erdmann, 1994](#)). These tools will serve as building blocks for modeling the force-motion interaction at the contacts involved in pushing manipulations. Table 2.1 lists the notation used within the rest of the chapter.

	$q = [x, y, \theta] \in SO(2)$	Configuration of the object
	\mathcal{F}_W	World frame fixed to the gripper.
	\mathcal{F}_S	Support frame, fixed to the support surface and coinciding with the world frame, \mathcal{F}_W . Located at the center of pressure at the contact.
	\mathcal{F}_O	Object frame, moving with the center of mass of the object
	\mathcal{F}_{P_i}	Pusher contact frame, moving with the i -th point contact modeling the pusher
	$w_{\text{support}} \in [f_x, f_z, m_y]^T$	Wrench exerted on the object by the support contact (in frame \mathcal{F}_O)
	$w_{\text{pusher}} \in [f_x, f_z, m_y]^T$	Wrench exerted on the object by the pusher contact (in frame \mathcal{F}_O)
	$w_s \in [f_x, f_z, m_y]^T$	Wrench at the support contact (in frame \mathcal{F}_S).
	$f_p \in [f_{n1}, f_{t1}, f_{n2}, f_{t2}, \dots]^T$	A vector of forces at pusher point contacts in $\mathcal{F}_{P1}, \mathcal{F}_{P1}, \dots$ respectively
	$v_{obj} \in [v_x, v_z, \omega_y]^T$	Object twist, i.e. generalized velocity, in \mathcal{F}_O
	$v_s \in [v_x, v_z, \omega_y]^T$	Object twist at the support contact in \mathcal{F}_S
	$J_s \in \mathbb{R}^{3 \times 3}$	Jacobian that maps v_{obj} to v_s
	$J_p = \begin{bmatrix} J_{P1} \\ J_{P2} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2n \times 3}$	Jacobian that maps v_{obj} to the velocity at the n pusher point contacts in the pusher contact frames ($\mathcal{F}_{P1}, \mathcal{F}_{P2}, \dots$).

2.3.1 Limit Surface

The limit surface is a common construction to model the frictional interaction at a planar patch contact. In this work, we use the limit surface to model the force-motion interaction between an object and a support contact. In cases (a) and (b) in Fig.2.4, the surface on which the object rests is the support contact. In cases (c) and (d), the gripper fingers are the support contacts. As summarized in Table 2.1, for object motions in the manipulation plane, the resultant friction wrench w_s offered by the support contact on the object is composed of the frictional forces (f_x, f_z) in the plane and the frictional torque (m_y) about the normal to the plane.

Goyal (1989) defined the boundary of the set of all possible friction wrenches that a contact can offer as the *Limit Surface*. Each wrench on the boundary of the limit surface corresponds to a different sliding motion of the object on the support plane. Based on the principal of maximum dissipation, the perpendicular to the limit surface at a given wrench provides

Table 2.1: (Left) Schematic of a prehensile pushing scene with important reference frame labeled. (Right) Mathematical notation used throughout the chapter.

the corresponding generalized velocity of the object (Goyal, 1989).

Related to the limit surface, the Gravito-Inertial Wrench Cone (GIWC) is a geometric construct in legged locomotion literature. However, the embedding of the principal of maximum dissipation into the limit surface construction provides the information of the sliding velocity at the contact, which GIWC can not.¹

The limit surface is smooth and convex if the support force distribution is finite everywhere. Howe and Cutkosky (1996); Xydas and Kao (1999) showed that an ellipsoidal approximation allows for a simpler representation of the limit surface geometry. In this work we will assume an ellipsoidal approximation of the limit surface, which has been shown to be computationally efficient for simulating and planning pushing motions (Lynch and Mason, 1996; Dogar and Srinivasa, 2011; Shi et al., 2015; Zhou and Mason, 2017).

Let $w_s = [f_x, f_z, m_y]$ be the frictional wrench on the object from the support contact in the support contact frame, \mathcal{F}_S . A mathematical representation of the ellipsoidal limit surface is given by:

$$w_s^T A w_s = 1 \quad (2.1)$$

where, for isotropic friction,

$$A = \begin{bmatrix} \frac{1}{(\mu_s N)^2} & & 0 \\ & \frac{1}{(\mu_s N)^2} & \\ 0 & & \frac{1}{(k\mu_s N)^2} \end{bmatrix}$$

such that μ_s is the friction coefficient between the contact and the object, N is the normal force at the contact and k is an integration constant. For a circular patch contact with a uniform pressure distribution at the contact, $k \approx 0.6r$ where r is the radius of the contact (Xydas and Kao, 1999; Shi et al., 2015). Note that the geometry of the limit surface is defined by the area and pressure distribution at the contact and scaled by normal force and friction coefficient at the contact.

When the object slides on the support contact, the friction wrench (w_s) at the contact intersects the limit surface, as shown in Fig.2.5. The direction of object twist in the contact frame (v_s) is given by the normal to the limit surface at the intersection point (Goyal, 1989) and can be computed as:

$$v_s \propto \frac{\partial}{\partial w_s} (w_s^T A w_s) \propto A w_s. \quad (2.2)$$

Conversely, if the object twist at the contact (v_s) is known, we can find the friction wrench as,

$$w_s = \frac{1}{\lambda} A^{-1} v_s \quad (2.3)$$

¹ The GIWC captures the set of wrenches the contacts between a robot and its environment can resist (Wieber, 2006; Caron et al., 2011). If the sum of the gravitational and inertial wrenches on the robot is inside the GIWC, it can be balanced by the friction and normal forces at the contacts and consequently the pose of the robot can be stable. This is similar to how the stability of a grasped object can be characterized by doing the Minkowski sum of the limit surfaces at the multiple support/finger contacts (Hong Lee and Cutkosky, 1991; Shi et al., 2015). Unlike the limit surface which, in combination with the principle of maximal dissipation, has been used to reason about sliding velocities, GIWC does not provide any information about the relative motion of the robot and the environment when the net gravito-inertial wrench is on the boundary or outside GIWC. For practical reasons, the locomotion community has not focused on sliding behaviors, while the manipulation community has tried to exploit them.

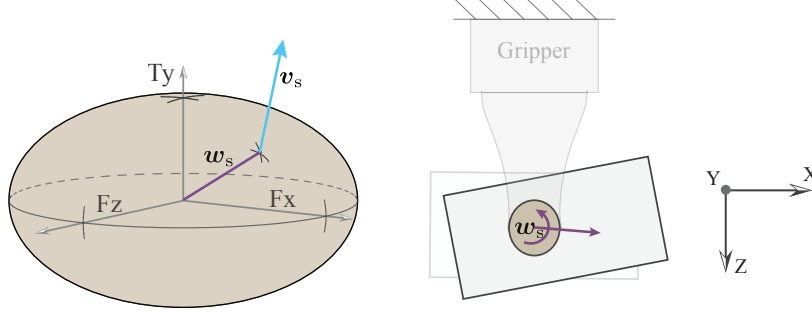


Figure 2.5: (Left) Ellipsoidal approximation of the limit surface at the finger contact. For a wrench w_s on the boundary of the limit surface, the twist at the contact v_s is normal to the limit surface. (Right) The wrench w_s and the corresponding object motion is shown in the manipulation scene.

Since w_s lies on the ellipsoidal limit surface, from (2.1), we can solve for λ :

$$\begin{aligned} \left(\frac{1}{\lambda}A^{-1}v_s\right)^T A \left(\frac{1}{\lambda}A^{-1}v_s\right) &= 1 \\ v_s^T A^{-T} A A^{-1} v_s &= \lambda^2 \\ v_s^T A^{-T} v_s &= \lambda^2 \\ \sqrt{v_s^T A^{-T} v_s} &= \lambda. \end{aligned}$$

Given that A is a diagonal matrix, $\lambda = \sqrt{v_s^T A^{-1} v_s}$. Rewriting (2.3):

$$w_s = \frac{A^{-1}v_s}{\sqrt{v_s^T A^{-1} v_s}} = \frac{\mu_s N B^{-1} v_s}{\sqrt{v_s^T B^{-1} v_s}} = \mu_s N \bar{w}_s \quad (2.4)$$

Here, $B = \text{Diag}(1, 1, k^{-2})$ and $\bar{w}_s = [\bar{f}_x, \bar{f}_z, \bar{m}_y]^T$ is the wrench on a unit limit surface ($\bar{w}_s^T B \bar{w}_s = 1$) which is scaled by $\mu_s N$ to produce the net frictional wrench w_s .

The force-motion mapping we have discussed so far is in the support contact frame. From (2.2) and (2.4), the direction of the object twist in the object frame (v_{obj}) for a given wrench on the limit surface can be computed as:

$$v_{obj} \propto J_s^{-1} B \bar{w}_s \quad (2.5)$$

where J_s^{-1} maps the object velocity from the support contact frame \mathcal{F}_s to the object frame \mathcal{F}_O .

2.3.2 Generalized Friction Cone

Coulomb's friction law is the basis for many of the models used to simulate frictional interaction between two rigid bodies. A Coulomb friction cone establishes the relationship between the normal force and the maximum friction force at a point contact.

Erdmann (1994) introduced the concept of the *generalized friction cone* (W) as the friction cone of the contact expressed in the reference frame of the object. The generalized friction cone for a line/patch contact modelled with multiple point contacts is the convex hull of the generalized friction cones for each constituent contact (Erdmann, 1993).

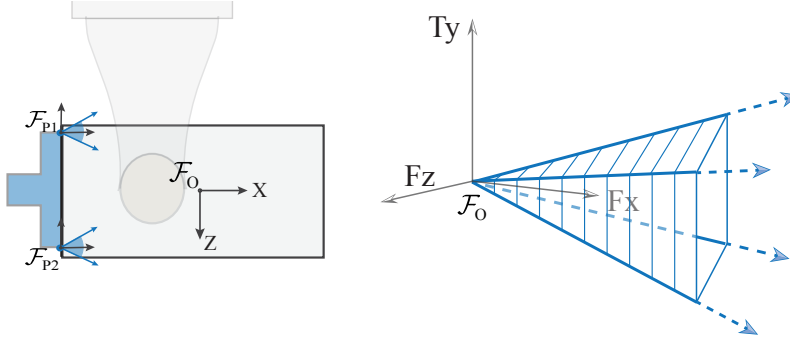


Figure 2.6: (Left) The line pusher contact in this example is modelled with two point contacts. The pusher contact frames and the object frame are drawn in the figure. (Right) The generalized friction cone of the pusher is the object frame representation of the set of forces the two point contacts can offer collectively.

We model the friction between the pusher and the object by constructing the generalized friction cone:

$$W_{\text{pusher}} = \{w_{\text{pusher}} = J_p^T \cdot f_p \mid f_p \in FC_{\text{pusher}}\} \quad (2.6)$$

where, J_p^T is the Jacobian that maps the contact forces f_p from the friction cone FC_{pusher} at the pusher point contacts to the object frame. Since the Coulomb friction cone represents the set of feasible forces a point contact can offer, the generalized friction cone captures the set of feasible wrenches the pusher can impose on the object. If the contact between the pusher and the object is maintained (sticking), the pusher can impose any wrench on the interior of the generalized friction cone. However, if the object slips with respect to the pusher, the pusher wrench on the object will be on the boundary of the generalized friction cone.

We model the interaction between support-and-object (limit surface) and between pusher-and-object (generalized friction cone) differently due to the different geometries and expected pressure distributions. For the support-object interaction, the ellipsoidal approximation to the limit surface provides a compact force-motion relationship at a support contact where we expect a constant normal force distribution during the manipulation. Whereas, the generalized friction cone is a better approach for modelling the friction at the pusher contact where we can have varying normal force distribution for every instantaneous push.

Now, with the chosen models for frictional contact, we formulate the mechanics of pushing in a plane.

2.3.3 Mechanics of Pushing

The motion of the pushed object is in accordance with the net wrench acting on the object. Under the quasi-static assumption, which is appropriate for slow pushing operations, the inertial forces on the object are negligible and force balance requires:

$$w_{\text{support}} + w_{\text{pusher}} + mg = 0 \quad (2.7)$$

Here, (2.7) is written in the object frame located at the center of gravity. w_{support} is the friction wrench provided by the support contact, w_{pusher} is the wrench exerted by the pusher, m is the mass of the object, g is the gravitational component in the plane of motion. The Jacobian J_s^\top maps the support contact wrench from the support contact frame (usually located at the center of pressure) of the support surface to the object frame. So, $w_{\text{support}} = J_s^\top \cdot w_s$ and (2.7) becomes:

$$J_s^\top \cdot w_s + J_p^\top \cdot f_p + mg = 0 \quad (2.8)$$

Equations (2.8), (2.6) and (2.5) define the system of equations to solve to determine the velocity of the object v_{obj} for any possible vector for pusher forces f_p . Solving the mechanics of pushing refers to solving (2.8) at every time instant while considering the possibility of sticking/sliding at the pusher-object and support-object contact interactions. Based on the sticking or sliding mode at the support and pusher contact, the wrenches at these contacts will be either inside or on the boundary of the limit surface and the generalized friction cone respectively, as discussed in Sec. 2.3.1 and Sec. 2.3.2. We can formulate the hybrid dynamics of pushing as a rigid-body dynamics problem which will take the form of a complementarity problem (Chavan-Dafle and Rodriguez, 2020).

Generating a motion plan for a given pushing manipulation task, e.g., moving an object on a ground or in a grasp, involves building a continuous series of instantaneous feasible object and pusher motions that satisfy the mechanics of pushing at every instant. In a trajectory optimization framework, this translates to including the complementarity constraints from the dynamics of pushing into the optimization problem and optimizing for the series of object-pusher states. In a sampling-based framework this translates to sampling instantaneous pusher-object motions that comply with (2.8), (2.6) and (2.5). Checking their feasibility involves solving the mechanics of pushing and keeping track of the pusher-object state including the possibility of sliding between them. Either way, solving the mechanics of pushing to predict the pusher-object motion and slipping/sticking at contacts is computationally expensive (Posa et al., 2014; Hogan and Rodriguez, 2016).

A direct action-effect mapping, i.e., the relationship between the pusher motion and object motion, avoids the burden of solving the hybrid mechanics of pushing. The existence of such a mapping is the key motivation of this work, described by the motion cone construction that we study in the following section.

2.4 Motion Cones for Planar Pushing

In this section, we present the general formulation for defining a motion cone for planar pushing. We discuss the nature of the constraints that define the motion cone for pushing on a horizontal surface (Fig.2.4 (a)) and for

more general pushing scenarios (Fig.2.4 (b-d)). We highlight some of the peculiarities of the case of pushing an object in the gravity plane, which forms the basis for the computation and application of the motion cones in the following sections.

The motion cone is the set of instantaneous object velocities that a pusher can impose on the object using the frictional interaction between them. If the pusher moves along a twist inside the motion cone, the contact between the object and the pusher sticks and the object follows the pusher motion. If the pusher moves along a twist outside the motion cone, there is sliding at the pusher-object interface and the object moves along a twist on the boundary of the motion cone. While the pusher has full control to move with any velocity, it can impose only a set of object velocities defined by the motion cone.

Mathematically, a motion cone is the largest set of object motions for which the net required wrench for the object motion can be balanced by a wrench on the interior of the generalized friction cone of the pusher. It is the set of object velocities for which constraint (2.8) holds true while $f_p \in FC_{\text{pusher}}$.

$$J_s^T \cdot w_s + J_p^T \cdot f_p + mg = 0, f_p \in FC_{\text{pusher}}$$

Using (2.4) and (2.6), we can rewrite the previous equation as:

$$\begin{aligned} \mu_s N J_s^T \cdot \bar{w}_s + k \bar{w}_{\text{pusher}} + mg &= 0 \\ \bar{w}_{\text{pusher}} \in W_{\text{pusher}}, k \in \mathbb{R}^+ \end{aligned} \quad (2.9)$$

where, k is the magnitude of the pusher force and \bar{w}_{pusher} is a unit pusher wrench and \bar{w}_s is a unit support wrench. To find the set of instantaneous object motions that satisfy the above equation, we rearrange it as:

$$\begin{aligned} J_s^T \cdot \bar{w}_s &= \frac{k}{-\mu_s N} \bar{w}_{\text{pusher}} + \frac{m}{-\mu_s N} g \\ \bar{w}_{\text{pusher}} \in W_{\text{pusher}}, k \in \mathbb{R}^+ \end{aligned} \quad (2.10)$$

Using (2.5) we can map the support contact unit wrench \bar{w}_s to the object twist v_{obj} . Hence, to find a motion cone, we first find the set of support contact wrenches (\bar{w}_s) that satisfy (2.10) and then map this wrench-set (W_s) to the set of object twists using (2.5). We denote this set of object twists, i.e. the motion cone, by V_{obj} ².

2.4.1 Motion Cone for Pushing on a Horizontal Surface

The presence of an external force, (such as the gravitational force) other than the pusher force, in the plane of motion complicates the mechanics and the structure of the motion cone. To explain this effect in detail, we will first consider the case of pushing an object on a horizontal surface where there is no such additional force.

² Mason (1986) defined the motion cone in terms of pusher twists which is a linear Jacobian transform of the motion cone V_{obj} . However, we will keep it in the object twist space because it is a more natural representation for planning object trajectories.

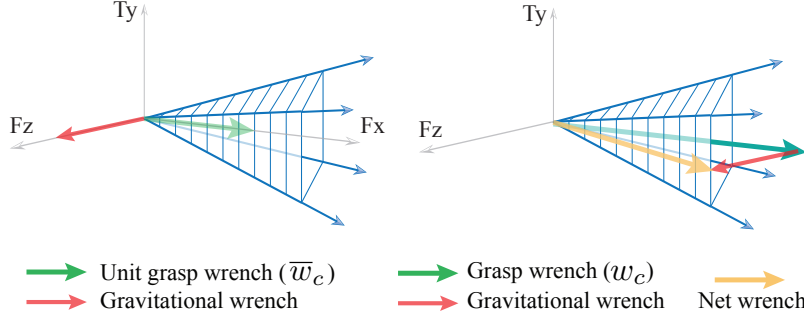


Figure 2.7: To make a push inside the gravity-free motion cone also stable in a scenario with gravity, the unit grasp wrench can be scaled such that the net pusher wrench required for the desired push falls inside/on the generalized friction cone of the pusher.

For the case of pushing on a horizontal plane, $\mathbf{g} = \mathbf{0}$. For an object on a flat support surface with uniform pressure distribution on the support (as is Fig.2.4- case (a) and (b)), the contact frame coincides with the object frame, $J_s = \mathbf{I}$ (identity matrix), because the support is infinite and spatially invariant. Then we can write (2.9) as:

$$\mu_s N \bar{\mathbf{w}}_s + k \bar{\mathbf{w}}_{\text{pusher}} = \mathbf{0}, \quad \bar{\mathbf{w}}_{\text{pusher}} \in \mathbf{W}_{\text{pusher}}, \quad k \in \mathbb{R}^+$$

$$\bar{\mathbf{w}}_s = \frac{-k \bar{\mathbf{w}}_{\text{pusher}}}{\mu_s N}, \quad \bar{\mathbf{w}}_{\text{pusher}} \in \mathbf{W}_{\text{pusher}} \quad (2.11)$$

The set of valid support contact wrenches that satisfy (2.11) is the negative of the generalized friction cone of the pusher, i.e. $\mathbf{W}_s = -\mathbf{W}_{\text{pusher}}$. By mapping \mathbf{W}_s through (2.5), we get the motion cone \mathbf{V}_{obj} .

Note that for the case of pushing on a horizontal surface, \mathbf{W}_s and \mathbf{V}_{obj} are both convex polyhedral cones like $\mathbf{W}_{\text{pusher}}$. Moreover, they are independent of the support normal force, i.e. the weight of the object mg , and friction at the support surface μ_s .

For more general pushing tasks however, $\mathbf{g} \neq \mathbf{0}$. From (2.10) we can see that, unlike for horizontal pushing, the system parameter (μ_s) and force magnitudes (k and N) influence the direction vectors of the wrench-set \mathbf{W}_s and motion cone \mathbf{V}_{obj} . In the next section we will focus on the case in Fig.2.4- (c) – pushing an object in a parallel-jaw grasp in the plane of gravity. There the gravitational force is not zero in the plane of motion and the Jacobians J_s are not always identity matrices since, in general, the support (finger) contact location changes in the object frame as the object is pushed in the grasp.

Note that the case in Fig.2.4-(b) can be modelled like (c) except the Jacobian $J_s = \mathbf{I}$ and only a part of the gravitational force is in the plane of motion. The case in Fig.2.4-(d) can be modelled like (c) except that only a fraction of the gravitational force acts in the plane of motion. The results derived for the case of pushing an object in the gravity plane then extend naturally to the cases in Fig.2.4-(b) and (d). However, the experimental validation necessary to certify this claim and to evaluate the accuracy of the underlying assumptions for cases (a) and (b) is out of scope of this work.

2.4.2 Motion Cone in the Gravity Plane

For a case similar to Fig.2.4-(c), but in a gravity-free world, we can simplify equation (2.10) by omitting the gravity term. We can then compute a convex polyhedral motion cone similar to that in the horizontal pushing case, but while taking the Jacobian J_s into consideration. We will refer to this motion cone as the *gravity-free motion cone* in the later discussions. This cone, similar to that for the case of pushing on the horizontal plane, is invariant to the friction at the finger contacts, the mass of the object, and the support normal force, i.e., the grasping force.

By rearranging (2.9), we see that an instantaneous object motion is feasible if the net wrench required for the object motion falls inside the generalized friction cone of the pusher.

$$-\mu_s N J_s^\top \cdot \bar{w}_s - m\mathbf{g} \in \mathbf{W}_{\text{pusher}} \quad (2.12)$$

The gravitational wrench $m\mathbf{g}$ on the object can pull the net required wrench in or out of the generalized friction cone of the pusher. Some of the motions in the gravity-free motion cone may not be stable pushes in the gravity plane, while for some object motions outside the gravity-free motion cone, the gravitational force can make them feasible.

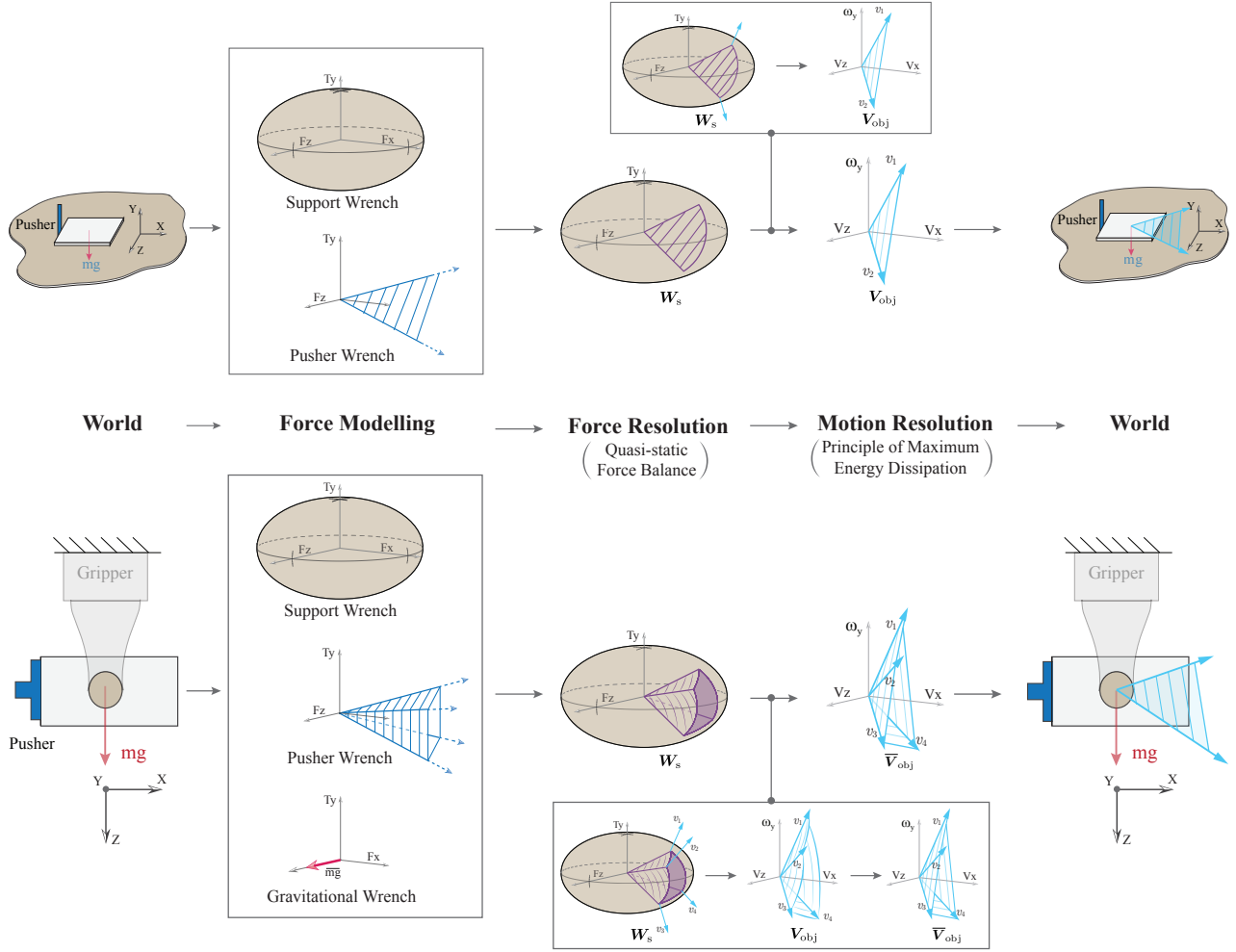
Proposition 1. *Any object motion inside the gravity-free motion cone can also be made feasible in the gravity plane by increasing the grasping force above a minimum threshold.*

Proof. For a motion inside a gravity-free motion cone, the support/grasp wrench direction lies on the interior the generalized friction cone of the pusher, i.e., $J_s^\top \cdot \bar{w}_s \in \mathbf{W}_{\text{pusher}}$. Because the gravitational wrench is of fixed magnitude, we can always scale the support/grasp wrench direction such that the net wrench (the vector sum of the gravitational wrench and the support/grasp wrench) is inside the generalized friction cone of the pusher. Fig.2.7 shows the graphical interpretation. In general, for a given μ_s , we can analytically find the bounds on the normal force N needed to pull the net wrench inside the generalized friction cone of the pusher. ■

A similar argument can be made for determining a minimum friction threshold at the finger contacts. For a given normal force N , we can analytically find the bounds on μ_s needed to pull the net wrench inside the generalized friction cone of the pusher.

Corollary 1.1. *With increased grasping force, more motions in the gravity-free motion cone are feasible in the gravity plane. For infinite grasping force, the gravity-free motion cone is equal to the motion cone in the gravity plane.*

Proof. From (2.12) and Proposition 1, as the grasping force increases, the net wrench for more twists or velocities from the gravity-free motion cone fall inside the generalized friction cone of the pusher.



In the gravity-free case, $J_s^T \cdot W_s = -W_{\text{pusher}}$.
As $N \rightarrow \infty$,

$$-\mu_s N J_s^T \cdot \bar{w}_s - m\vec{g} \rightarrow -\mu_s N J_s^T \cdot \bar{w}_s.$$

Then, $\forall J_s^T \cdot \bar{w}_s \in J_s^T \cdot W_s$:

$$-N\mu_s J_s^T \cdot \bar{w}_s - m\vec{g} \in W_{\text{pusher}}.$$

So, the set of object motions V_{obj} corresponding to W_s , which is the gravity-free motion cone, are also stable in the gravity case. ■

We see that, in theory, we can make any object motion in the gravity-free motion cone feasible under gravity by increasing the grasping force. With increased grasping force, more motions in the gravity-free motion cone are stable under gravity and become part of the motion cone. In practice, however, grippers have limited grasping force and often lack online force

Figure 2.8: Graphical illustration of the motion cone construction procedure applied to two different planar pushing systems: (top) pushing an object on a horizontal plane with a point pusher and (bottom) pushing an object in a grasp in the gravity plane. From left to right, the construction steps portray modelling of the forces involved in the task (Force Modelling), solving for the set of force-balance solutions in wrench-space (Force Resolution), and finally computing the set of object twists corresponding to the force-balancing solutions (Motion Resolution). This set of object twists is the motion cone.

control. Moreover, by limiting to the gravity-free motion cone, we omit object motions outside the gravity-free motion cone that otherwise would be feasible when considering the gravitational force on the object. Therefore, both for practical reasons and for capturing the rich mechanics of prehensile pushing, we need to find the motion cone under gravity for a given grasping force.

Note that for the case of pushing the object on a horizontal plane, the motion cone is invariant to the support normal force. However, for pushing on a support plane at an angle, as in Fig.2.4 -(b), a component of gravitational force acts in the manipulation plane and we do not have a freedom to change the support normal force, so computing the motion cone for a fixed support normal force is essential for these non-prehensile manipulation cases as well.

2.5 Computation and Experimental Validation of Motion Cone

In this section we present a closed-form computation of the object motion cone in the gravity case for a given grasping force and friction parameters. We discuss the graphical intuition for the mechanics and the computation of the motion cone. A large set of pushing experiments in different object-pusher configurations demonstrate that the analytically found motion cones characterize the the set of feasible object motions with an average of 80%+ accuracy.

2.5.1 Analytical Computation

For a known $\bar{\mathbf{w}}_{\text{pusher}} \in \mathbf{W}_{\text{pusher}}$, (2.10) is a set of three linear equalities with 4 unknowns, $\bar{\mathbf{w}}_s \in \mathbb{R}^3$ and $k \in \mathbb{R}^+$. We also know that $\bar{\mathbf{w}}_s = [\bar{f}_x, \bar{f}_z, \bar{m}_y]^T$ is a unit wrench that satisfies the ellipsoidal limit surface constraint:

$$\frac{\bar{f}_x^2}{1} + \frac{\bar{f}_z^2}{1} + \frac{\bar{m}_y^2}{(rc)^2} = 1 \quad (2.13)$$

Constraints (2.10) and (2.13) can be solved together analytically to find $\bar{\mathbf{w}}_s$ and k . Specifically, after substituting \bar{f}_x, \bar{f}_z , and \bar{m}_y from (2.10) into (2.13), (2.13) becomes a quadratic polynomial equation in k . Solving this quadratic equation with a constraint $k \in \mathbb{R}^+$ gives a unique solution for k . Substituting this value for k in (2.10) turns it into a set of three linear equalities with three unknowns $[\bar{f}_x, \bar{f}_z, \bar{m}_y]$ which can be solved for a unique solution. For prehensile pushing in the gravity plane, the relationship between \mathbf{W}_s and $\mathbf{W}_{\text{pusher}}$ is not linear as in the gravity-free case. To find the wrench-set \mathbf{W}_s , we need to sweep $\bar{\mathbf{w}}_{\text{pusher}}$ over the boundary of $\mathbf{W}_{\text{pusher}}$ and solve (2.10) and (2.13) iteratively.

Fig.2.8 illustrates the process for solving the constraints (2.10) and (2.13) together to compute the wrench cone \mathbf{W}_s and further map the wrench-set \mathbf{W}_s to the motion cone \mathbf{V}_{obj} , using equation (2.5). Intuitively we can read Fig.2.8 from left to right as follows:

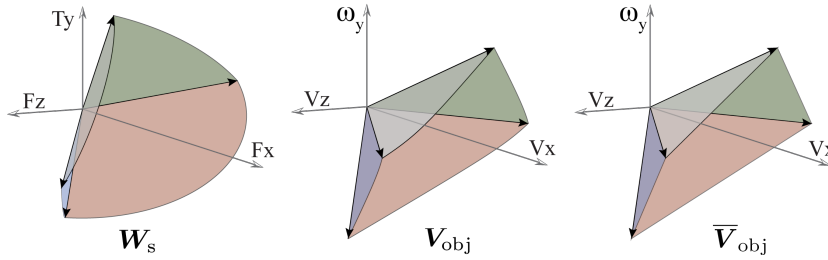


Figure 2.9: The analytically computed wrench cone (W_s), motion cone (V_{obj}), and polyhedral approximation to the motion cone (\bar{V}_{obj}) for the configuration in Fig.2.3 and 45 N grasping force. Note that the surfaces defining the motion cone (V_{obj}) are curved.

1. **Force Modelling** step involves formulating the set of forces present in the manipulation plane using the contact modelling techniques discussed in Sec. 2.3.1 and Sec. 2.3.2. The limit surface represents the set of friction wrenches that the support contacts can offer to prevent the motion between the object and the support surface. The generalized friction cone of the pusher is the set of wrenches the pusher can impose on the object. Together with other external forces in the manipulation plane, e.g., the gravitational component, it represents the set of external wrenches that act on the object.
2. **Force Resolution** refers to solving for the quasi-static force balance using (2.10) and (2.13). Solving for force balance means finding the subset W_s of the support limit surface that balances the set of external wrenches.
3. **Motion Resolution** interprets the set of object twists that corresponds to the force-balance solution set using (2.5). Graphically, this represents drawing normals to the subset W_s computed in the earlier step, to get the corresponding set of object twists V_{obj} . This set is the motion cone.

Fig.2.8 (top) shows the computation of the motion cone for pushing an object on a horizontal support plane with a point pusher. As discussed in Sec. 2.4.1, from (2.11), when there is no external wrench other than the pusher wrench acting on the object, the motion cone computation simplifies and W_s and V_{obj} are polyhedral cones. In Fig.2.8 (top) we observe that the motion cone is a polyhedral cone bounded by two twists v_1 and v_2 .

In more general cases where additional external forces, such as the gravitational force, are present in the manipulation plane, W_s and V_{obj} are not polyhedral cones as shown in Fig.2.8 (bottom). In a general planar pushing task, they can be well characterized as cones defined by low-curvature surfaces that intersect all in a point and pairwise in lines. Fig.2.9 shows the wrench cone W_s and motion cone V_{obj} computed analytically for one particular grasp-pusher configuration.

2.5.2 Polyhedral Approximation to the Motion Cone

As an object is pushed in a grasp, the position of the finger contacts in the object frame change, and consequently J_s and the motion cone V_{obj} also

change. We need to compute the motion cone iteratively as the object moves in the grasp, which requires computational efficiency.

As the boundary surfaces of the motion cone have low curvatures, we propose a polyhedral approximation (\bar{V}_{obj}) of the motion cone for efficient computation. Each edge of the polyhedral approximation of the motion cone is the object motion corresponding to an edge of the generalized friction cone of the pusher. Fig.2.8 and Fig.2.9 show example polyhedral approximations of the motion cone.

Procedure to compute polyhedral motion cone:

1. Solve (2.10) and (2.13) simultaneously to get \bar{w}_s for \bar{w}_{pusher} corresponding to every edge of W_{pusher} .
2. Define the set of \bar{w}_s computed in step 1 as the generators/edges of the support/grasp wrench-cone \bar{W}_s .
3. Map \bar{W}_s to the object twist space using (2.5) to get the polyhedral approximation \bar{V}_{obj} of the motion cone.

2.5.3 Experimental Validation of the Motion Cone

Fig.2.10 shows our manipulation platform equipped with an industrial robot arm (ABB IRB120), a parallel-jaw gripper (WEISS WSG32), two geometric features in the environment that act as pushers, and a Vicon system (Bonita) for accurate object tracking.

The theory from Sec. 2.4, states that the pusher sticks to the object for the pusher twists inside the motion cone while the twists outside the motion cone will result in slipping at the pusher contact. To evaluate the experimental validity of the polyhedral approximation of the motion cone, we collected data for the slip observed at the pusher contact for a variety of experimental settings.

Using the rectangular prism object listed in Table 2.12, we conducted six experiments across three configurations as shown in the leftmost column of Table 2.2. The first two configurations use the side pusher and the grasp is offset from the center of mass of the object by 0 mm and -10 mm respectively in the X direction. The third configuration uses a bottom pusher and the grasp is at the center of mass of the object.

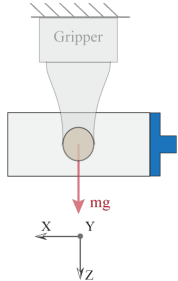
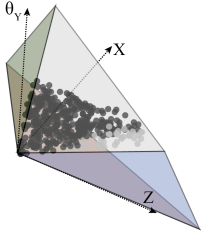
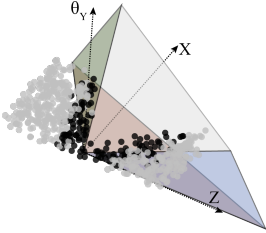
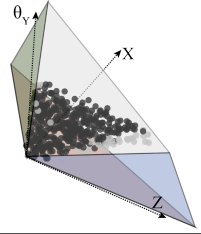
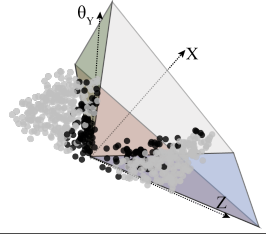
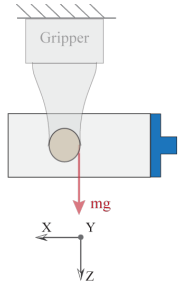
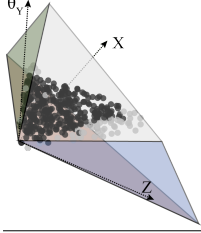
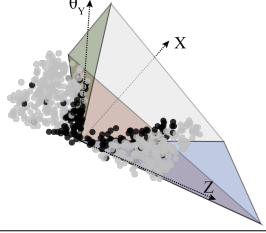
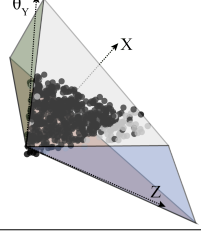
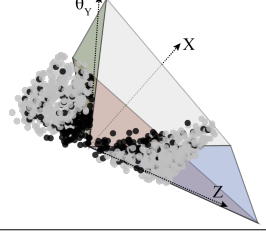
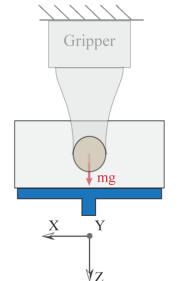
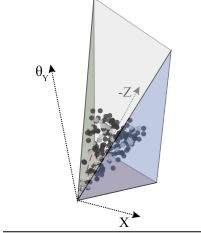
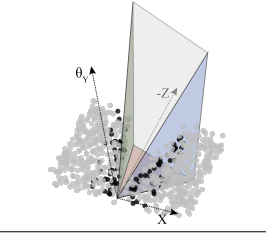
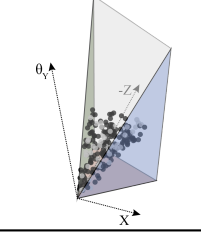
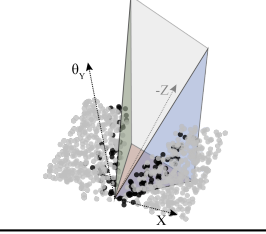
Experimental Set Up	Inside the Cone	Outside the Cone	Prediction Accuracy	
			No Slip	Slip
			90.2%	77.9%
			90.9%	77.6%
			82.9%	76.1%
			89.1%	73.0%
			76.5%	88.0%
			73.7%	85.0%

Table 2.2: We experimentally validate the polyhedral approximation of the motion cone across three configurations by characterizing thousands of random prehensile pushes by the slip observed at the pusher contact. The pushes that did slip are shown in grey, while those that did not are shown in black. For each configuration, we utilize a grasping force of 35 N (top) and 45 N (bottom). The percentages denote how many pushes lie inside the the motion cone that did not slip and how many pushes outside the cone that did slip.

For each configuration, we conducted two experiments, first with a grasping force of 35 N and then with 45 N. In Table 2.2, the results are shown on the top and bottom, respectively, for each configuration. For each experiment we executed 1000-1500 randomly sampled pushes and labeled each push as ‘slipping’ if the object moved with respect to the pusher/environment by more than 1.5 millimeters.

Due to kinematic constraints of the robot and the workspace, we limit the sampled pushes to the space of $[0, 10]$ mm, $[-5, 5]$ mm, $[-35, 35]^\circ$ displacements in $[X, Z, \theta_Y]$ when using the side pusher and $[-10, 10]$ mm, $[-8, 0]$ mm, $[-35, 35]^\circ$ when using the bottom pusher.

For each experiment, Table 2.2 lists two accuracy percentages. The first denotes percentage of pushes that lie inside the motion cone that did indeed stick, as predicted. The second denotes the percentage of pushes that lie outside the motion cone that did slip, as predicted. The analytically computed polyhedral approximation to the motion cone classifies the stick/slip at the pusher contact and characterizes the feasibility of the object motions with an average accuracy of 80%+ accuracy.

We observe that the mislabeled pushes are close to the boundaries of the motion cones. There could be a few reasons for these inaccurate predictions. The variation in the measured friction at the fingers or pusher contacts and variation in the grasping force can make the predictions close to the boundaries of the motion cone inaccurate. The motion cone construction builds upon the approximations of friction models at finger as well as pusher contacts and rigid-body assumptions. Therefore any unmodeled dynamics and compliance at the contacts affect its validity, especially close to the boundary of the motion cone. Sec. 2.8 explores different strategies to make the predictions more reliable.

2.6 Planning In-Hand Manipulations via Motion Cones

Motion cones abstract the algebra involved in solving the mechanics of pushing and provide direct bounds on the feasible object motions with pushing. For pusher motions inside the motion cone, the contact between the pusher and the object remains sticking, and the object follows the pusher. For pusher motions outside the motion cone, there is slipping at the pusher contact and the object motion is on the boundary of the motion cone. This mapping between the motions of the pusher and the object along with sticking/slipping contact mode resolution can be used for fast and efficient planning of pushing tasks.

In this section, we demonstrate the application of motion cones for planning in-hand manipulations with prehensile pushes. Particularly, we will constrain our planner to build manipulation strategies for which the pusher

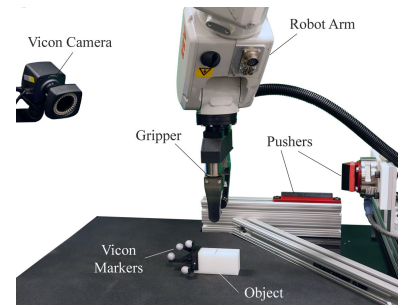


Figure 2.10: Experimental setup used for the data collection for motion cone validation as well as for the regrasp examples in Sec. 2.7 and Sec. 2.8.

contact sticks to the object. Lynch and Mason (1996) studied pushing motions for which the pusher sticks to the object when pushing on a horizontal surface, which they referred to as *Stable Pushing*. We call the equivalent prehensile version *Stable Prehensile Pushing* (Chavan-Dafle and Rodriguez, 2018).

2.6.1 Problem Formulation

An object is grasped in a parallel-jaw gripper and manipulated in the gravity plane by pushing it against contacts in the environment.

For the problem setup, we assume the following information about the manipulation system:

- Object geometry and mass.
- Initial and goal grasp of the object, specified by the locations of the finger contacts.
- Gripping force.
- Set of possible pusher contacts in the environment, specified by their geometries and locations in the object frame.
- Coefficients of friction of all contacts.

2.6.2 Planner Framework

Our proposed planning framework works at two levels. At the high-level, a T-RRT*-based planning architecture samples the configuration space of different grasps, similar to earlier work in Chavan-Dafle and Rodriguez (2018). At the low level, the planning tree is grown in the direction of the sampled grasp poses using the knowledge of locally reachable grasp poses in the form of motion cones.

For selective exploration, the T-RRT* framework relies on a transition test that filters the sampled configurations to prefer exploration in low configuration-cost regions (Jaillet et al., 2010; Devaurs et al., 2016). We define the configuration cost as the distance of the grasp from the goal. The transition test guides the stochastic exploration towards the goal grasp, while allowing the flexibility to explore high-cost transitions if they are necessary to get the object to the goal.

For effective connections, the T-RRT* algorithm uses the underlying RRT* (Karaman and Frazzoli, 2011) framework to make and rewire the connections in the tree at every step such that the local cost of the nodes is reduced when possible. We define the cost of a node as the sum of the cost of the parent node and the cost of the push to reach the sampled node from the parent node. We set the cost of a push to 0.1 if the parent node uses the same pusher as the child and 1 if there is a pusher switch-over. With our node cost definition, the planner generates pushing strategies that prefer fewer pusher switch-overs to push the object to the desired pose.

Algorithm 1 : In-Hand Manipulation Planner

```

input :  $q_{init}, q_{goal}$ 
output : tree  $\mathcal{T}$ 
 $\mathcal{T} \leftarrow$  initialize tree( $q_{init}$ )
generate_motionCones( $\mathcal{T}, q_{init}$ )
while  $q_{goal} \notin \mathcal{T}$  or  $\text{cost}(q_{goal}) > \text{cost threshold}$  do
   $q_{rand} \leftarrow$  sample random configuration( $\mathcal{C}$ )
   $q_{parent} \leftarrow$  find nearest neighbor( $\mathcal{T}, q_{rand}$ )
   $q_{sample} \leftarrow$  take unit step( $q_{parent}, q_{rand}$ )
  if  $q_{sample} \notin \mathcal{T}$  then
    if transition test( $q_{parent}, q_{sample}, \mathcal{T}$ ) then
       $q_{new} \leftarrow$  motionCone_push( $q_{parent}, q_{sample}$ )
      if transition test( $q_{parent}, q_{new}, \mathcal{T}$ ) and
        grasp maintained( $q_{new}$ ) then
           $q_{parent}^* \leftarrow$  optimEdge( $\mathcal{T}, q_{new}, q_{parent}$ )
          add new node( $\mathcal{T}, q_{new}$ )
          add new edge( $q_{parent}^*, q_{new}$ )
          generate_motionCones( $\mathcal{T}, q_{new}$ )
          rewire tree( $\mathcal{T}, q_{new}, q_{parent}^*$ )

```

Let q denote the configuration of the object, i.e., the pose of the object in the gripper frame, which is fixed in the world. In this work, we consider planar manipulations in a parallel-jaw grasp, so the configuration space \mathcal{C} is $[X, Z, \theta_y] \in \mathbb{R}^3$, i.e., the object can translate in the grasp plane (XZ) and rotate about a perpendicular (Y) to the grasp plane.

Algorithm 1 describes our in-hand manipulation planner. Let q_{init} and q_{goal} be an initial and desired pose of the object in the gripper frame respectively. The planner initiates a tree \mathcal{T} with q_{init} and generates motion cones at q_{init} .

While the desired object pose is not reached within some cost threshold, a random configuration q_{rand} is sampled. A nearest configuration q_{parent} to q_{rand} in the tree \mathcal{T} is found and an unit-step object pose q_{sample} from it towards q_{rand} is computed. Using the transition test, the planner evaluates if moving in the direction of q_{sample} from q_{parent} is beneficial. If it is, the *motionCone_push* routine computes an object configuration q_{new} closest to q_{sample} that can be reached using the motion cones at q_{parent} . It is further checked if moving towards q_{new} is beneficial and if q_{new} is an object configuration at which the grasp on the object is maintained. If both the criteria are satisfied, q_{new} is added to the tree such that the local costs of q_{new} and the nodes near q_{new} are reduced. The motion cones are generated online for every new node added to the tree.

Two important routines in Algorithm 1, particularly for this work, are *generate_motionCones* and *motionCone_push*.

generate_motionCones computes the polyhedral motion cones for a given

object configuration in the grasp for all possible external pushers using the procedure listed in Sec. 2.5.2. At every node, we will have the same number of motion cones as possible pushers and each motion cone represents the set of object configurations that can be reached from the current configuration using a specific pusher.

motionCone_push finds an object pose closest to the desired sampled pose q_{sample} that is feasible to reach with pushing. This computation is done using the motion cones at the parent node q_{parent} . If the object twist needed from the parent node pose to the sampled pose is already inside any of the motion cones at q_{parent} , the sampled pose can be reached. If the required object twist is outside all the motion cones at q_{parent} , a twist that is closest to the desired twist and inside one of the motion cones is selected by projecting the sampled twist on the motion cones as shown in Fig.2.11.

Motion cones define the set of feasible pushes, or in other words the set of reachable object configurations. Using motion cones as reachability bounds directly in the configuration space allows us to rapidly explore the configuration space of different object poses in the grasp and generate dynamically feasible pushing strategies for the desired in-hand regrasp.

2.7 Regrasp Examples and Experimental Results

We evaluate the performance of our proposed planner with examples of a parallel-jaw gripper manipulating a variety of objects listed in Table 2.12. The initial pose of an object in the gripper is treated as $[X, Z, \theta_Y] = [0, 0, 0]$. Table 2.13 lists the goal poses (in [mm, mm, deg.]) for different examples. While there are no comparable available algorithms that can solve the type of regrasps we are interested in, we provide comparisons with our own implementations of the same high-level planner paired with different approaches to solve the mechanics of prehensile pushing. These include sampling with rejection by a feasibility check for stable prehensile pushing (Chavan-Dafle and Rodriguez, 2018), and a more classical complementarity formulation (MNCP) that allows both sticking and slipping at the pusher contact (Chavan-Dafle and Rodriguez, 2020). We compare the performance in terms of planning time and the quality of the solutions. The planning times in Table 2.13 are the median times over 10 trials. All the computations are done in MATLAB R2017a on a computer with Intel Core i7 2.8 GHz processor.

In all the examples below, we assume three line pushers on the object, one on each side faces of the object parallel to the Z axis and one under the object parallel to the X axis. We use high friction line pushers, except in the first example, where we use low friction line pushers (Fig.2.14).

Shape	Material	Dim [L, B, H] (mm)	Mass (g)
square prism	Al 6061	100, 25, 25	202
rect. prism	Delrin	80, 25, 38	113
T-shaped	ABS	70, 25, 50	62

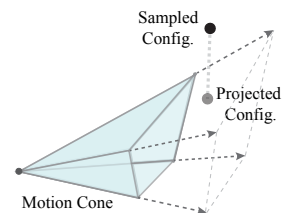


Figure 2.11: Since the sampled grasp configuration is outside the motion cone, it projected on the motion cone. The projected configuration is selected to grow the planning tree.

Table 2.12: Physical properties of the experimental objects

Manipulation	Goal [X, Z, θ_Y]	Motion Cone	Stable Check	MNCP
Horz. offset	20, 0, 0	0.45	2.83	592.8
[X, Z, θ_Y] regrasp	15, -13, 45	0.67	2.54	17684
T-shaped	25, 17.5, 0	0.54	0.82	32657

Table 2.13: Planning times (sec.) for approaches using motion cone, stable check [Chavan-Dafle and Rodriguez \(2018\)](#) and MNCP [Chavan-Dafle and Rodriguez \(2020\)](#) for unit-step propagation

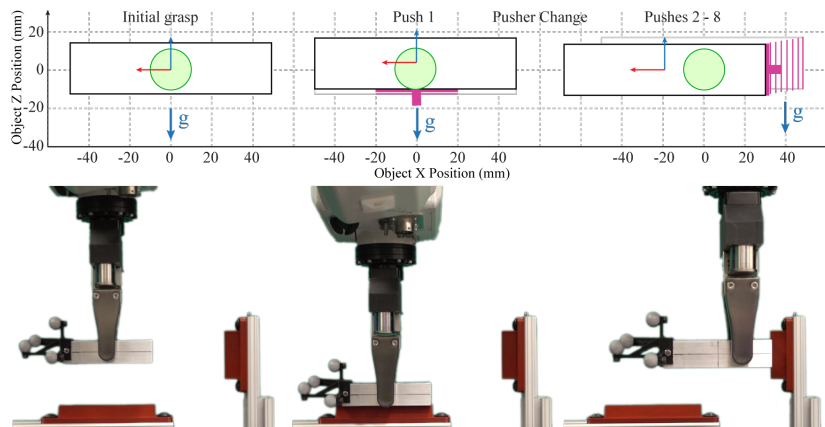


Figure 2.14: Simulation and experimental run for a pushing strategy to regrasp the aluminum object with low friction pushers. In the simulation figure (top), the finger and pusher contacts are shown in green and magenta color respectively.

2.7.1 Regrasping an object offset to the center

In this example, the goal is to regrasp the square prism horizontally 20 mm offset from the center. We use low friction pushers first. [Kolbert et al. \(2017\)](#) showed that for a similar setting, if the object is pushed horizontally in the grasp it slides down as it moves sideways in the grasp. For low-friction pushers, our planner generates a strategy where the object is first pushed up using the bottom pusher and then the side pusher is used to virtually keep the object stationary while the fingers slide up and along the length of the object as seen in Fig.2.14. This plan is similar to the one found in [Chavan-Dafle and Rodriguez \(2020, 2018\)](#).

When we replace the pushers with high-friction pushers (pushers with rubber coating), the planner estimates that the desired horizontal object motion lies inside the motion cone for the side pusher at the initial and all the subsequent grasp poses until the goal, i.e, simply pushing from the side is a valid pushing strategy as shown in Fig.2.15.

2.7.2 Regrasp in [X, Z, θ_Y]

The goal in this example is to regrasp the rectangular prism requiring a net object motion in all three dimensions [X, Z, θ_Y]. Similar to [Chavan-Dafle and Rodriguez \(2018\)](#), our planner finds a strategy to achieve the regrasp using only one pusher. In fact, as we can see in Fig.2.16, the pushing strategy our planner generates is smoother and seems to avoid unnecessary object motions seen in the strategy shown in [Chavan-Dafle and Rodriguez \(2018\)](#)-Fig.1.

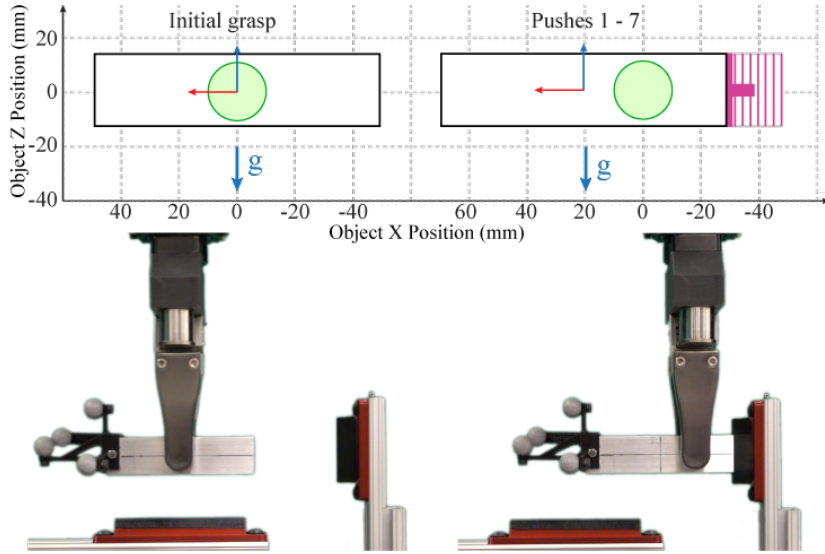


Figure 2.15: Simulation and experimental run for a pushing strategy to regrasp the aluminum object with high friction pushers.

2.7.3 Manipulating a non-convex object

In this example, the goal is to regrasp a T-shaped object. The goal pose is such that a greedy approach to push the object directly towards the goal will result in losing the grasp on the object. As shown in Fig.2.17, our planner comes up with a pushing strategy that respects the geometric constraints of the problem and moves the object to the desired pose without losing the grasp.

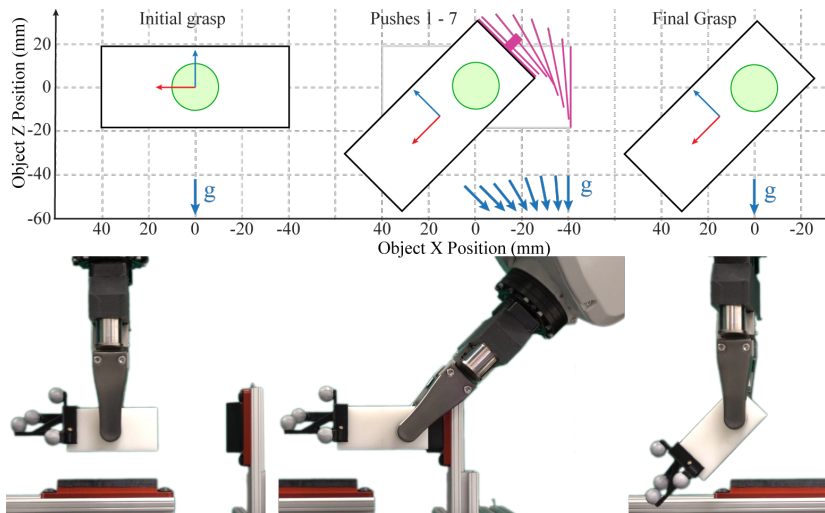


Figure 2.16: A pushing strategy for $[X, Z, \theta_y]$ regrasp. In simulation, the direction of gravity remains constant in the pusher frame, because in real experiments, the pushers are fixed features in the environment.

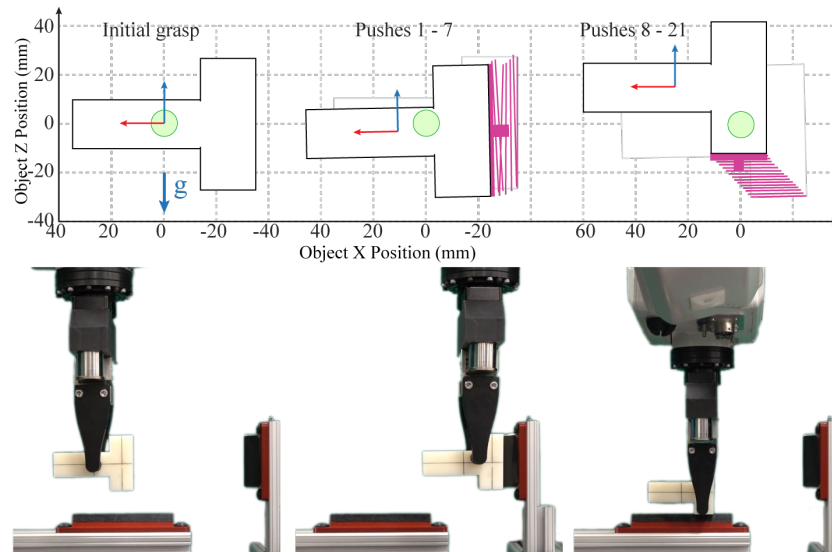


Figure 2.17: Simulation and experimental run to manipulate a T-shaped object. Snapshots of the experimental run are shown in Fig.2.2

2.8 Robust In-Hand Manipulations via Motion Cones

Sec. 2.6 and Sec. 2.7 describe an application of motion cones to plan in-hand manipulations with stable prehensile pushes. In this section, we discuss constraining the planning tree to propagate through a conservative interior of motion cones for increased robustness. We show that we can exploit the structure of motion cones to derive manipulation strategies that are robust against uncertainties in the friction parameters at the finger and pusher contacts, grasping force, and mass of the object.

2.8.1 Robustness to friction variation at the pusher-object contact

For a given object mass, given friction at the finger contacts, and given grasping force, the motion cone is a function of the friction at the pusher. For a higher friction coefficient, the generalized friction cone is wider and so is the motion cone. There is a monotonic relationship between the friction at the pusher and the volume of the motion cone. This relationship can be exploited to derive pushing strategies robust to uncertainty in the friction at the pusher.

Proposition 2. *Any object velocity inside the motion cone for a given pusher friction coefficient is inside the motion cone for a higher friction coefficient at the pusher.*

Proof. Let W_{pusher} be the generalized friction cone of a pusher and let W_{pusher}^+ be the generalized friction cone for the same pusher but with a higher coefficient of friction. From the definition of the generalized friction cone in (2.6), it is a linear mapping of the Coulomb friction cone/s at the constituent contact/s. A Coulomb friction cone for a given coefficient of friction is a subset of a friction cone with a higher coefficient of friction. Therefore, the generalized friction cone is

also the subset of a generalized friction cone for a higher coefficient of friction:

$$\mathbf{W}_{\text{pusher}} \subset \mathbf{W}_{\text{pusher}}^+ \quad (2.14)$$

If an object motion is inside the motion cone, the net required wrench for the object motion lies inside the generalized friction cone of the pusher, i.e., repeating (2.12):

$$-\mu_s N \mathbf{J}_s^\top \cdot \bar{\mathbf{w}}_s - m\mathbf{g} \in \mathbf{W}_{\text{pusher}} \quad (2.15)$$

From (2.14) and (2.15),

$$-\mu_s N \mathbf{J}_s^\top \cdot \bar{\mathbf{w}}_s - m\mathbf{g} \in \mathbf{W}_{\text{pusher}}^+ \quad (2.16)$$

Since the net required wrench lies inside the friction cone of the pusher, the object motion is inside the motion cone. ■

If there is uncertainty in the friction coefficient at the pusher, we can obtain a robust strategy by planning with the lower bound on the coefficient. The resulting strategy will produce the same net object motion in the grasp even if the true friction coefficient at the pusher is higher.

This property is also verified by our experiments from Sec. 2.5.3. For our experiment where we offset the grasp -10 mm in the x -direction from the center of mass of the object and use a grasping force of 45-N, our estimated friction coefficient at the pusher is 0.5. Given this coefficient, as stated in Table 2.2, 89.1% of the pushes within the computed motion cone stick, as predicted. If we compute our motion cone with a coefficient of 0.25, then 95.6% of the pushes lie within this reduced cone.

2.8.2 Robustness to friction variation at the gripper-object contact

For a given friction coefficient at the pusher, the motion cone is a function of the object mass and the friction coefficient at the finger contacts, and the grasping force.

Unlike the monotonic relationship of the motion cone to the friction at the pusher, the dependence of the motion cone on the friction force at the fingers is not straight-forward. For increased friction at the fingers, either by increasing the grasping force or by increasing the coefficient of friction, we do not necessarily get a wider motion cone. Instead the motion cone shifts towards the gravity-free motion cone as shown in Fig.2.18³. Therefore, some of the object motions inside the motion cone, particularly those that are feasible by exploiting gravity, may no longer be feasible.

We propose two approaches to address this problem and to achieve robust pushing strategies under the uncertainty in the friction at the fingers.

Modified Motion Cone. In this approach, we find a subset of a motion cone such that any object motion inside it will always be feasible with any friction at the gripper higher than a chosen value.

³ At infinite friction at the gripper the motion cone is equal to the the gravity-free motion cone as shown in Corollary 1.1

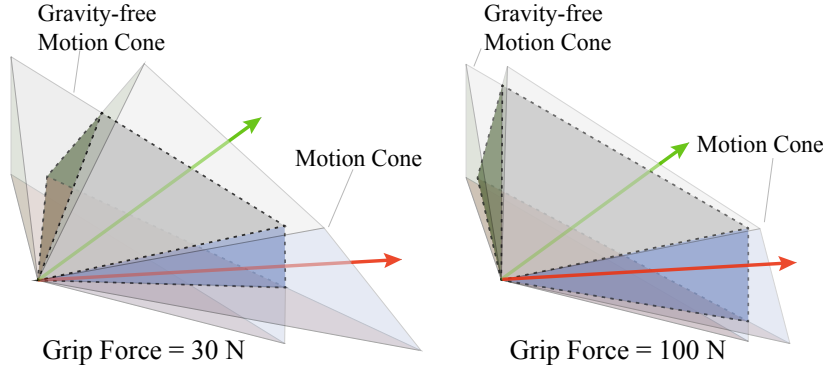


Figure 2.18: The motion cone shifts towards the gravity-free motion cone as the friction at the gripper increases due to the increased grip force. The intersection of the gravity-free motion cone and the motion cone is shown by the dotted region for the two grip forces. The motions inside the motion cone that are outside the gravity-free motion cone are feasible only by exploiting the gravitational force on the object (in Z direction). For higher friction at the gripper than expected, these motions may no longer be feasible. One of such motions that is shown in red falls outside the motion cone when the grip force is increased. All the motions inside the intersection however, will always be feasible for the higher than expected friction at the gripper.

Proposition 3. *Any motion inside the intersection of the gravity-free motion cone and the motion cone, for a given friction at the gripper, will always be inside the motion cone for a higher friction at the fingers or/and a higher grasping force.*

Proof. If an object motion is inside the motion cone, from (2.12), the net required wrench is inside the generalized friction cone of the pusher:

$$-\mu_s N J_s^T \bar{w}_s - m\mathbf{g} \in \mathbf{W}_{\text{pusher}}. \quad (2.17)$$

If the object motion is inside the gravity-free motion cone:

$$-\mu_s N J_s^T \bar{w}_s \in \mathbf{W}_{\text{pusher}}$$

or simply,

$$-J_s^T \bar{w}_s \in \mathbf{W}_{\text{pusher}} \quad (2.18)$$

Let μ_s^+ be a higher friction coefficient at the fingers and N^+ be a higher grasping force. The net wrench required for the same object motion under the higher finger friction conditions is:

$$-\mu_s^+ N^+ J_s^T \bar{w}_s - m\mathbf{g}.$$

Which can be rewritten as:

$$-k J_s^T \bar{w}_s - \mu_s N J_s^T \bar{w}_s - m\mathbf{g}, \quad k \in \mathbb{R}^+. \quad (2.19)$$

From (2.17), (2.18), and (2.19):

$$-k J_s^T \bar{w}_s - \mu_s N J_s^T \bar{w}_s - m\mathbf{g} \in \mathbf{W}_{\text{pusher}}$$

Since the net required wrench lies inside the friction cone of the pusher, the object motion is inside the motion cone. ■

In the case of uncertainty in the friction at the gripper, to generate robust pushing strategies, we can constrain the planner to the intersection motion cones computed with lower bounds on the friction coefficients and the grasping force. Such a pushing strategy will produce the same regrasp outcome for the expected variation in the friction at the gripper.

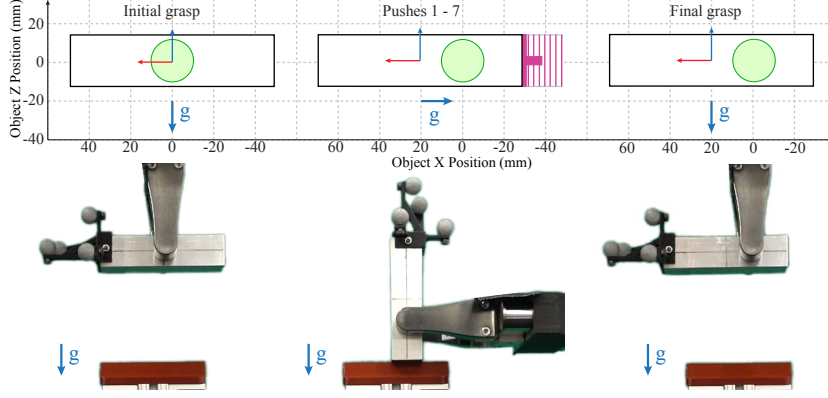


Figure 2.19: Simulated motion of the object and snapshots of the experimental run for a pushing strategy to offset the object in the grasp using low coefficient pushers.

Gravity-aligned Pusher. Recent work showed that we can generate robust prehensile pushes in the presence of gravitational force by careful selection of pusher contact placement [Chavan-Dafle and Rodriguez \(2018\)](#). Specifically, we demonstrate that by aligning the pusher normal along the gravitational force on the object, the dynamics of pushing and the motion cone becomes invariant to all the system parameters except for the friction at the pusher.

Proposition 4. *The motion cone for gravity-aligned pushers is invariant to the object mass, grasping force and friction coefficient at the finger contacts.*

Proof. If the pusher is aligned such that the pusher contact normal is along the direction of gravity, the gravitational force on the object is entirely balanced by the part of the normal force at the pusher. Then, for $\bar{w}_{\text{pusher}} \in \mathcal{W}_{\text{pusher}}$, the dynamic check for a feasible prehensile push (2.9) becomes:

$$\begin{aligned} -\mu_s N J_s^T \cdot \bar{w}_s - mg &= k \bar{w}_{\text{pusher}} \\ -\mu_s N J_s^T \cdot \bar{w}_s - k_2 \bar{w}_{\text{pusher}_n} &= k \bar{w}_{\text{pusher}} \\ -\mu_s N J_s^T \cdot \bar{w}_s &= k \bar{w}_{\text{pusher}} + k_2 \bar{w}_{\text{pusher}_n} \end{aligned}$$

where $k, k_2 \in \mathbb{R}^+$ and $\bar{w}_{\text{pusher}_n}$ is the unit normal wrench at the pusher contact. Since $\bar{w}_{\text{pusher}_n} \in \mathcal{W}_{\text{pusher}}$, we can rewrite the previous constraint as:

$$-\mu_s N J_s^T \cdot \bar{w}_s = k_3 \bar{w}_{\text{pusher}} \quad k_3 \in \mathbb{R}^+$$

where \bar{w}_{pusher} is different from before, but also in $\mathcal{W}_{\text{pusher}}$. As μ_s and N are scalar constants, we further simplify it as:

$$-J_s^T \cdot \bar{w}_s = k_4 \bar{w}_{\text{pusher}} \quad k_4 \in \mathbb{R}^+ \quad (2.20)$$

Now, we can write the dynamics condition for a feasible prehensile pushing with gravity balancing pushers as:

$$-J_s^T \cdot \bar{w}_s \in \mathcal{W}_{\text{pusher}} \quad (2.21)$$

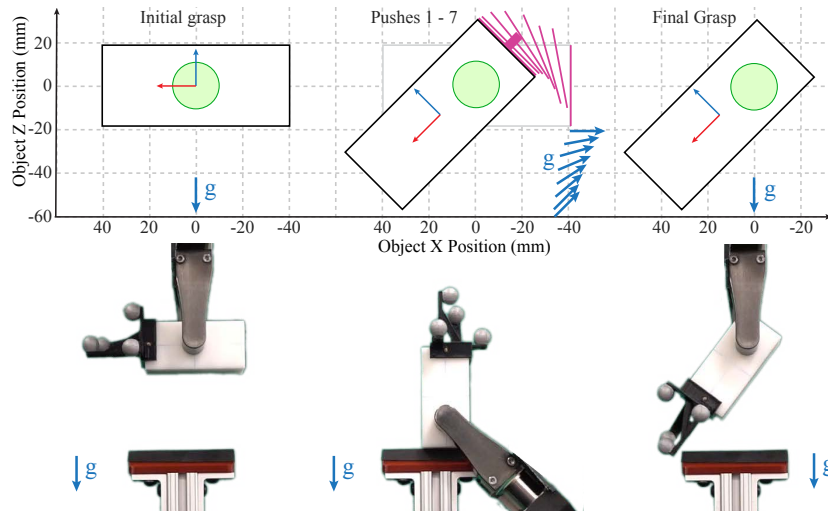


Figure 2.20: Simulated motion of the object and snapshots of the experimental run for a general regrasp in $[X, Z, \theta_y]$.

Equations (2.20) and (2.21) show that the dynamics of pushing, and the consequent motion cone for the gravity-aligned pusher, are invariant to the object mass, grasping force, and the friction coefficient at the fingers. ■

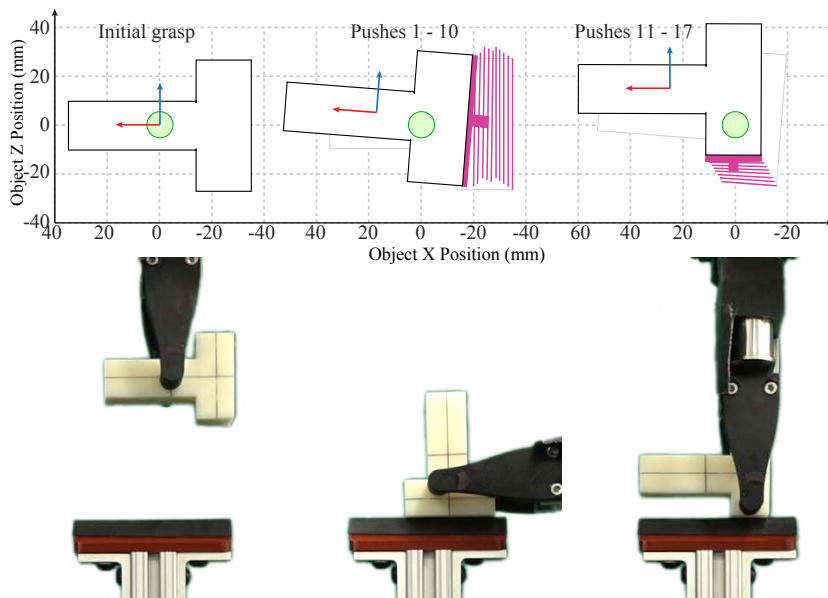


Figure 2.21: Simulated motion of the object and snapshots of the experimental run for the T-shaped object.

As discussed in Sec. 2.8.1, we can further make the pushing strategies robust to the uncertainty at the pusher contact by planning with lower bounds on the expected coefficient of friction at the pusher.

Fig.2.19, Fig.2.20, and Fig.2.21 show the regrasps in Table 2.13 planned with the constraint of using only the bottom pusher which is aligned along the direction of gravity.

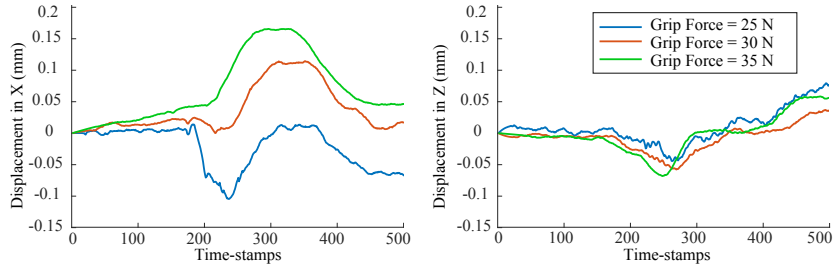


Figure 2.22: Displacement of the object with respect to the environment for the regrasp action shown in Fig. 2.19. As expected, during pushing, the object sticks to the environment and moves by a negligible amount as the fingers slide on it.

To validate the invariance in the outcome of the regrasp actions planned when the friction at the fingers is changed, we varied the grasping force and executed the same pushing strategy. As the grasping force changes, the friction force at the fingers change. However, as shown in the plots in Fig. 2.22 and Fig. 2.23, the variation in the outcome of the pushing strategy for the three different grasping forces is negligible. This supports the expected invariance. Moreover, the minimal slip observed at the pusher contacts confirms that the pushing actions selected by the planner result in stable prehensile pushes.

It should be noted that using conservative friction values at the pusher as proposed in Sec. 2.8.1 or modified motion cones as in Sec. 2.8.2 provides robustness, however, it reduces the volume of the effective motion cones, i.e., the dynamically-reachable configuration space. This can adversely affect planning time, especially for problems with "narrow-corridors". The method proposed in this section however, does not suffer from such an effect since the volume of the motion cones in this case does not change significantly. We observe negligible (fractions of a second) difference in the planning times for the examples considered in this section compared to those in Sec. 2.7.

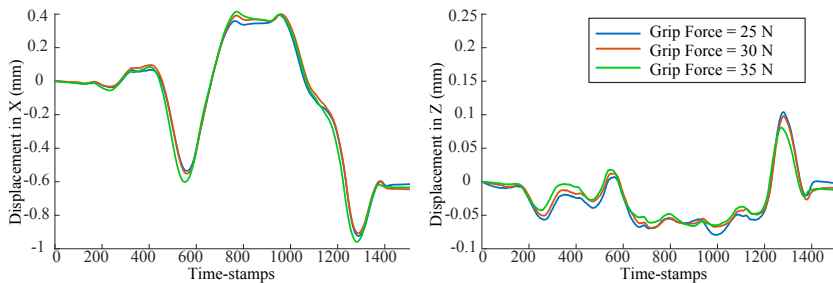


Figure 2.23: Displacement of the object with respect to the environment for the regrasp action shown in Fig. 2.20.

2.9 Discussion

A motion cone is the maximal set of dynamically feasible velocities that a pusher can transmit to a pushed object. It also describes the set of motions of the pusher that yield sticking contact with the object during the push. It

abstracts away the dynamics of frictional pushing and provides direct bounds on the reachable space for pushing tasks.

In this work we extend the concept of motion cones to a general set of planar pushing tasks with external forces such as the gravitational force in the plane of motion. We show that the motion cone for a general planar push is defined as a cone with low-curvature facets, and propose a polyhedral approximation for efficient computation.

We demonstrate the use of motion cones as the propagation step in a sampling-based planner for in-hand manipulation. Combining a T-RRT*-based high level planning framework and a *motion cone*-based dynamics propagation, the planner builds in-hand manipulation strategies with sequences of continuous prehensile pushes in a fraction of a second. Furthermore, we show that by constraining the planner to propagate the planning tree through a conservative interior of the motion cones and/or with specific selection of the pusher alignment, we can generate pushing strategies that are robust against uncertainty in the physical parameters of the system.

Throughout this chapter, we emphasize the experimental validation of the theoretical contribution and planning framework. Thousands of pushes in different object-grasp-pusher configuration corroborate the analytical computation of motion cones. With a variety of regrasp experiments we show that the pushing strategies generated by our planner result in motions with minimal slip at the pusher contact and negligible variation in the outcome of the regrasp.

The motion cone provides direct knowledge of the set of reachable configurations. It allows us to explore the connectivity of the configuration space for planning through regions/volumes of the configuration space for contact-rich manipulations (Brock and Kavraki, 2001; Morales et al., 2007; Shkolnik et al., 2009; Shkolnik and Tedrake, 2011). Moreover, motion cones, as bounds on pusher actions or as bounds on the object motions, have an adequate form to be incorporated into trajectory optimization frameworks to plan pushing strategies. We believe that the extension and application of motion cones to more general settings provides new opportunities for fast and robust manipulation through contact.

3

Robust Planning for Multi-stage Forceful Manipulation

3.1 Introduction

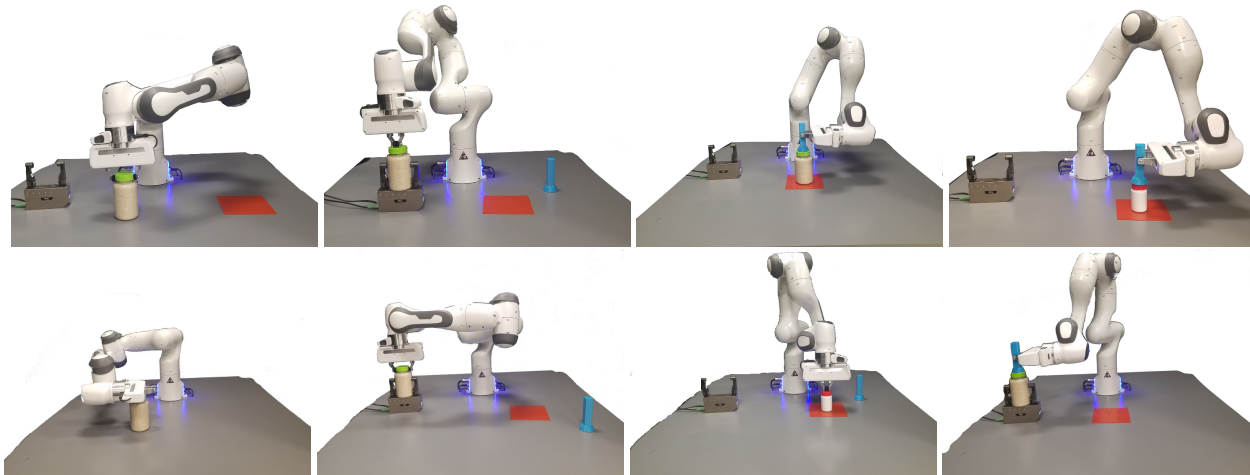
Our goal is to enable robots to plan and execute *forceful manipulation tasks* such as cutting a vegetable, opening a push-and-twist childproof bottle, and twisting a nut. While all tasks that involve contact are technically forceful, we refer to forceful manipulation tasks as those where the ability to generate and transmit the necessary forces to objects and their environment is an active limiting factor which must be reasoned over and planned for. Respecting these limits might require a planner, for example, to prefer a robot configuration that can exert more force or a grasp on an object that is more stable.

Forceful operations, as defined by (Chen et al., 2019), are the exertion of a wrench (generalized force/torque) at a point on an object. These operations are intended to be quasi-statically stable, i.e. the forces are always in balance and produce relatively slow motions, and will generally require some form of fixturing to balance the applied wrenches. For example, to open a push-and-twist childproof bottle the robot must exert a downward force on the cap while applying a torque along the axis of the bottle. The robot must be in a configuration that allows it to apply enough force to accomplish the task, and also securing the bottle to prevent it from moving during the task.

To accomplish these complex, multi-step forceful manipulation tasks, robots need to make discrete decisions, such as, for example, whether to push on the bottle cap with the fingers, the palm or a tool, and whether to secure the bottle via frictional contact with a surface, with another gripper or with a vise. The robot must also make continuous decisions such as the choice of grasp pose, robot configurations and robot trajectories. Critically, all these decisions interact in relatively complex ways to achieve a valid task execution.

Fig.3.1 illustrates that there are *different strategies* for completing the task of opening a push-and-twist childproof bottle. Each strategy's viability depends on the robot's choices and on the environment. For example, a strategy that uses the friction from the table to secure the bottle, as shown

This research has been published at ICRA 2021 and IJRR 2023 (Holladay et al., 2021, 2023).



in the top leftmost corner, would fail if the table can only provide a small amount of friction. Instead, the robot would need to find a significantly different strategy, such as securing, or fixturing, the bottle via a vise, as shown in the top center-left.

Choosing a strategy corresponds to making some of the aforementioned discrete decisions, *e.g.* deciding how to fixture the bottle. We define strategies as sequences of parameterized high-level actions. Each action is implemented as a controller parameterized by a set of constrained discrete and continuous values, such as robot configurations, objects, grasp poses, trajectories, etc. Our goal is to find both a sequence of high-level actions (a strategy) and parameter values for those actions, all of which satisfy the various constraints on the robot's motions.

To produce valid solutions for a wide range of object and environment configurations, the robot must be able to consider a wide range of strategies. As discussed above, small changes, such as decreasing a friction coefficient, may necessitate an entirely new strategy. In a different environment, the robot may need to first move some blocking object out of the way (Fig.3.5) or relocate an object in order to achieve a better grasp. Approaches that attempt to explicitly encode strategies in the form of a policy, *e.g.* via a finite state machine or a fixed action sequence, will generally fail to capture the full range of feasible strategies (Michelman and Allen, 1994; Holladay et al., 2019). Methods that attempt to learn such a policy will need a very large number of interactions to explore this rich and highly-constrained solution space.

We propose addressing forceful manipulation problems by planning over the combinatorial set of discrete/continuous choices. We use an existing *task and motion planning* (TAMP) system, PDDLStream, which reduces this type of hybrid discrete/continuous planning problem to a sequence of discrete planning problems via focused sampling of the continuous and discrete parameters (Garrett et al., 2020b).

Figure 3.1: Opening a childproof bottle involves executing a downward-push and twist on the cap, while fixturing the bottle. Our system can reason over a combinatorial number of strategies to accomplish this forceful manipulation task, including push-twisting with various parts of its end effector, push-twisting with a tool (in blue), fixturing with a vise (in grey), fixturing against the table, or fixturing against a high-friction rubber mat (in red).

To enable this approach to forceful manipulation, we introduce force-related constraints: the requirement to fixture objects and the *forceful kinematic chain*. The forceful kinematic chain constraint captures whether the robotic and frictional joints of the kinematic chain are “strong” enough to exert the forces and torques to perform forceful operations. These constraints are integrated into the TAMP framework.

Furthermore, we enable the planner to choose strategies that are *robust to uncertainty* by formulating this as cost-sensitive planning, where the cost of an action is tied to its probability of success in open-loop execution, given perturbations in parameters for the force-based constraints.

This chapter makes the following contributions:

- Characterize forceful manipulation in terms of constraints on forceful kinematic chains
- Generate multi-step plans that obey force- and motion-related constraints using an existing TAMP framework.
- Formulate finding plans that are robust to uncertainty in the physical parameters of forceful kinematic chains as cost-sensitive planning.
- Demonstrate planning and robust planning for forceful manipulation in three domains (opening a childproof bottle, twisting a nut on a bolt, cutting a vegetable)

3.2 Related Work

This chapter focuses on enabling force-based reasoning in planning multi-step manipulation tasks. In this section we review various strategies for incorporating force-based reasoning and force-related constraints across various levels of planning: from single step actions to fixed action sequences to task and motion planning.

3.2.1 Force-Reasoning in Single Actions

Several papers have considered force requirements for generating specialized motions. Gao et al. use learning from demonstration to capture “force-relevant skills”, defined as a desired position and velocity in task space, along with an interaction wrench and task constraint (Gao et al., 2019). Berenson et al. incorporated a torque-limiting constraint into a sample-based motion planning to enable manipulation of a heavy object (Berenson et al., 2009). These papers consider force constraints when generating individual actions, while we consider force constraints over a sequence of actions.

One important category of reasoning with respect to force actions is stabilizing, or fixturing an object. The goal of fixturing is to fully constrain an object or part, while enabling it to be accessible (Asada and By, 1985). Fixture planning often relies on a combination of geometric, force and friction

analyses (Hong Lee and Cutkosky, 1991). There are various methods of fixturing including using clamps (Mitsioni et al., 2019), custom jigs (Levi et al., 2022) or using another robot to directly grasp or grasp via tongs (Watanabe et al., 2013; Stückler et al., 2016; Zhang et al., 2019). Additionally some strategies, such as fixtureless fixturing and shared grasping, rely on friction and environmental contacts to fixture without additional fixtures or tools (Chavan-Dafle and Rodriguez, 2018; Hou et al., 2020). In this chapter we consider fixturing via grasping and environmental contacts.

3.2.2 Force-Reasoning in Sequences of Actions

Several papers have considered reasoning over forces across multi-step interactions. Chen et al. define “forceful operations” as a 6D wrench f applied at a pose p with respect to a target object (Chen et al., 2019). In this chapter, we adopt their definition of forceful operations to characterize the type of interactions our system plans for. Chen et al. focus on searching over environmental and robot contacts to stabilize an object while a human applies a series of forceful operations. In our work, a robot must both stabilize the object and perform the forceful operation.

Manschitz et al. termed “sequential forceful interaction tasks” as those characterized by point-to-point motions and an interaction where the robot must actively apply a wrench (Manschitz et al., 2020). The method first learns, from demonstrations, a set of movement primitives, which are then sequenced by learning a mapping from feature vectors to activation of the primitives. The goal of their work is enable the robot to reproduce the demonstrations and therefore the sequencing is pre-scripted.

Michelman and Allen formalize opening a childproof bottle via a finite state machine, where the robot iteratively rotates the cap, while pressing down, and then attempts to lift it until the cap moves (Michelman and Allen, 1994). If the cap is not yet free, the robot continues to rotate the cap. In their formulation, the bottle is fixtured and some of the continuous parameters, such as the grasp, are fixed.

Holladay et al. frames tool use as a constraint satisfaction problem where the planner must choose grasps, arm paths and tool paths subject to force and kinematic constraints such as joint torque limits, grasp stability, environment collisions, etc. (Holladay et al., 2019). The planner outputs a fixed sequence of actions and thus the force and kinematic constraints do not impact the sequence of actions, i.e. the choice of strategy. In contrast, in this thesis, we are interested in searching over various possible strategies.

3.2.3 Multi-step Planning with Constraints

Solving for a sequence of actions parameterized by constrained and continuous values lies at the heart of multi-modal motion planning (MMMP) (Hauser and Latombe, 2010; Hauser and Ng-Thow-Hing, 2011) and task and motion

planning (TAMP) (Garrett et al., 2020a). MMMP plans motions that follow kinematic modes, e.g. moving through free space or pushing an object, and motions that switch between discrete modes, e.g. grasping or breaking contact, where each mode is a submanifold of configuration space.

TAMP extends MMMP by incorporating non-geometric state variables and a structured action representation that supports efficient search (Gravot et al., 2005; Plaku and Hager, 2010; Kaelbling and Lozano-Pérez, 2011; Srivastava et al., 2014; Toussaint, 2015; Dantam et al., 2016; Garrett et al., 2017).

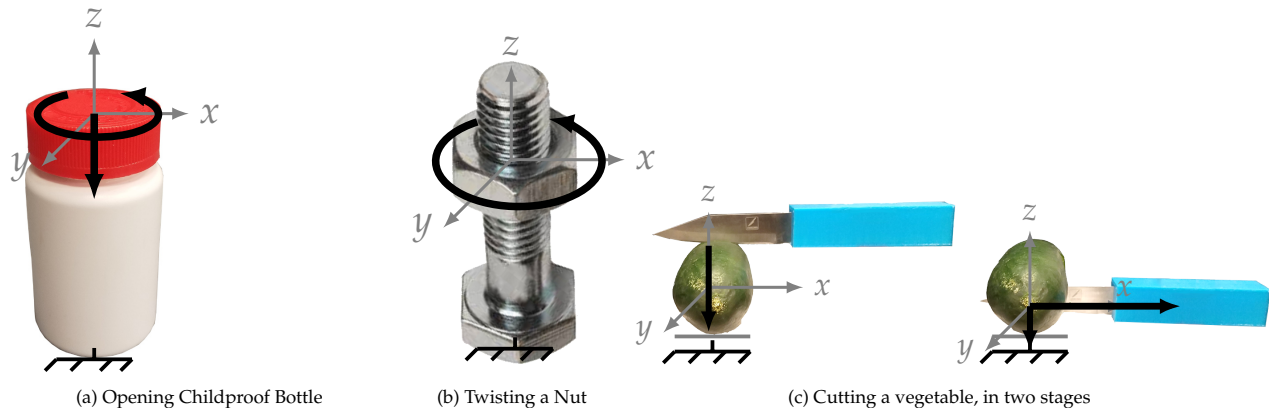
Most TAMP algorithms, although not all (Toussaint et al., 2020), have focused on handling collision and kinematic constraints. This paper focuses on integrating force-based constraints with an existing TAMP framework, PDDLStream (Garrett et al., 2020b), which we discuss in Sec. 3.6.

Most similar to our work, Toussaint et al. formulate force-related constraints that integrate into a trajectory-optimization framework (LGP) for manipulation tasks (Toussaint et al., 2020). While LGP can search over strategies, in the aforementioned paper the strategy was provided and fixed. While Toussaint et al. take a more generic approach to representing interaction which can capture dynamic manipulation, their use of 3D point-of-attack (POA) to represent transmitted 6D wrenches, fails short of representing the frictional constraints of the contact patches.

Levihn and Stilman present a planner that reasons over which combination of objects in the environment will yield the appropriate mechanical advantage for unjamming a door (Levihn and Stilman, 2014). The type of the door directly specifies which strategy to use (lever or battering ram) and the planner considers the interdependencies of force-based and geometric-based decisions for each application. The planner is specific to the domain and does not provide a general framework for planning with force constraints.

Several other systems consider forces either as a feasibility constraint or as a cost. In assessing feasibility for assembly plans, Lee and Wang (1993) account for the amount of force required to connect two pieces in an assembly. Akbari et al. (2015) focus on incorporating “physics-based reasoning” in a TAMP system that sequences push and move actions by formulating action costs with respect to power consumed and forces applied. Again, each of these planners present a domain-specific approach to considering force.

In this work, we consider generating plans that are robust to state uncertainty, with a particular focus on physical properties of the objects. Several TAMP frameworks approach uncertainty by planning in belief space, the space of probability distributions over underlying world states (Kaelbling and Lozano-Pérez, 2013; Hadfield-Menell et al., 2015; Garrett et al., 2020c). These planners generate action sequences that can also involve information-gathering sensing actions. In contrast, our system executes open-loop plans and focuses on uncertainty that impacts the force-related constraints.



3.3 Problem Domain

We define *forceful manipulation* as a class of multi-step manipulation tasks that involve reasoning over and executing forceful operations. Drawing from [Chen et al. \(2019\)](#), a forceful operation is a robot action where the robot exerts a 6D force/torque wrench $([f_x, f_y, f_z, t_x, t_y, t_z])$ on an object or the environment. In addition to the geometric constraints (such as collision-free trajectories) that characterize many multi-step manipulation tasks, forceful manipulation tasks are also characterized by constraints relating to the ability to exert wrenches.

Specifically, we constrain that the robot, including any grasped object, must be “strong” or “stable” enough to exert the forceful operation, i.e. the robotic system must be able to exert the desired wrench of the forceful operation without experiencing excessive force errors or undesired slip. We additionally constrain that any object that the forceful operation is acting on must be secured, or fixtured, in a way that prevents its motion.

Our aim is to perform forceful manipulation tasks in a wide range of environments, where there is variation in the number, type, poses and physical parameters, such as masses and friction coefficients, of the objects. We also characterize *robust* planning for forceful manipulation tasks as generating plans whose success is robust to uncertainty in these physical parameters.

To ground our work in concrete problems, we consider three example tasks within the forceful manipulation domains: (1) opening a childproof bottle (2) twisting a nut on a bolt and (3) cutting a vegetable. For each domain, we define the forceful operation(s) that represents the task (also called the task wrench(es)), strategies for exerting those forceful operation(s) and what object must be fixtured, We then discuss various methods of fixturing objects.

3.3.1 Childproof Bottle Opening

In the first domain, the objective is to open a push-and-twist childproof bottle, as introduced in [Sec. 4.1](#). We specify the push-twist, required before removing the cap, as the forceful operation of applying wrench $(0, 0, -f_z, 0, 0, t_z)$

Figure 3.2: (a) Opening a childproof bottle involves executing a push-twist on the cap, while fixturing the bottle. (b) Twisting a nut requires exerting a torque about the nut, while fixturing the bolt. (c) To cut, the robot first press down vertically and then slices horizontally. The object being cut must be fixtured.

in the frame of the cap (Fig.3.2(a)), where we assume f_z and t_z are given. As illustrated in Fig.3.1, the robot can apply this wrench through a variety of possible contacts: a grasp, fingertips, a palm or a grasped pusher tool. If using the latter three contacts, the robot can reason over applying additional downward force. While performing the forceful operation, the robot must fixture the bottle.

3.3.2 Nut Twisting

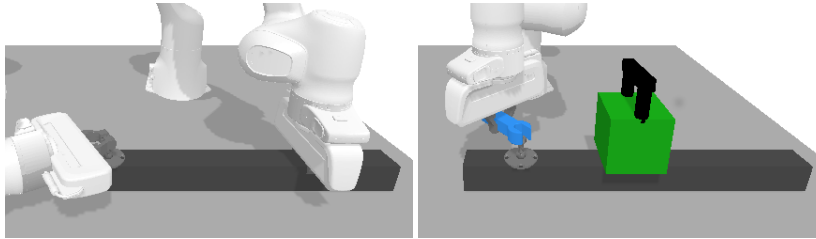


Figure 3.3: To twist a nut on the bolt, the robot can use either its fingers or a spanner (in blue). While twisting, the robot must fixture the beam that the bolt is attached to. Here we show two fixturing strategies: using another robot to grasp the beam and weighing down the beam with a large mass (in green).

In the second domain, the robot twists a nut on a bolt by applying the wrench $(0, 0, 0, 0, 0, t_z)$ in the frame of the nut (Fig.3.2(b)), where we assume t_z is given. As shown in Fig.3.3, the robot can make contact either through a grasp or through a grasped tool, i.e. a spanner¹. When twisting the nut, the robot must also fixture the beam holding the bolt.

We do not consider the more general task of twisting a nut *until* it is tight, which would require a feedback loop.

¹ To avoid confusion between a wrench (the tool) and a wrench (a 6D force-torque) we use the British term “spanner” to refer to the tool.

3.3.3 Vegetable Cutting

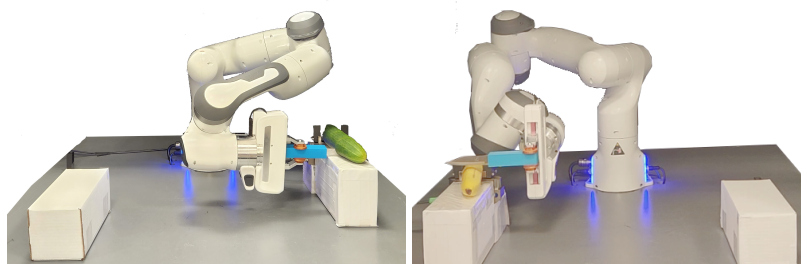
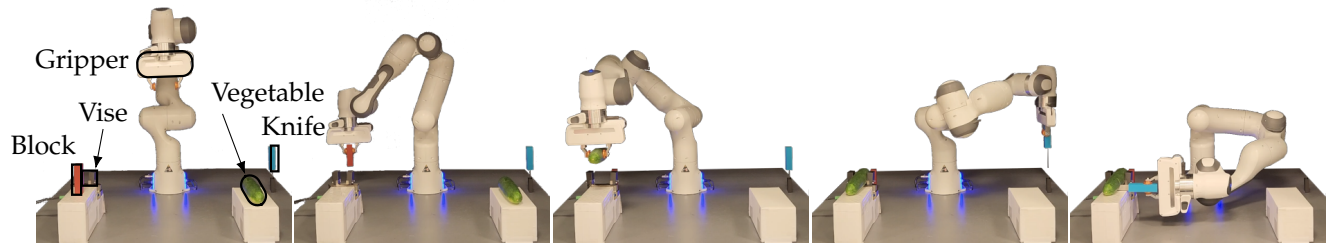


Figure 3.4: The robot uses a knife to cut an object, while fixturing the object. Here a vise is used to fixture a cucumber (left) and a banana (right) while they are being cut with a knife (blue).

In the third domain, the robot uses a knife to cut a vegetable (as shown in Fig.3.4). Cutting is a complex task that involves fracture, friction and changing contacts (Jamdagni and Jia, 2019, 2021; Mu and Jia, 2022). There have been a variety of approaches to tackling this cutting-edge topic such as learning the task-specific dynamics (Mitsioni et al., 2019; Zhang et al., 2019; Rezaei-Shoshtari et al., 2020), developing specialized simulators (Heiden et al., 2021), and proposing adaptive controllers (Zeng and Hemami, 1997a;



Long et al., 2013, 2014). In this work, we adopt a simplified approach to cutting.

We take inspiration from Mu et al. (2019) and assume that, while being cut, the object will have negligible deformation and that dynamic effects are insignificant. Similarly to their proposed cutting process, we formulate cutting as a two-stage process where the knife first exerts downward force, followed by a translational slice. Thus, this task has two forceful operations: the downward force $(0, 0, -f_{z,0}, 0, 0, 0)$ and the translational slice $(-f_x, 0, f_{z,1}, 0, 0, 0)$, as visualized in Fig.3.2(c). We assume that $f_{z,0}$, $f_{z,1}$ and f_x are given. While cutting the object, the robot must also fixture it.

3.3.4 Fixturing

While performing any forceful operation, the robot must fixture the object it is exerting force on to prevent its motion. There are a wide variety of ways to fixture, which are not unique to any particular domain. In this work, as shown across Fig.3.1, Fig.3.3, Fig.3.4, we present several different fixturing methods such as:

- Grasping the object with another robot
- Grasping the object in a vise
- Weighing the object down with a heavy weight
- Exerting additional downward force to secure the object with friction from a surface

For the second method, in practice we use a table-mounted robot hand as the vise. For the last method, the frictional surface can either be the table, or higher-friction rubber mats and the robot can exert this additional downward force through various contacts: fingertips, a palm or a grasped pusher tool.

3.4 Approach

Forceful manipulation tasks are characterized by constraints related to exerting force. We view a robotic system, composed of the robot joints, grasps and other possible frictional contacts, as a *forceful kinematic chain*.

Figure 3.5: The goal of the robot is to cut the vegetable, using the knife (in blue). The vegetable must be fixtured, which can be achieved using the vise. However, the robot cannot secure the vegetable in the vise because a red block is preventing a collision-free placement. Our system constructs a plan where the robot first picks up the red block and places it on the table, out of the way. The robot can then pick up the vegetable and fixture it in the vise. Next, the robot grasps the knife and uses it to cut the fixtured vegetable.

For example, in the bottom rightmost example of Fig.3.1, the robot has constructed a *chain* of joints composed of the robot’s actual joints, the robot’s grasp on the tool and the tool’s contact with the cap.

When the robot is performing a forceful operation, we can capture whether the system is strong enough to exert the task wrench by assessing if the forceful kinematic chain is maintained, i.e. if each joint is stable under the imparted wrench. We informally use the word *stable* to refer to an equilibrium of the forces and torques at all the joints.

For each class of joint, we describe a mathematical model that characterizes the set of wrenches that the joint can resist and thus the joint is stable if the imparted wrench lies within this set. To test the constraint, the planned task wrench combined with the wrench due to gravity, is propagated through the joints of the forceful kinematic chain and each of the joints are evaluated for their stability.

Given our domain description, there are two forceful kinematic chains when performing a forceful operation: the chain of the system exerting the task wrench and the chain of the system fixturing the object. Returning to the bottom rightmost example of Fig.3.1, the exertion chain was described above and the fixturing chain is one joint: the vise’s grasp on the bottle.

We need a planning framework that generates multi-step manipulation plans that are flexible to a wide-range of environments and that respect various force- and motion-related constraints. We opt to cast this as a *task and motion planning (TAMP)* problem, where the robot must find a feasible strategy, or sequence of actions, to complete the task.

Each action is implemented by a parameterized controller and is associated with constraints relating the discrete and continuous parameter values that must be satisfied for the controller to achieve its desired effect. The discrete parameters are values such as objects, regions, robot arms and the continuous parameters are values such as robot configurations, poses, paths, wrenches, etc.

Solving a TAMP problem corresponds to finding a valid sequence of actions and finding the discrete and continuous parameters of those actions that satisfy the constraints. These two problems are tightly connected, since the force- and motion-related constraints impact whether it’s possible to find a valid sequence of actions.

As an illustration of this connection, we return back to the example mentioned in Sec. 4.1, where the table top does not provide enough friction to fixture the bottle. This corresponds to a forceful kinematic chain where it is impossible to find a set of parameters to make it stable. In this case, the planner searches for a new set of actions, such as picking and placing the bottle into a vise, in order to create a different, stable forceful kinematic chain.

As another example, Fig.3.5 shows the robot completing a sequence of actions in order to cut a vegetable while it is fixtured in a vise. Since the red

block on the vise prevents the robot from directly placing the vegetable in the vise, the robot constructs a plan to move the red block out of the way before fixturing and cutting the vegetable.

Both of these examples illustrate that the interleaved constraint evaluation and action search in the TAMP framework is critical to enabling the planner to solve the tasks in a variety of environments.

In order for plans to reliably succeed in a variety of environments, the robot must also be able to account for uncertainty in the world. In this work, we focus on uncertainty in the physical parameters of the stability models used to evaluate the forceful kinematic chain constraint. For example, if the stability of a grasp used during a forceful operation is dependent on precise value of a friction coefficient, this choice of grasp is not very robust to uncertainty.

In order to find plans that are *robust*, we use cost-sensitive planning, where the planner searches for a plan whose cost is below a user-defined cost threshold. We relate the cost of an action to the probability that the forceful kinematic chain involved in the action is stable under uncertainty in the physical parameters. This formulation allows the planner to reason over which strategies and which parameters of those strategies lead to more robust, reliable execution.

In this work we assume a quasi-static physics model and, as input, are given geometric models of the robot, the objects and the environment along with the poses of each object. Estimates of physical parameters, such as the object’s mass and center of mass, and friction coefficients, are known. In robust forceful manipulation, we relax the need for exact estimates of physical parameters, instead using ranges.

3.5 Forceful Kinematic Chain

We have defined a *forceful kinematic chain* as the series of joints in a system, including robotic and frictional joints. In forceful manipulation tasks, the robot aims to exert wrenches through this chain. The forceful kinematic chain constraint evaluates whether each joint in the system is in equilibrium in the face of the task wrench and gravity (Fig.3.6). For each joint, we propagate these wrenches to the frame of the joint and evaluate if that wrench lies in the set of wrenches that the joint can resist. If every joint in the chain is stable, the constraint is satisfied.

For joint types, we consider planar frictional joints and robot manipulator joints and define the mathematical models that characterize the set of wrenches each joint type can resist. While our treatment in this chapter is limited to these joint types, alternative joints or models could easily be integrated, such as non-planar frictional joints (Xu et al., 2020).

In defining planar frictional joints we consider an example joint: the robot’s grasp on the blue pusher tool in Fig.3.6. In the three directions of

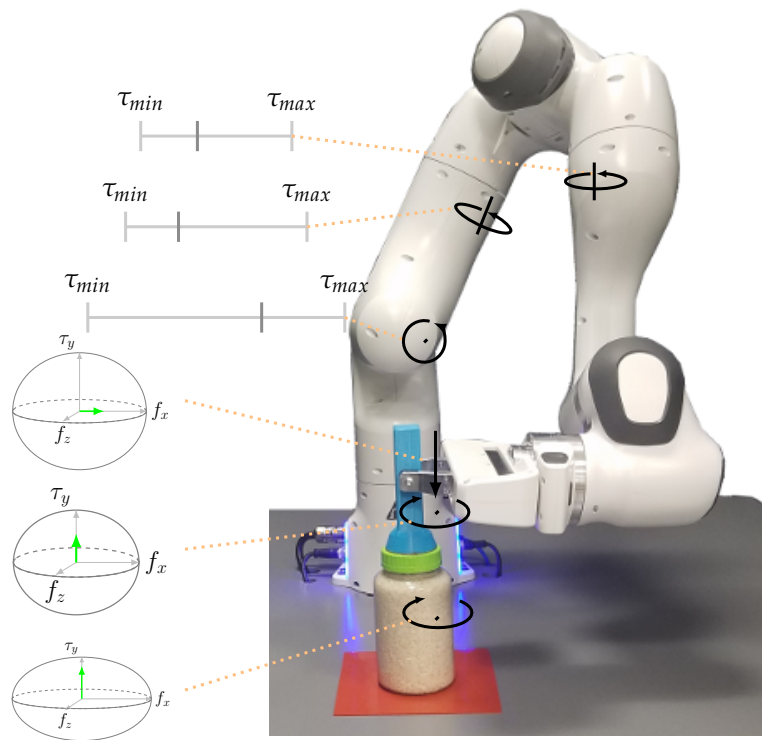


Figure 3.6: Along each joint of the forceful kinematic chain, we first project the expected wrench into the subspace defined by each joint and then verify if the joint is stable under that wrench. The figure illustrates the wrench limits for each joint: For circular patch contacts, we check the friction force against a limit surface ellipsoidal model and for each robot joint we check against the 1D torque limits.

motion outside the plane of the grasp, the motion of the tool in-hand is prevented by the geometry of the hand, i.e. we assume the fingers are rigid such that the tool cannot translate or rotate by penetrating into the hand. Thus, any wrenches exerted in those directions are resisted kinematically by non-penetration reaction forces, which we assume are unlimited. In the other three directions, motion within the plane of the grasp is resisted by frictional forces. We represent the boundary of the set of possible frictional wrenches in the three dimensional friction subspace of the plane of contact with a limit surface (Goyal et al., 1991)². We utilize two ways to approximate the limit surface, depending on the characteristics of the planar joint (Sec. 3.5.1, Sec. 3.5.2).

For the robot’s joints, the set of wrenches that can be transmitted are bound by the joint torque limits (Sec. 3.5.3).

² While these same tools were introduced in Chapter 2, we reintroduce them here, focusing on their specific application within the context of the forceful kinematic chain.

3.5.1 Limit Surface for Small Circular Patch Contacts

For small circular patch contacts with uniform pressure distributions, we use an *ellipsoidal approximation* of the limit surface (Xydas and Kao, 1999). The ellipsoid is centered in the contact frame, $w = [f_x, f_z, \tau_y]$, and, for isotropic

friction, is defined by $w^T Aw = 1$ where:

$$A = \begin{bmatrix} \frac{1}{(N\mu)^2} & & 0 \\ & \frac{1}{(N\mu)^2} & \\ 0 & & \frac{1}{(Nk\mu)^2} \end{bmatrix}$$

where μ is the friction coefficient, N is the normal force and, for a circular patch contact we approximate $k \approx 0.6r$ where r is the radius of the contact (Xydas and Kao, 1999; Shi et al., 2017).

Having transformed the wrench into the contact frame, we check if this wrench lies in the ellipsoid, which would indicate a stable contact:

$$\frac{f_x^2}{(N\mu)^2} + \frac{f_z^2}{(N\mu)^2} + \frac{m_y^2}{(Nk\mu)^2} < 1. \quad (3.1)$$

As an example, in Fig.3.7 we compare two possible grasps on a knife. For each grasp, we visualize the limit surface and an exerted wrench, transformed to the contact frame. In this example we consider the first step of vegetable cutting: exerting the downward force.

For both grasps, the friction coefficient, normal grasping force and radius of contact (μ , N , r respectively) are the same, so the shape of the ellipsoid is the same. Each grasp, however, varies the contact frame and grasp plane that the wrench is transformed into. In addition to impacting the magnitude of the generated torque, this defines which three directions must be resisted due to friction forces, as captured by the limit surface.

In both grasps, transforming the downward force of this cutting action into the contact frame generates a substantial amount of torque that the grasp must resist.

However, the grasp shown in Fig.3.7-top, which grasps the top of the handle, largely relies on frictional forces to resist this torque. We can imagine if the robot were to use this grasp, the knife could pivot in the robot's hand as it moved down to cut the vegetable. As illustrated by the projected wrench (in red) falling outside of the ellipsoid, this grasp is unstable with respect to the forceful operation.

In contrast, in the grasp shown in Fig.3.7-bottom, which grasps the side of the handle, the large force and torque are largely resisted kinematically and the grasp is very stable. Again, looking at the grasp, the geometry of the fingers, rather than friction, prevents the knife from sliding or pivoting in the hand.

3.5.2 Limit Surface for More General Patch Contacts

For contacts with more irregular shapes than a circle and with less uniform pressure distributions, we directly model the contact patch as a set of point contacts, each with its own normal force (localized pressure) and its own friction limits. Given a contact patch we model the force it can transmit as

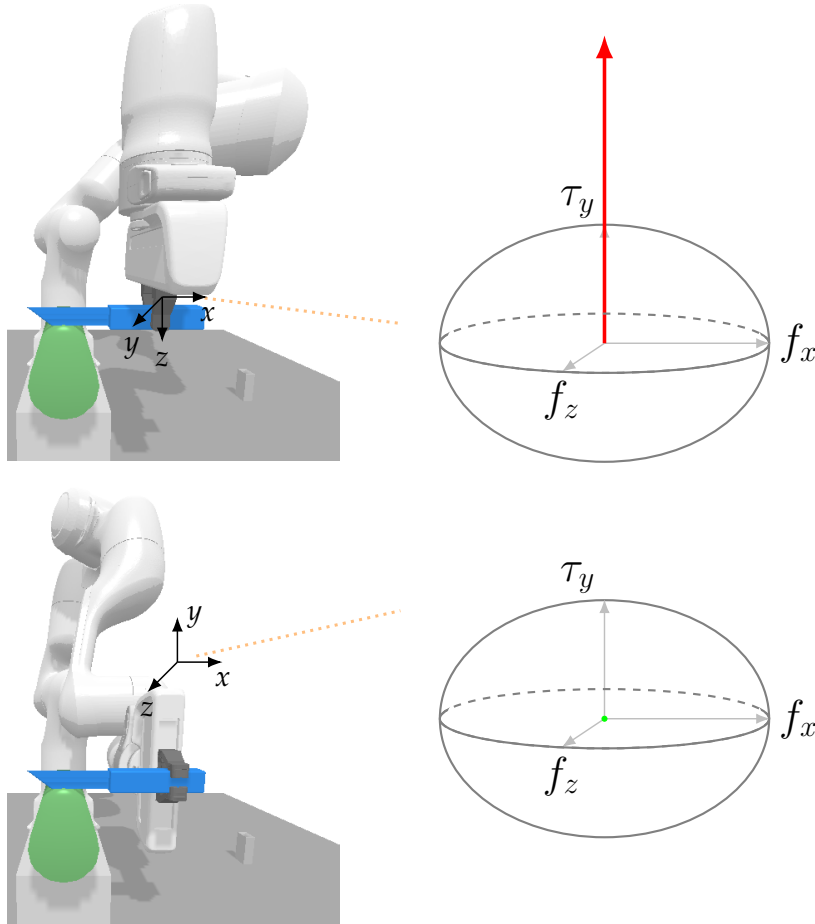


Figure 3.7: Here we show two possible grasps on the knife as it cuts a vegetable. For each grasp we visualize the corresponding limit surface with the propagated task wrench. The top grasp is not stable, as the wrench lies outside the boundary of the limit surface. In contrast, the bottom grasp, which leverages kinematics to resist the large torque, is stable.

the convex hull of generalized friction cones placed at the corners of the patch. Generalized friction cones, based on the Coulomb friction model, represent the frictional wrench that a point contact can offer (Erdmann, 1994). We represent the friction cone, FC, at each point contact with a polyhedral approximation of generators:

$$FC = \{(\mu, 0, 1), (-\mu, 0, 1), (0, \mu, 1), (0, -\mu, 1)\} \quad (3.2)$$

for a friction coefficient, μ (Lynch and Park, 2017). These generators can be scaled by the applied normal force. Given this approximation, the generalized friction cone can be written as:

$$V = \{v = J_f^T F \mid F \in FC\} \quad (3.3)$$

where J_f^T is the Jacobian that maps contact forces f from the contact frame, where FC is defined, to the object frame. If the exerted wrench, in the reference frame of the patch contact, lies in the convex hull of V , the frictional wrench can resist the exerted wrench and the contact is stable.

As an example, in the nut-twisting domain (Fig.3.3), we use the generalized friction cone to model the contact patch between the table and the beam holding the bolt, placing friction cones at the four corners of the beam. In evaluating the stability of fixturing the beam to the table via a heavy weight, the applied normal force, determined by the mass and location of the weight, is modeled as a simply supported 1D beam with a partially distributed uniform load.

3.5.3 Torque Limits

The last type of joint we consider are the joints of the robot, where the limit of each joint is expressed via its torque limits. We relate the wrenches at the end effector to robot joint torques through the manipulator Jacobian, J_m . Specifically, given a joint configuration q and wrench w , the torque τ experienced at the joints is modeled by $\tau = J_m^T(q)w$. The forceful kinematic chain is stable if the expected vector of torques τ does not exceed the robot's torque limits τ_{lim} :

$$J_m^T(q)w_{ext} < \tau_{lim}. \quad (3.4)$$

3.6 PDDLStream

Task and motion planning (TAMP) algorithms solve for a sequence of parameterized actions for the robot to take, also called the strategy or plan skeleton, and the hybrid parameters of those actions (Garrett et al., 2020a). The parameters are discrete and continuous values such as robots, robot configurations, objects, object poses, grasping poses, regions, robot paths, wrenches, etc. These parameters are subject to constraints, such as requiring that all paths are collision-free. The parameters of forceful manipulation tasks are also subject to the forceful kinematic chain constraints, which evaluates if each joint in the chain is stable in the face of an exerted wrench.

In order to find sequences of parameterized actions that satisfy a wide-range of constraints, we use PDDLStream, a publicly available TAMP framework (Garrett et al., 2020b). It has been demonstrated in a variety of robotics domains, including pick-and-place in observed and partially-observed settings (Garrett et al., 2020c).

In this section, we begin with some introduction to PDDL and then discuss how PDDLStream extends PDDL to enable planning over discrete and continuous parameters.

3.6.1 PDDL Background

A key challenge in solving TAMP problems is solving for the hybrid (discrete/continuous), constrained parameters. If all of the parameters were discrete, we could apply domain-independent classic planning algorithms from AI planning to search for sequences of actions. These planners use

predicate language, specifically the Planning Domain Definition Language (PDDL) (McDermott et al., 1998), to define the problem.

PDDL is inspired by STRIPS (Stanford Research Institute Problem Solver), a problem domain specification language developed for Shakey, a mobile robot that traveled between rooms and manipulated blocks via pushing (Nilsson, 1984). We next briefly describe PDDL in the context of a Shakey example. In the following subsection we will discuss how PDDL can be augmented to address problems with hybrid parameters.

In PDDL the state of the world is defined by a set of facts. A fact captures a relationship among state variables. We denote variables with italics symbols, e.g. d , $robot$, r . When defining a domain in PDDL, we specify fact types, such as $(\text{Door } d)$, $(\text{Room } r)$, $(\text{InRoom } robot\ r)$.

We denote constant variables with bold symbols, e.g. \mathbf{d}_1 , $\mathbf{robot}_{\text{shakey}}$, \mathbf{r}_6 . Hence, the fact $(\text{Door } \mathbf{d}_1)$ captures that in the world, there exists a door \mathbf{d}_1 . The fact $(\text{InRoom } \mathbf{robot}_{\text{shakey}}\ \mathbf{r}_6)$ captures that the robot $\mathbf{robot}_{\text{shakey}}$ is in room \mathbf{r}_6 .

$(\text{Door } \mathbf{d}_1)$ is an example of a static fact, which will remain constant throughout the problem. $(\text{InRoom } \mathbf{robot}_{\text{shakey}}\ \mathbf{r}_6)$ is an example of a dynamic fact, also known as a fluent, whose truth value may change over time, e.g. as the robot moves between rooms.

The action space is defined via a set of *operators*. Each operator is composed of a controller, a set of variables, a set of preconditions and a set of effects. The controller is a policy that issues a sequence of parameterized robot commands. Preconditions define the requirements for executing the controller, i.e. the facts, involving the operator variables, which must be true in the state in order to execute the controller. The effects capture how the world changes as a result of executing the controller, i.e. what facts have been added or removed from the state.

As an example, the `go_thru` operator is parameterized by a robot $robot$, a door d , a start room r_s and a goal room r_g (Nilsson, 1984). The controller drives the robot by following a set of waypoints, going from the start to the goal room, moving through the door. Two preconditions are that the robot is in room r_s , i.e. $(\text{InRoom } robot\ r_s)$, and that the door d connects the two rooms, i.e. $(\text{Connect } d\ r_s\ r_g)$. The two effects of executing this operator are that the robot is no longer in the starting room and is instead in the goal room, i.e. $(\neg (\text{InRoom } robot\ r_s))$ and $(\text{InRoom } robot\ r_g)$, respectively.

An operator is *lifted* if the variables are unassigned, i.e. `go_thru` ($robot$, d , r_s , r_g). A *ground operator* is instantiated with constant values assigned to all variables, i.e. `go_thru` ($\mathbf{robot}_{\text{shakey}}$, \mathbf{d}_2 , \mathbf{r}_4 , \mathbf{r}_7) where $\mathbf{robot}_{\text{shakey}}$, \mathbf{d}_2 , \mathbf{r}_4 , \mathbf{r}_7 satisfy the constraints defined by the preconditions. Given a ground operator, we can ground the controller, in this case mapping each room and door to fixed locations. The controller plans and executes a path driving to the discrete locations.

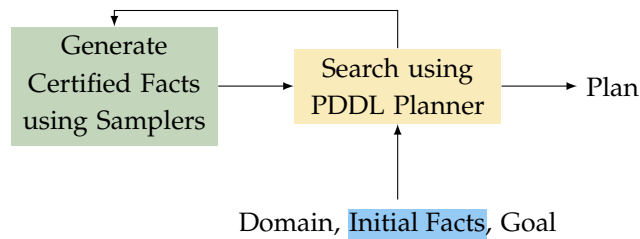


Figure 3.8: Algorithmic Flow of PDDL-Stream. The samplers are used to certify static facts. These facts, together with the domain, initial facts and goal, serve as input to a PDDL planner, which searches for a plan. If a plan cannot be found, the algorithm generates more certified facts via the samplers.

Given an set of facts that define the initial state, a set of facts defining the goal condition and a set of lifted operators, the planner searches for a sequence of ground operators to achieve the goal. Note that this involves solving for the sequence of operators *and* for valid groundings of each operator, i.e. finding parameter values of each operator that satisfy the constraints.

PDDL can be used to describe problems in finite domains, hence all parameter values are *discrete*. In order to use PDDL for robotics, we need to extend this language to include *continuous* values.

3.6.2 Algorithmic Overview

PDDLStream extends PDDL to include continuous domains by allowing for the *sampling* of continuous values. To generate these values, PDDL-Stream introduces *streams* that sample continuous parameters subject to constraints. These streams, or samplers, output static facts that the parameters are *certified* to satisfy.

For example, a configuration sampler could sample a collision-free joint configuration \mathbf{q} for a robot arm. The static fact output by this stream would be $(\text{Conf } \mathbf{q})$. A grasp sampler could generate a 6D grasp \mathbf{g} on an object \mathbf{o} , certifying the fact $(\text{Grasp } \mathbf{o} \ \mathbf{g})$. By capturing continuous parameters and their constraints using PDDL facts, PDDLStream can use discrete classic planners to plan over a hybrid search space.

To understand how sampling and search are integrated in PDDLStream it is helpful to draw an analogy to the popular motion planning algorithm of probabilistic roadmaps (PRMs) (Kavraki et al., 1996). PRMs uses discrete graph search algorithms to solve a motion planning problem in the continuous space of robot configurations. To do this, PRMs first sample configurations and represent them as nodes in a graph. The edges of a PRM represent the one action the robot can make: moving from one configuration to another. Given an initial state, goal and graph, domain-independent graph search algorithms can be used to search for a path.

Likewise, PDDLStream first samples parameters by executing streams to certify facts. Since the parameters often must satisfy a complex set of constraints, the details of the sampling procedures are often more complex than in PRMs. The certified facts generated by the samplers are added to the initial state. Given an initial state, goal and set of lifted operators, a

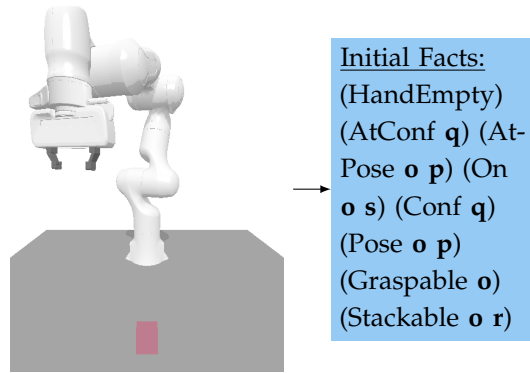


Figure 3.9: Pick-and-Place Example. A set of facts characterize the state. Here the robot starts at some configuration **q** and a graspable pink block **o** starts at some pose **p** on the surface region **r**.

discrete, domain-independent PDDL planner can be used to search for a plan composed of ground operator instances.

This algorithmic procedure is illustrated in Fig.3.8. The “Incremental” PDDLStream algorithm alternates between this process of sampling and searching until a plan is found, similarly to the way a PRM could continue to sample additional configurations and search the graph. In practice, we use the “Focused” PDDLStream algorithm, which more intelligently and efficiently samples in a lazy fashion, as detailed in [Garrett et al. \(2020b\)](#).

3.6.3 Specifying a Pick-and-Place Domain

In this section we use a pick-and-place domain as an example of specifying domains using PDDLStream, highlighting how discrete and continuous parameters are captured. Specifying a domain requires defining a set of fact types, lifted operators and samplers. Both the facts and actions are specified in PDDL, while the samplers are implemented in Python.

Facts. Facts can capture relationships over discrete variables, like objects, and continuous variables, such as robot configurations or grasp poses. Fig.3.9 shows a set of facts that characterize a scene with a robot and pink block on a surface. The fact (Graspable **o**) defines that the object **o** can be grasped, (Pose **o p**) defines that **p** is a pose of object **o**, (AtConf **q**) defines that the robot is at some configuration **q**, and (On **o r**) defines that an object **o** is on top of some surface region **r**. Here **o** and **r** are discrete parameters while **p** and **q** are continuous.

Facts can also be derived from other facts, e.g. the fact that a robot is holding an object **o**, (Holding **o**), is true if there exists a grasp **g** such that the robot is at some grasp, i.e. (AtGrasp **o g**). These are called *derived facts*.

Operators. Operators are parameterized by discrete and continuous values. As an example, the `pick` action is parameterized by an object to be grasped **o**, the pose of that object **p**, a grasp on the object **g**, a configuration **q** and a trajectory **t**. Two preconditions of the action are that the robot isn’t already holding something, (HandEmpty), and that the object is at the pose

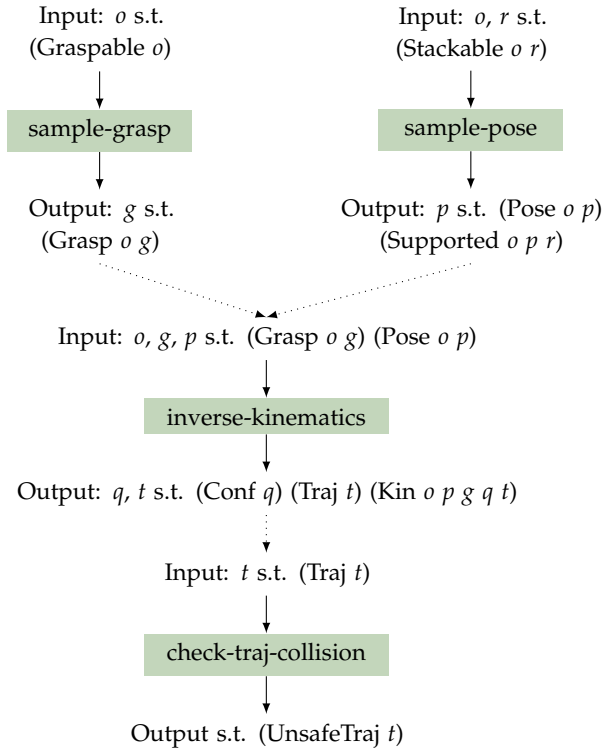


Figure 3.10: Pick-and-Place Example. Each sampler takes as input some values such that those values satisfy some constraint. The samplers (highlighted in green), output either new parameter values that are certified to satisfy some constraint or simply a certification that the inputs satisfy a constraint. Samplers can be conditioned upon each other such that the output of one sampler is the input to another, as shown by the dotted line.

(AtPose $o p$). Two effects of the operator are that the robot’s hand is no longer empty, $(\neg (\text{HandEmpty}))$ and that the robot is now at the grasp, (AtGrasp $o g$). We define the pick controller to be a series of commands: executing trajectory t to move towards the object, closing the hand, executing trajectory t in reverse to back up.

Samplers. Samplers are conditional generators that output static facts that *certify* that the set of parameters satisfy some set of constraints. Fig.3.10 shows several possible samplers, and how samplers can be conditioned on the output of other samplers.

For example, `sample-grasp` takes in an object o that is constrained to be graspable, and returns a grasp g that is certified to be a grasp of the object, as represented by the generation of the fact (Grasp $o g$). Likewise, `sample-pose` takes in an object o and region r such that the object can be stacked on the region, i.e. (Stackable $o r$). The output of this sampler is a pose p that is certified to be a valid pose of the object o and that if object o is at pose p , that object o is supported by region r .

The grasp g and pose p generated by these two samplers, along with the object o , can be inputs to an `inverse-kinematics` sampler which outputs a configuration q and trajectory t . The relationship between the variables is captured in the static fact (Kin $o p g q t$): the trajectory t starting from configuration q grasps object o at pose p with grasp g .

In addition to generative samplers, there are also test samplers which evaluate to true or false based on whether the input variables satisfy some

constraint. These samplers do not generate new parameters and instead only add facts about existing parameters.

As an example, a test stream is used to evaluate whether a trajectory t is collision-free. This is a condition over the entire state, i.e. the trajectory must be collision-free with respect to all objects in the environment. Evaluating if a trajectory is collision-free by an universal qualifier (`forall`) is expensive for PDDL planners. Therefore, instead we can use negation to evaluate an existential quantifier (`exists`). Specifically, we use the test sampler `check-traj-collision` to certify the fact `(UnsafeTraj t)` if there exists any object o at pose p such that trajectory t is in collision with o at p^3 . We can then enforce that trajectories are collision-free by using `(not (Unsafe t))` as a precondition.

³ Both o and p are omitted from Fig.3.10 for clarity.

3.6.4 Search Procedure

Given the domain specified in the previous subsection, we consider one step of the search, visualized in Fig.3.11

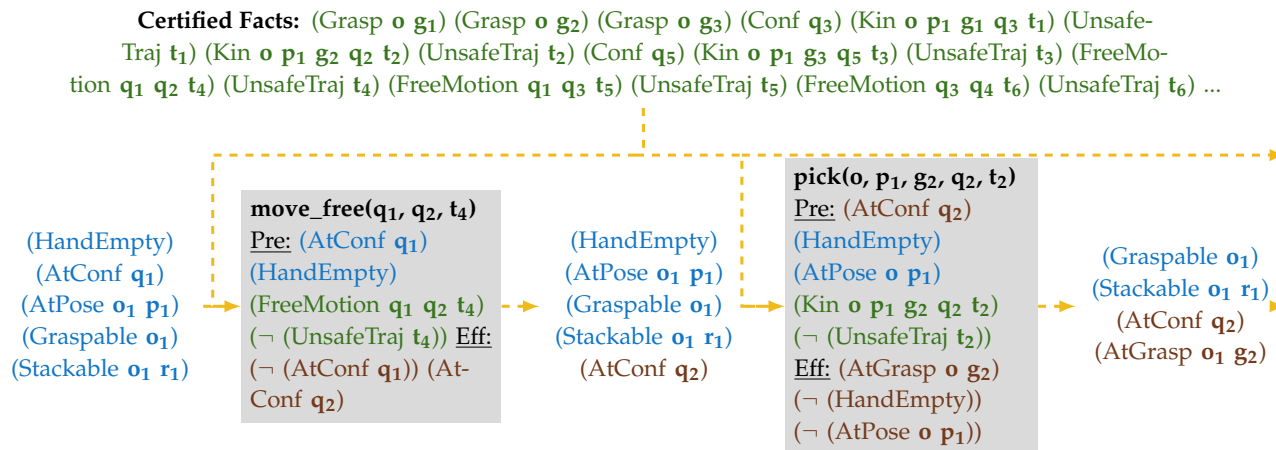
The state is defined by all of the facts that are true. At the beginning of the search, this is the set of initial facts (in blue) and the set of facts certified by the samplers (in green). In this example pick-and-place domain, Fig.3.9 gives the initial facts and Fig.3.10 visualizes some of the streams.

As stated above, the facts certified by the samplers are static. As an example, if a grasp sampler generates a grasp on an object, the fact capturing this grasp, `(Grasp o g)`, is always true. Thus, in the search, the certified facts output by the samplers are always a part of the state. The facts from the initial state may be static facts, such as `(Graspable o)`, or fluents, such as `(AtConf q1)`. These latter types of facts may be added or removed, as a result of the operators' effects, as the robot acts in the environment.

Given the state composed of all of these facts, the planner now conducts a search for a sequence of actions that can be taken to achieve the goal. For an action to be valid, all of its preconditions must be met, i.e. all of the precondition facts must be true in the state. The facts could be true either because the fact was certified by a sampler, or because it was part of the initial state or because it was added to the state as the effect of a previous action.

In Fig.3.11, the `move_free` action can be taken because all of the preconditions are satisfied. If the planner elected to take this action, the effects of the action would update the current state (in brown). The search continues forward based on this updated state, searching for valid actions.

Following the algorithmic loop in Fig.3.8, if the search is unsuccessful in finding a sequence of actions to the goal, the algorithm would generate more certified facts via the samplers and search again.



3.7 Incorporating Force into Planning

PDDLStream provides a framework for solving TAMP problems. Sec. 3.6.3 showed how to specify the standard pick-and-place domain, highlighting the fact types, lifted operators and samplers needed to capture the actions and the domain constraints, which relate to kinematic and geometric feasibility.

In order to leverage PDDLStream for forceful manipulation tasks, and thus extend PDDLStream’s range of applicability, we encode the the forceful kinematic chain constraint and the fixturing requirement. We do so by adding fact types, lifted operators and samplers. Additionally, each domain has domain-specific elements, although we find that many samplers are reused across domains. As an illustrative example for how these pieces come together, in Sec. 3.7.1 we detail the domain specification for the nut-twisting task. Through this example we also explore what modeling effort is required to specify a domain.

In addition to finding plans, in Sec. 3.7.2 we discuss how we use cost-sensitive planning in PDDLStream to find robust plans for forceful manipulation tasks.

3.7.1 PDDLStream for Forceful Manipulation

We use nut-twisting as an example of how the framework of PDDLStream is used for forceful manipulation tasks. The lifted operators, derived facts and samplers for this domain are defined in Table 3.1. We first require the standard pick-and-place operators (along with their related derived facts and samplers), some of which were already described in Sec. 3.6.3: `move_free`, `move_holding`, `pick`, `place`.

We next include the elements (in light grey) common across all of our forceful manipulation domains, which capture the forceful kinematic chain constraint and enable fixturing. As discussed in Sec. 3.3.4, there are several different fixturing strategies so in this domain we focus on two of them

Figure 3.11: Pick-and-Place Example. Expanded view of the PDDLStream search procedure. The state is composed of facts from the initial set of facts (in blue) and the set of certified static facts generated by the samplers (in green). An action is feasible if all of the facts of the preconditions are met. The state is then updated with the resulting effects (in brown). In this example, the first action is to `move` from configuration q_1 to q_2 . Having taken this action, the next is to pick up an object o at pose p_1 using grasp g_2 . The result of the search would be the sequence of ground operators, i.e. [`move_free`(q_1 , q_2 , t_4), `pick`(o , p_1 , g_2 , q_2 , t_2)].

Operator	Preconditions	Effects
move_free	$(\text{AtConf } a \ q_1) \wedge (\text{HandEmpty } a) \wedge$ $(\text{FreeMotion } a \ q_1 \ q_2 \ t) \wedge (\neg (\text{TrajUnsafe } a \ t))$	$(\neg (\text{AtConf } a \ q_1))$ $\wedge (\text{AtConf } a \ q_2)$
move_holding	$(\text{AtConf } a \ q_1) \wedge (\text{Movable } o) \wedge (\text{AtGrasp } a \ o \ g_o) \wedge$ $(\text{HoldingMotion } a \ o \ g_o \ q_1 \ q_2 \ t) \wedge (\neg (\text{TrajUnsafe } a \ t))$	$(\neg (\text{AtConf } a \ q_1))$ $\wedge (\text{AtConf } a \ q_2)$
pick	$(\text{AtConf } a \ q) \wedge (\text{HandEmpty } a) \wedge ((\text{Movable } o) \wedge$ $(\text{AtPose } o \ p_o) \wedge (\text{Kin } a \ o \ p_o \ g_o \ q \ t) \wedge (\neg (\text{TrajUnsafe } a \ t)))$	$(\text{AtGrasp } a \ o \ g_o)$ $\wedge (\neg (\text{AtPose } o \ p_o))$ $\wedge (\neg (\text{HandEmpty } a))$
place	$(\text{AtConf } a \ q) \wedge (\text{AtGrasp } a \ o \ g_o) \wedge$ $(\text{Kin } a \ o \ p_o \ g_o \ q \ t) \wedge (\neg (\text{TrajUnsafe } a \ t))$	$(\text{AtPose } o \ p_o)$ $\wedge (\text{HandEmpty } a)$ $\wedge (\neg (\text{AtGrasp } a \ o \ g_o))$
hand_twist	$(\text{AtConf } a \ q_0) \wedge (\text{HandEmpty } a) \wedge (\text{Nut } o_n) \wedge (\text{Bolt } o_b) \wedge$ $(\text{AtPose } o_n \ p_n) \wedge (\text{On } o_n \ o_b) \wedge (\text{Fixtured } o_b \ w) \wedge$ $(\text{StableGrasp } o_n \ g_n \ w) \wedge (\text{StableJoints } a \ t \ w) \wedge$ $(\text{NutHandMotion } a \ o_n \ p_n \ g_n \ w \ q_0 \ q_1 \ t) \wedge (\neg (\text{TrajUnsafe } a \ t))$	$(\text{Twisted } o_n \ w)$ $\wedge (\neg (\text{AtConf } a \ q_0))$ $\wedge (\text{AtConf } a \ q_1)$
tool_twist	$(\text{AtConf } a \ q_0) \wedge (\text{Nut } o_n) \wedge (\text{Bolt } o_b) \wedge (\text{AtPose } o_n \ p_n) \wedge (\text{On } o_n \ o_b) \wedge$ $(\text{Spanner } o_t) \wedge (\text{AtGrasp } a \ o_t \ g_t) \wedge (\text{Fixtured } o_b \ w) \wedge$ $(\text{StableGrasp } o_t \ g_t \ w) \wedge (\text{StableJoints } a \ t \ w) \wedge$ $(\text{NutToolMotion } a \ o_t \ o_n \ p_n \ g_t \ w \ q_0 \ q_1 \ t) \wedge (\neg (\text{TrajUnsafe } a \ t))$	$(\text{Twisted } o_n \ w)$ $\wedge (\neg (\text{AtConf } a \ q_0))$ $\wedge (\text{AtConf } a \ q_1)$

(a) Operators

Derived Facts	Definition
$(\text{On } o \ r)$	$\exists p_o ((\text{Supported } o \ p_o \ r) \wedge (\text{AtPose } o \ p_o))$
$(\text{Holding } o)$	$\exists a, g_o (\text{AtGrasp } a \ o \ g_o)$
$(\text{TrajUnsafe } a_1 \ t_1)$	$(\exists o, p ((\text{AtPose } o \ p) \wedge (\neg (\text{ObjCollisionFree } a_1 \ t_1 \ o \ p)))) \vee$ $(\exists a_2, q_2 ((\text{AtConf } a_2 \ q_2) \wedge (\neg (\text{ArmCollisionFree } a_1 \ t_1 \ a_2 \ q_2))))$
$(\text{HoldingFixtured } o \ w)$	$\exists g, a (\text{AtGrasp } a \ o \ g) \wedge (\text{StableGrasp } o \ g \ w)$
$(\text{WeightFixtured } o \ w)$	$\exists o_1, p_1 ((\text{Weight } o_1) \wedge (\text{AtPose } o_1 \ p_1) \wedge (\text{On } o_1 \ o) \wedge (\text{StableWeighDown } o_1 \ p_1 \ o \ w))$
$(\text{Fixtured } o \ w)$	$((\text{HoldingFixtured } o \ w) \vee (\text{WeightFixtured } o \ w))$

(b) Derived Facts

Sampler	Inputs	Outputs	Certified Facts
sample-pose	$o \ r$	p_o	$(\text{Pose } o \ p_o) \wedge (\text{Supported } o \ p_o \ r)$
sample-grasp	$a \ o$	g_o	$(\text{Grasp } a \ o \ g_o)$
inverse-kinematics	$a \ o \ p_o \ g_o$	$q \ t$	$(\text{Kin } a \ o \ p_o \ g_o \ q \ t) \wedge (\text{Conf } q) \wedge (\text{Traj } t)$
plan-free-motion	$a \ q_0 \ q_1$	t	$(\text{FreeMotion } a \ q_1 \ q_2 \ t) \wedge (\text{Traj } t)$
plan-holding-motion	$a \ q_1 \ q_2 \ o \ g_o$	t	$(\text{HoldingMotion } a \ o \ g_o \ q_1 \ q_2 \ t) \wedge (\text{Traj } t)$
test-arm-collision	$a_1 \ a_2 \ t_1 \ q_2$		$(\text{ArmCollisionFree } a_1 \ a_2 \ t_1 \ q_2)$
test-obj-collision	$a_1 \ t_1 \ o \ p$		$(\text{ObjCollisionFree } a_1 \ t_1 \ o \ p)$
test-grasp-stable	$a \ o \ w \ g_o$		$(\text{StableGrasp } o \ g_o \ w)$
test-joints-stable	$a \ t \ w$		$(\text{StableJoints } a \ t \ w)$
test-weight-stable	$o_1 \ p_1 \ o \ w$		$(\text{StableWeighDown } o_1 \ p_1 \ o \ w)$
plan-nut-hand	$a \ o_n \ p_n \ g_n \ w$	$q_0 \ q_1 \ t$	$(\text{NutHandMotion } a \ o_n \ p_n \ g_n \ w \ q_0 \ q_1 \ t) \wedge$ $(\text{Conf } q_1) \wedge (\text{Conf } q_2) \wedge (\text{Traj } t)$
plan-nut-tool	$a \ o_t \ o_n \ p_n \ g_n \ w$	$q_0 \ q_1 \ t$	$(\text{NutToolMotion } a \ o_t \ o_n \ p_n \ g_n \ w \ q_0 \ q_1 \ t) \wedge$ $(\text{Conf } q_1) \wedge (\text{Conf } q_2) \wedge (\text{Traj } t)$

(c) Samplers

Table 3.1: The specification of the nut twisting domain via the lifted operators, derived facts and samplers. Elements colored in light grey are common across all forceful manipulation domains. Elements colored in darker grey are specific to the nut-twisting domain. Throughout the table we use the symbols: a is a robot arm, o is an object, p_o is a pose of object o , g_o is a grasp on object o , q_i is a configuration, r is a region, t is a trajectory, w is a wrench. Specific to this domain: o_n, o_b and o_t refer to the nut, bolt and spanner, respectively.

(`HoldingFixtured` and `WeightFixtured`), while the rest are detailed in Appendix B. In all forceful manipulation domains we add the fact type (`Wrench w`) where w is a 6D wrench and a coordinate frame.

Finally, we define two domain-specific operators, `hand_twist` and `tool_twist` (and related samplers, all colored in darker grey), which correspond to the robot twisting the nut with its hand or with a grasped tool, respectively.

Forceful Kinematic Chain Constraint. To incorporate the forceful kinematic chain constraint into the PDDLStream framework, we assess the stability of the chain using test samplers. The facts certified by the test samplers serve as preconditions for the operators that exert forceful operations.

For example, to assess the planar frictional joints formed by the robot’s grasp, we define fact (`StableGrasp o g w`) which is true if the grasp g on object o is stable under the wrench w . This fact is certified by the test sampler `test-grasp-stable`, which uses the limit surface model discussed in Sec. 3.5.1 to evaluate the stability of the joint.

For an operator that applies a forceful operation using a grasped object, we use (`StableGrasp o g w`) as a precondition to constrain the grasp to be stable. In the context of the nut-twisting domain, `hand_twist` uses this precondition to evaluate the robot’s grasp on the nut and `tool_twist` uses it to evaluate the robot’s grasp on the tool.

We also use the (`StableGrasp o g w`) fact to derive the (`HoldingFixtured o w`) fact, which defines if an object o is fixtured from wrench w using some grasp g .

As another example, to evaluate if robot joints are stable we define the fact (`StableJoints a t w`) which is true if the trajectory t executed on robot arm a is stable under wrench w . This is certified by the test sampler `test-joints-stable`, which evaluates the torques experienced at each configuration in the trajectory are within the arm’s torque limits.

Fixturing. In addition to the forceful kinematic chain constraint, we require that while the robot is exerting a forceful operation on an object, the object must be fixtured. We propose implementing a variety of fixturing methods through various lifted operators and derived facts, which use forceful kinematic chain test samplers to evaluate the stability of the fixturing chain.

In the context of the nut-twisting domain, we restrict our focus to two fixturing methods: fixturing an object by holding it or by weighing it down with another heavy object. Thus, in this domain, the fact (`Fixtured o w`) is satisfied if either (`HoldingFixtured o w`) or (`WeightFixtured o w`) are true. The latter fact is derived from the fact (`StableWeighDown o1 p1 o w`), which is certified by the test sampler `test-weight-stable`. This sampler uses the generalized friction cone from Sec. 3.5.2 to evaluate stability.

Neither of these two fixturing methods required adding new operators, since sequences of moves, picks and places can be constructed to either

stably grasp the fixturing object or have a robot place a weighing-down object on the fixtured object. However, other fixturing methods (detailed in Appendix B), such as operating a vise, include adding operators, as well as test samplers and facts, to the domain.

Domain-Specific Actions. Given the forceful manipulation extensions, we now detail the domain-specific additions. In the nut-twisting domain the robot can impart the forceful operation to twist the nut either by making contact with its fingers or through a grasped spanner. To enable this, we define two new operators in Table 3.1.

The controller for the `hand_twist` operator grasps the nut with the hand, forcefully twists the nut with the hand and then releases the nut. Likewise, the controller for `tool_twist` makes contact with the nut via the grasped spanner, forcefully twists the nut with the spanner and then breaks contact between the nut and the spanner. The trajectory for each operator is generated by the samplers `plan-nut-hand` and `plan-nut-tool`, respectively. The twisting step of the trajectory involves planning a path that applies the forceful operation. Across all domains, we use Cartesian impedance control to exert wrenches, as detailed in Appendix A.1.

As stated in Sec. 3.7.1, the forceful kinematic chain constraint is implemented as preconditions for each operator. When twisting the nut with the robot hand (`hand_twist`), the forceful kinematic chain composed of the robot joints and the robot’s grasp on the nut. Likewise, when twisting with the spanner (`tool_twist`), the chain is composed of the robot joints, the robot’s grasp on the spanner and the spanner’s grasp on the nut. Grasps are evaluated via (`StableGrasp o g w`) and robot joints are evaluated via (`StableJoints a t w`). The sampler `plan-nut-tool` evaluates the stability of the spanner’s grasp on the nut.

We also constrain that the bolt, o_b is fixtured, using any of the available fixturing methods. In Appendix B we describe the domain-specific elements for the childproof bottle and vegetable cutting domain.

Modeling Effort. We next consider what is required in applying this framework to new settings. We consider two categories of modifications: incorporating new assessments of the forceful kinematic chain and fixturing constraints and incorporating new domain-specific elements.

Sec. 3.5 defines several mathematical models for assessing the stability of a joint. These models were incorporated as test samplers within PDDL-Stream. Since the framework is agnostic to the Pythonic implementation of the sampler, it is straightforward to swap out various stability models.

Adding a new joint type, and its corresponding stability model, requires adding both the corresponding sampler and the certified fact. Adding a new fixturing method may require adding operators to utilize the fixturing method (e.g. incorporating vise fixturing required adding vise actuation actions). It also requires identifying and integrating the appropriate stability assessment.

Critically, once such additions are made, they can be reused across many

Algorithm 2 Compute $P[\text{success}(\text{operator})]$

```

1: Given: Chain  $c$ , wrench  $w$ , parameters  $p$ 
2: Initialize:  $i = 0$ 
3: procedure FOR  $J=0:N$ 
4:    $p_\epsilon \leftarrow$  Perturb  $p$  by epsilon
5:   if STABILITYCHECK( $c, w, p_\epsilon$ ) then
6:      $i += 1$ 
7: return  $\frac{i}{n}$ 

```

domains. In our experience, this allows domains to build off of each other, decreasing the implementation effort with each new domain. For example, to model the contact patch between the beam and the table in the nut twisting domain, we incorporated the generalized friction cone. Thus, when implementing the cutting domain, we reused that same abstraction to model the contact patch between the vegetable and the table.

Solving a new task often involves adding domain-specific operators to capture a new action space (e.g. for cutting we had to add the `slice-cut` operator). We approach adding a new operator by first defining the preconditions and effects that characterize the kinematics and geometry. For example, if an operator uses a tool (as `tool-twist` does), then a precondition is that the robot must be grasping the tool. Next, if the operator is exerting a forceful operation, the relevant forceful kinematic chains must be stable. This requires identifying the joints within the chain and the appropriate corresponding stability models and adding the stability checks as preconditions.

Finally, we must write the sampler that defines the operator’s parameterized controller. This step involves combining the existing lower-level controllers (joint-space position controller, grasp controller, guarded move controller, cartesian impedance controller, etc.) to create the desired behavior. We find that even with operator-specific parameterized controllers, there often is significant overlap across domains with respect to how to combine the low-level controllers.

3.7.2 Robust Planning

Given the ability to generate plans that satisfy motion- and force-based constraints, we now aim to produce *robust* plans. In particular, we focus on protecting against stability-based failures along the forceful kinematic chains due to *uncertainty* in physical parameters. For example, we want to discourage the system from selecting a grasp where a small change in the friction coefficient would break the stability of the grasp, leading to the object slipping.

To generate robust plans, we associate each operator with a probability of success $P[\text{success}(\text{operator})]$. As given in Algorithm 2, we assess the probability of an operator’s success via Monte Carlo estimation, i.e. we

draw sets of parameter values, where each parameter is perturbed by some uniformly-sampled epsilon, and evaluate the stability of each forceful kinematic chain. We perturb, when applicable, parameters such as the friction coefficient, the planned applied wrench, the contact frame and the effective size of the contact patches⁴.

As an example, Fig.3.12 visualizes how sampling various grasp parameters essentially generates a large set of possible limit surfaces. By evaluating the stability with respect to all of them, we capture how the grasp stability is impacted by uncertainty.

We define the cost of an operator as:

$$\text{cost}(\text{operator}) = -\log(P[\text{success}(\text{operator})]). \quad (3.5)$$

The cost of a plan is then the sum over the cost of all the operators in the plan. Minimizing this cost is equivalent to maximizing the plan success likelihood.

To generate robust plans, we use PDDLStream’s *cost-sensitive planning* where, given non-negative, additive operator costs that are functions of the operators’s parameters, the planner searches for a plan that is below a user-provided cost threshold. Cost functions are specified in the domain by adding that the effect of an operator is increasing the total cost of the plan by the value computed by the operator’s cost function.

With this cost definition, the cost threshold corresponds to the probability of succeeding during open-loop execution given uncertainty in the physical parameters.

3.8 Empirical Evaluation

Using the childproof bottle domain, we demonstrate how the planner finds a wide variety of solutions and how the feasibility of these solutions depends on the environment. In each of the three domains, we show how accounting for uncertainty by planning robustly leads to the robot making different choices, both with respect to the strategy and with respect to the continuous choices.

Simulation and real robot videos showing a variety of strategies in each of the three domains are available online⁵.

3.8.1 Exploring Strategies

We demonstrate the range of the plans that the planner generates by considering two settings in the childproof bottle domain (Table 3.13). The robot can fixture the bottle using a variety of possible strategies including stably grasping with another robot, using a vise or using frictional contact with the surface, where the surface is either the table or a high-friction rubber mat⁶. The robot can apply the push-twist operation through a variety of contacts: a grasp, fingertips, a palm or a grasped pusher tool.

⁴ The details of the perturbations used in the experiments in Sec. 3.8.2 are given in Appendix A.3

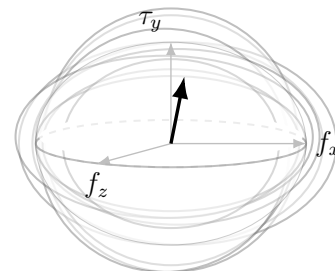


Figure 3.12: By sampling over the friction coefficient, μ , radius of contact r and task wrench, we can assess if a contact is stable in face of uncertainty in those parameters.

⁵ <https://www.youtube.com/watch?v=r1L-gxufFuY>

⁶ In this domain we do not consider the fixturing method of weighing down the bottle, since this is not geometrically feasible.

Fixturing Ablation		
Fixturing Method	# Steps	Planning Time (SE)
Surface(Table) *	4	177 (51)
Robot Grasp	6	60 (9.4)
Surface(Mat)**	8	142 (73)
Vise grasp**	9	95 (35)
Push-Twisting Ablation		
Pushtwist Method	# Steps	Planning Time (SE)
Grasp	4	37 (1.8)
Palm	4	25 (3.1)
Fingertip*	4	63 (35)
Pusher Tool**	8	40 (5.1)

Table 3.13: For each setting, we provide the number of steps for each strategy and the average planning time in seconds (and standard error) over five runs. *: Utilized a higher friction coefficient μ to increase feasibility **: Invalidated shorter strategies to force to planner to find these longer strategies.

In the first setting, we search over several possible fixturing strategies, fixing the push-twisting strategy to use a grasp contact. In this setting the bottle starts at a random location on the table. In the second setting, we search over all possible push-twisting strategies. In this setting the bottle starts on a high-friction rubber mat in order to leverage the friction of the surface for fixturing.

Because the underlying search over strategies in PDDLStream biases towards plans with the fewest actions, we incrementally invalidate the shorter strategies once found by the planner in order to force exploration of the alternative, longer strategies. For example, in the fixturing setting we invalidate fixturing via a robot grasp as a feasible strategy by removing the second arm from the environment.

Fixturing Setting. We first consider searching over various fixturing strategies. Looking first for the strategy with the fewest number of actions, the planner first tries to fixture the bottle against the table surface by applying additional downward force. However, the friction coefficient between the table and bottle is small enough that this is not a viable strategy: even when applying maximum downward force the robot cannot fixture the bottle. So instead, the planner fixtures using the second robot to grasp the bottle.

Removing the second arm from the environment forces the planner to discover new strategies. One strategy employed is to use a pick-and-place operation to move the bottle to a high-friction rubber mat, where it is possible to sample enough additional downward force such that the bottle is fixtured. Another strategy is to use a pick-and-place operation to move the bottle into the vise, where it can be fixtured.

Push-Twisting Setting. We next consider searching over various push-twisting strategies. Three of the twisting strategies, contacting via a grasp, the fingertips or the palm, are of equivalent length, and are therefore equally attempted when searching over strategies. We can view the viability of finding successful parameters to these actions, and thus probability of employing that

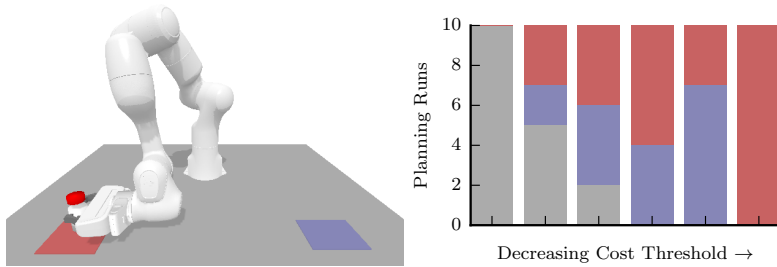


Figure 3.14: In opening the childproof bottle, the robot can fixture against the low-friction table (▩), a medium-friction mat (▨) or a high-friction mat (▧). For each cost threshold we run the planner ten times, noting which surface is used. As the cost threshold decreases, the robot is forced to more frequently use higher friction surfaces that are more robust to uncertainty.

Cost Threshold	Planning Time (SE)
∞	6.9 (0.1)
0.5	41 (14)
0.4	69 (17)
0.3	77 (10)
0.2	94 (18)
0.1	127 (29)

Table 3.15: We evaluate how decreasing the cost threshold impacts what fixturing surface the planner uses in the childproof bottled domain (Fig.3.14). As the cost threshold decreases, the planning time increases.

strategy, as how easy it is to sample satisfying values.

For example, twisting the cap by pushing down with the palm is only stable if, given the values of the radius of the palm and the friction coefficient, the system can exert enough additional downward force. Decreasing either the radius or the friction coefficient narrows the space of feasible downward forces. Since the fingertips have a smaller radius, as compared to the palm, it is harder to sample a set of satisfying values for the fingertips.

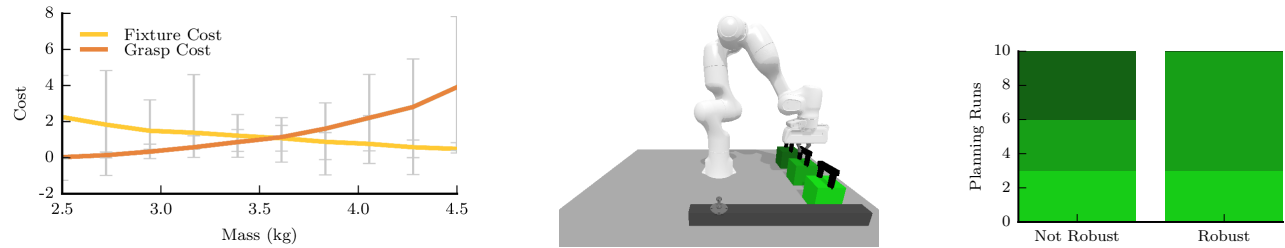
In each of these settings, we demonstrate that the planner finds a variety of different strategies and that the choice of strategy adapts to what is feasible in the environment. This adaptability allows the planner to generalize over a wide range of environments.

3.8.2 Generating Robust Plans

We next examine how leveraging cost-sensitive planning enables the robot to generate more robust plans.

Childproof Bottle Domain. In the childproof bottle domain, we explore how accounting for robustness impacts what surface the robot uses to fixture against. As visualized in Fig.3.14(left), the robot can choose between fixturing on three surfaces: the low-friction table, the medium-friction blue mat or the high-friction red mat. For all planning instances we start the bottle on the table and set the friction coefficient between the table and the bottle to be just high enough to make fixturing feasible. At each cost threshold, we run the planner ten times.

If the planner does not account for robustness, the robot fixtures with the table every time. This is because doing so is feasible and results in the shortest plan.



When considering robustness, the planner evaluates that fixturing using the table produces a feasible but brittle plan, resulting in a high cost plan that is unlikely to succeed. To avoid this, the planner completes a pick-and-place action to relocate the bottle to one of the two mats that have a higher friction coefficient and thus offers a more robust fixturing surface. As we decrease the cost threshold, the planner is forced to use exclusively the high-friction red mat.

Table 3.15 shows how decreasing the cost threshold increases the planning time. The choice of the cost threshold can therefore be viewed as trading off between the probability of the plan’s success and the time to find the plan.

Nut Twisting Domain. In the nut-twisting domain we explore robustness by considering a scenario where the robot must choose between several weights, of varying mass, to fixture the beam with.

First, for a given mass, we sample 100 placement locations along the beam holding the bolt and evaluate two robustness metrics: how robustly the weight fixtures the beam and how robustly the robot is able to grasp (and therefore move) the weight to this placement. Fig.3.16-left shows the trade-off: a heavier weight more easily fixtures the beam but is harder to grasp robustly. In finding a robust plan, and hence a low-cost plan, the planner is incentivized to act like Goldilocks and pick the weight that best balances this trade-off.

We can see this in action when running the planner in a setting with three weights of various masses (2.6, 3.5, 4.4 kg), shown in Fig.3.16-center, where a darker color corresponds to a larger mass. Fig.3.16-right shows that when the planner does not account for robustness, the three weights are selected equally, since all can be used to produce feasible plans. However, when accounting for robustness the planner more often selects the medium weight, which balances the trade off between the cost functions. In both instances, the planner was run ten times.

Vegetable Cutting Domain. In the vegetable cutting domain, we explore how planning robustly impacts the continuous choice of the grasp on the knife. While our grasp set defines grasps along every face of the rectangular handle, for clarity of example, we restrict the planner to top grasps along the length of the tool, such as the grasp shown in Fig.3.7(top).

We return to the two possible grasps on the knife shown in Fig.3.7. The side grasp, shown on the bottom, relies on normal reaction forces to resist

Figure 3.16: In the nut-twisting domain we consider the trade-off between the grasp cost and the fixturing cost. On the left, at each weight value, we randomly sample, 100 times, the pose of the weight along the beam and the grasp on the weight. Since, at the extremes, some costs evaluate to infinity, we plot the median and a 95% confidence interval. We then demonstrate how the trade off impacts the choices made by the planner by considering an environment in which there are three possible masses, as shown in the center. The robot can fixture using the 2.6kg mass (■), the 3.5kg mass (■) or the 4.4kg mass (■). Without accounting for robustness, the robot chooses any of the masses. When planning robustly, the robot more often picks the medium weight, which balances the trade-off in costs. In both cases we run the planner ten times, noting which weight is used.

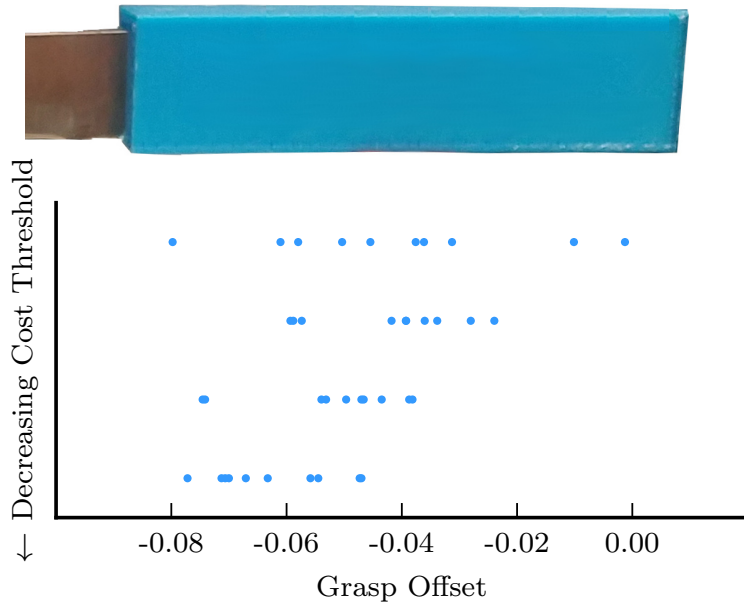


Figure 3.17: In the vegetable cutting domain, we show how robust planning leads the planner to select grasps that are closer to the blade of the knife, because doing so creates a smaller torque that the grasp needs to resist. For each cost threshold, the planner is run 10 times and the grasping offset is plotted. An offset of 0 corresponds to a grasp at the butt of the knife.

the torque experienced while exerting the downward force, the first half of the cutting action. As such, the grasp is very stable and robust, regardless of the location of the grasp along the length of the handle (the annotated x -axis). In contrast, the top grasp (shown in Fig.3.7(top)) relies on frictional forces to resist the exerted torque. The stability of the top grasp varies based on its location along the handle and by evaluating the robustness of these grasps, we can empower the planner to make a more informed choice.

To demonstrate this, we restrict our grasp set to only those that grasp the knife from the top (i.e. like Fig.3.7(top)). Additionally, we consider cutting a softer object, reducing the magnitude of the downward force ($f_{z,0}$) such that it is feasible to find a stable grasp. Specifically, by default we set $f_{z,0} = 5N$, while for the soft object we use $f_{z,0} = 3N$. We refer to this as the high force setting and low force setting, respectively.

Across several cost thresholds, considering only the cost with respect to the grasp stability during the downward cut, we plot the location of the grasp along the handle in Fig.3.17. For each cost threshold, the planner was run ten times. When the planner does not account for robustness, shown at the top of Fig.3.17, the robot selects any grasp along the handle. Accounting for robustness, as the cost threshold decreases, the planner selects grasps that are closer to the blade and thus create a smaller lever arm, decreasing the amount of torque the grasp must be stable with respect to.

We next evaluate how this choice impacts the robot’s ability to successfully slice, by testing on three different foods of varying hardness: cucumbers, bananas with the peel and bananas without the peel. From the continuous grasp set, we use three representative grasps, shown in Table 3.18: grasping the side of the handle (`side_grasp`), grasping from the top of the handle

close to the blade (`top_grasp_close`) and grasping from the top of the handle close to the butt of the knife (`top_grasp_far`).

For each grasp and each food we execute the same downward force-motion, without evaluating stability or robustness. For the cucumber and the banana with the peel we repeat this fifteen times. For the banana without the peel we repeat this ten times. Table 3.18 classifies the results into three categories: success (fully slicing through the object), partial success (slicing through at least half of the object) and failure (not significantly slicing the object).

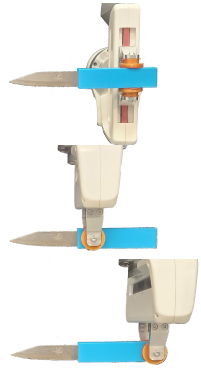
While we cannot precisely know the required downward force ($f_{z,0}$) for each food, we qualitatively assess the results. From our experience, the banana with the peel requires a force slightly higher than the low force setting, as considered in Fig.3.17. As stated previously, a planner that does not account for robustness would consider `top_grasp_close` and `top_grasp_far` equally while a robust planner would prefer `top_grasp_close`. The `top_grasp_far` cannot slice through the banana with the peel (all fifteen attempts are failures) while the `top_grasp_close` is able to cut through, at least partly, the majority of the time. In contrast, the banana without the peel corresponds to a soft enough object such that all grasps can successfully slice through.

Finally, we estimate that the cucumber requires the high force setting. As predicted in Fig.3.7, the `side_grasp` is sufficiently stable to slice through the object. However neither `top_grasp_close` nor `top_grasp_far` succeed because for both, the forceful kinematic chain breaks (the knife moves considerably within the robot’s end effector). While the robust planner, considering only grasps from the top, evaluates the `top_grasp_close` to be robust enough with respect to the low force setting, this is inadequate for the cucumber, whose true required force is significantly larger.

As a final note, one might wonder why we do not fix the planner to always use `side_grasp`, since Table 3.18 demonstrates it to be the most robust grasp.

First, we currently use a generic uninformed grasp sampler that is common across all of the handled tools (the pusher tool in the childproof bottle domain, spanner in the nut twisting domain and knife in cutting domain). Prescribing a preferred grasp for each tool and task, via an informed grasp sampler, would both increase the modeling effort and decrease the generalization. Instead, with an uninformed sampler, we allow the planner to reason over the best grasp for the specific scenario. As shown in Table 3.15, this is at some computational cost.

Second, while the side grasp may seem to be the best grasp with respect to this constraint, in the context of a multi-step manipulation problem, there are many, often competing, constraints. While for simplicity we have focused on evaluating the grasp with respect to the downward force, the grasp must also be stable with respect to the translational slice. Additionally, as shown in






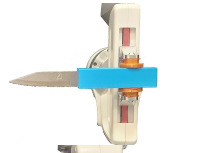


			
	6/9/0	7/6/2	10/0/0
	0/0/15	3/8/4	10/0/0
	0/0/15	0/0/15	9/1/0
			

Table 3.18: We evaluate slicing success for three grasps (from top to bottom: `side_grasp`, `top_grasp_close` and `top_grasp_far`) across three foods. For the cucumbers and bananas with peel we perform 15 iterations, for the banana without the peel we perform 10. We classify each interaction as a success / partial success / failure.

Fig.3.4, Fig.3.5 and Fig.3.7, due to the geometry of the robot’s end effector, `side_grasp` is only collision-free if the object is significantly raised. For foods where such a grasp is not strictly necessary, the planner can select a more reachable grasp. This flexibility is critical to enabling the planner to adapt to a variety of different environments.

3.9 Discussion

This chapter proposes a planning framework for solving forceful manipulation tasks. We define forceful manipulation as a class of multi-step manipulation tasks that involve reasoning over and executing forceful operations, where forceful operations are defined as the robot applying a wrench at a pose.

3.9.1 Summary of Contributions

Solving forceful manipulation tasks requires planning over a hybrid space of discrete and continuous choices that are coupled by force and motion constraints. We frame the primary force-related constraint as the system’s ability to stably exert the desired task wrench. To capture this, we propose the forceful kinematic chain constraint which evaluates if every joint in the chain is stable under the application of the imparted wrench and gravity. For each class of joint in the chain, which may be robot joints, grasps or frictional contacts, we discuss a model for evaluating its stability.

In addition to the forceful kinematic chain constraint, the planner must reason over other force-related constraints, such as the requirement to fixture objects, and over motion constraints, such as the requirement to find collision-free paths. To plan multi-step sequences that respect these constraints, we augment an existing task and motion planning framework, PDDLStream (Garrett et al., 2020b). We illustrate our system in three example domains: opening a push-and-twist childproof bottle, twisting a nut on a bolt and cutting a vegetable.

While PDDLStream finds a satisficing plan, the plan may not be robust to uncertainty. To find robust plans, we propose using cost-sensitive planning to select actions that are robust to perturbations. We specifically focus on uncertainty in the physical parameters that determine the stability of the forceful kinematics chains. Our demonstrations show how cost-sensitive planning enables the robot to make more robust choices, both with respect to the strategy, such as what fixturing method to use, and the continuous choices, such as which grasp to pick.

3.9.2 *Future Directions*

In this work we assume that the planner has access to many different parameters, such as the magnitude and form of the forceful operation, object models and object poses. We also assume the domain is given in the form of the fact types and lifted operators, with their preconditions, effects and controllers. Techniques from machine learning, combined with this planning framework, could be used to relax these assumptions and enable wider generalizations (Konidaris et al., 2018; Wang et al., 2021; Silver et al., 2021; Liang et al., 2022).

4

Briefly-Dynamic Manipulation with Uncertainty and Dead Ends

4.1 Introduction

We want to enable robots to leverage dynamic, non-prehensile manipulation actions, like shoving or toppling. As an example, Fig.4.1 shows how giving the object a quick shove allows the robot to efficiently relocate the block to the edge of the table, which is outside of the robot's reach. In general, expanding a robot's repertoire to include these actions empowers it to manipulate objects that are just out of reach, to manipulate objects that are too heavy or otherwise difficult to grasp, to reorient an object as a pregrasp manipulation maneuver (King et al., 2013) and, as shown, to extend the robot's reach.

Generating plans that sequence such dynamic actions presents two challenges. First, there is significant *uncertainty* in the outcome of these actions. Fig.4.1 shows that shoving could result in the block sliding a short distance before stopping, toppling over or even sliding off the table.

While this uncertainty results from various sources of partial information (i.e. uncertainty on the table-block friction coefficient, uncertainty in the block's center of mass), we choose to model this as action uncertainty, i.e. uncertainty in the resulting outcome of the action¹. Additionally, while any action has a degree of uncertainty (Mason, 2018), we focus on the dynamic non-prehensile actions as cases where the uncertainty must be reasoned over and planned for.

The second challenge is that executing these dynamic actions may result in outcomes that make it *impossible to achieve the goal*. Returning to Fig.4.1, if, in trying to move the block to the table's edge, the block slides off of the table, the fixed-based manipulator would not be able to retrieve it. In a more cluttered environment, the wrong action could lead to an object being stuck or wedged in a corner.

While we use manipulation with dynamic actions as an illustrative instance, the problem of planning uncertainty with irrecoverable outcomes

¹ Note that this has also been referred to as future state uncertainty (Kaelbling and Lozano-Pérez, 2013).

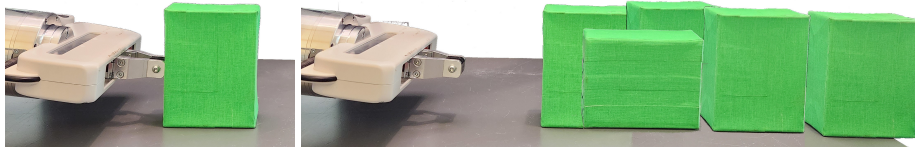


Figure 4.1: The robot gives a quick shove in order to push the green block across the table. As shown, there are many possible outcomes as a result of this action, including the object toppling over or teetering on the edge of the table. In this work we focus on enabling robots to leverage dynamic actions like this while accounting for uncertainty and irrecoverable outcomes, or dead ends.

appears in many contexts. Critical to each of these methods is identifying and then avoiding both the areas of state space from which it is impossible to reach the goal, referred to as *dead ends* (Little and Thiebaux, 2007), and the areas that lead to these dead ends, sometimes referred to as *traps* (Lipovetzky et al., 2016).

Within trajectory generation this can be framed as avoiding areas of inevitable collisions (Fraichard and Asama, 2004). In the context of dynamical systems, Hamilton-Jacobi reachability analysis can be applied to guarantee avoidance of undesirable states (Bansal et al., 2017). Techniques in classical planning often focus on identifying dead-end and trap formulas, in the context of a discrete, factored domain (Kolobov et al., 2010, 2012a) Policy generation methods for MDPs (Markov Decision Processes), also operating on discrete states, are augmented with an iterative or pre-processing step to identify and remove dead-end states (Kolobov et al., 2011, 2012b).

Drawing inspiration from motion planning, we propose a sample-based manipulation planner, GUARD (Guiding Uncertainty Accounting for Risk and Dynamics) for acting in the presence of uncertainty and dead ends. In face of these dual challenges, rather than search for an optimal strategy, the planner adopts a cautious approach, focusing ensuring that, even in the face of uncertainty, the robot does not take a risky action which could lead to a dead end.

We coarsely model the transition function of the action by learning outcome volumes, which define the space of possible resultant configurations. Given this model, we compute *danger zones* as continuous areas of configuration space that capture both dead ends and traps, where the latter of which are computed by taking the preimage of the former. By constraining the sample-based planner to avoid danger zones, we ensure that the robot never takes an action that leads to a dead end. With this guard, we interleave planning and execution: generating and executing an action, resensing the world and repeating until the goal is reached.

To validate GUARD, we explore a series of rearrangement tasks, where the goal is move a target object in to a goal region amongst possible clutter. We focus on environments where dead ends exist but are avoidable (Kolobov et al., 2012b). We present preliminary experimental results in simulation that show that GUARD is able to generate plans that evade the dead ends and successfully reach the goal state.

4.2 Related Work

This work focuses on planning under uncertainty in situations where there are irrecoverable outcomes, i.e. areas of state space from which it is impossible to achieve the goal. In this section we review several thrusts within planning and control that have focused on a similar domains.

4.2.1 Inevitable Collision States

Fraichard and Asama introduce the idea of *inevitable collision states* (ICS) as states where, regardless of the future controls, a robot will collide with the environment (Fraichard and Asama, 2004). For example, if a robot is driving towards a wall, there is a point at which it is too late for the robot to slow to a stop or change direction, thus resulting in a collision. Within air traffic control literature, these are also known as danger zones (Teo and Tomlin, 2003).

ICS have been extended to consider dynamic obstacles (Petti and Fraichard, 2005), uncertainty in obstacle locations (Bautin et al., 2010) and sets of ICS, for improved computational efficiency. (Althoff et al., 2011). ICS assume a continuous dynamics function and focus on collision as the only source of dead ends.

4.2.2 Backwards Reachable Tubes in Dynamical Systems

In context of dynamical systems, Hamilton-Jacobi (HJ) reachability analysis is a verification method for providing safety and performance guarantees (Bansal et al., 2017). One use case of this is verifying that a robot avoids a given, target set of undesirable states.

Avoiding the target set requires the system to also avoid the Backwards Reachable Tube (BRT), the set of states such that trajectories that start from this set can reach some target set. Solving for this, and the safe set of controls, is often framed as a two-player differential game, sometimes called a reach-avoid game (Zhou et al., 2012; Fisac et al., 2015), in which one player, representing the robot, is attempting to steer away from the target set and the other player, representing disturbances from the world, is attempting to steer into the target set. This game is resolved by solving an associated Hamilton-Jacobi-Isaacs (HJI) equation (Evans and Souganidis, 1984). While often restricted to continuous dynamics, HJ reachability analysis for computing BRTs has recently been extended for hybrid systems with controlled and forced transitions (Borquez et al., 2024). There are several parallels between BRTs and our danger zones, which are discussed further in Sec. 4.6.

4.2.3 *Dead-End Detection*

In a factored, discrete state space, where the state is given as a conjunction of literals, it is possible to detect or learn the set of dead-end states.

A dead-end formula is a formula that is true only in states where no goal state is reachable (Lipovetzky et al., 2016). We refer to these states as dead-end states. Lipovetzky et al. (2016) introduces the idea of trap formula as a formula such that if a state satisfies the formula, all states reachable from that state also satisfy the formula. If the trap is mutually exclusive with the goal, then it is a dead-end trap formula, i.e. it is only true in dead-end states (Chrupa et al., 2017).

SixthSense proposes characterizing dead-end states by learning a compact set of literals, nogoods, whose truth implies that the goal is no reachable (Kolobov et al., 2010, 2012a). Dead ends have also been detected by using critical-path heuristics (Steinmetz and Hoffmann, 2016), by detecting action reversability (Morak et al., 2020) and by framing detection as a binary classification problem (Ståhlberg et al., 2021). Steinmetz and Hoffmann (2017) expands existing dead-end detectors to additionally detect traps online. While we draw inspiration from the terminology of dead ends and traps, we cannot directly apply these dead-end detection methods in our continuous, geometric state space.

4.2.4 *Dead Ends in MDPs*

Several MDP formulations and policy-generation algorithms have been proposed to address environments with dead ends. The Stochastic Shortest Path (SSP) problem is an MDP that generalizes the classic deterministic shortest path problem (Bertsekas, 1996). However, SSPs cannot capture environments with dead ends since they assume that there exists at least one proper policy, i.e. a policy that reaches a goal state from any state with probability 1. This does not hold for dead-end states, which cannot reach the goal state. As a consequence, without discounting, solvers like value iteration will not converge in environments with dead ends (Kolobov et al., 2012b).

To address this, Kolobov et al. (2011) proposed the Generalized Stochastic Shortest Path (GSSP) problem, whose MDP-defining tuple includes an initial state. The aforementioned assumption is relaxed to the condition that there exists a proper policy starting from the initial state, rather than from all states, thus allowing for dead ends.

To find a partial policy rooted at the start state, Kolobov et al. (2011) proposes FRET (Find, Revise, Eliminate Traps). FRET iterates between FR: using the Find-and-Revise (Bonet and Geffner, 2003) framework as a heuristic search to update the value function and ET: first using using Tarjan’s algorithm (Tarjan, 1972) on the reachability graph of the value function to identify trap states and then modifying the value function of those states to discourage further exploration. This finds a reward-maximizing policy, within

the set of proper policies.

This framing and algorithmic strategy is similar to [Koenig and Liu \(2002\)](#), who focus on finding a plan that maximizes expected total reward among all plans that achieve the goal. The proposed algorithm identifies traps in a similar style to FRET and eliminates those states but remove them from state space.

[Kolobov et al. \(2012b\)](#) proposed three classes of MDPs whose formulation frames reaching a dead-end state as a cost for the policy the pay. The dead ends are framed as being avoidable (SSPADE), unavoidable and incurring a finite cost (SSPUDE) and unavoidable and incurring an infinite cost (iSSPUDE). To formally balance dead end avoidance with cost, [Teichteil-Königsbuch \(2012\)](#) proposed Safest and Stochastic Shortest Path Problems (S³P) MDPs. S³P MDPs aim to find a policy that minimizes a goal-cost criteria from among policies that maximize a goal-probability criteria.

Finally, GSSPs enable the formulation of MAXPROB MDPs, where the objective is to maximize the probability of reaching the goal, rather than hitting a dead end ([Kolobov et al., 2011](#)).

Two drawbacks of the MDP approaches are, in the context of our domain, that they require an a priori discretization of the state space and that computing a policy, even a partial policy, is very computationally expensive. In contrast, our sample-based search algorithm allows for an adaptive discretization.

4.3 Problem Domain

We consider a rearrangement problem, where the robot’s goal is to manipulate a target object into a goal region. The environment is a horizontal surface, which may have obstacles and other objects. The robot’s action space is a discrete set of non-prehensile dynamic actions with a high degree of action uncertainty. We capture this uncertainty by approximating the set of feasible successor states of an action using learned outcome volumes. To introduce the domain, we next detail the environment setup and the robot’s action space.

4.3.1 Environment

We consider a robot R operating on a finite surface, T that has a boundary $b(T)$. If an object is moved outside the boundary of the surface, the object is unreachable and cannot be acted on. For example, if the surface is a table top, once the object falls off the edge of the table, we would consider it unreachable to the robot.

We consider three types of objects on the surface: the target object, obstacles and non-interactables. There is one, movable target object o_T that the robot aims to move into the goal region. Obstacles, $o \in O_{ob}$, are fixed objects

on the surface. Non-interactables, $o \in O_{ni}$ are objects that the robot and target object cannot contact, similar to the interaction constraint defined in Saleem and Likhachev (2020). As an example, we could imagine disallowing interaction with fragile objects, such as a glass vase. By contrast, the robot and target object can contact, but not penetrate, obstacles.

First, we approximate the geometry of the target object to be a rectangular prism². For the rectangular prism we define three sets of planar, rectangular faces $f \in F_{rp}$. We additionally discretize the planar orientation of the target object. Through these assumption, the configuration of the target object is defined as:

$$q_{o_T} = (f_j, \theta_k, p) \quad (4.1)$$

where $f_j \in F_{rp}$ is the face of the object in contact with the surface, $\theta_k \in \Theta$ is the discretized planar orientation, and $p = (x, y) \in \mathbb{R}^2$ is the planar position. We define $m_{o_T} = (f_j, \theta_k)$ to be the object mode. Hence, given a fixed mode, the configuration space of the object is \mathbb{R}^2 .

Finally, the goal region, G is defined as polygonal region on the surface.

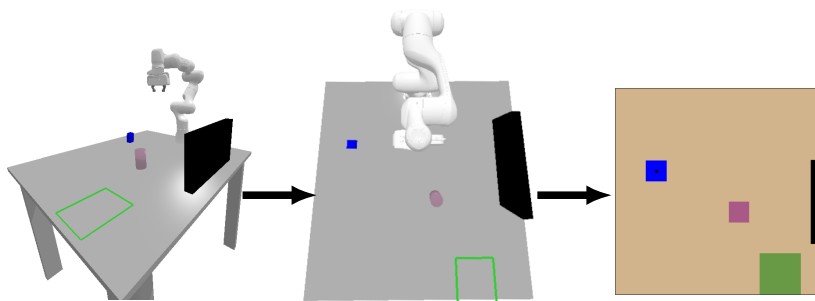


Fig.4.2-(left) shows an example environment where the goal area is highlighted in green, the target object is the blue rectangular prism, the black walls serve as obstacles and the purple cylinder is a glass vase that is a non-interactable. We abstract the 3D world into a 2D planar representation where all obstacles and non-interactables are represented by planar polygons corresponding to their projection onto the surface. Fig.4.2-(right) shows the corresponding 2D abstraction for the aforementioned example environment. Here, fixing the orientation, we show three of the modes corresponding to the different faces of the target object.

Fig.4.3 shows three environments that serve as the experimental domains in Sec. 4.9. The first environment (left), `corner` has no objects on the table top other than the target object. The goal region is in the far corner edge of the table and thus the robot must manipulate the object without letting it fall over the edge. The second environment (middle), `slipperyslope` presents a goal at the start of a long hallway that ends at the edge of the table. Here if the robot overshoots the goal region, the hallway is not wide enough

² We use Trimesh (Dawson-Haggerty et al.) to compute a bounding box, given an STL model. We assume we have models for all of the target objects.

Figure 4.2: The lefthand side show an example environment viewed from the side. The environment includes with a robot, surface (the grey table top), a target object (the blue rectangular prism), an obstacle (in black), a glass vase serving as a non-interactable (in purple) and a goal region (highlighted in green). The center shows this same environment from an overhead view. The righthand side shows the factored 2D representation of our 3D environments. Each of the colors correspond to the lefthand side, but the robot is omitted. This shows one mode, for a fixed choice of target object orientation and face.

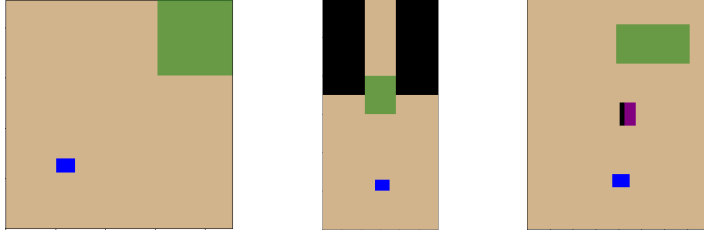


Figure 4.3: We consider three experimental domains: `corner` (left), `slipperyslope` (middle) and `glasswall` (right). The target object, at its starting configuration, is blue, the goal is the highlighted green region, obstacles are black and non-interactables are purple.

for the robot to bring the target object back. In the third environment (right), `glasswall`, the robot must manipulate the object without cracking the right-side of a wall, which is covered in glass (shown in purple), which is marked as a non-interactable. Each of these domains present avoidable dead ends (Kolobov et al., 2012b), where it is feasible to reach the goal while entirely avoiding dead ends.

4.3.2 Actions

The focus of this work is planning with actions where it is necessary to reason over the uncertainty in the outcome. As representative examples, we consider the dynamic and non-prehensile action of `fast-push`.

In the action `fast-push`, shown in Fig.4.1, the robot dynamically pushes an target object (Agboh and Dogar, 2020). This phase, where the object is sliding, and possibly toppling, is bounded and we only execute one action at the end of the previous action’s horizon, i.e. after all objects have come to rest. Thus, we are planning and executing briefly-dynamic manipulation, i.e. manipulation with dynamic phases that have finite time horizon (Mason, 2001).

This action is parameterized by the approach direction ϕ , specified by an angle in the world frame. We define a discrete set of angles Φ and so the action set is a discrete set of ground actions: $A = \{\text{fast-push}(\phi) \text{ for } \phi \in \Phi\}$.

To capture the action uncertainty, for each ground action a , we learn an *outcome volume* $V_\rho(a, f_j, \theta_k)$ that defines possible region(s) of subsequent configurations as a result of taking ground action a for object in mode $m_{o_T} = (f_j, \theta_k)$. $V_\rho(a, m_{o_T})$ is the nominal outcome volume that is independent of the planar position p thus does not account for obstacles. $\mathbf{V}_\rho(a, (f_j, \theta_k, p)) = \mathbf{V}_\rho(a, q_{o_T})$ is a grounded outcome volume, i.e. the outcome volume at a particular planar position p . The ground outcome volume is computed accounting for obstacles in the environment, with the simplifying assumption that if the target object contacts an object, the object halts. Hence we do not model more complex dynamics, such as the object bouncing off of an obstacle or rotating after partial contact. As an action may cause the object to transition in multiple, different modes, the outcome volume is defined as a set of regions over these modes.

The nominal outcome volume is learned from data collected on the real

robot and, more specifically, $V_\rho(a, f_j, \theta_k)$ captures the convex hull of $\rho\%$ of the data³ Thus, a higher ρ corresponds to a more conservative estimation of the outcomes. Note that outcome volumes define the space of possible resultant configurations and do not capture a probability distribution.

Currently we are using relatively few samples to construct the outcome volumes and so our estimates are quite unreliable. In principle, this could be improved by acquiring more training samples or adjusting the estimates to be more or less conservative.

In between each `fast-push` action, we plan a `move` action which is used to relocate the robot to the start of the next action. We assume the `move` action has negligible uncertainty.

4.4 GUARD: Algorithmic Overview

Given the dynamic actions and the outcome volumes that approximate their dynamics, the goal is to plan a sequence of actions the goal, which is defined as manipulating the target object to within the goal region. One strategy for handling this high degree of action uncertainty is adopt a reactive planning strategy where the robot generates a plan, executes the first action, senses the new state of the world and repeats, continuing this process until the goal is reached.

This strategy, however, relies on on the idea that no action is catastrophic and there are no dead ends (Little and Thiebaux, 2007). Stated another way, this assumes that the robot will always have the ability to try again and replan and that there is never an action that, once taken, make it impossible for achieve the goal. Essentially, straight-forward reactive planning does not empower the robot to act cautiously in the face of danger.

Given the possibility of dead ends, we want to encourage the robot to not take irrecoverable actions, actions from which it cannot replan and try again. To enable this, we propose first offline characterizing *danger zones*, areas of configuration space where it is either impossible or unlikely to achieve the goal. Danger zones can be viewed as capturing dead-end states, but in the continuous configuration space. With this, we can constrain the sample-based graph planners to explore configuration space while avoiding danger zones. Online we then employ a reactive replanning strategy, but with safety guards to discourage the planner from taking an action that would make it impossible to achieve the goal.

In the following sections we present our algorithm GUARD: Guiding Uncertainty Accounting for Risk and Dynamics. It first constructs two core data structures: `ActionRegions` and `DangerZones`. `ActionRegions` define where, in the target object’s configuration space, each ground actions is feasible. An action is feasible to execute if the target object is collision-free and stably supported and if the swept volume of the robot’s execution motion in the robot’s collision-free, reachable workspace. `ActionRegions`

³ There may be multiple possible convex hulls that satisfy this condition; we employ the simple following strategy. We construt a circle of radius r at the centroid of the set of points. We iteratively increase r until $\rho\%$ of the points are contained within the circle. We then return the convex hull of this set of points.

capture whether the preconditions of executing an action are satisfied.

From this, we next compute `DangerZones`, which are also defined in the target object’s configuration space. We construct zone 0, z_0 , as the areas of configuration space where it is not feasible to reach the goal, either because there are no feasible actions (as characterized by the `ActionRegions`) or because the area is disconnected, in configuration space, from the goal. By computing preimages, we define z_i defines the union of the set of all configurations from which all actions reach z_{i-1} . Note that due to the formulation of the ρ -outcome volumes, we only capture the possibility of transitioning, not the probability, and a higher ρ -value will lead to a more conservative approximation. Summarizing, `DangerZones` thus characterize, at varying levels, areas of configuration space the robot wants to avoid.

Given these two data structures, we propose two sample-based graph planners that interleave search and execution: `GUARD-G` and `GUARD-T`. Both algorithms build graphs in the object’s configuration space, with the constraint of avoiding `DangerZones`, thus ensuring that the robot never takes an action that leads to a dead end. Online, the algorithms augment the graph to find a sequence of actions. Following the reactive planning strategy, we execute the first action in the sequence, resense the environment and repeat until the goal is achieved. We can view generating the sequence of actions at each step as justification for taking the first action.

We detail the computation of `ActionRegions` and `DangerZones` in Sec. 4.5 and Sec. 4.6, respectively. We briefly discuss theoretical guarantees (Sec. 4.7) that are used to inspire the search algorithms presented in Sec. 4.8.

4.5 *GUARD: Action Regions*

`ActionRegions` provide a continuous mapping, in the object’s configuration space, of the set of feasible actions. A ground action is feasible at a configuration if the object and robot are not in collision with the environment, the object is stably supported by the surface and the action is within the robot’s reachable workspace⁴.

We compute an action region mapping within each mode, m_{o_T} . We start by computing, for each mode and action, where the action is *infeasible*. We take the complement of the union of these areas, thus computing the areas where each action is feasible. Within a mode, we combine these areas for each action (via a shattering algorithm, Fig.4.4) to produce the `ActionRegions` mapping.

Returning to the first step, for each mode m_{o_T} and each ground action a we compute the following infeasible areas by taking Minkowski differences. Given the factored representation of the object’s configuration, we compute these areas in \mathbb{R}^2 , making this computation efficient.

- We define the object as not being stable supported by the surface if the center of mass of the object is outside the boundary of the surface. Hence

⁴ For example, for a mobile robot we compute the reachable workspace as the set of configurations where the robot is entirely supported by the surface. For a fixed-based robot, we assume access to a precomputed, possibly simplified, computation of its workspace

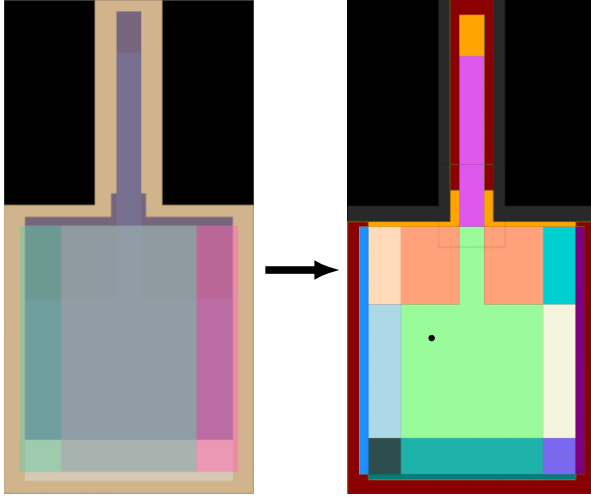


Figure 4.4: (Left) In the `slippery slope` environment, for one mode, we overlay the `FeasibleArea` for four actions. (Right) The result of the shattering algorithm is to compute all possible intersections of the subsets of `FeasibleAreas` such that the space is partitioned into regions where each region has a specified set of actions, thus defining the `ActionRegions`. Given this representation, we can query a specific point in configuration space (shown as the black dot) to retrieve the set of feasible actions.

this allows the object to extend or overhang from the surface, as long as it is stably supported. For each boundary $b \in b(T)$, $\text{support}(o_T, b)$ computes, in the object frame, the subset of the object that must be within the boundary in order for the object to be stably supported⁵. The set of configurations where the object is not stably supported is:

$$B_o = \{b \ominus \text{support}(o_T, b) \mid \forall b \in b(T)\} \quad (4.2)$$

- Let $(o_T, R(a))$ be, in the object frame, the combined polygon of the target object o_T and the swept volume of the robot R executing action a . The set of configurations where either the robot or the object would be in collision with an obstacle $o \in O_{ob}$ is:

$$B_{ob} = \{o \ominus (o_T, R(a)) \mid \forall o \in O_{ob}\} \quad (4.3)$$

- Likewise, we can compute the set of configurations where either the robot or the object would be in collision with a non-interactable object $o \in O_{ni}$ in the same fashion:

$$B_{ni} = \{o \ominus (o_T, R(a)) \mid \forall o \in O_{ni}\} \quad (4.4)$$

Finally, we require the swept volume of the robot R executing action a to be within the robot's reachable workspace. The set of configurations outside the reachable workspace is:

$$B_R = \{q \mid q \notin \text{reachableWorkspace}(R, a)\} \quad (4.5)$$

We can then compute the areas where each action is feasible by taking the complement of the union of these areas:

$$\text{FeasibleArea}(a) = \neg(B_o \cup B_{ob} \cup B_{ni} \cup B_R) \quad (4.6)$$

⁵ We assume a uniform mass distribution and a given center of mass.

For each mode, we next combine the feasible areas for each action to provide a continuous mapping of action feasibility, hence constructing the `ActionRegions`.

For each possible combination of actions (e.g. $\{a_0\}$, $\{a_1, a_3\}$, $\{a_0, a_1\}$, etc.) we intersect the corresponding feasible areas to define the region where that combination of actions is feasible. Visualized in Fig.4.4, we refer to this as the shattering process and it can be thought of taking the Venn diagram of all of the feasible areas and splitting up the various subsets.

Having constructed the `ActionRegions`, a simple lookup function can be used to define the function $A_F(q_{o_T}) = \{a_1, a_2, \dots\}$, which outputs the discrete list of ground actions that are feasible in configuration q_{o_T} .

4.6 GUARD: Danger Zones

`DangerZones` capture the dead-end areas of the object's configuration space, i.e. areas where it is impossible to reach the goal z_0 , and areas that might unavoidably lead to these dead-end areas, z_i .

4.6.1 Zone 0

We define zone 0, z_0 as areas in object's configuration space where it is impossible to reach the goal. We construct this in two steps.

First, z_0 includes the regions where there are no feasible actions, as computed by the region generation process detailed in Sec. 4.5. Second, z_0 also includes regions where, based on a region-based reachability graph, it is impossible to reach the goal.

Each action region is a vertex in region-based reachability graph. A directed edge exists between two vertices (ar_0, ar_1) if there exists an action a such that a is feasible in region ar_0 and, by taking action a within region ar_0 it is feasible to transition to ar_1 , accounting for obstacles. This reachability is computed by taking Minkowski sum of region ar_0 and the ρ -outcome volume $V_\rho(a, m_{ar_0})$ where $\rho = 1$ and m_{ar_0} is the mode of region ar_0 . Given this graph, we define $AR_{de}(G)$ as the set of action regions where the action region's corresponding vertex does not have a path in the graph to any vertex for a goal action region, for a goal G .

Summarizing:

$$z_0(G) = \{q \mid |A_F(q)| = 0\} \cup AR_{de}(G) \quad (4.7)$$

As a slight abuse of notation, we omit the G and thus z_0 defines a continuous set of dead-end states — if the target object enters z_0 , it is impossible to reach the goal G .

4.6.2 Preimages and Subsequent Zones

For z_0 , we define the preimage of z_0 under an action a as:

$$\text{preimage}_a(z_0, \rho) = \{q \mid \mathbf{V}_\rho(a, q) \cap z_0 \neq \emptyset\} \quad (4.8)$$

Hence $\text{preimage}_a(z_0, \rho)$ is the set of configurations where ρ -outcome volume overlaps with z_0 . Stated another way, some portion of $\rho\%$ of the outcomes of taking action a from configuration q for $s \in \text{preimage}_a(z_0, \rho)$ will result a configuration within z_0 . We can compute this preimage by again taking the Minkowski sum of zone 0 and the ρ -outcome volume, while accounting for obstacles in the environment.

From this we define subsequent zones z_i , for $i > 0$, as:

$$z_{i,\rho} = \{q \mid q \in \text{preimage}_a(z_{j < i}, \rho) \forall a \in A_F(q)\} \cup z_{i-1,\rho} \quad (4.9)$$

First, we compute the set of configurations for which every feasible action, that configuration is in the preimage of previous zones. Therefore, every possible action results in some portion of the $\rho\%$ outcomes within $z_{j < i}$. To define $z_{i,\rho}$, we union this set with all previous zones, thus nesting zones.

The result of this is that we cannot guarantee that any of the actions are safe in the sense that they not transition the object into a subsequent zones. However, this also does not guarantee that every action will necessarily transition the object into subsequent zones. In this context, higher ρ values correspond to more conservative approximations and it may not even be likely that such an action transitions the object into subsequent zones (the condition is only that it is possible).

Generalizing (4.8) to z_i :

$$\text{preimage}_a(z_i, \rho) = \{q \mid \mathbf{V}_\rho(a, q) \cap z_i \neq \emptyset\}. \quad (4.10)$$

Hence, if z_0 defines the dead-end area of configuration space, the subsequent z_i represent the slippery slope to this dead end. As an illustrative example, Fig. 4.5 shows the zones for the `slippery slope` environment. The darkest red corresponds to z_0 and two subsequent zones are colored in increasing brighter reds. Unsurprisingly, the riskiest areas are near the edges of the surface and in the narrow hallway, where the constrained space limits available actions.

The formulation of the subsequent zones (i.e. z_i for $i > 0$) mirror, in varying degrees, the definitions of ICS, BRTs and traps, as described in Sec. 4.2. z_i captures the area of configuration space where all feasible actions *could* transition the object into a dead-end state, but are not guaranteed to. This is a weaker condition than ICS and traps, where entering into an inevitable collision state or trap ensures that that a collision or dead end, respectively, are going to occur. BRTs capture where the system cannot guarantee that the undesirable target set will be avoided, which is closer to the `DangerZones` formulation.

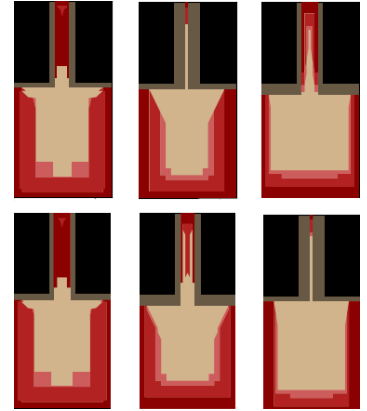


Figure 4.5: `DangerZones` in the `slippery slope` domain. Here we show the zones across six modes, capturing the three object faces and two orientations. The obstacles (and configuration space obstacles) are given in black (and grey). We color z_0 in dark red and show z_1 and z_2 in brighter shades.

4.7 Theoretical Guarantees

We next show how leveraging of `DangerZones` empowers the robot to choose actions that avoid dead ends. Recall from Sec. 4.6 that z_0 defines the set of dead ends and z_i defines the union of the set of all configurations from which all actions reach z_{i-1} . Following in the definition of z_i , at the limit, we define z_∞ .

We start by assuming that the learned ρ -outcome volumes perfectly capture the $\rho\%$ of outcomes, i.e. that there is at most a $(1 - \rho)$ probability of the outcome not being within the learned volume. We conservatively assume that all of the unmodeled outcomes lead to a danger zone, z_i . Given this, for a plan of length n that never enters z_∞ , with probability $(1 - (1 - \rho))^n$, the robot can execute the entire plan without entering z_∞ .

Likewise, following the same assumptions, for a plan of length n that never enters z_n , with probability $(1 - (1 - \rho))^n$, the robot can execute the entire plan without entering z_0 .

We now set $\rho = 1$, hence all outcomes are modeled by the learned volume. Under this assumption, we show that:

Proposition 5. *If the robot only takes actions that avoid z_∞ , the robot will never reach a dead-end state.*

Proof. Assume for the sake of contradiction that the robot reaches a dead-end state. Therefore, there exists a step k in which the robot transitioned into z_0 . By the construction of the `DangerZones`, this implies that at step $k - 1$ the robot was in z_1 . Backtracking further, this implies that at step 0 the robot was in z_{k+1} . By definition, $z_{k+1} \subset z_\infty$. However, since we assumed the robot never entered z_∞ , this is a contradiction. Therefore, the robot will never reach a dead-end state. ■

In practice, we do not compute to z_∞ and we cannot guarantee that the learned volumes perfectly capture the dynamics. Even still, these ideas motivate our search algorithms: by selecting actions outside of the `DangerZones`, the robot can avoid dead ends and continue to act. This enables the robot, in the face of uncertainty, to continue replanning and trying again. Here we focus on certifying against reaching a dead end, which is a weaker condition than guaranteeing the robot achieves the goal.

Finally, we note that ρ and n , the maximum zone computed, act as knobs that control the conservatism of the planner. While the planner we present in this work treats them as fixed parameters, an alternative strategy could initialize them to high values, and iteratively decrease them until a plan is found, in an attempt to find the safest plan possible.

4.8 GUARD: Search

We propose two sample-based manipulation search algorithms, GUARD-G and GUARD-T, to generate actions. Taking inspiration from the results in Sec. 4.7, we constrain the search algorithm’s exploration to avoid `DangerZones` in order to ensure that the robot never takes an action that leads to a dead end.

Both search algorithms construct graphs where a vertex v_i corresponds to a configuration q_i of the target object. Two vertices are connected by a directed edge, annotated with an action, if the configuration of the child vertex is contained within the ρ -outcome volume of that action taken from the configuration of the parent vertex. This captures that, through this action, it is *possible* to reach the child vertex from the parent vertex.

For a fixed n , we constrain the graph such that the robot can only take actions that avoid entering z_n . Doing so requires imposing two *guard criteria*. First, we only sample configurations q_i that are outside of z_n . Second, for each q_i we compute the feasible action set, $A_F(q_i)$. Let $A_G(q_i) \subseteq A_F(q_i)$ be the *guarded action set*, the set of actions where $V_\rho(a, q_i)$ has no intersection with the preimage of z_n , hence $V_\rho(a, q_i)$ does not intersect with z_{n+1} . This insures that, for any outcome, any subsequent configuration, within the ρ -outcome volume, there exists a feasible action that satisfies the first criteria, an action that is not in z_n . If $A_G(q_i)$ is non-empty, i.e. there exist guarded actions, configuration q_i can be added to the graph as vertex v_i .

At each step, we augment the graph, respecting the above condition, and search for a plan, i.e. a path from the current configuration to a configuration within the goal region. This path corresponds to a sequence of actions⁶. Following in the framework of reactive planning, the robot executes the first action in the plan, observes the resulting configuration and repeats this process until it reaches the goal.

The nature of the graph augmentation differs between the two search algorithms. GUARD-G builds a graph prior to any execution and continues to augment it in a multi-query fashion, inspired by Probabilistic Roadmaps (PRM) (Kavraki et al., 1996). GUARD-T constructs a new tree at each step, in a single-query fashion inspired by Rapidly-Exploring Random Trees (RRT) (LaValle and Kuffner, 2001a). As explained below, they differ both in their reuse of computation and in the perspective their edge set captures with respect to determinization.

4.8.1 GUARD-G

GUARD-G computes a graph in an offline stage before searching and expanding the graph online. We construct a multi-edge directed graph $G = (V, E)$. To generate vertices, we rejection sample target object configurations, based on the vertex guard criteria. Some $k\%$ of the time, we explicitly sample target

⁶ Given a plan, we lazily (Bohlin and Kavraki, 2000) evaluate if there exists a feasible path for the robot to execute this action sequence. If there does not, we invalidate the plan and continue searching.

Algorithm 3 GUARD-T(q_s)

```

1:  $T.$ Init( $q_s$ )
2: while TIMEREMAINING() do
3:    $q_{rand} =$  RANDOMCONFIG()
4:    $q_{near} =$  NEARESTNEIGHBOR( $T, q_{rand}$ )
5:    $a =$  COMPUTEBESTCONTROL( $q_{near}, q_{rand}$ )
6:    $q_{new} =$  DETERMINIZEOUTCOME( $q_{near}, a$ )
7:   if ISSAFENODE( $q_{new}$ ) then
8:      $T.$ AddNode( $q_{new}$ )
9:      $T.$ AddEdge( $q_{near}, q_{new}, a$ )
10:  if INGOAL( $q_{new}$ ) then
11:     $p =$  EXTRACTPATH( $T, q_s, q_{new}$ )
12:    return GETFIRSTACTION( $p$ )

```

object configurations from the goal region. We add all possible edges and thus, for a vertex v_i representing configuration q_i , the edge set is:

$$E = \{(v_i, v_j, a) \mid a \in A_G(q_i) \wedge q_j \in V_\rho(a, q_i)\} \quad (4.11)$$

Depending on the density of the graph, a vertex could have multiple edges with the same action a , corresponding to various possible outcomes as a result of executing that action. Hence, we can view the edge set of the graph as offering a many-outcomes determinization, rather than a single-outcome and all-outcomes determinization (Yoon et al., 2007). While the robot, in executing the action, cannot control the outcome, given the guard conditions, we are know that whichever outcome occurs, will not lead to a dead end.

Having constructed the graph, online we interleave search, graph expansion and execution. At each step, we add the current configuration as a vertex in the graph, along with its corresponding edges. We search for a path to the goal⁷. If no path exists, we expand the graph by sampling more vertices and adding the corresponding edges. If after a fixed number of expansions, we are still unable to find a plan, we declare a no-path failure.

Given a plan, we execute the first action and repeat until the target object has reached the goal region.

4.8.2 GUARD-T

GUARD-T constructs a search tree from scratch at each step. The tree construction method (Algorithm 3) is a variant of the standard RRT algorithm (LaValle and Kuffner, 2001a). At each step, we sample an target object configuration, q_{rand} , sampling from the goal region $k\%$ of the time (Line 3). Since this configuration is only used to encourage expansion, we do not apply the vertex guard criteria. We next compute q_{near} , the closest node in T to q_{rand} ⁸.

We select an action $a \in A_G(q_{near})$ that most moves in the direction of

⁷ As a minor optimization, to discourage the planner from alternating between a variety of possible routes, we attempt to reuse previously found plans. To do so, we cache the action sequence found at the first step and, if action sequence is still feasible from the resulting configuration and still reaches for the goal, we continue to use this sequence rather than searching for a new one. If the sequence has become infeasible, we search anew. Additionally, in searching for paths, we select the shortest one (with respect to the number of actions). Hence while we make no guarantees with respect to overall path length, we do still bias towards shorter paths.

⁸ Given the factored configuration representation, the distance function is a weighted combination of the euclidean distance between the planar positions, the distance between the discretized orientations and an indicator function over the faces.

q_{rand} . Specifically, we compute which action’s ρ -outcome volumes minimize the distance to q_{rand} .

To generate a new object configuration q_{new} we randomly sample from the ρ -outcome volumes associated action a (Line 6). If q_{new} satisfies the vertex guard criteria, we add edge (v_{near}, v_{new}, a) to the tree, where v_i represents to configuration q_i . We terminate once a plan to the goal is found or a maximum exploration time limit is reached. Like GUARD-G, we execute the first action in the plan, observe and repeat until the goal is reached.

We highlight two particular ways that GUARD-T differs from GUARD-G. First, GUARD-T constructs a new tree at each step, forcing it to repeatedly re-explore the configuration space. Experimentally we’ll see how this, unsurprisingly, slows the algorithm down. Second, the edge expansion technique of sampling an outcome more closely aligns with a single-outcome determinization strategy which also, in some ways, limits the exploration⁹.

4.9 Experiments

We present a preliminary evaluation of GUARD across the three environments shown in Fig.4.3. First, we describe the three baselines that we compare against, each of which perform a search not informed by DangerZones. We next describe the experimental setup and reporting procedure before finally describing the experimental results.

4.9.1 Algorithms

We compare against three baselines: GUARD-GB, GUARD-TB and DDR. GUARD-GB and GUARD-TB are ablations of GUARD-G and GUARD-T (respectively) where DangerZones are not computed and graph creation is not constrained by the criteria described in Sec. 4.8.

The final baseline, DDR (Discrete Determinize-and-Replan), operates in a discrete configure space. We discretize the surface into a series of equally-sized grid blocks. The action set of each block is composed only of actions that are feasible for every region that overlaps with the block, hence ensuring that an action is only deemed feasible in a block if it is feasible at every location within the block. We also explicitly construct a transition function by computing the set of reachable blocks¹⁰.

Given this discretization, DDR interleaves search and execution. At each step, we use Dijkstra algorithm to search for a path, leveraging the same distance function as GUARD-T and adopting a single-outcome discretization that selects the outcome, corresponding to a block, randomly¹¹. Like GUARD, given a path, we execute the first step in the path and repeat until the target object has reached the goal region. In contrast to the GUARD algorithms, which achieve adaptive discretization through sampling, DDR adopts a fixed, uniform discretization of the state space.

⁹ We could alternatively consider both a version of GUARD-G which constructs a new graph at each step and a version of GUARD-T that saves the tree structure, building off of it at each step rather than starting anew

¹⁰ For each block b , for each feasible action a of that block, we compute the reachable area by taking the Minkowski sum of the block at the ρ -outcome volume $V_\rho(a, m_b)$ where $\rho = 1$ and m_b is the mode of the block b , while accounting for obstacles. As a simplification, we assign a uniform transition probability for any block b' that intersects with this reachable area. Unsurprisingly, this is the most computationally-expensive step of the preprocessing.

¹¹ It would be equally valid, and an interesting comparison, to consider an all-outcome discretization as well as a stronger baseline that accounts for DangerZones and respects the guard criteria.

Environment	Algorithm	Results				Time		# Steps (SE)
		S	MS	NP	NA	Offline (SE)	Online (SE)	
Corner	GUARD-G	10	0	0	0	22.5 (0.32)	19.3 (4.9)	23.2 (3.4)
	GUARD-T	8	2	0	0	14.3 (0.0)	246 (53)	27.4 (4.6)
	GUARD-GB	7	1	0	2	3.01 (0.03)	6.77 (3.1)	26.6 (3.6)
	GUARD-TB	0	0	5	5	2.08 (0.0)	29.2 (5.1)	8.4 (2.3)
	DDR (0.05, 0.05)	4	0	2	4	4280 (0.0)	810 (120)	22.5 (2.8)
SlipperySlope	GUARD-G	7	0	3	0	48.3 (0.36)	21.3 (7.8)	15.5 (2.5)
	GUARD-T	4	2	4	0	59.0 (0.0)	255 (39)	29.3 (4.2)
	GUARD-GB	8	0	2	0	6.9 (0.033)	11.9 (7.6)	8.1 (1.0)
	GUARD-TB	4	0	3	3	4.42 (0.0)	22.1 (8.0)	7.3 (0.97)
	DDR (0.05, 0.05)	2	0	6	2	1950 (0.0)	49.3 (11)	5.2 (1.1)
GlassWall	GUARD-G	9	0	1	0	133 (1.7)	69.9 (16)	17.4 (3.2)
	GUARD-T	10	0	0	0	132 (0.0)	97.0 (14)	17.5 (1.7)
	GUARD-GB	1	0	7	2	14.9 (0.12)	39.9 (4.8)	16.4 (4.8)
	GUARD-TB	0	0	7	3	10.8 (0.0)	120 (15)	8.3 (1.3)
	DDR (0.05, 0.05)	1	0	3	6	3220 (0.0)	277 (64)	8.5 (2.2)

4.9.2 Experimental Setup

For each environment, each algorithm is evaluated across 10 trials. We classify the output of each iteration into four categories:

Success (S) The target object was successfully moved into the goal region.

Max Steps (MS) As a practical matter, we set an upper limit on the number of actions the planner can execute before terminating. This failure mode does not necessarily mean that it is impossible to reach the goal, only that the robot did not reach it within the step limit.

No Path (NP) The planner was unable to find a path from the current configuration to the goal region. Again, this does not necessarily mean that it is impossible to reach the goal. Instead, it could correspond to the graph not being large enough, i.e. that more graph augmentations would be needed for GUARD-G or GUARD-GB or that a larger time limit is needed for GUARD-T and GUARD-TB. For DDR, the randomized determinization choice could leave no path to the goal.

No Actions (NA) The planner has reached a configuration where there are no feasible actions. This corresponds to a dead end.

We also record the computation time, which we split into the offline time and online time. For GUARD-T the offline time is computation of the `ActionRegions` and `DangerZones`. For GUARD-G this additionally

Table 4.1: We evaluate the two proposed algorithms and three baselines across the three experimental domains visualized in Fig.4.3. We report the outcomes, for ten iterations, provide the computation time, in seconds, and the number of steps. For DDR the spatial discretization is provided. All parameters were kept constant except for `slipperyslope` where GUARD-T and GUARD-TB used a longer max time.

includes the initial graph construction. The offline time for GUARD-TB and GUARD-GB is analogous, excluding the computation of `DangerZones`. The offline time for DDR is the instantiation of the MDP. Finally, while we are not optimizing with respect to path length, we also include the average number of steps in the execution iteration.

For all algorithms, we use a conservative $\rho = 0.99$.

4.9.3 Results

Table 4.1 presents the results of each of the algorithms across the three domains. GUARD is able to generate action sequences that avoid dead ends, with zero No Action (NA) results, in contrast to the three baselines. While GUARD-G and GUARD-T have several No Path (NP) results, as stated above this is, in our experience, a consequence of limiting the computation time of the planner. In the case of GUARD-G, we use a random, uninformed vertex sampling strategy, when more focused sampling could reduce computation time. Further experiments would be needed to confirm both hypotheses.

Unsurprisingly, while the offline time for GUARD-G is slightly longer compared to GUARD-T due to the graph construction step, GUARD-T has a significantly longer online computation time. As mentioned in Sec. 4.8, this is largely because GUARD-T constructs a new tree from scratch each time, hampering it from reusing computation in the way GUARD-G does. Additionally, as compared to the baselines, the additional offline time required for both GUARD algorithms is dominated by the construction of the region-based reachability graph.

With respect to computation time, DDR requires a large amount of offline computation time in order to construct the discrete state space and, in particular, to explicitly construct the transition function. We are exploring lazily constructing the transition function as well as decreasing the online computation time.

Finally, we note that these results are preliminary and a more rigorous evaluation of GUARD is needed to validate its properties.

4.10 Discussion

We present GUARD, a sample-based planning algorithm that addresses manipulation under uncertainty while accounting for dead ends. In a world where some actions can make it impossible to achieve the goal, GUARD adopts a cautious approach. We construct danger zones that capture where certain actions may lead to dead ends, and then constrain the search algorithms to avoid these zones, thus ensuring the robot can always continue acting in its pursuit of the goal. We show some initial experiment results to verify our method in comparison to a few baselines that do not explicitly account for dead ends.

In addition to the discrete action space, GUARD’s representations pose several limitations. By planning in a 2D space, abstracting the target object to a template shape and only considering sticking contact in object-obstacle interactions, we simplify the dynamics of the dynamic actions.

The work presented here is preliminary and we are interested in several immediate future steps. First, a more rigorous experimental evaluation is needed to fully validate GUARD. In addition to more environments and iterations, as discussed in Sec. 4.8, there are a number of additional ablations and baselines that would further illuminate important algorithmic choices. We focused on a relatively small action space (four *fast-push* action) and thus it would be interesting to incorporate other dynamic actions such as toppling or flipping. Finally, GUARD is a very cautious algorithm, operating on the assumption that dead ends are entirely avoidable. A natural extension would be to allow GUARD to modulate its level of caution, even allowing for tradeoffs in the case of unavoidable dead ends.

5

Conclusion

This thesis focuses on enabling robots to robustly perform complex, multi-step manipulation tasks, like chopping vegetables or wielding a wrench. These tasks require the robot to solve for both the sequences of actions and the discrete and continuous parameters of those actions, subject to constraints relating to both geometry and physics. To tackle this, we adopt a model-based approach wherein we equip the robot with composable models of the world and develop planning frameworks that use the models to sequence complex behaviors. We propose both models and algorithms that enable robots to make choices that are robust to uncertainty. We consider three manipulation domains: in-hand manipulation, forceful manipulation and briefly-dynamic manipulation.

First we focus on enabling robots to regrasp objects by repeatedly pushing against features in the environment, thus reorienting the object in-hand (Chapter 2). To enable this, we propose both a compact model that defines how the pushed object will move in-hand and a planning algorithm that propagates the model forward in order to generate a sequence of many reorienting pushes. Our model extends the notion and construction of the motion cone, thus abstracting away the complex physics that govern how a grasped object can move when it is pushed against the environment. This abstraction defines the set of reachable object motions, which enables the planner to find strategies an order of magnitude faster than previous methods. Additionally, we show how operating within a subset of this mapping enables robust pushing.

Next, we define forceful manipulation tasks as those where the ability to generate and transmit the necessary force to objects is an active limiting factor that constrains the robot's choices (Chapter 3). Opening a childproof-bottle, twisting a nut on a bolt and chopping vegetables serve as three illustrative instances of forceful manipulation tasks. We introduce force-related constraints that explicitly consider torque and frictional limits and integrate these, alongside motion constraints, into an existing state-of-the-art task and motion planning (TAMP) framework. We propose using cost-sensitive planning to find strategies that are robust to bounded uncertainty in the physical

parameters of the force-related constraints.

Finally, we consider briefly-dynamic manipulation, where action with bounded periods of dynamics, like shoving or toppling, are sequenced together (Chapter 4). While utilizing dynamic actions enables robots to expand their dexterity, such actions are also characterized by the fact that their outcomes are non-deterministic and can lead to dead-ends. We propose learning a coarse model of the outcome dynamics which captures this uncertainty. We contribute a sample-based planning framework, GUARD, that uses this model to continuously characterize the dead-end areas of configuration space and then search for plans that avoid these areas.

Below we briefly describe avenues of future research as well as a few concluding remarks.

5.1 Future Directions

While there are many possible future direction, we highlight expanding the role of reactivity and feedback.

Throughout this thesis, we consider several different paradigms with respect to open-loop versus closed-loop execution. Our in-hand manipulation planner executes a series of open-loop quasistatic pushes in an open-loop fashion (open-open). Our forceful manipulation TAMP framework executes a closed-loop series of actions in an open-loop fashion (closed-open). Finally, our briefly-dynamic manipulation planner executes a series of open-loop dynamic pushes in a closed-loop fashion (open-closed)¹.

In a sense, this captures our belief that future manipulation systems should include combinations of closed and open-loop planning and control. However, the thesis does not investigate two core research questions with respect to this structure: (1) what are the sources of feedback? and, perhaps more connected to the core of this thesis, (2) how is that feedback integrated into our decision-making frameworks? Investigating both of these questions exciting opportunities to develop more reactive frameworks that can handle richer sources of uncertainty, unexpected outcomes and disturbances.

For example, as an initial step in this direction, we proposed using the robot's proprioception as a source of feedback in the context of manipulating objects with unknown force requirements and integrated simple strategies for error-handling and replanning in a TAMP framework². In this context, the feedback updated the robot's model of the force required to open a heavy drawer.

More broadly, information-gathering actions could be leveraged alongside robust planning to refine estimates of uncertain quantities and update our models of the world, enabling the robot to improve with experience.

¹ We admit that the thesis is therefore missing an instance of closed-closed, which would complete the quadrant.

² Project led by Rachel Lu. See <https://www.youtube.com/watch?v=TMkYot4rAbo> for details

5.2 *Concluding Remarks*

The vision of this thesis is to take a step towards a world in which robots are cooking dinner at home, packing supplies in hospitals and cleaning up messy classrooms. We argue that fundamental to solving these tasks is enabling robots to reason over geometry and physics, such that they can tackle the marvelous messiness of manipulation. We take a model-based approach, contributing models and algorithms, because this places the onus on the robot to reason over how to solve the hybrid, combinatorial constraint satisfaction problem induced by these manipulation missions. We believe this is critical to enabling the ultimate goal – robot behavior that generalizes across a wide array of tasks and environments.

Bibliography

Wisdom Agboh and Mehmet Dogar. [Pushing fast and slow: task-adaptive planning for non-prehensile manipulation under uncertainty](#). In *WAFR*, pages 160–176. Springer, 2020.

Aliakbar Akbari, Muhayyuddin, and Jan Rosell. [Task and motion planning using physics-based reasoning](#). In *ETFA*, pages 1–7. IEEE, 2015.

Alin Albu-Schaffer, Christian Ott, Udo Frese, and Gerd Hirzinger. [Cartesian impedance control of redundant robots: Recent results with the DLR-light-weight-arms](#). In *ICRA*, volume 3, pages 3704–3709. IEEE, 2003.

Daniel Althoff, Christoph Brand, Dirk Wollherr, and Martin Buss. [Computing unions of inevitable collision states and increasing safety to unexpected obstacles](#). In *IROS*, pages 3114–3119. IEEE, 2011.

Haruhiko Asada and Andre By. [Kinematics of Workpart Fixturing](#). In *ICRA*, volume 2, pages 337–345. IEEE, 1985.

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire Tomlin. [Hamilton-jacobi reachability: A brief overview and recent advances](#). In *CDC*, pages 2242–2253. IEEE, 2017.

Antoine Bautin, Luis Martinez-Gomez, and Thierry Fraichard. [Inevitable collision states: A probabilistic perspective](#). In *ICRA*, pages 4022–4027. IEEE, 2010.

Dmitry Berenson, Siddhartha Srinivasa, Dave Ferguson, and James Kuffner. [Manipulation planning on constraint manifolds](#). In *ICRA*, pages 625–632. IEEE, 2009.

Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. [Task Space Regions: A framework for pose-constrained manipulation planning](#). *IJRR*, 30(12):1435–1460, 2011.

Dimitri Bertsekas. [Dynamic programming and optimal control](#). *JORS*, 47(6): 833–833, 1996.

Robert Bohlin and Lydia Kavraki. [Path planning using lazy PRM](#). In *ICRA*, volume 1, pages 521–528. IEEE, 2000.

Blai Bonet and Hector Geffner. *Faster heuristic search algorithms for planning with uncertainty and full feedback*. In *IJCAI*, pages 1233–1238. Citeseer, 2003.

Javier Borquez, Shuang Peng, Yiyu Chen, Quan Nguyen, and Somil Bansal. *Hamilton-Jacobi Reachability Analysis for Hybrid Systems with Controlled and Forced Transitions*. *RSS*, 2024.

Oliver Brock and Lydia Kavraki. *Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces*. In *ICRA*, volume 2, pages 1469–1474. IEEE, 2001.

Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. *Leveraging Cone Double Description for Multi-contact Stability of Humanoids with Applications to Statics and Dynamics*. In *RSS*, 2011.

Nikhil Chavan-Dafle and Alberto Rodriguez. *Stable Prehensile Pushing: In-Hand Manipulation with Alternating Sticking Contacts*. In *ICRA*, pages 254–261. IEEE, 2018.

Nikhil Chavan-Dafle and Alberto Rodriguez. *Regrasping by Fixtureless Fixturing*. In *CASE*, pages 122–129. IEEE, 2018.

Nikhil Chavan-Dafle and Alberto Rodriguez. *Sampling-based Planning of In-Hand Manipulation with External Pushes*. In *ISRR*, pages 523–539. Springer, 2020.

Nikhil Chavan-Dafle, Rachel Holladay, and Alberto Rodriguez. *In-Hand Manipulation via Motion Cones*. In *RSS*, 2018.

Nikhil Chavan-Dafle, Rachel Holladay, and Alberto Rodriguez. *Planar in-hand manipulation via motion cones*. *IJRR*, 39(2-3):163–182, 2020.

Lipeng Chen, Luis F. C. Figueredo, and Mehmet Dogar. *Manipulation Planning Using Environmental Contacts to Keep Objects Stable under External Forces*. In *Humanoids*, pages 417–424. IEEE, 2019.

Lukás Chrpa, Nir Lipovetzky, and Sebastian Sardina. *Handling non-local dead-ends in Agent Planning Programs*. In *IJCAI*, pages 971–978, 2017.

Neil Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia Kavraki. *Incremental Task and Motion Planning: A Constraint-Based Approach*. In *RSS*, 2016.

Dawson-Haggerty et al. Trimesh. URL <https://trimesh.org/>.

Joris De Schutter and Hendrik Van Brussel. *Compliant robot motion I. A formalism for specifying compliant motion tasks*. *IJRR*, 7(4):3–17, 1988.

- Didier Devaurs, Thierry Siméon, and Juan Cortés. *Optimal Path Planning in Complex Cost Spaces With Sampling-Based Algorithms*. *TASE*, 13(2): 415–424, 2016.
- Mehmet Dogar and Siddhartha Srinivasa. *Push-grasping with dexterous hands: Mechanics and a method*. In *IROS*, pages 2123–2130. IEEE, 2010.
- Mehmet Dogar and Siddhartha Srinivasa. *A Framework for Push-grasping in Clutter*. In *RSS*, 2011.
- Michael Erdmann. *Multiple-point contact with friction: Computing forces and motions in configuration space*. In *IROS*, pages 163–170. IEEE, 1993.
- Michael Erdmann. *On a Representation of Friction in Configuration Space*. *IJRR*, 13(3):240–271, 1994.
- Michael Erdmann. *An Exploration of Nonprehensile Two-Palm Manipulation*. *IJRR*, 17(5):485–503, 1998.
- Lawrence Evans and Panagiotis Souganidis. *Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations*. *Indiana University mathematics journal*, 33(5):773–797, 1984.
- Jaime Fisac, Mo Chen, Claire Tomlin, and S Shankar Sastry. *Reach-avoid problems with time-varying dynamics, targets and constraints*. In *HSCC*, pages 11–20, 2015.
- Thierry Fraichard and Hajime Asama. *Inevitable collision states—a step towards safer robots?* *Advanced Robotics*, 18(10):1001–1024, 2004.
- Xiao Gao, Jie Ling, Xiaohui Xiao, and Miao Li. *Learning Force-Relevant Skills from Human Demonstration*. *Complexity*, 2019.
- Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. *Sample-Based Methods for Factored Task and Motion Planning*. In *RSS*, 2017.
- Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. *Integrated Task and Motion Planning*. *Annual review of control, robotics, and autonomous systems*, 2020a.
- Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. *PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning*. In *ICAPS*, volume 30, pages 440–448, 2020b.
- Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. *Online replanning in belief space for partially observable task and motion problems*. In *ICRA*, pages 5678–5684. IEEE, 2020c.

Suresh Goyal. *Planar sliding of a rigid body with dry friction: Limit surfaces and dynamics of motion*. PhD dissertation, Department of Mechanical Engineering, Cornell University, 1989.

Suresh Goyal, Andy Ruina, and Jim Papadopoulos. *Planar sliding with dry friction part 1. limit surface and moment function*. *Wear*, 143(2):307–330, 1991.

Fabien Gravot, Stephane Cambon, and Rachid Alami. *aSyMov: a planner that deals with intricate symbolic and geometric problems*. In *ISRR*, pages 100–110. Springer, 2005.

Dylan Hadfield-Menell, Edward Groshev, Rohan Chitnis, and Pieter Abbeel. *Modular task and motion planning in belief space*. In *IROS*, pages 4991–4998. IEEE, 2015.

Kris Hauser and Jean-Claude Latombe. *Multi-modal motion planning in non-expansive spaces*. *IJRR*, 29(7):897–915, 2010.

Kris Hauser and Victor Ng-Thow-Hing. *Randomized multi-modal motion planning for a humanoid robot manipulation task*. *IJRR*, 30(6):678–698, 2011.

Eric Heiden, Miles Macklin, Yashraj Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. *DiSECT: A Differentiable Simulation Engine for Autonomous Robotic Cutting*. *RSS*, 2021.

Francois Hogan and Alberto Rodriguez. *Feedback control of the pusher-slider system: a story of hybrid and underactuated contact dynamics*. In *WAFR*, 2016.

Neville Hogan. *Impedance Control: An Approach to Manipulation: Part I - Theory*. *Journal of Dynamic Systems, Measurement, and Control*, 107(1): 1–7, 1985.

Anne Holladay, Robert Paolini, and Matthew Mason. *A general framework for open-loop pivoting*. In *ICRA*, pages 3675–3681. IEEE, 2015.

Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. *Force-and-Motion Constrained Planning for Tool Use*. In *IROS*, pages 7409–7416. IEEE, 2019.

Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. *Planning for Multi-stage Forceful Manipulation*. In *ICRA*, pages 6556–6562. IEEE, 2021.

Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. *Robust planning for multi-stage forceful manipulation*. *IJRR*, 43(3):330–353, 2023.

Soo Hong Lee and Mark Cutkosky. *Fixture Planning With Friction*. *Journal of Manufacturing Science and Engineering*, 113(3):320–327, 1991.

- Yifan Hou. **Robust Planar Dynamic Pivoting by Regulating Inertial and Grip Forces**. Master's thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2017.
- Yifan Hou and Matthew Mason. **Robust execution of contact-rich motion plans by hybrid force-velocity control**. In *ICRA*, pages 1933–1939. IEEE, 2019.
- Yifan Hou, Zhenzong Jia, and Matthew Mason. **Manipulation with Shared Grasping**. In *RSS*, 2020.
- Robert Howe and Mark Cutkosky. **Practical Force-Motion Models for Sliding Manipulation**. *IJRR*, 15(6):557–572, 1996.
- Léonard Jaillet, Juan Cortés, and Thierry Siméon. **Sampling-Based Path Planning on Configuration-Space Costmaps**. *Transactions on Robotics*, 26(4):635–646, 2010.
- Prajwal Jamdagni and Yan-Bin Jia. **Robotic cutting of solids based on fracture mechanics and FEM**. In *IROS*, pages 8252–8257. IEEE, 2019.
- Prajwal Jamdagni and Yan-Bin Jia. **Robotic Slicing of Fruits and Vegetables: Modeling the Effects of Fracture Toughness and Knife Geometry**. In *ICRA*, pages 6607–6613. IEEE, 2021.
- Leslie Pack Kaelbling and Tomás Lozano-Pérez. **Hierarchical task and motion planning in the now**. In *ICRA*, pages 1470–1477. IEEE, 2011.
- Leslie Pack Kaelbling and Tomás Lozano-Pérez. **Integrated task and motion planning in belief space**. *IJRR*, 32(9-10):1194–1227, 2013.
- Sertac Karaman and Emilio Frazzoli. **Sampling-based Algorithms for Optimal Motion Planning**. *IJRR*, 30(7):846–894, 2011.
- Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. **Probabilistic roadmaps for path planning in high-dimensional configuration spaces**. *Transactions on Robotics and Automation*, 12(4):566–580, 1996.
- Jennifer King, Matthew Klingensmith, Christopher Dellin, Mehmet Dogar, Prasanna Velagapudi, Nancy Pollard, and Siddhartha Srinivasa. **Pregrasp Manipulation as Trajectory Optimization**. In *RSS*, 2013.
- Sven Koenig and Yaxin Liu. **The interaction of representations and planning objectives for decision-theoretic planning tasks**. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(4):303–326, 2002.
- Roman Kolbert, Nikhil Chavan-Dafle, and Alberto Rodriguez. **Experimental Validation of Contact Dynamics for In-Hand Manipulation**. In *ISER*, pages 633–645. Springer, 2017.

Andrey Kolobov, Mausam, and Daniel Weld. [SixthSense: Fast and reliable recognition of dead ends in MDPs](#). In *AAAI*, volume 24, pages 1108–1114, 2010.

Andrey Kolobov, Mausam, Daniel Weld, and Hector Geffner. [Heuristic search for generalized stochastic shortest path MDPs](#). In *ICAPS*, volume 21, pages 130–137, 2011.

Andrey Kolobov, Mausam, and Daniel Weld. [Discovering hidden structure in factored MDPs](#). *Artificial Intelligence*, 189:19–47, 2012a.

Andrey Kolobov, Mausam, and Daniel Weld. [A theory of goal-oriented MDPs with dead ends](#). In *UAI*, 2012b.

George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. [From skills to symbols: Learning symbolic representations for abstract high-level planning](#). *JAIR*, 61:215–289, 2018.

Ingo Kresse and Michael Beetz. [Movement aware Action Control - Integrating Symbolic and Control-theoretic Action Execution](#). In *ICRA*, pages 3245–3251. IEEE, 2012.

Vikash Kumar, Yuval Tassa, Tom Erez, and Emanuel Todorov. [Real-time behaviour synthesis for dynamic hand-manipulation](#). In *ICRA*, pages 6808–6815. IEEE, 2014.

Steven LaValle and James Kuffner. [Randomized kinodynamic planning](#). *IJRR*, 20(5):378–400, 2001a.

Steven LaValle and James Kuffner. [Rapidly-Exploring Random Trees: Progress and Prospects](#). *Algorithmic and Computational Robotics*, pages 303–307, 2001b.

Sukhan Lee and Fu-Chung Wang. [Physical reasoning of interconnection forces for efficient assembly planning](#). In *ICRA*, pages 307–313. IEEE, 1993.

Gal Levi, Yoav Golan, and Amir Shapiro. [SIMJig-Smart Independent Minimalist Jig](#). *Robotics and Automation Letters*, 7(2):3396–3403, 2022.

Martin Levihn and Mike Stilman. [Using Environment Objects as Tools: Unconventional Door Opening](#). In *IROS*, pages 2502–2508. IEEE, 2014.

Jacky Liang, Mohit Sharma, Alex LaGrassa, Shivam Vats, Saumya Saxena, and Oliver Kroemer. [Search-based task planning with learned skill effect models for lifelong robotic manipulation](#). In *ICRA*, pages 6351–6357. IEEE, 2022.

Nir Lipovetzky, Christian Muise, and Hector Geffner. [Traps, invariants, and dead-ends](#). In *ICAPS*, volume 26, pages 211–215, 2016.

- Iain Little and Sylvie Thiebaux. *Probabilistic planning vs. replanning*. In *ICAPS Workshop on IPC: Past, Present and Future*, pages 1–10, 2007.
- Philip Long, Wisama Khalil, and Philippe Martinet. *Modeling & control of a meat-cutting robotic cell*. In *ICAR*, pages 1–6. IEEE, 2013.
- Philip Long, Wisama Khalil, and Philippe Martinet. *Force/vision control for robotic cutting of soft materials*. In *IROS*, pages 4716–4721. IEEE, 2014.
- Kevin Lynch and Matthew Mason. *Stable Pushing: Mechanics, Controllability, and Planning*. *IJRR*, 15(6):533–556, 1996.
- Kevin Lynch and Frank Park. *Modern Robotics*. Cambridge University Press, 2017.
- Kevin Lynch, Hitoshi Maekawa, and Kazuo Tanie. *Manipulation And Active Sensing By Pushing Using Tactile Feedback*. In *IROS*, pages 416–421. IEEE, 1992.
- Simon Manschitz, Michael Gienger, Jens Kober, and Jan Peters. *Learning Sequential Force Interaction Skills*. *Robotics*, 9(2):45, 2020.
- Matthew Mason. *Compliance and force control for computer controlled manipulators*. *Transactions on Systems, Man, and Cybernetics*, 11(6): 418–432, 1981.
- Matthew Mason. *Mechanics and Planning of Manipulator Pushing Operations*. *IJRR*, 5(3):53–71, 1986.
- Matthew Mason. *Mechanics of robotic manipulation*. MIT press, 2001.
- Matthew Mason. *Toward robotic manipulation*. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:1–28, 2018.
- Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL-the planning domain definition language*, 1998.
- Paul Michelman and Peter Allen. *Forming complex dextrous manipulations from task primitives*. In *ICRA*, pages 3383–3388. IEEE, 1994.
- Ioanna Mitsioni, Yiannis Karayiannidis, Johannes Stork, and Danica Kragic. *Data-Driven Model Predictive Control for the Contact-Rich Task of Food Cutting*. In *Humanoids*, pages 244–250. IEEE, 2019.
- Michael Morak, Lukas Chrupa, Wolfgang Faber, and Daniel Fišer. *On the reversibility of actions in planning*. In *KR*, volume 17, pages 652–661, 2020.
- Marco Morales, Roger Pearce, and Nancy Amato. *Analysis of the evolution of C-Space models built through incremental exploration*. In *ICRA*, pages 1029–1034. IEEE, 2007.

Xiaoqian Mu and Yan-Bin Jia. [Physical Property Estimation and Knife Trajectory Optimization During Robotic Cutting](#). In *ICRA*, pages 2700–2706. IEEE, 2022.

Xiaoqian Mu, Yuechuan Xue, and Yan-Bin Jia. [Robotic cutting: Mechanics and control of knife motion](#). In *ICRA*, pages 3066–3072. IEEE, 2019.

Nils Nilsson. [Shakey the robot](#). 1984.

Christian Ott. [Cartesian impedance control: The rigid body case](#). In *Cartesian Impedance Control of Redundant and Flexible-Joint Robots*, pages 29–44. Springer, 2008.

Stéphane Petti and Thierry Fraichard. [Safe motion planning in dynamic environments](#). In *IROS*, pages 2210–2215. IEEE, 2005.

Erion Plaku and Gregory Hager. [Sampling-based motion and symbolic action planning with geometric and differential constraints](#). In *ICRA*, pages 5002–5008. IEEE, 2010.

Michael Posa, Cecilia Cantu, and Russ Tedrake. [A Direct Method for Trajectory Optimization of Rigid Bodies Through Contact](#). *IJRR*, 33(1): 69–81, 2014.

Sahand Rezaei-Shoshtari, David Meger, and Inna Sharf. [Learning the Latent Space of Robot Dynamics for Cutting Interaction Inference](#). In *IROS*, pages 5627–5632. IEEE, 2020.

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

Jian Shi, J. Zachary Woodruff, and Kevin Lynch. [Dynamic in-hand sliding manipulation](#). In *IROS*, pages 870–877. IEEE, 2015.

Jian Shi, J Zachary Woodruff, Paul Umbanhowar, and Kevin Lynch. [Dynamic in-hand sliding manipulation](#). *Transactions on Robotics*, 33(4):778–795, 2017.

Alexander Shkolnik and Russ Tedrake. [Sample-Based Planning with Volumes in Configuration Space](#). In *arXiv*, 2011.

Alexander Shkolnik, Matthew Walter, and Russ Tedrake. [Reachability-guided sampling for planning under differential constraints](#). In *ICRA*, pages 2859–2865. IEEE, 2009.

Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. [Learning symbolic operators for task and motion planning](#). In *IROS*, pages 3182–3189. IEEE, 2021.

- Avishai Sintov and Amir Shapiro. [Swing-up regrasping algorithm using energy control](#). In *ICRA*, pages 4888–4893. IEEE, 2016.
- Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. [Combined task and motion planning through an extensible planner-independent interface layer](#). In *ICRA*, pages 639–646. IEEE, 2014.
- Simon Ståhlberg, Guillem Frances, and Jendrik Seipp. [Learning Generalized Unsolvability Heuristics for Classical Planning](#). In *IJCAI*, pages 4175–4181, 2021.
- Marcel Steinmetz and Jörg Hoffmann. [Towards clause-learning state space search: Learning to recognize dead-ends](#). In *AAAI*, volume 30, 2016.
- Marcel Steinmetz and Jörg Hoffmann. [Search and Learn: On Dead-End Detectors, the Traps they Set, and Trap Learning](#). In *IJCAI*, pages 4398–4404, 2017.
- David Stewart and J. C. Trinkle. [An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic Collisions and Coulomb Friction](#). *International Journal for Numerical Methods in Engineering*, 39(15): 2673–2691, 1996.
- Jörg Stückler, Max Schwarz, and Sven Behnke. [Mobile manipulation, tool use, and intuitive interaction for cognitive service robot cosero](#). *Frontiers in Robotics and AI*, 3:58, 2016.
- Balakumar Sundaralingam and Tucker Hermans. [Relaxed-Rigidity Constraints: In-Grasp Manipulation using Purely Kinematic Trajectory Optimization](#). In *RSS*, 2017.
- Robert Tarjan. [Depth-first search and linear graph algorithms](#). *SIAM journal on computing*, 1(2):146–160, 1972.
- Florent Teichteil-Königsbuch. [Stochastic safest and shortest path problems](#). In *AAAI*, volume 26, pages 1825–1831, 2012.
- Rodney Teo and Claire Tomlin. [Computing danger zones for provably safe closely spaced parallel approaches](#). *Journal of Guidance, Control, and Dynamics*, 26(3):434–442, 2003.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. [MuJoCo: A physics engine for model-based control](#). In *IROS*, pages 5026–5033. IEEE, 2012.
- Marc Toussaint. [Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning](#). In *IJCAI*, pages 1930–1936, 2015.

Marc Toussaint, Jung-Su Ha, and Danny Driess. *Describing Physics For Physical Reasoning: Force-based Sequential Manipulation Planning*. In *IROS*. IEEE, 2020.

J. C. Trinkle, J.S. Pang, S. Sudarsky, and G. Lo. *On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction*. *ZAMM - Journal of Applied Math. and Mechanics*, 77(4):267–279, 1997.

Francisco Viña B, Yiannis Karayiannidis, Christian Smith, and Danica Kragic. *Adaptive control for pivoting with visual and tactile feedback*. In *ICRA*, pages 399–406. IEEE, 2016.

Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. *Learning compositional models of robot skills for task and motion planning*. *IJRR*, 40(6-7):866–894, 2021.

Yoshiaki Watanabe, Kotaro Nagahama, Kimitoshi Yamazaki, Kei Okada, and Masayuki Inaba. *Cooking Behavior with Handling General Cooking Tools based on a System Integration for a Life-sized Humanoid Robot*. *Journal of Behavioral Robotics Paladyn*, 4(2):63–72, 2013.

P.-B. Wieber. *Holonomy and Nonholonomy in the Dynamics of Articulated Motion*, pages 411–425. Springer, 2006.

Jingyi Xu, Michael Danielczuk, Eckehard Steinbach, and Ken Goldberg. *6DFC: Efficiently Planning Soft Non-Planar Area Contact Grasps using 6D Friction Cones*. In *ICRA*, pages 7891–7897. IEEE, 2020.

Nicholas Xydias and Imin Kao. *Modeling of contact mechanics and friction limit surfaces for soft fingers in robotics, with experimental results*. *IJRR*, 18(9):941–950, 1999.

Sung Wook Yoon, Alan Fern, and Robert Givan. *FF-Replan: A Baseline for Probabilistic Planning*. In *ICAPS*, volume 7, pages 352–359, 2007.

Ganwen Zeng and Ahmad Hemami. *An adaptive control strategy for robotic cutting*. In *ICRA*, volume 1, pages 22–27. IEEE, 1997a.

Ganwen Zeng and Ahmad Hemami. *An overview of robot force control*. *Robotica*, 15(5):473–482, 1997b.

Kevin Zhang, Mohit Sharma, Manuela Veloso, and Oliver Kroemer. *Leveraging Multimodal Haptic Sensory Data for Robust Cutting*. In *Humanoids*, pages 409–416. IEEE, 2019.

Jiaji Zhou and Matthew Mason. *Pushing revisited: Differential flatness, trajectory planning and stabilization*. In *ISRR*, 2017.

Zhengyuan Zhou, Ryo Takei, Haomiao Huang, and Claire Tomlin. *A general, open-loop formulation for reach-avoid games*. In *CDC*, pages 6501–6506. IEEE, 2012.

Appendix

This appendix provides supplementary material to Chapter 3, describing additional implementation details and experimental domains.

A Implementation Details

In the context of Chapter 3, we provide some implementation details with respect to the controller used to exert wrenches, the specific motion planning and grasping tools used in the kinematic samplers and the parameters of the robust planning experiments.

A.1 Controller for Exerting Wrenches

Each of the operators that apply the forceful operation, e.g. `hand_twist` and `tool_twist` in the nut twisting domain, are associated with a controller that must exert a wrench. While there are several control methods for exerting wrenches explicitly, such as force control (Zeng and Hemami, 1997b) or hybrid position-force control (Mason, 1981; De Schutter and Van Brussel, 1988; Hou and Mason, 2019), we opt to use a Cartesian impedance controller. Cartesian impedance control regulates the relationship between force and motion by treating the interplay of interaction forces and motion derivatives as a mass-spring-damper system (Hogan, 1985; Albu-Schaffer et al., 2003; Ott, 2008).

We can exert a wrench by offsetting the target Cartesian pose to be below the point of contact and adjusting the impedance parameters (Kresse and Beetz, 2012). Intuitively this exerts a wrench by compressing the spring rendered by the robot. We chose to only vary the stiffness matrix, K_p , and set the damping matrix, K_d , to be approximately critically damped, i.e. $K_d = 2\sqrt{K_p}$. As shown in Fig.1 we can experimental characterize the relationship between the exerted wrench, the pose offset and the stiffness. The planner uses this experimental relation to select, given the stiffest possible setting, the desired pose offset. Thus, each of the samplers that generate Cartesian impedance paths must generate the series of setpoints in $SE(3)$, accounting for the pose offset, and the stiffnesses.

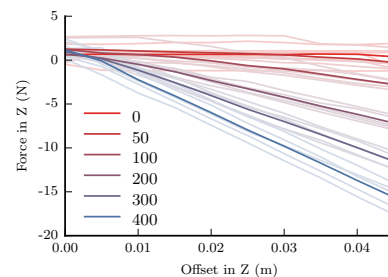


Figure 1: Across varying stiffnesses with Cartesian impedance control, we plot the experimental relation between the offset in the commanded offset in the z direction and the exerted force in z, as measured by an external force-torque sensor. For each stiffness we plot all five experimental runs, bolding the average. The result shows that the relation between the offset and force exerted is nearly linear.

A.2 Motion Planning and Grasping

Collision-free motion planning, such as what is used in samplers `plan-free-motion` and `plan-holding-motion`, can be implemented with any choice of motion planner. We use a python implementation of bidirectional RRT (BiRRT) (LaValle and Kuffner, 2001b). In these settings, a considerable fraction of the overall PDDLStream planning time is spent performing motion planning. A more efficient implementation or algorithm would significantly speed up the reported results.

Any grasp generation method could be used to generate grasps for samplers such as `sample-grasp`. For each graspable object we define the grasp set using Task Space Regions (TSRs), which are a compact method for describing pose constraints that allows for random sampling (Berenson et al., 2011).

A.3 Robust Planning Experimental Parameters

As detailed in Sec. 3.7.2, to generate robust plans we assess the probability of a forceful kinematic chain being stable via Monte Carlo estimation. Here we briefly describe how each parameter was perturbed for the experiments in Sec. 3.8.2.

The friction coefficient is additively perturbed by a value uniformly sampled between $[-0.1, 0.1]$. The applied wrench is multiplicatively perturbed by a value uniformly sampled between $[0.5, 1.5]$. The contact frame is additively perturbed by a value uniformly sampled between $[-5mm, 5mm]$ if the frictional joint is formed by a robot grasp and $[-10mm, 10mm]$ otherwise. The contact patch size (r , in the context of the ellipsoidal limit surface) is recomputed.

B Further Domain Specification

In Sec. 3.7 we detailed the lifted operators, fact types and samplers used for the nut-twisting domain. The domain was composed of elements from the pick-and-place domain, the forceful manipulation additions and domain-specific additions. In this section we first detail more forceful manipulation additions before providing the domain-specific elements for the childproof bottle and vegetable cutting domains.

B.1 Forceful Manipulation Additions

In Sec. 3.7.1 and Sec. 3.7.1 we detailed the augmentations needed to enable forceful manipulation in the PDDLStream framework. In that example we discussed how the forceful kinematic chain constraint was integrated into the domain and discussed two possible fixturing methods: (`HoldingFixtured o w`), which uses a second robot to grasp the object

to be fixtured, and $(\text{WeightFixtured } o \ w)$, which fixtures by weighing down the object with a heavy mass.

Table 1 lists the lifted operators, derived facts and samplers that are used to enable the other fixturing strategies that can be used across forceful manipulation domains: $(\text{ViseFixtured } o \ w)$ and $(\text{SurfaceFixtured } o \ w)$

One way to fixture an object is to place it in a vise: $(\text{ViseFixtured } o \ w)$. This fact is verified by the test sampler `test-vise-grasp-stable`, which uses the limit surface models to evaluate the stability of the grasp. To use a vise we add the lifted operators `open_vise` and `close_vise` along with their sampler `plan-vise-motion`. These operators actuate the vise (implemented here as a table-mounted robotic hand) and have as a precondition that the object to be fixtured is already in the vise, which can be achieved through a pick-and-place motion.

Another way to fixture is to create a frictional grasp by exerting a downward force on the object through a contact, thus sandwiching the object between the contact and the surface $(\text{SurfaceFixtured } o \ w)$. The exerted downward force can be thought of as the grasping force of the frictional grasp. This creates two joints: the joint between the force-exerting contact and the object (evaluated by `test-contact-stable`) and the joint between the object and the surface (evaluated by `test-surface-stable`). For both joints we use the limit surface models to evaluate stability.

We also introduce operators to make and break contact with either a grasped tool or an end effector contact of the robot (fingers or palm): `pushin_tool`, `pushout_tool`, `pushin_rcontact`, `pushout_rcontact` (with samplers `plan-tool-contact` and `plan-rcontact-contact`). The operators that make contact (`pushin_tool` and `pushin_rcontact`) make contact by exert a downward force on the object, leveraging Cartesian Impedance control, described in Sec. A.1.

In each domain we define a derived fact $(\text{Fixtured } o \ w)$ to define which fixturing methods are valid to use.

B.2 Childproof Bottle Domain

Table 2 lists the lifted operators, derived facts and samplers that are specific to the childproof bottle domain. The robot can impart the forceful operation to push-twist the cap through a variety of possible contacts: a grasp, fingertips, a palm or a grasped pusher tool. Correspondingly, we have a lifted operator for each of these contact types: `grasp_twist`, `contact_twist`, `tool_twist`. The operator `contact_twist` is parameterized by what end effector contact the robot is using, which we define as being either the fingertips or a palm.

The controller for the `grasp_twist` operator grasps the cap with the hand, forcefully pushes down while twisting the cap with the hand and then

releases the cap. The controller for the `contact_twist` operator performs the same sequence only instead of grasping the cap, frictional planar contact is made. Likewise the controller for the `tool_twist` operator is the same sequence, but where a grasped tool is making and breaking contact with the cap. The trajectories are generated by the samplers `plan-grasp-twist`, `plan-contact-twist` and `plan-tool-twist` respectively. Unsurprisingly, the implementation of the three samplers has a tremendous amount of overlap with small, parameterized differences. We use a Cartesian Impedance controller, detailed in Sec. A.1 to exert the forceful push and twist wrench for all three operators.

While the task requires that the robot exert a wrench $w = (0, 0, -f_z, 0, 0, t_z)$, we allow `contact_twist` and `tool_twist` to sample a wrench w_e that exerts additional downward force, i.e. $w_e = (0, 0, -f_{z++}, 0, 0, t_z)$ where f_{z++} is a sampled value such that $f_{z++} > f_z$. If the bottle is being fixtured via `(SurfaceFixtured o w)` this additional force increase the stability of the frictional contacts.

Like in the nut twisting domain, any operator that applies a forceful operation has preconditions that check the forceful kinematic chain constraint. Note that for the operators where the robot is exerting w_e the stability is evaluated with respect to this wrench.

For each of these operators, the bottle must be fixtured, which can either be achieved by using a second arm to grasp the bottle (`(HoldingFixtured o w)`), using the frictional contact with the surface (`(SurfaceFixtured o w)`) or using a vise (`(ViseFixtured o w)`). Likewise if the operator is exerting w_e the fixturing is evaluated with respect to this. In this domain we slightly modify the formulation of `(SurfaceFixtured o w)` to account for the fact that the additional force is made in conjunction with the push-twist operations.

We additionally add a lifted operator that removes the cap from the bottle, `remove_cap`, which is only feasible after we have used one of the push-twist operators. The controller for this operator grasps the cap with the hand and lifts the cap up.

B.3 Vegetable Cutting Domain

Table 3 lists the lifted operator, derived fact and sampler that are specific to the vegetable cutting domain. The robot imparts the forceful operation to cut the vegetable by using a grasped knife.

We define one new operator: `slice_cut`. The controller for this operator makes contact with the vegetable via the grasped knife, exerts a downward force with the knife and then exerts a translational slice with the knife. These trajectories are generated by sampler `plan-slice-motion` and, like in the other domains, we use Cartesian Impedance control to exert the wrenches.

Since this operator involves two wrenches, the wrench of the downward

force w_0 and the wrench of the translation slice w_1 , the forceful kinematic chains must be stable with respect to both wrenches. Hence the preconditions evaluate the stability of the vegetable's fixturing and the stability of the grasp on the knife for w_0 and w_1 .

In this domain the vegetable must be fixtured, which can be achieved either by using a second arm to grasp the vegetable (`HoldingFixtured o w`), using the frictional contact with the surface (`SurfaceFixtured o w`) or using a vise (`ViseFixtured o w`).

Action	Preconditions	Effects
open_vise	$(\text{ViseHand } v) \wedge (\neg (\text{Movable } o)) \wedge (\text{AtPose } o p_o) \wedge$ $(\text{AtGrasp } v o g_o) \wedge (\text{ViseMotion } v o p_o g_o w t)$	$(\text{Movable } o)$ $(\neg (\text{AtGrasp } v o g_o))$
close_vise	$(\text{ViseHand } v) \wedge (\text{Movable } o) \wedge (\text{AtPose } o p_o) \wedge$ $(\text{On } o v) \wedge (\text{ViseMotion } v o p_o g_o w t)$	$(\text{AtGrasp } v o g_o)$ $(\neg (\text{Movable } o))$
pushin_tool	$(\text{AtConf } a q_0) \wedge (\text{AtPose } o p_o) \wedge (\text{Pusher } o_p) \wedge$ $(\text{AtGrasp } a o_p g_p) \wedge (\text{StableGrasp } o_p g_p w) \wedge$ $(\text{SampleWrench } w o) \wedge$ $(\text{ContactToolMotion } a o_p g_p o p_o p_{oc} w q_0 q_1 t) \wedge$ $(\neg (\text{TrajUnsafe } a t))$	$(\text{InContact } o o_p c p_{oc} w)$ $(\neg (\text{AtConf } a q_0))$ $(\text{AtConf } a q_1)$
pushout_tool	$(\text{AtConf } a q_0) \wedge (\text{AtPose } o p_o) \wedge (\text{Pusher } o_p) \wedge$ $(\text{AtGrasp } a o_p g_p) \wedge (\text{InContact } o o_p c p_{oc} w) \wedge$ $(\text{ContactToolMotion } a o_p g_p o p_o p_{oc} w q_0 q_1 t) \wedge$ $(\neg (\text{TrajUnsafe } a t))$	$(\neg (\text{InContact } o o_p c p_{oc} w))$ $(\neg (\text{AtConf } a q_0))$ $(\text{AtConf } a q_1)$
pushin_rcontact	$(\text{AtConf } a q_0) \wedge (\text{AtPose } o p_o) \wedge$ $(\text{RContact } c) \wedge (\text{SampleWrench } w o) \wedge$ $(\text{ContactMotion } a c o p_o p_{oc} w q_0 q_1 t) \wedge$ $(\neg (\text{TrajUnsafe } a t))$	$(\text{InContact } o o_p c p_{oc} w)$ $(\neg (\text{AtConf } a q_0))$ $(\text{AtConf } a q_1)$
pushout_rcontact	$(\text{AtConf } a q_0) \wedge (\text{AtPose } o p_o) \wedge$ $(\text{RContact } c) \wedge (\text{InContact } o o_p c p_{oc} w) \wedge$ $(\text{ContactMotion } a c o p_o p_{oc} w q_0 q_1 t) \wedge$ $(\neg (\text{TrajUnsafe } a t)) \wedge$	$(\neg (\text{InContact } o o_p c p_{oc} w))$ $(\neg (\text{AtConf } a q_0))$ $(\text{AtConf } a q_1)$

(a) Actions

Derived Fact	Definition
$(\text{ViseFixtured } o w)$	$\exists v g_o (\text{ViseHand } v) \wedge (\text{AtGrasp } v o g_o) \wedge (\text{StableViseGrasp } o g_o w)$
$(\text{SurfaceFixtured } o w)$	$\exists p_o c p_{oc} r (\text{AtPose } o p_o) \wedge (\text{On } o r) \wedge (\text{InContact } o c p_{oc} w_e) \wedge$ $(\text{StableObjContact } c p_{oc} o w_e w) \wedge (\text{StableSurfaceContact } o p_o r w_e w)$

(b) Derived Facts

Sampler	Inputs	Outputs	Certified Facts
sample-push-wrench	o	w	$(\text{SampleWrench } w o) \wedge (\text{Wrench } w)$
plan-vise-motion	$v o p_o g_o w$	t	$(\text{ViseMotion } v o p_o g_o w t) \wedge (\text{Traj } t)$
plan-tool-contact	$a o_p g_p o p_o p_{oc} w$	$q_0 q_1 t$	$(\text{ContactToolMotion } a o_p g_p o p_o p_{oc} w q_0 q_1 t) \wedge$ $(\text{Conf } q_0) (\text{Conf } q_1) (\text{Traj } t)$
plan-rcontact-contact	$a c o p_o p_{oc} w$	$q_0 q_1 t$	$(\text{ContactMotion } a c o p_o p_{oc} w q_0 q_1 t) \wedge$ $(\text{Conf } q_0) \wedge (\text{Conf } q_1) \wedge (\text{Traj } t)$
test-vise-grasp-stable	$v o g_o w$		$(\text{StableViseGrasp } o g w)$
test-contact-stable	$c p_{oc} o w_e w$		$(\text{StableObjContact } c p_{oc} o w_e w)$
test-surface-stable	$o p_o r w_e w$		$(\text{StableSurfaceContact } o p_o r w_e w)$

(c) Samplers

Table 1: The lifted operators, derived facts and samplers common across forceful manipulation domains. Throughout the table we use the symbols: a is a robot arm, c is a robot contact (fingertips or palm), o is an object, p_o is a pose of object o , g_o is a grasp on object o , p_{ij} is a relative pose between two objects i and j , q_i is a configuration, r is a region, t is a trajectory, w is a wrench, v is a vise.

Action	Preconditions	Effects
grasp_twist	$(\text{AtConf } a \ q_0) \wedge (\text{HandEmpty } a) \wedge (\text{Cap } o_c) \wedge (\text{Bottle } o_b) \wedge$ $(\text{AtPose } o_c \ p_c) \wedge (\text{AtPose } o_b \ p_b) \wedge (\text{Fixtured } o_b \ w) \wedge$ $(\text{StableGrasp } o_c \ g_c \ w) \wedge (\text{StableJoints } a \ t \ w) \wedge$ $(\text{GraspPushMotion } a \ o_c \ o_b \ p_c \ p_b \ g_c \ w \ q_0 \ q_1 \ t) \wedge$ $(\neg (\text{TrajUnsafe } a \ t))$	$(\text{Twisted } o_c \ w) \wedge$ $(\neg (\text{AtConf } a \ q_0)) \wedge$ $(\text{AtConf } a \ q_1)$
contact_twist	$(\text{AtConf } a \ q_0) \wedge (\text{HandEmpty } a) \wedge (\text{Cap } o_c) \wedge (\text{Bottle } o_b) \wedge (\text{RContact } n) \wedge$ $(\text{AtPose } o_c \ p_c) \wedge (\text{AtPose } o_b \ p_b) \wedge (\text{ExtraWrench } w \ w_e) \wedge$ $(\text{Fixtured } o_b \ w_e) \wedge (\text{StableJoints } a \ t \ w) \wedge$ $(\text{ContactPushMotion } a \ n \ o_c \ o_b \ p_c \ p_b \ w_e \ q_0 \ q_1 \ t) \wedge$ $(\neg (\text{TrajUnsafe } a \ t))$	$(\text{Twisted } o_c \ w) \wedge$ $(\neg (\text{AtConf } a \ q_0)) \wedge$ $(\text{AtConf } a \ q_1)$
tool_twist	$(\text{AtConf } a \ q_0) \wedge (\text{Cap } o_c) \wedge (\text{Bottle } o_b) \wedge (\text{PushTool } o_t) \wedge$ $(\text{AtPose } o_c \ p_c) \wedge (\text{AtPose } o_b \ p_b) \wedge (\text{AtGrasp } a \ o_t \ g_t) \wedge$ $(\text{ExtraWrench } w \ w_e) \wedge (\text{Fixtured } o_b \ w_e) \wedge$ $(\text{StableGrasp } o_t \ g_t \ w_e) \wedge (\text{StableJoints } a \ t \ w) \wedge$ $(\text{ToolPushMotion } a \ o_t \ o_c \ o_b \ p_c \ p_b \ g_t \ w_e \ q_0 \ q_1 \ t) \wedge$ $(\neg (\text{TrajUnsafe } a \ t))$	$(\text{Twisted } o_c \ w) \wedge$ $(\neg (\text{AtConf } a \ q_0)) \wedge$ $(\text{AtConf } a \ q_1)$
remove_cap	$(\text{AtConf } a \ q) \wedge (\text{HandEmpty } a) \wedge (\text{Cap } o_c) \wedge (\text{Bottle } o_b) \wedge$ $(\text{AtPose } o_b \ p_b) \wedge (\text{AtPose } o_c \ p_c) \wedge (\text{Twisted } o_c \ w) \wedge$ $(\text{UncapMotion } a \ o_c \ o_b \ p_b \ g \ q \ t) \wedge$ $(\neg (\text{TrajUnsafe } a \ t))$	$(\text{AtGrasp } a \ o_c \ g) \wedge$ $(\neg (\text{AtPose } o_c \ p_c)) \wedge$ $(\neg (\text{HandEmpty } a)) \wedge$ $(\text{Uncapped } o_c \ o_b)$

(a) Actions

Derived Fact	Definition
$(\text{Fixtured } o \ w)$	$((\text{HoldingFixtured } o \ w) \vee (\text{SurfaceFixtured } o \ w) \vee (\text{ViseFixtured } o \ w))$

(b) Derived Facts

Sampler	Inputs	Outputs	Certified Facts
sample-wrench	w_1	w_2	$(\text{ExtraWrench } w_1 \ w_2) \wedge (\text{Wrench } w_2)$
plan-uncap-motion	$a \ o_c \ o_b \ p_b \ g_c$	$q \ t$	$(\text{UncapMotion } a \ o_c \ o_b \ p_b \ g_c \ q \ t) \wedge (\text{Conf } q) \wedge (\text{Traj } t)$
plan-grasp-twist	$a \ o_c \ o_b \ p_c \ p_b \ g_c \ w$	$q_0 \ q_1 \ t$	$(\text{GraspPushMotion } a \ o_c \ o_b \ p_c \ p_b \ g_c \ w \ q_0 \ q_1 \ t) \wedge$ $(\text{Conf } q_0) \wedge (\text{Conf } q_1) \wedge (\text{Traj } t)$
plan-contact-twist	$a \ n \ o_c \ o_b \ p_c \ p_b \ w$	$q_0 \ q_1 \ t$	$(\text{ContactPushMotion } a \ n \ o_c \ o_b \ p_c \ p_b \ w \ q_0 \ q_1 \ t) \wedge$ $(\text{Conf } q_0) \wedge (\text{Conf } q_1) \wedge (\text{Traj } t)$
plan-tool-twist	$a \ o_t \ o_c \ o_b \ g_t \ p_c \ p_b \ w$	$q_0 \ q_1 \ t$	$(\text{ToolPushMotion } a \ o_t \ o_c \ o_b \ p_c \ p_b \ g_t \ w \ q_0 \ q_1 \ t) \wedge$ $(\text{Conf } q_0) \wedge (\text{Conf } q_1) \wedge (\text{Traj } t)$

(c) Samplers

Table 2: The domain-specific lifted operators, derived facts and samplers for the childproof bottle domain. Throughout the table we use the symbols: a is a robot arm, o is an object, p_o is a pose of object o , g_o is a grasp on object o , q_i is a configuration, r is a region, t is a trajectory, w is a wrench. Specific to this domain: n , o_b , o_c and o_t refer to the robot contact, bottle, cap and pusher tool, respectively.

Action	Preconditions	Effects	
slice_cut	$(\text{AtConf } a \ q_0) \wedge (\text{Cutttable } o_v) \wedge (\text{AtPose } o_v \ p_v) \wedge (\text{Knife } o_k) \wedge$ $(\text{AtGrasp } a \ o_k \ g_k) \wedge (\text{Fixtured } o_v \ w_0) \wedge (\text{Fixtured } o_v \ w_1) \wedge$ $(\text{StableGrasp } o_k \ g_k \ w_0) \wedge (\text{StableGrasp } o_k \ g_k \ w_1) \wedge$ $(\text{StableJoints } a \ t \ w_0) \wedge (\text{StableJoints } a \ t \ w_1) \wedge$ $(\text{SliceCutMotion } a \ o_k \ o_v \ g_k \ p_c \ w_1 \ w_2 \ q_0 \ q_1 \ t) \wedge$ $(\neg (\text{TrajUnsafe } a \ t))$	$(\text{Sliced } o_v) \wedge$ $(\neg (\text{AtConf } a \ q_0)) \wedge$ $(\text{AtConf } a \ q_1)$	
(a) Actions			
Derived Fact	Definition		
	$(\text{Fixtured } o \ w) \ ((\text{HoldingFixtured } o \ w) \vee (\text{SurfaceFixtured } o \ w) \vee (\text{ViseFixtured } o \ w))$		
(b) Derived Facts			
Sampler	Inputs	Outputs	Certified Facts
plan-slice-motion	$a \ o_k \ o_v \ g_k \ p_v \ w_1 \ w_2$	$q_0 \ q_1 \ t$	$(\text{SliceCutMotion } a \ o_k \ o_v \ g_k \ p_v \ w_1 \ w_2 \ q_0 \ q_1 \ t) \wedge$ $(\text{Conf } q_0) \wedge (\text{Conf } q_1) \wedge (\text{Traj } t)$
(c) Samplers			

Table 3: The domain-specific lifted operators, derived facts and samplers for the vegetable cutting domain. Throughout the table we use the symbols: a is a robot arm, o is an object, p_o is a pose of object o , g_o is a grasp on object o , q_i is a configuration, r is a region, t is a trajectory, w is a wrench. Specific to this domain: o_k and o_v refer to the knife and the vegetable, respectively.

Index

- GUARD, 92
- Action Regions, 93
- Action Uncertainty, 85
- Backwards Reachable Tube, 87
- Briefly-Dynamic Manipulation, 91
- Cartesian Impedance Control, 119
- Coulomb's friction law, 29, 65
- Danger Zones, 95
- Dead-End States, 88
- Fixturing, 58, 60, 74
- Forceful Kinematic Chain, 61, 62, 74
- Forceful Manipulation, 53
- Forceful operations, 53, 56, 58
- GSSP, 88
- Inevitable Collision States, 87
- Limit surface, 23, 25, 27, 63
 - Ellipsoidal Approximation, 63
 - Generalized Friction Cone, 23, 29, 65
- MMMP, 57
- Motion Cone, 21, 25
 - Gravity-free motion cone, 34
- Outcome Volume, 91
- PDDL, 67
 - effects, 67
 - fact, 67
 - operators, 67
 - preconditions, 67
- PDDLStream, 68
 - cost-sensitive planning, 77
 - streams, 68
- Principle of Maximal Dissipation, 23
- Quasistatic, 30, 53
- SSP, 88
- STRIPS, 67
- T-RRT*, 41
- TAMP, 57, 61, 66
- Torque Limits, 66