Robotic Manipulation Primitives

by

Eric Huang

Submitted to the Robotics Institute
in partial fulfillment of the requirements for the degree of

Ph.D. in Robotics

at the

CARNEGIE MELLON UNIVERSITY

April 2021

CMU-RI-TR-21-14

Author .....
Robotics Institute
May 4, 2021

Certified by .....
Matthew T. Mason
Professor
Thesis Supervisor

Accepted by .....
David Wettergreen
Chairman, Department Committee on Graduate Theses

Robotic Manipulation Primitives

by

Eric Huang

Submitted to the Robotics Institute
on May 4, 2021, in partial fulfillment of the
requirements for the degree of
Ph.D. in Robotics

Abstract

The central theme in robotic manipulation is that of the robot interacting with the world through physical contact. We tend to describe that physical contact using specific words that capture the nature of the contact and the action, such as *grasp*, *roll*, *pivot*, *push*, *pull*, *tilt*, *close*, *open* etc. We refer to these situation-specific actions as manipulation primitives. Due to the nonlinear and nonsmooth nature of physical interaction, roboticists have devoted significant efforts towards studying individual manipulation primitives.

This thesis begins with the study of pulling and pushing primitives. First, we derive exact bounds on the motion of a pulling object and use these bounds to plan pulling trajectories which guarantee convergence. Second, we show that large-scale multi-object rearrangement is achievable using only simple pushing primitives. However, in spite of the close relation between pushing and pulling, the required engineering effort within each project limited our ability to generalize across projects or to new scenarios.

This limitation motivates the main contribution of this thesis. We contribute a complete and general framework to autogenerate manipulation primitives. To do so, we develop the theory and computation of contact modes as a means to classify and enumerate manipulation primitives. The contact modes form a graph, specifically a lattice. Our algorithm to autogenerate manipulation primitives performs graph-search on the contact mode lattice and solves a quadratic program to generate each primitive. We hope that our contributions will lead to more general approaches for robotic manipulation.

Thesis Supervisor: Matthew T. Mason
Title: Professor

Acknowledgments

I would like to thank the professors who have helped me while I was at Georgia Tech: Mike Stilman, Andrea Thomaz, and Byron Boots. I would like to thank Matt Mason for taking me on as a PhD student at CMU. I would like to thank my PhD committee which includes Chris Atkeson, Aaron Johnson, and Kevin Lynch. I would also like to thank my the members and staff of the Manipulation Lab for supporting me over all these years. They are Robbie Paolini, Ankit Bhatia, Jiaji Zhou, Zhengzong Jia, Yifan Hou, Reuben Aronson, Xianyi Cheng, Pragna Mannam, Jon King, and Jean Harpley. I would like to thank Kuai Kuai Jin for her support. Finally, I would like to thank my family for always supporting me on my journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A central theme in robotics is that of robots interacting with the world through physical contact. We tend to describe that physical contact using specific words that capture the nature of the contact and the action. Reaching into a sink full of dishes and utensils, a robotic manipulator could choose to *cage* a fork against the corner and *scoop* it out. In warehouse automation, robots *grasp* items out of a bin and move them to their next destination. In a home environment, a robot may need to *push* chairs out of the way in order to navigate the dining room. A legged robot that has *slipped* on an icy surface needs to *stand-up* to regain its footing. Within the parlance of robotics research, we often refer to these situation-specific actions as behaviors, actions, skills, or *primitives*, the term favored in this thesis.

We desire a robot which can replicate the entire range of human skills. However, the reality is that physical interaction is difficult. Frictional contacts are non-linear and discontinuous in both theory and practice [82]. Coefficients of friction are usually not known beforehand, and even if they were, the pressure distributions at the location of contact are indeterminant [77]. It is also hard to even estimate the set of manipulation skills used by humans or animals. Byrne [15] documented 72 functionally distinct manipulation primitives used by foraging mountain gorillas. Nakamura [85] studied human grasping behaviors in grocery stores and noticed that existing taxonomies cannot categorize all the observed behaviors.

Within the taxonomy of physical interaction, this thesis is primarily focused on manipulation. Researchers have devoted significant efforts towards studying and recreating various manipulation primitives. Mechanics-based approaches to creating manipulation primitives have a long history. Mason's thesis was the first to study robotic pushing. A detailed analysis of the mechanics involved in robotic pushing has lead to multiple force-motion models [118, 52, 38] and motion bounds [91, 5,

78]. These results have been used to derive primitives such as two-point stable pushing [73] and uncertainty reducing grasping [120]. Researchers have also studied tumbling [99, 74] and pivoting primitives [4, 47, 49]. Peg-in-hole is a classic problem in robotic manipulation. There are still variants of peg-in-hole being studied today, such as the work on shallow depth insertion from Kim [61]. Extrinsic dexterity refers to robotic manipulation using external resources such as gravity or environmental contacts [20]. Hou developed a manipulation technique known as shared grasping which uses external contacts as secondary "hand" [50]. Chavan-Dafle a prehensile pushing primitive which uses external contacts to repositions objects with the hand [21, 19]. Ultimately, there are more research papers analyzing robotic manipulation primitives than can be reasonably listed.

Numerical optimization is another way to solve robotic manipulation problems. Stewart and Trinkle [101] published a contact-implicit time-stepping framework which forms the basis for much of the algorithms in this area. Posa et al. [93] proposed a direct method for solving contact-implicity trajectory optimization (CITO) problems leveraging Sequential Quadratic Programming (SQP). A key source of difficult is the hardness of contact constraints, i.e. a contact can only generate a force if the distance is zero. Tassa et al. [104] developed a soft contact model which was easier to optimize and used it to animate various manipulation tasks. Önol et al. [87] proposed a direct contact-implicit trajectory optimization method based on a variable smooth contact model and successive convexification. This is a challenging area of research which is undergoing active development.

Machine learning is an important tool for robotic manipulation. Learning from demonstration (LfD) is an area which aims to enable robots to learn manipulation primitives from human teachers [23]. Dynamical movement primitives (DMP) [56] are a well-known interaction model for learning dynamic tasks from demonstrations, such as swinging a racquet. Robots are also well-suited for learning through self-supervision. Learning grasping through repeated interactions has lead to spectacular breakthroughs in automated bin picking [92]. Probablistic models of grasping can adapt to outcomes which mechanics models fail to predict [90, 89]. However, for this approach to succeed, the robot requires strong priors to bias it towards meaningful interactions.

Manipulation primitives are the building blocks which enable robots to accomplish more complex tasks. Researchers combine action models with search-based planning methods to generate manipulation motions for a wide range of tasks. In the early work of in-hand regrasping [69], grasp gaits are obtained from searching on a grasp map developed from the force closure model. For nonprehensible two-palm manipulation, Erdmann [34] proposed get the set of possible motions by partitioning

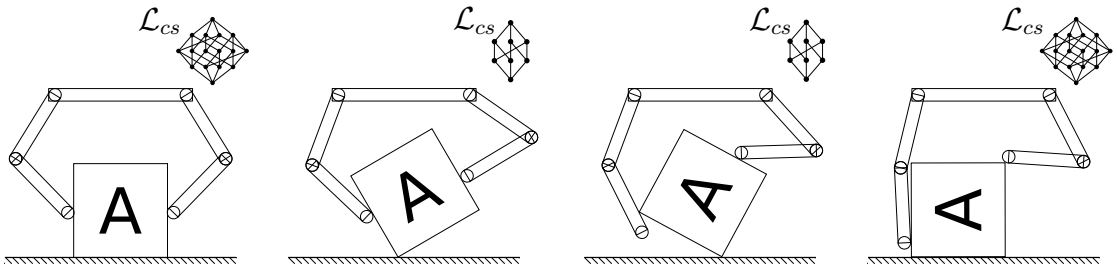


Figure 1-1: Manipulation sequence comprised of autogenerated manipulation primitives with contacting-separating mode lattices visualized.

the configuration space through primitive operations under different contact modes, and generate motion plans by searching. More recently, Chavan-Dafle et al. [21] combined high-level sampling based planning with the motion cones to achieve prehensible in-hand manipulation with external pushes. Hou et al. [49] proposed a fast planning framework using two reorientation motion primitives for object reoritation problems. All these works adopt the structure of combining high-level planning methods with handwritten low-level task models, which is difficult to generalize and scale.

1.1 Contributions

This thesis explores a primitive-based approach to robotic manipulation. Our methodology is grounded in the mechanics of robotic manipulation. We apply mathematical tools, ranging from variational analysis to computational geometry, to derive novel insights into the behavior, structure and, ultimately, automatic generation of robotic manipulation primitives. We use these primitives to create planning algorithms which solve difficult manipulation problems. We summarize each individual contribution below.

Robotic Pulling with Exact Convergence Bounds: We studied the quasi-static motion of a planar slider being pushed or pulled through a single contact point assumed not to slip [53]. The main contribution is to derive a method for computing exact bounds on the object's motion for classes of pressure distributions where the center of pressure is known but the distribution of support forces is unknown. The second contribution is to show that the exact motion bounds can be used to plan robotic pulling trajectories that guarantee convergence to the final pose. The planner was tested on the task of pulling an acrylic rectangle to random locations within the robot workspace. The generated plans were accurate to 4.00mm $\pm$ 3.02mm of the

target position and 4.35 degrees ± 3.14 degrees of the target orientation.

Robotic Pushing for Large-Scale Rearrangement: We also studied large-scale, multi-object pushing [54]. We introduced a new robotic tabletop rearrangement system, and presented experimental results. The system searches through policy rollouts of simulated pushing actions, using an Iterated Local Search technique to escape local minima. In real world execution, the system executes just one action from a policy, then uses a vision system to update the estimated task state, and replans. The system accepts a fully general description of task goals, which means it can solve the singulation and separation problems addressed in prior work, but can also solve sorting problems and spell out words, among other things. The tasks involve rearranging as many as 30 to 100 blocks, sometimes packed with a density of up to 40%. The high packing factor forces the system to push several objects at a time, making accurate simulation difficult, if not impossible. Nonetheless, the system achieves goals specifying the pose of every object, with an average precision of ± 1 mm and $\pm 2^\circ$.

Autogenerated Manipulation Primitives: These previous attempts demonstrated the value different primitives can bring to different scenarios and motivated the preeminent contribution of this thesis: a general method for autogenerating manipulation primitives. We believe this approach will be widely useful. First, autogeneration of a manipulation primitive library adds robustness to failure. A single primitive may fail, but a library gives the robot multiple options to try until success. Second, autogeneration removes the burden of writing task specific code.

In order to automatically generate manipulation primitives, we develop the theory of contact modes as a mathematically precise method for classifying and distinguishing between different manipulation primitives [55]. In a hybrid dynamical system with multiple rigid bodies, the relative motions of the contact points on two colliding bodies may be classified as separating, sticking (moving together), or sliding. Given a physical contact model, the active contact modes determine the dynamic equations of motion. Analogously, the set of all possible (valid) contact mode assignments enumerates the set of all possible dynamical flows of the hybrid dynamical system at a given state. Naturally, queries about the kinematics or dynamics of the system can be framed as computations over the set of possible contact modes. This motivated us to investigate efficient ways to compute the set of contact modes. To that end, we developed the first efficient 3D contact mode enumeration algorithm [55]. The algorithm is exponential in the degrees of freedom of the system and polynomial in the number of contacts. The exponential term is unavoidable and an example is provided. Prior work in this area has only demonstrated efficient contact mode enumeration in 2D for a single rigid body [77, 41].

We then used our contact mode enumeration algorithm to create a novel method for automatically generating robotic manipulation primitives [22]. Given a desired object velocity, the current state, and the dynamic equations of motion, our method generates a partially ordered list of manipulation primitives which would best approximate the object motion while satisfying contact dynamics. The generated manipulation primitives are uniquely specified by their contact mode. Our method is based on two key observations. First, specifying the contact mode reduces contact and friction force constraints into equality and inequality constraints. This in turn allows us to solve for the controls using quadratic programming. Second, the contact mode lattice is effectively a graph data structure. This allows us to use graph search techniques when designing our algorithm. Our algorithm walks along the contact mode lattice and solves a quadratic program at each visited node. In this way, we iteratively build a library of distinct manipulation primitives. Finally, we demonstrated our autogenerated manipulation primitives in various manipulation planning problems.

1.2 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 analyzes the mechanics of pulling an object with a single point of contact. Chapter 3 studies robotic pushing applied to large-scale, multi-object rearrangement. Autogenerated manipulation primitives are introduced in two parts, Chapters 4 and 5. The former chapter establishes the theory and computation of contact modes. The latter chapter discusses how to autogenerate manipulation primitives from the graph (lattice) formed by the contact modes. Chapter 6 summarizes the contributions of this thesis.

Chapter 2

Robotic Pulling with Exact Convergence Bounds

2.1 Introduction

Pushing (or pulling) planar objects with fixed contact is difficult to model in both theory and practice. First, pressure distributions of objects are statically indeterminant (barring the case of three-point support with known center of mass). Second, surface imperfections lead to spatial variability in both the pressure distribution and coefficient of friction [114]. Though several force-motion models for pushing exist [118, 52, 38], the above sources of indeterminacy ultimately lead to errors in the predicted velocity of the pushed object.

If the motion cannot be predicted, then another option is to find bounds on the velocity of the pushed object. This problem was first raised in Mason's thesis on robotic pushing [76]. In the case of fixed contact pushing, this is equivalent to finding bounds on angular velocity of the object as it is pushed through the contact point. To this end, we develop the first algorithm that finds *exact* angular velocity bounds on the object's motion over all pressure distributions with shared center of pressure. Moreover, the bounds are exact for many additional classes of pressure distributions that have not been considered before.

Dealing with uncertainty is a fundamental challenge in robotics [107]. We demonstrate how our bounds can be applied to planning for robotic pulling under action uncertainty. Robotic pulling is a general-purpose manipulation skill for positioning and orienting objects. The proposed planner uses the angular velocity bounds to find actions that reduce the uncertainty in the system, i.e. close the distance between the integrated orientation bounds. Moreover, given a suitable initialization, the planner

finds trajectories that guarantee the uncertainty at the final pose converges to a very small value.

The rest of this chapter is organized as follows. Section 2.2 discusses related work. Section 2.3 summarizes the relevant background on planar pushing needed to understand our work. Section 2.4 develops several theoretical results needed to prove the correctness of our algorithmic contributions. Section 2.5 introduces the exact angular velocity bound algorithm and the algorithm for planning pulling trajectories under action uncertainty. Section 2.6 presents our experimental results. Section 2.7 gives concluding remarks.

2.2 Related Work

We can categorize prior work bounding the motion of a pushed object according to the tightness of the bounds and the basic assumptions from which those bounds are deduced, such as knowledge of the center of pressure location. The bisector bound restricts the feasible rotation centers to a half-space delimited by the perpendicular bisector between the contact point and center of pressure [78]. Alexander and Maddocks bounded the set of feasible rotation centers to lie within the minimum object-enclosing vertical strip perpendicular to the wrench applied by the pusher [5]. Peshkin and Sanderson bounded the motion of a pushed object by computing the set of feasible rotation centers of the minimum object-enclosing disk centered at the object's center of pressure [91]. In all prior work, the bounds are conservative but not exact. That is, the bounds include every feasible motion, but also include infeasible motions.

To the extent of our knowledge, prior work has used robotic pulling significantly less compared to its counterpart, robotic pushing. The stable equilibrium that occurs when pulling along a straight line was recognized, without proof, by Mason [75], Lynch and Mason [72], and Berretty et al. [11]. For completeness, we provide a proof in Subsection 2.4.1. To the best of our knowledge, Berretty et al. is the only work, apart from ours, to explicitly take advantage of this stability [11]. However, the method of Berretty et al. is limited to orienting asymmetrical convex polygons. In contrast, our method can both position and orient any pullable object, regardless of its geometry. Our planner is also related to the work of Lynch and Mason [73]. They invented an algorithm that plans stable pushes using two contact points. However, their use of weaker bounds [91, 5, 78] leads to more conservative strategies.

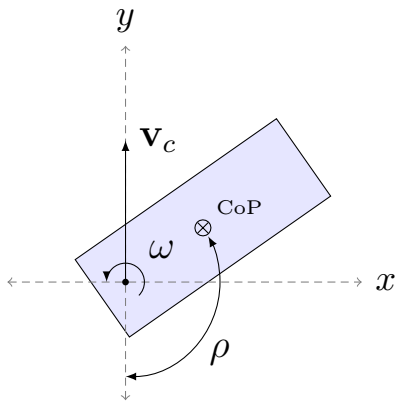


Figure 2-1: Coordinate frame associated with a press-pulled slider, where $\mathbf{v}_c$ is the contact point velocity, ω is the angular velocity of the slider, ρ is the angular deviation from the stable pulling configuration.

2.3 Background

This section provides the relevant background needed to understand our contributions. Subsection 2.3.1 serves two purposes. First, it defines the notation and terminology used throughout this chapter. Second, it frames our work in the context of the quasi-static theory of planar pushing. Subsection 2.3.2 summarizes the moment envelope construct used to jointly reason about pressure and frictional torques. This construct is particularly relevant for Subsection 2.5.2, which introduces an algorithm that computes the feasibility of a given pushed object velocity.

2.3.1 Planar Pushing Subject to Friction

In this work, we treat planar pushing as the manipulation skill where the robot contacts a rigid body at a point and pushes (or pulls) that fixed contact point along a trajectory. Our following presentation of the quasi-static analysis of planar pushing follows that of [5, 78]. A quasi-static analysis seeks to balance contact forces, gravity, and other applied forces while neglecting inertial forces [78].

Let the *generalized velocity*, or *twist*, of a planar rigid body be $\mathbf{v}^+ = [v_x, v_y, \omega]^T$, where v_x and v_y are the linear velocities of a reference point and ω is the angular velocity about that point. Taking the origin as the reference point, the velocity of a point $\mathbf{x} = [x_1, x_2]^T$ on the body is then given by $\mathbf{v}(\mathbf{x}) = [v_x, v_y]^T + \omega\hat{\mathbf{k}} \times \mathbf{x}$ with

$\hat{\mathbf{k}} = [0, 0, 1]^T$, and can be written in matrix notation as

$$\mathbf{v}(\mathbf{x}) = A(\mathbf{x})\mathbf{v}^+, \tag{2.1}$$

where

$$A(\mathbf{x}) = \begin{bmatrix} 1 & 0 & -x_2 \\ 0 & 1 & x_1 \end{bmatrix}. \tag{2.2}$$

Where convenient, we will convert freely between twists and their equivalent formulation, rotation centers. The mappings

$$[v_x, v_y, \omega] \rightarrow [-\frac{v_y}{\omega}, \frac{v_x}{\omega}, \omega] \tag{2.3}$$

$$[r_x, r_y, \omega] \rightarrow [r_y\omega, -r_x\omega, \omega] \tag{2.4}$$

map a twist at the origin to a rotation center and angular velocity about that rotation center and vice versa.

We define the *contact frame* to be the coordinate frame where the origin is the contact point and the y-axis aligns with the contact point velocity (see Figure 2-1 for an example). Let ρ be the *angular deviation* from the stable pulling configuration to the rigid body's center of pressure. The stable pulling configuration occurs when the center of pressure is collinear with and behind the direction of the pulling motion (proof in Subsection 2.4.1). The contact point is assumed to be pushed with unit speed, yielding a body twist $\mathbf{v}^+ = [0, 1, \omega]^T$. Assuming Coulomb's law of sliding friction, the total frictional force and moment of the body at the origin are

$$\mathbf{f}_f = -\mu \int_R \frac{A(\mathbf{r})\mathbf{v}^+}{\|A(\mathbf{r})\mathbf{v}^+\|} p(\mathbf{r}) dA \tag{2.5}$$

$$\mathbf{m}_f = -\mu \int_R \mathbf{r} \times \frac{A(\mathbf{r})\mathbf{v}^+}{\|A(\mathbf{r})\mathbf{v}^+\|} p(\mathbf{r}) dA, , \tag{2.6}$$

where μ is the coefficient of friction (static and dynamic), R is the region of the rigid body in contact with the plane, $\mathbf{r}$ is a point in R, $\mathbf{v}(\mathbf{r})$ is the body point velocity given by (2.1) and $p(\mathbf{r})$ is a pressure distribution over R.

When the contact point is pushed at constant velocity, the quasi-static assumption states that the total moment at the contact point is zero. Because a point contact cannot generate any torque, this implies the total frictional moment (2.6) must also be zero. This leads to the following constraint on the possible motions of an object.

2.1 Definition. An angular velocity ω and its corresponding twist $\mathbf{v}^+ = [0, 1, \omega]^T$

are said to be **feasible** if there exists a pressure distribution $p(\mathbf{r})$ such that the resulting total frictional moment is zero.

Note that both the angular velocity and its corresponding twist are taken with respect to the contact point frame. However, using equations (2.5) and (2.6) directly to check feasibility can lead to difficulties when the integrand's denominator $\|A(\mathbf{r})\mathbf{v}^+\|$ is zero. The next formulation obviates that difficulty.

The principle of minimal dissipation states that the motion of the pushed body minimizes the instantaneous work dissipated by friction [5]. That is, the motion minimizes the following:

$$\begin{array}{ll} \underset{\mathbf{v}^+}{\text{minimize}} & \mu \displaystyle\int_R \|A(\mathbf{r})\mathbf{v}^+\| p(\mathbf{r}) dA \\ \text{subject to} & \mathbf{v}^+ \in \mathcal{C}. \end{array} \tag{2.7}$$

We take $\mathcal{C} = \{[0, 1, \omega]^T, \omega \in \mathbb{R}\}$ so that the contact point motion is aligned with the coordinate frame. We call the objective in (2.7) the *frictional dissipation function*, $\mathrm{P}(\mathbf{v}^+)$. The principle of minimal dissipation is equivalent to the quasi-static model of planar sliding with friction [5]. In fact, the quasi-static motion constraints are identical to the first order optimality conditions of (2.7). To see this, take the Lagrangian of (2.7)

$$\mathrm{L}(\mathbf{v}^+, \lambda) = \mathrm{P}(\mathbf{v}^+) + \lambda_x v_x^+ + \lambda_y (v_y^+ - 1), \tag{2.8}$$

where λ_x and λ_y are Lagrangian multipliers, and set gradient of L with respect to $\mathbf{v}^+$,

$$\nabla \mathrm{L}(\mathbf{v}^+, \lambda) = \mu \int_R \frac{A(\mathbf{r})^T A(\mathbf{r})\mathbf{v}^+}{\|A(\mathbf{r})\mathbf{v}^+\|} p(\mathbf{r}) dA + \left[\lambda_1, \lambda_2, 0\right]^T, \tag{2.9}$$

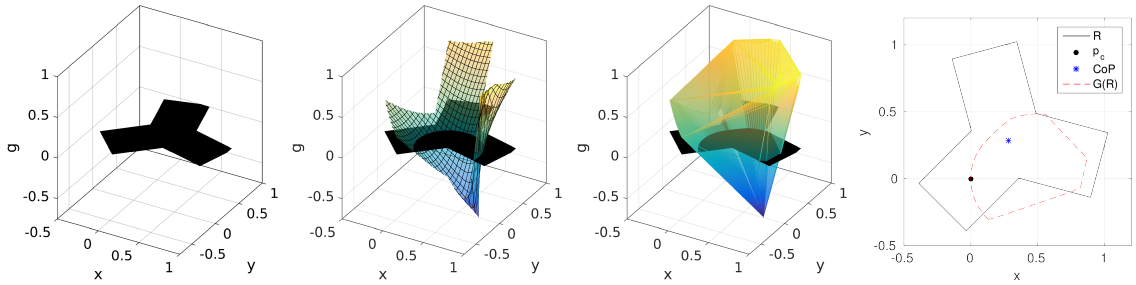
to zero. When equation 2.9 is written out element-wise, we end up with the following first order conditions on the force and the moment

$$\mathbf{f}_f = \left[\lambda_x, \lambda_y\right]^T \tag{2.10}$$

$$\mathbf{m}_f = 0, \tag{2.11}$$

that is, the quasi-static motion model.

In addition to avoiding the zero denominator issue of equations (2.5) and (2.6), we prefer using the principle of minimal dissipation in the proofs of our main results (Section 2.4) because the frictional dissipation equation (2.7) is continuous and convex in $\mathbf{v}^+$.



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

Figure 2-2: Example moment envelope for a trigonal 2D object with rotation center $x_{\text{IC}} = 0.75$. (a) Support region R. (b) Normalized-moment surface $G(R)$. (c) Convex moment envelope of $G(R)$. (d) Intersection of the moment envelope and the xy-plane. The intersection bounds the set of feasible centers of pressure with zero moment.

2.3.2 Frictional Moment Envelopes

The frictional moment envelope provides a nice geometric model of the constraints between the center of pressure and moment at the contact point [78]. In this subsection, we show that the moment envelope is important because it reduces the feasibility test of a particular angular velocity to a point-in-convex-hull test. An example frictional moment envelope is illustrated step-by-step in Figure 2-2.

A frictional moment envelope has as its parameters the support region R and twist $\mathbf{v}^+$. Let f_0 be the total normal force. Then the function

$$g(\mathbf{x}) = -\mu f_0 \, \mathbf{x} \times \frac{A(\mathbf{x})\mathbf{v}^+}{\|A(\mathbf{x})\mathbf{v}^+\|} \tag{2.12}$$

evaluates the frictional torque that would result from a unit normalized pressure at $\mathbf{x}$. Let G map R into a surface in $\mathbb{R}^3$ by associating each point $\mathbf{x} \in R$ with its maximum potential moment $g(\mathbf{x})$, i.e.

$$G(\mathbf{x}) = \begin{bmatrix} x \\ y \\ g(\mathbf{x}) \end{bmatrix}, \tag{2.13}$$

and let $\hat{p} = p/f_0$ be the normalized pressure. Then the set $\{\int_R G(\mathbf{r})\hat{p}(\mathbf{r})dA \,|\, \int_R \hat{p}(\mathbf{r})dA = 1\}$ is the convex hull of the surface $G(R)$. Moreover, any point in the convex hull of

$G(R)$ satisfies

$$\int_R G(\mathbf{r})\hat{p}(\mathbf{r})dA = \int_R \begin{bmatrix} x \\ y \\ g(\mathbf{x}) \end{bmatrix} \hat{p}(\mathbf{r})dA$$
$$= \begin{bmatrix} x_0 \\ y_0 \\ \int_R g(\mathbf{x})\hat{p}(\mathbf{r})dA \end{bmatrix}. \tag{2.14}$$

Thus, the convex hull of $G(R)$ is the set of all feasible centers of pressure and frictional moments for a given support region R and twist $\mathbf{v}^+$. We refer to the convex hull of $G(R)$ as the *moment envelope generated by* $\mathbf{v}^+$. Therefore, given a center of pressure $[x_0, y_0]^T$, an angular velocity ω is feasible if and only if the point $[x_0, y_0, 0]^T$ is contained in the moment envelope generated by ω.

2.4 Theory

This section covers our theoretical contributions. Subsection 2.4.1 proves the existence and uniqueness of the stable equilibrium during pulling. This result motivates our main contributions. Subsection 2.4.2 lays the theoretical groundwork necessary for proving the correctness of the exact angular velocity bound algorithm introduced in 2.5.1. Subsection 2.4.3 extends the angular velocity bounds to orientation bounds. This subsection completes the theoretical tools needed in Subsection 2.5.3 to generate robotic pulling trajectories that guarantee convergence to the final pose.

2.4.1 Stable Equilibrium When Pulling

For completeness, we prove the existence and uniqueness of the stable equilibrium that occurs when pulling a rigid body. The existence of the stable equilibrium was observed, without proof, in Mason [75], Lynch and Mason [72], and Berretty et al. [11]. The robotic pulling trajectories generated in Subsection 2.5.3 automatically use the stable equilibrium to reduce pose uncertainty.

2.2 Theorem. For pulling of a rigid body in the plane, the rigid body converges to the state where its center of pressure is collinear with the pulling direction.

Proof. We know from Theorem 7.4 in [78] that the rigid body translates when the center of pressure is already collinear with the pulling direction. Now, suppose the center of pressure is strictly to the right of the pulling direction (as in Figure 2-1).

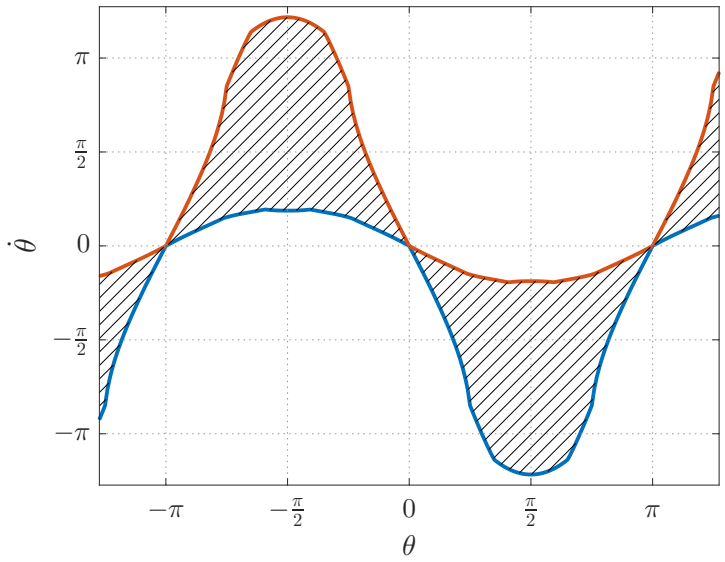


Figure 2-3: Example angular velocity bounds for the trigonal 2D object from Figure 2-2. The object is oriented such that the stable pulling configuration corresponds to $\theta = 0$. The orange curve is the upper bound α. The blue curve is the lower bound β.

Then by Theorem 7.4 in [78], the rigid body rotates clockwise about the contact point, i.e. has angular velocity $\omega < 0$. Let $\rho \in (\pi, 0)$ be the angular deviation of the rigid body. Since ρ is monotonically decreasing and $\omega = 0$ if $\rho = 0$ or π, we see that ρ converges to 0 in the limit as $t \to \infty$. The case when the center of pressure is strictly to the left of the line of motion follows from symmetry. □

2.4.2 Properties of Angular Velocities Bounds

We prove that the set of feasible angular velocities for an object with known center of pressure is connected and bounded. These two properties justify the use of a bisection search to locate the minimum and maximum angular velocities in Subsection 2.5.1.

We begin by citing a proposition from variational analysis used in our proofs of the main results.

2.3 Proposition. Suppose $P(u) := \arg\min_x f(x, u)$ with $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ continuous and level-bounded in x locally uniformly in u. Then the set-valued mapping $P(u)$ is outer-semicontinuous and locally bounded.

Proof. Proposition adapted from Corollary 7.42 and Theorem 1.17 in [97]. □

2.4 Theorem. For pulling of a planar rigid body with known center of pressure, the set of all feasible angular velocities is connected.

Proof. Let Ω be the set of feasible angular velocities, w.r.t. (2.7), for a given support region R with known center of pressure. Suppose Ω is non-empty. Let $\omega_1, \omega_2 \in \Omega$ and let p_1, p_2 be their corresponding pressure distributions. Then the new distribution $p_t = tp_1 + (1-t)p_2$, with $t \in [0,1]$, shares the same center of pressure as p_1, p_2. Define the function

$$f(\omega, t) = \mu \int_R \|A(\mathbf{r})\mathbf{v}^+(\omega)\| p_t(\mathbf{r}) dA, \tag{2.15}$$

where $\mathbf{v}^+ : \omega \to [0, 1, \omega]^T$ and $\operatorname{dom} f := \mathbb{R} \times [0,1]$. By inspection, f is continuous. For all $t \in [0,1]$ and $\alpha \in \mathbb{R}$, the set $\{(\omega, t) \mid f(\omega, t) \leq \alpha\}$ is bounded because $f \to \infty$ as $|x| \to \infty$. Hence, f is level-bounded in ω locally uniformly in t, and f satisfies the conditions of Proposition 2.3.

The image of a connected set by an outer-semicontinuous set-valued mapping whose values are nonempty and connected is connected [46]. Let $P(t) := \arg\min_\omega f(\omega, t)$. For a given t, the set $P(t)$ is convex-valued and therefore connected. Since f is continuous and level-bounded, the set is also nonempty (Theorem 1.9 [97]). Therefore, the image $P([0,1])$ contains the interval connecting ω_1 and ω_2. Because the choice of ω_1, ω_2 was arbitrary, Ω is connected. □

In general, Theorem 2.4 holds for any convex set of pressure distributions with known center of pressure.

2.5 Corollary. For pulling of a planar rigid body with known center of pressure, the set of all feasible angular velocities is bounded.

Proof. We prove Corollary 2.5 separately for the cases when the rigid body's center of pressure lies strictly in the right-half plane, left-half plane, or on the y-axis.

Suppose the center of pressure lies in the right-half plane of the contact frame. Then by Theorem 7.4 of [78] the angular velocity of the pulled body is strictly negative and hence, 0 bounds Ω from above. To establish a lower bound, we appeal to the following two properties about the set of feasible rotation centers when pushing or pulling with sticking contact. First, the rotation center must lie on the x-axis. Second, the rotation center must lie behind the line bisecting the contact point and the center of pressure [78]. We observe that any feasible rotation center must have the form $[x, 0]$ with $x > x^* > 0$, where x^* is the intersection of the bisecting line and the x-axis. Recall that, by convention, we fix the contact point velocity to

$\mathbf{v}_c = [0, 1]^T$. We compute the angular velocity ω^* at $[0, x^*]$ using $\mathbf{v}_c$ and see that Ω is bounded from below by

$$\omega^* = -\frac{\|\mathbf{v}_c\|}{x^*}. \tag{2.16}$$

Thus, Ω is bounded. The case when the center of pressure lies in the left-half plane follows from a symmetrical argument.

When the center of pressure is collinear with the pulling direction, the rigid body translates and $\Omega = \{0\}$ [78]. □

2.4.3 Integrated Orientation Bounds

We prove that the angular velocity bounds derived in Subsection 2.4.2 integrate into bounds on the orientation of the pulled body.

Let θ be the orientation of the rigid body in the world frame and let u and l be upper and lower bounds on the orientation in the world frame. Let the function $\omega(\rho)$ map from the angular deviation to the angular velocity of the rigid body. Likewise, let the functions $\alpha(\rho)$ and $\beta(\rho)$ map from the angular deviation to the upper and lower angular velocity bounds. An example phase-plot of $\alpha(\rho)$ and $\beta(\rho)$ is illustrated in Figure 2-3. Note the stable equilibrium at $\theta = 0$.

We assume that the pulling trajectory $\gamma : \mathbb{R} \to \mathbb{R}^2$ can be approximated by a finite number of straight line segments of equal length. Given such a γ, the pulling angle $\phi(t) = \tan^{-1}(\dot{\gamma}_y(t), \dot{\gamma}_x(t))$ is a piece-wise constant (step) function. Let $v(t) = \|\dot{\gamma}(t)\|$. As the planar rigid body is pulled along γ with unit velocity, the state and bounds change according to the dynamical system

$$\dot{x} = \cos(\phi(t)) \tag{2.17}$$
$$\dot{y} = \sin(\phi(t)) \tag{2.18}$$
$$\dot{\theta} = \omega(\theta - \phi(t) + \pi) \tag{2.19}$$
$$\dot{u} = \alpha(u - \phi(t) + \pi) \tag{2.20}$$
$$\dot{\ell} = \beta(\ell - \phi(t) + \pi), \tag{2.21}$$

where the expression $z - \phi(t) + \pi$ maps an orientation z in the global frame to the angular deviation.

2.6 Proposition. For pulling of a rigid body with known initial pose, the orientation of the body is bounded above and below by u and ℓ.

Proof. Suppose the proposition is false and θ crosses the bound u at time t_0, i.e. $u(t_0) = \theta(t_0)$ and $\theta(t) > u(t)$ immediately afterwards. Let the line segment of γ

at t_0 be indexed by i and have length ε. Then $\phi(t)$ is constant for t in the range $[t_0, (i+1)\varepsilon)$. Pick t_1 from said range such that $\theta(t_1) > u(t_1)$. Because ϕ is constant in $[t_0, t_1]$, we can apply separation of variables to solve differential equation (2.20) and get

$$\int_{u(t_0)}^{u(t_1)} \frac{1}{\alpha(x - \phi(t_0))} dx = \int_{t_0}^{t_1} dt. \tag{2.22}$$

This result shows that we can integrate the inverse of an angular velocity function to compute the amount of time required to reach a particular orientation. However, we can also apply separation of variables to the function ω. Observe that because α is an upper bound on ω

$$\alpha(x - \phi(t_0)) \geq \omega(x - \phi(t_0)), \tag{2.23}$$

the resulting integral of ω satisfies

$$\int_{u(t_0)}^{u(t_1)} \frac{1}{\alpha(x - \phi(t_0))} dx \leq \int_{u(t_0)}^{u(t_1)} \frac{1}{\omega(x - \phi(t_0))} dx. \tag{2.24}$$

This indicates that it takes less time for u to reach $u(t_1)$ compared to θ, and therefore, $\theta(t_1) \leq u(t_1)$, a contradiction. Therefore, the upper bound holds and the lower bound follows from a similar argument. □

2.5 Methods

In this section, we synthesize the materials in Sections 2.3 and 2.4 into an algorithm for computing exact angular velocity bounds and a method for planning convergent trajectories using the computed bounds. The former is detailed in Subsection 2.5.1 and extended in Subsection 2.5.2 and the latter is detailed in Subsection 2.5.3.

2.5.1 Exact Angular Velocity Bound Algorithm

Algorithm 1 finds exact angular velocity bounds for a given support region R and center of pressure $[x_0, y_0]^T$. It uses bisection search to estimate the end-points of Ω. This choice is justified because Ω is connected and bounded by Theorem 2.4 and Corollary 2.5.

To initialize the bisection search, we need an angular velocity in Ω and two angular velocities above and below the bounds of Ω. First, we compute an angular velocity ω in Ω. We find a feasible assignment of pressures P such that the center of pressure

Algorithm 1 Exact Angular Velocity Bounds

1: **function** FIND EXTREMA(R, x_0, y_0)
2: **if** x_0 is 0 **then return** $[0, 0]$
3: $P \leftarrow \min_{\mathbf{x}} \mathbf{0}$ s.t. $R\mathbf{x} = [x_0, y_0]^T, L \leq \mathbf{x} \leq U$
4: $x_r \leftarrow$ COMPUTE ROTATION CENTER(R, P)
5: $\omega \leftarrow -\|\mathbf{v}_c\|/x_r$
6: $l \leftarrow 0$
7: $\omega_1 \leftarrow$ BISECTION SEARCH(R, x_0, y_0, l, ω)
8: $u \leftarrow \omega$
9: **do**
10: $u \leftarrow 2u$
11: $\mathbf{v}^+ \leftarrow [\mathbf{v}_c^T, u]^T$
12: $G \leftarrow \{-\mathbf{x} \times A(\mathbf{x})\mathbf{v}^+/\|A(\mathbf{x})\mathbf{v}^+\| \mid \mathbf{x} \in R\}$
13: **while** $[x_0, y_0, 0]^T \in$ CONVHULL(G)
14: $\omega_2 \leftarrow$ BISECTION SEARCH(R, x_0, y_0, u, ω)
15: $l \leftarrow$ MIN(ω_1, ω_2)
16: $u \leftarrow$ MAX(ω_1, ω_2)
17: **return** $[l, u]$
18: **function** BISECTION SEARCH(R, x_0, y_0, α, β)
19: **while** $\varepsilon < |\alpha - \beta|$ **do**
20: $\omega \leftarrow (\alpha + \beta)/2$
21: $\mathbf{v}^+ \leftarrow [\mathbf{v}_c^T, \omega]^T$
22: $G \leftarrow \{-\mathbf{x} \times A(\mathbf{x})\mathbf{v}^+/\|A(\mathbf{x})\mathbf{v}^+\| \mid \mathbf{x} \in R\}$
23: **if** $[x_0, y_0, 0]^T \in$ CONVHULL(G) **then**
24: $\beta \leftarrow \omega$
25: **else**
26: $\alpha \leftarrow \omega$
27: **return** $(\alpha + \beta)/2$

is $[x_0, y_0]$ (line 3). From P, the resultant rotation center $[x_r, 0]$ can be computed using the root-finding method in [76] (line 4). Lastly, we convert the rotation center into the angular velocity $\omega \in \Omega$ (line 5). Of the two out-of-bound angular velocities l and u, we can set l to 0 (line 6). The other can be found by repeatedly doubling ω until the resulting angular velocity is no longer feasible (lines 8-13). As a reminder, we test the feasibility of a given angular velocity ω' by checking whether the point $[x_0, y_0, 0]^T$ is contained in the associated frictional moment envelope (see Section 2.3.2). Now that ω, u, and l have been computed, we pass them into the bisection

search to compute the boundary points of Ω (lines 7 and 14).

The run-time of the algorithm is $\mathcal{O}(d\,n \log n)$, where d is the number of significant digits returned and n is the number of points in the discretization of R. This computation is relatively expensive to perform online. In Section 2.5.3, we avoid recomputing angular velocity bounds by fitting Fourier series to a set of pre-computed orientation-bound pairs.

2.5.2 Improving on Exact Angular Velocity Bounds

The exact angular velocity bounds computed in Section 2.5.1 result in slow convergence towards the stable pulling equilibrium point (for experimental measurements, see Subsection 2.6.1). Consequently, wide bounds cause our planner to generate long trajectories that exceed the robot's workspace in order to satisfy tolerances on the final pose uncertainty.

In this subsection, we show how to modify the constraints on the pressure distributions from which the bounds were computed. This allows us to restrict pressure distributions to smaller subclasses and thus achieve tighter angular velocity bounds. Let $\mathcal{C}$ be the class of normalized pressure distributions over a region R with center of pressure $[x_0, y_0]$. Now, suppose we had a convex subclass $\mathcal{K}$ of pressure distributions such that $\mathcal{K} \subset \mathcal{C}$. Regrettably, the point-in-convex-hull feasibility test only works for $\mathcal{C}$. However, we can setup an alternative feasibility test with respect to $\mathcal{K}$ by solving the linear program

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & \left\| \sum_R g(\mathbf{r}) p(\mathbf{r}) \right\| \\ \text{subject to} \quad & p \in \mathcal{K}, \end{aligned} \tag{2.25}$$

where p is a discretized pressure distribution, $g(\mathbf{r})$ is the unit-torque function from equation (2.12), and the summation is over points $\mathbf{r} \in R$. A given angular velocity ω is feasible if and only if the linear program (2.25) finds a pressure distribution p such that the objective $\|\sum_R g(\mathbf{r})p(\mathbf{r})\|$ is 0 and $p \in \mathcal{K}$.

Several options exists for the choice of $\mathcal{K}$. In our experiments, we use

$$\mathcal{K} = \{p \mid 0 \leq p_i \leq U, p \in \mathcal{C}\}, \tag{2.26}$$

where $U \leq 1$ is an upper bound on the discretized pressures. The upper bound U controls the percentage of R guaranteed to be in contact with the surface, i.e. has non-zero pressure. For example, if we set $U = 2/N$, where N is the number of points in the discretization of R, then, by the pigeon-hole principle, at least 50% of R is

always in contact with the surface. Our implementation solves linear program (2.25) using `Gurobi` [40].

2.5.3 Planning Convergent Trajectories for Robotic Pulling

We use control-limited Differential Dynamic Programming (DDP) [106] to plan convergent trajectories for robotic pulling. At a high level, DDP solves a trajectory optimization problem where the objective is to reach a target pose $[x_T, y_T, \omega_T]^T$ with small uncertainty, i.e. the integrated orientation bounds are within ϵ of ω_T.

Let the discretized state be $\mathbf{x}_i = [x_i, y_i, u_i, l_i]^T$, $i \in [1, N]$. Let the discretized controls be $\mathbf{u}_i = [d_i, \phi_i]^T$, $i \in [1, N]$, where d_i is the distance to travel and ϕ_i is the heading in the global frame. We use the following first order approximation of the discretized dynamics in our trajectory optimizer

$$x_{i+1} = x_i + d_i \cdot \cos(\phi_i) \tag{2.27}$$
$$y_{i+1} = y_i + d_i \cdot \sin(\phi_i) \tag{2.28}$$
$$u_{i+1} = u_i + d_i \cdot \hat{\alpha}(u_i - \phi_i + \pi) \tag{2.29}$$
$$l_{i+1} = \ell_i + d_i \cdot \hat{\beta}(l_i - \phi_i + \pi) \tag{2.30}$$
$$h_{i+1} = h_i + d_i, \tag{2.31}$$

where the functions $\hat{\alpha}$ and $\hat{\beta}$ are Fourier series approximations of the upper and lower angular velocity bounds (to avoid costly in-loop computations) and the additional state h_i measures the cumulative distance pulled.

We use the following cost functions to bias the trajectory optimizer towards finding convergent trajectories. We set the running cost $\mathcal{L}(\mathbf{x}_i, \mathbf{u}_i)$, $i \in [1, N-1]$ to zero. We set the final cost to be

$$\mathcal{L}_F(\mathbf{x}_N, h_N) = \mathbf{k}^T \mathcal{L}_{\boldsymbol{\delta}}(\mathbf{x}_N - \mathbf{x}_F) + \lambda h_N^2, \tag{2.32}$$

where $\mathcal{L}_{\boldsymbol{\delta}}$ is the vectorized version of the Pseudo-Huber loss function[1]

$$\mathcal{L}_\delta(a) = \sqrt{a^2 + \delta^2} - \delta, \tag{2.33}$$

$\mathbf{x}_F = [x_T, y_T, \omega_T, \omega_T]^T$ is the target state, $\mathbf{k}$ and $\boldsymbol{\delta}$ are the slope and width, respectively, of the vectorized Pseudo-Huber loss function, and λ is the distance penalty coefficient. Note that the target upper and lower orientation bounds are equal to ω_T in the target state $\mathbf{x}_F$. This ensures the generated trajectory minimizes uncertainty

[1] This function approximates an ℓ_1 norm for $a > \delta$.

in the final orientation (due to Proposition 2.6). Finally, we initialize our trajectory optimizer using paths generated from Dubins' curves. Pushing with sticking contact shares similar dynamics with the simple car [29, 117].

2.6 Experiments

2.6.1 Comparison of Angular Velocity Bounds

In this experiment, we compare distance-to-convergence for our exact angular velocity bounds and the previous best bound, i.e. the Peshkin bound [91]. We test the bounds over the objects in the MIT Pushing Dataset [114] and randomly generated bipods, tripods, and quadrapods. The generated n-pods were chosen to have circumcircle diameters similar to the MIT objects, roughly 0.16m.

For each MIT object, we pick 10 even spaced contact points on the boundary of the object. We generate 30 random n-pods for each category and took the contact point to be the center of a random pod (similar to pulling the leg of a chair). We compute distance-to-convergence in the following manner. Let γ be an angular velocity bound (can be upper or lower). We orient the object such that the center of pressure is 90 degrees away from the stable configuration. Next, we simulate a pulling trajectory while integrating γ and stop when the integral converges to within 1 degree of the stable configuration. The distance travelled is the distance-to-convergence[2].

The experimental results are collected in Table 2.1. The Peshkin bound computes the feasible angular velocities for the circumcircle enclosing the object. As a result, it underestimates the slowest angular velocity bound and its distance-to-convergence can be twice are far as compared to the exact bound. When feasible pressure distribution are restricted such that at least 50% of the object is in contact with the surface, the distance-to-convergence of the exact bound is reduced by another factor of two. Because the exact bound converges within 3/4 a meter, it is serviceable for manipulating the MIT objects on a large table. Naturally, smaller objects or tighter bounds are required for smaller tables.

2.6.2 Robotic Pulling on a Tabletop

Figure 2-4 shows the experimental setup that we used to test the robotic pulling trajectories generated by the planning algorithm in Subsection 2.5.3.

[2]Note that this distance is independent of the pulling velocity, see Equation (2.16).

	Exact	Exact-50%	Peshkin
MIT	0.670±0.141	**0.362**±0.073	1.354±0.501
	0.172±0.047	**0.243**±0.053	0.162±0.041
Bipod	0.762±0.210	**0.692**±0.213	0.899±0.239
	0.516±0.218	**0.574**±0.216	0.273±0.095
Tripod	0.765±0.133	**0.688**±0.143	1.180±0.244
	0.522±0.162	**0.576**±0.157	0.340±0.108
Quadrapod	0.880±0.120	**0.749**±0.114	1.207±0.235
	0.417±0.099	**0.489**±0.098	0.329±0.096

Table 2.1: Comparison of distance-to-convergence (in meters) for different objects and angular velocity bounds. The top and bottom values in each cell correspond to distances from the upper and lower angular velocity bounds, respectively. See Section 2.6.1 for the experimental setup.

Experimental data was collected using an ABB 140 manipulator equipped with a conical finger. The test object was a laser-cut acrylic rectangle (75mmx50mmx6.35mm) with 8 holes at the edges and corners. The conical finger moved the acrylic rectangle by pulling inside the holes. A 5 camera OptiTrack motion capture system was set up to record ground truth position of the object in 2D with a accuracy of 2mm. To compensate sensing error, the holes on the rectangle were oversized to have a 3mm radius. We used MDF board as our surface material.

We computed angular velocity bounds for the acrylic rectangle over pressure distributions restricted to have at least 25% of the object is in contact with the surface. The slope $\mathbf{k}$ of the Pseudo-Huber Loss function for the DDP planner was set to $[5000, 5000, 1000, 1000]$ and the width $\boldsymbol{\delta}$ was set to $[0.01, 0.01, 0.02, 0.02]$. The distance penalty λ was set to 40. For each pulling trial, we generated random start and end poses within the vision system's field of view. The planner was evaluated for all eight contact points and the lowest cost trajectory that remained within the robot workspace was executed on the robot at 25mm/s linear speed. The final pose was then recorded by the motion capture system.

We collected eighty trials of robotic pulling. Of those eighty, we discarded the four trials where our planner failed to find trajectories that satisfied workspace constraints. The average absolute displacement from the target pose was 4.00mm ± 3.02mm. The average absolute angular displacement from the target pose was 4.35

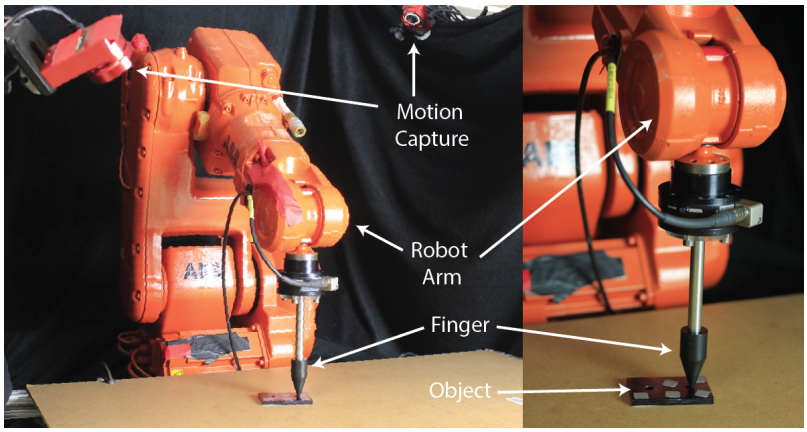


Figure 2-4: Hardware setup for robotic pulling experiments.

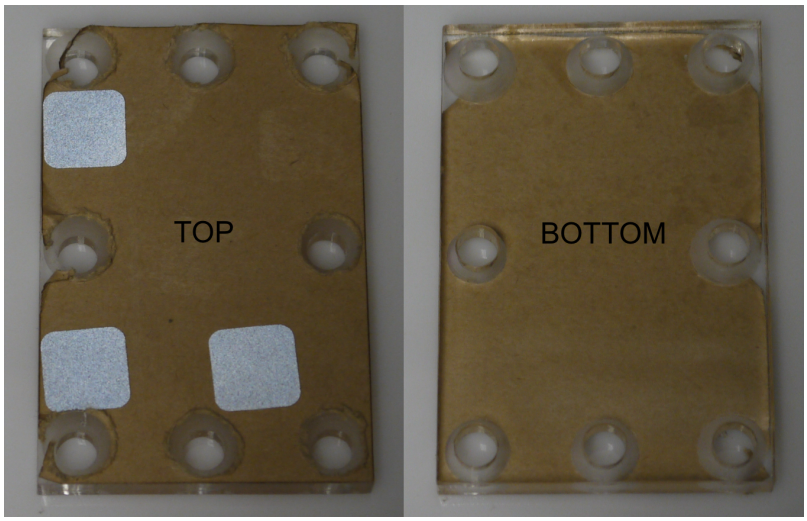


Figure 2-5: The top and bottom view of the acrylic rectangle with motion capture markers and 8 holes for pulling.

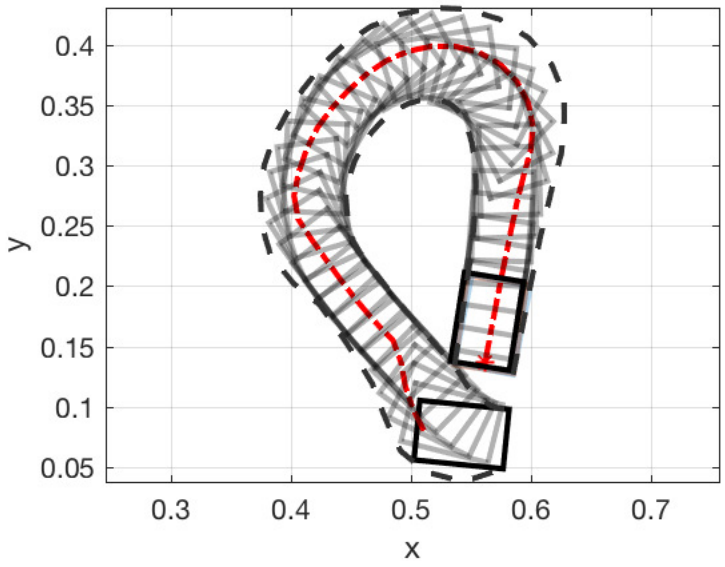


Figure 2-6: (Dashed red line) The planned robotic pulling trajectory. (Dashed black line) The area swept by the possible poses computed by our bounds during the planned trajectory. (Grey rectangles) The measured poses of the object when the trajectory was executed on a real robot.

degrees $\pm$ 3.14 degrees. Note the hole radius introduces a systematic error of 3mm to the final pose because the puller contacts the edge of the hole, not the center. Overall, our experimental results support the claim that the planner finds convergent pulling trajectories. An example trial is visualized in Figure 2-6.

2.7 Conclusion

In this chapter, we derive a method for computing exact bounds on the object's motion for classes of pressure distributions where the center of pressure is known but the distribution of support forces is unknown. We also show these exact motion bounds can be used to plan robotic pulling trajectories that guarantee the pulled object converges to the final pose. We validate our planner on a real robotic system and show that the generated trajectories obtain low errors on the final pose of the object.

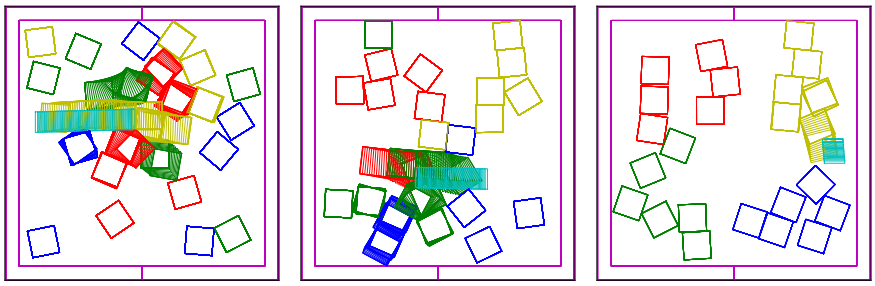
Chapter 3

Robotic Pushing for Large-Scale Rearrangement

3.1 Introduction

Past robotics research has identified certain scenarios in which robots need to *rearrange* multiple objects in their environment in order to accomplish a goal. One early example of such a scenario is the problem of navigation among moveable obstacles (NAMO) introduced by Wilfong [113]. Later on, Stilman and Kuffner [102] demonstrated the possibility of real-time NAMO in tight household spaces containing upwards of 90 pieces of furniture, despite NAMO problems being fundamentally NP-hard to solve [27]. Home environments are also a natural setting for scenarios involving pick-and-place rearrangement planning. The most difficult problems in this domain are *non-monotone*, i.e. the solutions require moving objects more than once. Recently, Krontiris and Bekris [65] and Han et al. [42] presented algorithms which handle non-monotonicity during pick-and-place rearrangement on shelves and tabletops, respectively; however, finding optimal solutions is often NP-hard [42].

This chapter focuses on *non-prehensile* rearrangement planning, a type of rearrangement which emphasizes the need for multi-contact manipulation. For the rest of this thesis, the term rearrangement planning implies the non-prehensile variant. We categorize non-prehensile rearrangement planning problems based on whether the objective is to singulate, separate, reposition, arrange, or sort objects. We define *singulate* as isolating a single object for grasping [45, 66, 25, 81, 28, 2, 116], *separate* as acheiving a minimum separating distance between all objects [17, 31], *reposition* as moving a subset of the objects into target positions (no orientation constraints) [62, 63, 64, 43, 44, 115, 10], *arrange* as moving the objects into a target configura-



(a) Action 5 (b) Action 83 (c) Action 133

Figure 3-1: Example pushes generated by our algorithm as it sorts 24 blocks by color.

tion (with position and orientation constraints) [6], *sort* as grouping objects based on some similarity metric.

Prior work has been limited to solving a *large-scale* local rearrangement problem, i.e. singulation where the behaviors of non-target objects (obstacles) are largely inconsequential, or a *small-scale* global rearrangement problem, such as separating a maximum of 12 objects [18] or repositioning 2 out of 6 objects max [43, 44]. We define a *local* rearrangement problem as one whose solution only requires the robot to interact with a small neighborhood around the object(s) of interest. Likewise, a rearrangement problem is *global* if the solution requires interaction with all objects in the scene. Historically, global problems are harder. For instance, Zeng et al. [116] could singulate 1 object out of 30 (1/30) from a random initial configuration but only 1/6 from a packed initial configuration. Table 3.1 provides an extensive list of prior results.

Our work introduces the first algorithm capable of solving large-scale global rearrangement planning problems. We validate our algorithm on rearrangement problems which contain up to 100 objects and packing factors of up to 40%. We believe this result is impressive (and surprising) for a number of reasons. First, the planner must reason about complex multi-object, multi-contact dynamics. Second, the problem is strongly non-monotonic. In the tightly packed environments we study, objects clump together and must be acted upon *en masse*. Finally, the search space is large. The continuous action space and large number of objects generates very high branching factors.

This chapter introduces Iterated Local Search (ILS) with an annealing ε-greedy policy as the first known algorithm for solving large-scale global rearrangement planning problems. The ILS algorithm consists of two components, the inner the local search, i.e. the annealing ε-greedy policy, and the outer loop with iterates over local

searches. The ε-greedy policy starts from the current state, and randomly selects and rolls out pushing actions. If the action is greedy, then the rollout terminates when the distance to goal is minimized. A non-greedy action would continue to some predetermined maximum. By repeating the random selection of mostly greedy actions, the policy rollout would produce a sequence of actions and a corresponding system trajectory. This local search would tend to descend the gradient, but might get stuck in a local minimum. We solve this issue using an iterated local search. Two departures are used to define the iterated local search. First, each local search selectively uses non-greedy actions – actions which will proceed a considerable distance without regard to distance-to-goal. These non-greedy actions are employed according to an annealing schedule. They appear frequently at the beginning of the local search, and less frequently later on. Second, the local search is repeated several times until a sequence of actions which improves the overall cost-to-goal is found. The result is a plan that might be kicked out of local minima several times but always moves closer to the goal.

Our results have effectively shown that a simple greedy heuristic search can solve non-prehensile rearrangement planning problems which are more difficult than those previously considered. Our algorithm and results suggest that this problem space is *approximately convex*. We believe this novel insight will be important for any future work in this domain. Moreover, our algorithm transfers from simulation to real-world experiments without any parameter tuning, uncertainty modeling, or learning.

The rest of this chapter is organized as follows. Section 3.2 discusses related work in more depth and quantitatively compares past results in Table 3.1. Section 3.3 provides a background for our algorithm by formally defining non-prehensile rearrangement problems and Markov Decision Processes. Section 3.4 describes our algorithm by first introducing ILS and then the annealing ε-greedy policy. Section 3.6 presents our simulated and real-world experimental results. Section 3.7 discusses the reasons behind our algorithm's performance. Section 3.8 reviews our algorithm's limitations and proposes future work, while Section 3.9 gives concluding remarks.

3.2 Related Work

Chang et al. [17] and Hermans et al. [45] both proposed a pushing policy for singulating objects based on visual boundary information. Laskey et al. [66] learned a deep grasping policy which pushes clutter away to achieve a grasp on a target object. King et al. [62, 63, 64] published a number of different planning methods which used pushing primitives to reposition a single object in the presence of limited clutter. Eitel et al. [31] learned a push proposal network for separating objects. Haustein et al.

[44] solve non-prehensile rearrangement planning problems by employing a variant of Rapidly-Exploring Random Trees (RRTs) augmented with a learned state generator and a learned policy (action generator). Similar to their previous work [43], Haustein et al. [44] restricted the pusher motion to lie in-plane, which likely makes their problems as difficult as NAMO[1]. Danielczuk et al. [25] proposed two novel deterministic pushing policies for singulating objects in a bin using only one push. Unfortunately, their results lacked any post-push grasping success rates. Yuan et al. [115] adopted Deep Q-Learning (DQN) to rearrangement planning, noting how both Atari games and tabletop pushing tasks are essentially 2D; however, their results were limited to repositioning 1 out of 2-4 objects. Muhayyuddin et al. [81] generated plans for non-prehensile reach-to-grasp tasks using Kinodynamic Motion Planning by Interior and Exterior Cell Exploration (KPIECE) with uncertainty propagation and safe motion biases. Dogar and Srinivasa [28] introduced push-grasping, i.e. singulating an object for grasping using non-prehensile actions, and described an action library approach to generating push-grasp trajectories. Agboh and Dogar [2] formulate push-grasping as an online stochastic trajectory optimization problem. Bejjani et al. [10] also applied Deep Q-Learning to rearrangement planning problems but were only able reposition 3 out of 3 objects. Anders et al. [6] placed and then pushed individual blocks one at a time into a packed configuration using forward search through in belief space with a learned transition model. Though they demonstrated the ability to rearrange objects into target positions and orientations, they were only able to handle a very specific type of goal configuration, i.e. a corner-pyramid configuration supported by the bottom and right walls. Zeng et al. [116] used Q-learning to train a fully convolutional network which labeled pixels as potential pushing or grasping actions locations. They tested on scenes with 30 randomly placed objects and scenes with 6 tightly-packed objects.

To the best of our knowledge, all related papers (see Table 3.1) only address a single category at a time. Moreover, no prior work has demonstrated the capability to arrange objects to arbitrary configurations or sort objects. Though Anders et al. [6] demonstrated arranging, they were only able to handle a very specific corner-pyramid configuration, whereas our algorithm can arrange blocks into any character or word (Section 3.6). This limitation indicates the lack of a *general* approach. With the sole exception of singulation problems, Table 3.1 shows that our algorithm outperforms all prior work with respect to the number of objects and packing factors they can handle.

[1]In our opinion, this restriction is unnecessary given that the target task is tabletop rearrangement

	max packing factor					max # objects	success
	singulate	separate	reposition	rearrange	sorting		
Chang et al. [17]	-	12/12†, 0.22*	-	-	-	12/12, 0.22*	98%
Hermans et al. [45]	1/6, 0.07*	-	-	-	-	1/6, 0.07*	72%
Laskey et al. [66]	1/4, 0.20*	-	-	-	-	1/4, 0.20*	65%
King et al. [62, 63, 64]	-	-	1/6‡, 0.09*	-	-	1/6‡, 0.09*	100%
Eitel et al. [31]	-	8/8, 0.10*	-	-	-	8/8, 0.10*	57%
Haustein et al. [43, 44]	-	-	2/6‡, 0.04*	-	-	2/6‡, 0.04*	99%
Danielczuk et al. [25]	1/10, 0.21*	-	-	-	-	1/10, 0.21	?%
Yuan et al. [115]	-	-	1/4, 0.02*	-	-	1/4, 0.02*	70%
Muhayyuddin et al. [81]	1/40‡, 0.25*	-	-	-	-	1/40‡, 0.25*	90%
Dogar et al. [28, 2, 10]	1/16‡, 0.17	-	3/3‡, 0.07*	-	-	1/16‡, 0.17	93%
Anders et al. [6]	-	-	-	9/9†, 0.20*	-	9/9†, 0.20*	96%
Zeng et al. [116]	**1/30**†, **0.47***	-	-	-	-	1/30†, 0.47*	81%
Our work	1/33, 0.41	**25/25**, **0.31**	**32/32**, **0.20**	**32/32**, **0.20**	**24/24**, **0.31**	**100/100**, **0.10**	91%

Table 3.1: Comparison of prior work with ours in non-prehensile rearrangement planning. Authors are ordered by latest publication date. All numbers are given in terms of the max achieved. †Grasping included as a action primitive. ‡Pusher motion restricted to be in-plane. *Packing factor estimated from images in paper.

3.3 Background

3.3.1 Push Planning for Rearrangement Tasks

We take the definition of table-top rearrangement planning from King et al. [62]. Let there be n (not necessarily unique) moveable objects on a flat surface. Assuming the objects do not roll or topple, the n objects form the state space $S = \mathbb{R}^{3n}$. Given a start state $s \in S$, the goal of table-top rearrangement planning is to find a sequence of non-prehensile actions $a_1, \ldots, a_n$ which rearrange the objects into any state g in the goal set $G \subset S$.

Similar to prior work, we restrict non-prehensile actions to pushes. We define a pushing action as the tuple $a = \langle h, v, t_v, greedy \rangle$, where $h \in \mathbb{R}^3$ is the pre-push pose of the pusher, $v \in \mathbb{R}^2$ is the pusher velocity (no rotation, i.e. a straight line push), t_v is the pushing duration, and $greedy \in \{0, 1\}$ determines whether the action is evaluated greedily or not. In practice, we also fix a maximum greedy distance d_g and a maximum random distance d_r at which to stop evaluating the action. We assume the pushing actions are evaluated in a quasi-static environment, i.e. inertial forces are negligible [78].

3.3.2 Markov Decision Processes

In this work, we model the rearrangement planning problem as a Markov Decision Process (MDP) with continuous state and action spaces [103]. Recall that an MDP $\mathcal{M}$ is described by a five-tuple $\langle S, A, T, R, \gamma \rangle$, where $S \subseteq \mathbb{R}^N$ is an N-dimensional state space, $A \subseteq \mathbb{R}^D$ is an D-dimensional action space, $T(s, a, s')$ is the probability that action a applied to state s will lead to state s' in the next time step, $R(s, a, s')$ is the immediate reward received after transitioning from state s to state s' due to action a, and $\gamma \in [0, 1]$ is the discount factor on future rewards. A non-deterministic policy $\pi(s, a)$ specifies the probability that the agent, or robot, chooses action a in state s. The expected value of a policy $V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t R_t$ is the discounted sum over the expected rewards starting at state s and following policy π. The solution to an MDP is an optimal policy π^* which maximizes $V^\pi(s), \forall s \in S$.

3.3.3 Policy Rollouts

A *policy rollout* generates a solution sequence $a_1, \ldots, a_n$ to an MDP $\langle S, A, T, R, \gamma \rangle$ with start s and goals G by sampling and rolling out actions from a policy $\pi(s, a)$ until a terminal state is reached. Not unexpected, the policies themselves determine

the effectiveness of this solution method. Thus, Subsection 3.4.1 describes a policy suitable for non-prehensile rearrangement planning.

3.4 Methods

3.4.1 ε-greedy Pushing Policy

This section adopts the classic ε-greedy policy formulation to continuous, non-prehensile action spaces [103]. Given a probability ε, an ε-greedy policy π_ε selects random actions with probability ε and greedy actions with probability $1-\varepsilon$. To extend π_ε to continuous action spaces, we have π_ε execute greedy actions using a *steering function* [67]. To increase sample efficiency, we have π_ε sample pushing actions from an *object-centric* action space [63].

Steering Functions

We adopted the steering function from a class of sample-based planning algorithms known as Rapidly-Exploring Random Trees (RRT) [67]. Let a be a greedy action and suppose taking action a at state s generates the trajectory $\mathcal{T}(t)$, $t \in [0, \tau]$. Given a goal set G, the steering function returns the state s' along the trajectory $\mathcal{T}(t)$ which minimizes the cost-to-goal, that is

$$s' = \arg\min_{x \in \mathcal{T}} \{\|x - g\| \mid g \in G\}. \tag{3.1}$$

Greedy execution using a steering function means action a is only applied up until we reach state s', or up to time $\mathcal{T}^{-1}(s')$. If a is random (not greedy) then the action is taken in full, i.e. $s' = \mathcal{T}(\tau)$. Note, examples of distance functions are given in Subsections 3.5.1 and 3.5.2.

Object-Centric Sampling

An action is said to be object-centric if it interacts in a targeted way with a single object [63]. In this work, we sample an object-centric action in two phases. First, select an object not at its goal, i.e. $\|p-g\| > \delta$, where p is the object pose, g is the goal pose, and δ is the goal tolerance. Next, we sample a pusher velocity v from a velocity set $V \subseteq \mathbb{R}^2$ and a collision-free pre-push pose h such that the pusher motion from h with velocity v intersects the selected object. Furthermore, if the action is greedy,

Algorithm 2 Iterated Local Search

```
1: function ITERATED-LOCAL-SEARCH(s, g)
2:   x ← s
3:   T ← [ ]
4:   for i ∈ RANGE(N_max) do
5:     T' ← LOCAL-SEARCH(x, g)
6:     if DIST(T') < DIST(T) then
7:       T ← T ∪ T'
8:       x ← FINAL-STATE(T)
9:     if TERMINAL(x) then
10:      return T
11:  return T
```

we sample velocities from the positive half-space $V^+ = \{v | v \cdot (g - x) \geq 0, v \in V\}$ instead of V (for efficiency).

Examples of V include $V_{com} = \{v \,|\, \|v\| = 1\}$, the set of unit velocities which pass through the object center, $V_{NESW} = \{(\cos\theta, \sin\theta) \,|\, \theta = n\pi/2, n \in \mathbb{Z}\}$, the set of cardinal directions, and $V_{OCT} = \{(\cos\theta, \sin\theta) \,|\, \theta = n\pi/4, n \in \mathbb{Z}\}$, the set of principal directions.

3.4.2 Iterated Local Search

The Iterated Local Search (ILS) meta-algorithm iteratively builds a sequence of solutions generated by the embedded local search heuristic [71]. At each iteration, ILS runs the heuristic once and saves the returned sequence only if it reduces the cost to goal. Algorithm 2 lists ILS pseudo-code.

In this work, we use an annealing ε-greedy policy rollout as the embedded heuristic. We schedule ε according to a temperature-controlled acceptance function (lines 5 and 6). This raises the chance of a non-greedy action significantly when the number of search iterations is very small, which helps kick the system out of local minima.

3.5 Implementation

Recall that the steering function applies an action a up to the point where a distance function between the rolled out state and the goal is minimized. Subsections (3.5.1 and 3.5.2) describe the two distance functions used in our algorithm.

Algorithm 3 Annealing ε-greedy Policy Rollouts

function LOCAL-SEARCH(s, g)
  $x \leftarrow s$
  $\mathcal{T} \leftarrow [\,]$
  for $i \in$ RANGE(N_{search}) **do**
    $\rho \leftarrow \rho_0 / i$
    $\varepsilon \leftarrow 1 \,/\, \left(1 + \exp \frac{1}{\rho}\right)$
    $a \leftarrow \pi_\varepsilon(x)$
    $\mathcal{T} \leftarrow \mathcal{T} \cup$ ROLLOUT(x, a)
    $x \leftarrow$ FINAL-STATE$(\mathcal{T})$
    if TERMINAL(x) **then**
      return $\mathcal{T}$
  return $\mathcal{T}$

3.5.1 Weighted Euclidean Distance Function

Given a set of object poses $p_1, \ldots, p_n$, corresponding goal poses $g_1, \ldots, g_n$, and non-negative weights $w_1, \ldots, w_n$, the weighted Euclidean distance function returns

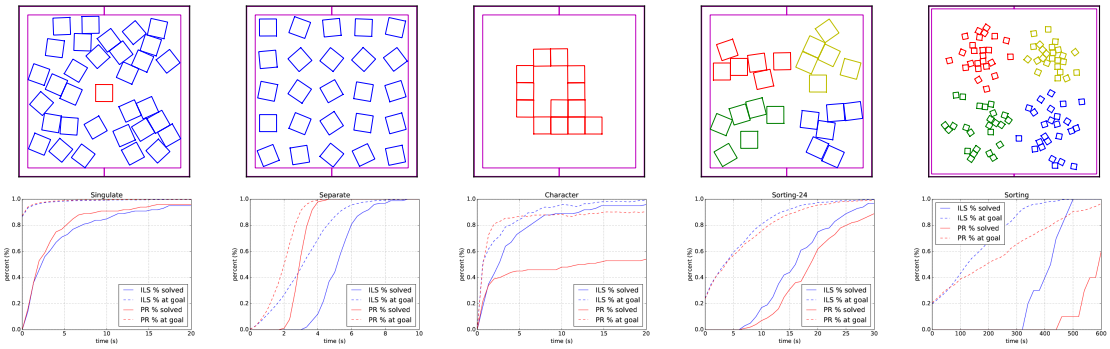
$$Dist = \sum_{i=1}^{n} \|p_i - g_i\|_{w_i}. \tag{3.2}$$

Note, we first wrap the angular component of p_i and g_i to be within $[0, 2\pi/m_i)$, where m_i is the number of radial symmetries of object i.

3.5.2 Linear Assignment Distance Function

Suppose that some objects are not unique. Because non-unique objects can be assigned to their corresponding goals in any permutation, we must modify the distance function to additionally solve the *assignment problem* [59].

Let each set of identical objects O_k be assigned an index k. Then we can write the set of object poses and goal poses for O_k as $p_1^k, \ldots, p_{n_k}^k$ and $g_1^k, \ldots, g_{n_k}^k$, respectively, where $n_k = \|O_k\|$. Let a $n \times n$ cost matrix C be comprised of the weighted Euclidean distances between all pairs of object poses p_i^k and goals g_j^k. The linear assignment problem is to find an bijective assignment $A_k : \{1, \ldots, n_k\} \to \{1, \ldots, n_k\}$ such that



(a) Singulate (b) Separate (c) Character (d) Sorting-24 (e) Sorting-100

Figure 3-2: Top: Example final states for each problem. Bottom: Plots of the percentage of problems solved over time for ILS (solid-blue) and ε-greedy policy rollouts (solid-red) and the percentage of objects at goal over time for ILS (dotted-blue) and ε-greedy policy rollouts (dotted-red).

the cost function

$$\sum_{i=1}^{n_k} C_{i,A_k(i)} \tag{3.3}$$

is minimized. This problem can be solved in $O(n^3)$ time using the Jonker-Volgenant algorithm [59]. The total distance across all sets of identical objects is given by

$$\sum_{k=1}^{n_o} \sum_{i=1}^{n_k} C_{i,A_k(i)}, \tag{3.4}$$

where n_o is the number of object sets.

3.6 Experiments

3.6.1 Simulation Problem Setups

We used the following simulation environment setup in our experiments. The objects consisted of 4×4 colored blocks. The workspace was restricted to a 40×40 square with a virtual out-of-bounds region with thickness 2. We introduced the virtual out-of-bounds region to give the robot enough space to push blocks out from edges and corners. For an example workspace, see Figure 3-1. Unless otherwise specified, the robot was equipped with a 3×0.5 fence pusher. All algorithms were run on a

computer with an Intel i7-7820x CPU (3.5 MHz, 16 threads). The simulation environment was implemented in Box2D [16]. Note, we have limited our presented results to the hardest problems in each category. We have also validated our algorithm on smaller versions of these problems.

Singulate

This problem required the robot to singulate the block nearest to the center of the work space away from the 32 other blocks (total 33 blocks). For the target block, we used a weighted euclidean distance function with goal $g_t = [0, 0, 0]$, weights $w_t = [1, 1, 0]$, and goal tolerance $\delta_t = 0.5$. For all other blocks, we used a linear assignment distance function with $g_i \in [\pm 9, \pm 9, 0]$, $w_i = [1, 1, 0]$, and $\delta_i = 9$. Packing factor ≈ 0.41.

Separate

This problem required the robot to separate 25 blocks into a 5×5 grid. We used a linear assignment distance function with $g_{i,j} = [15 - 7.5i, 15 - 7.5j, 0]$, $w_{i,j} = [1, 1, 0]$, $\delta_{i,j} = 0.1$, and $i, j \in [0, 5)$. The robot was equipped with a 0.5×0.5 square pusher to help it squeeze between tightly packed blocks.

Character

This problem required the robot to rearrange 4×4-sized blocks into characters from the alphabet. The number of blocks in each character ranged from 3 to 13. We used a linear assignment distance function with g_i as the i-th block's location in the character, $w_i = [1, 1, 5]$, and $\delta_i = 0.1$. Max packing factor ≈ 0.16.

Sorting-24

This problem required the robot to sort 4 sets of 6 blocks by color (red, blue, yellow, and green). We used a weighted Euclidean distance function for each color c with each goal g_c being one of $[\pm 9, \pm 9, 0]$, $w_c = [1, 1, 0]$, and $\delta_c = 9$. Packing factor ≈ 0.30.

Sorting-100

This problem is the same as *Sorting-24* except with 25 blocks in each set (total 100), a workspace of size 125×125, $g_c \in [\pm 30.25, \pm 30.25, 0]$, and $\delta_c = 30.25$. Packing factor $\approx 10\%$.

3.6.2 Simulation Results

Both ILS and ε-greedy policy rollouts were used to solve the above problems. Both algorithms used the velocity set V_{NESW}, as we found it to have good performance across all problems. ILS used an initial temperature of $\rho_0 = 1.7$ for all problems. The following parameters are given in the order of problem numbering. For both algorithms, we set $d_g = 20, 20, 20, 20, 50$ and $d_r = 8, 4, 4, 4, 25$. For ε-greedy policy rollouts, we set ε to 0.85, 0.999, 0.999, 0.85, and 0.75. We set time limits for both algorithms to 20 s, 20 s, 20 s, 30 s, and 600 s, and ran trials of size 100, 100, 100, 100, and 10. We collected our results in Tables 3.2 and 3.3.

	# objects, pf	% objects at goal	successes
Singulate	1/33, 0.41	99.73%	$95\% \pm 2\%$
Separate	25/25, 0.31	100%	$100\% \pm 0\%$
Character	13/13, 0.16	99.42%	$\mathbf{96}\% \pm \mathbf{2}\%$
Sorting-24	24/24, 0.30	99.66%	$\mathbf{97}\% \pm \mathbf{2}\%$
Sorting-100	100/100, 0.10	100%	$\mathbf{100}\% \pm \mathbf{0}\%$

Table 3.2: ILS results

	# objects, pf	% objects at goal	successes
Singulate	1/33, 0.41	99.81%	$96\% \pm 2\%$
Separate	25/25, 0.31	100%	$100\% \pm 0\%$
Character	13/13, 0.16	90.56%	$54\% \pm 5\%$
Sorting-24	24/24, 0.30	95.585%	$89\% \pm 3\%$
Sorting-100	100/100, 0.10	96.7%	$60\% \pm 5\%$

Table 3.3: ε-greedy results

We observed that ILS statistically outperforms ε-greedy policy rollouts on *Character*, *Sorting-24*, and *Sorting-100*. *Character* problems required a large number of actions to solve, which can make the wrong non-greedy action very costly. The main advantage of ILS over ε-greedy is that ILS can erase mistakes by throwing away bad sequences. This feature also translates into better search efficiency, which explains the higher success rate in *Sorting-100*. ILS uses the time saved from undoing mistakes to search for better actions.

3.6.3 Real-World Problem Setups

We used the following hardware setup in our physical experiments. The objects consisted of 25.4 mm (1 inch) square blocks of various colors. The workspace was restricted to a 420 mm by 310 mm rectangle with a virtual out-of-bounds region of thickness 20 mm. The workspace surface was made from transparent, scratch resistant acrylic so that the objects could be tracked using AprilTags [112] from underneath. The robot was equipped with a 19.05 mm flat fence pusher (Figure 3-3a). The pusher was mounted on a linear potentiometer with spring return. The potentiometer was used to detect contact of the fence pusher against the top of the blocks, typically resulting from errors in sensing. On top-contact, the pusher was repositioned 3 mm behind and the action retried. We tested our algorithm on the following problems.

Sorting-32

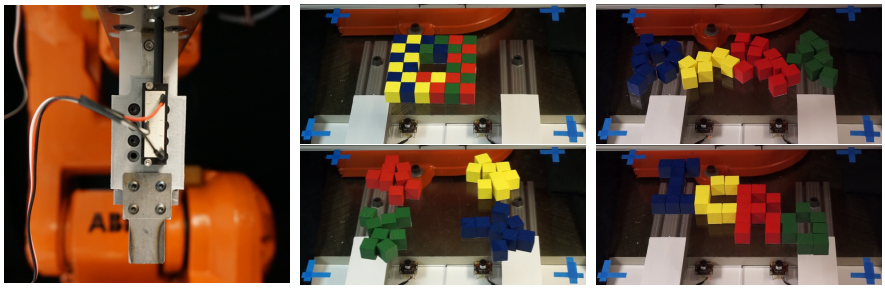
The goal of this problem is to sort 4 sets of 8 blocks by color. We used a weighted Euclidean distance function with $g_c \in [\pm 95\,\text{mm}, -495\,\text{mm} \pm 67.5\,\text{mm}, 0]$, $w_c = [1, 1, 0]$, and $\delta_c = 60\,\text{mm}$. Similar to [116], the blocks are adversarially packed to increase the problem difficulty. Initial and final configurations are shown in Figure 3-3b.

Word-IcRa

The goal of this problem is to simultaneously rearrange 9 blue, 6 yellow, 10 red, and 7 green blocks into the letters "IcRa", respectively. We used a linear assignment distance function for each letter with g_i^k as the i-th block's location in the k-th letter, $w_i^k = [1, 1, 31.75]$, and $\delta_i^k = 3$. We mixed upper/lower case letters due to constraints on the workspace and total available numbers of blocks of each color. Initial and final configurations are shown in Figure 3-3c.

3.6.4 Real-World Results

The ε-greedy pushing policy was used to solve the *Sorting-32* and *Word-IcRa* problems 5 times each. We did not use ILS in our real world experiments because executing an open-loop sequence of actions quickly leads to divergent states. We propose a method to overcome this difficulty in Section 3.8. For both problems, execution time was capped to 30 minutes. We set ε to 0.75 and 0.9999, $d_g = 100, 100$, $d_r = 50, 25$, and the velocity set to V_{NESW}. Our results are presented in Table 3.4.



(a) Fence pusher (b) Sorting-32 (c) Word-IcRa

Figure 3-3: Images of our experimental setup for physical experiments. (a) Fence pusher with linear potentiometer. (b) Initial and final configurations for the *Sorting-32* problem. (c) Initial and final configurations for the *Word-IcRa* problem. The blue corner tape in (b) and (c) denote the workspace limits of the robot.

	# objects, pf	% objects at goal	successes
Sorting-32	32/32, 0.20	100%	5/5
Word-IcRa	32/32, 0.20	98.75%	3/5

Table 3.4: Real-world experimental results

The ε-greedy policy failed twice on the *Word-ICRA* problem. In one failure, the policy cycled between dislodging a yellow block from the counter-space (hole) in "R" and trapping the yellow block back in "R". In the other failure, the policy alternated between fixing the legs of "R" and fixing the counter-space of "a".

3.7 Discussion

3.7.1 Simulation to Reality

Notably, we transferred the pushing policies from simulation to the real world without *any* tuning or system identification. We hope the following discussion of pushing dynamics can elucidate why this works. By using a fence pusher, our system is designed to take advantage of two-point stable pushing [73]. Moreover, the turning direction easily maps from simulation to reality given low measurement error. Lastly, running our algorithm online allows the robot to iteratively correct errors between simulation and reality.

3.7.2 Parameter Selection

In our experience, the most effective method for tuning parameters consisted of visualizing the difficult scenarios and then tweaking the various algorithmic components to raise the probability of overcoming these scenarios. For instance, we observed that longer fence pushers, e.g. length 8, struggled to separate blocks because the pusher tended to make sticking contact with more than one block. Using a smaller pusher with length 3 opened up the action space significantly more. During sorting, we observed off-color blocks would get trapped in a group of similarly colored blocks. To free the off-color block, we had to make sure the policy could sample non-greedy actions for the off-color block which started from the other side of the on-color blocks. The character rearrangement problem required long sequences of small corrections, hence the high ε and low d_r.

3.7.3 Computational Complexity

We conjecture that NP-complete sliding puzzles, such as the n-puzzle, can be reduced to an instance of non-prehensile rearrangement [96]. While we abstain from a proof here, this conjecture suggests the following parallels. Like the n-puzzle, non-prehensile rearrangement planning problems can be solved in polynomial time; however, finding an optimal solution is NP-hard [96].

3.8 Limitations & Future Work

We did not validate our algorithm on problems with non-convex objects as this work was primarily intended to demonstrate large-scale global rearrangement. In addition, the presented algorithm does not provide any theoretical guarantees on probabilistic completeness or solvability.

3.9 Conclusion

This chapter provided algorithmic insight into the nature of pushing multiple objects in clutter. Specifically, we demonstrated that policy rollouts with a greedy action space are sufficient for push planning on table-top rearrangement tasks. We also showed that using an iterated local search technique can help escape local minima and improve results. We successfully applied this algorithm to singulating, separating, arranging, and sorting large-scale clutter in simulated and physical experiments.

Chapter 4

Autogenerated Manipulation Primitives I

4.1 Introduction

When a moving robot contacts its environment, the points of the resulting contact manifold may be sliding, sticking, or separating. Under the Coulomb model of friction, the frictional force is either opposite to the velocity of a sliding contact point, oriented in any direction for a sticking contact point, or zero for a separating contact point. As each individual contact mode imparts complementary dynamic equations, the set of all (valid) contact mode assignments enumerates the set of all possible flows for the system [58]. (Here, by valid we mean kinematically feasible, i.e. there exists a generalized velocity $\dot{q}$ that generates the correct mode at each contact point.) Given the one-to-one mapping between contact modes and dynamics, we advocate that efficient contact mode enumeration will be a useful tool for the simulation, analysis, and control of robotic systems that make and break contact with the environment.

Our main contribution is to report the first efficient algorithm for contact mode enumeration in 3D which is exponential in the number of degrees of freedoms of the system and polynomial in the number of contact points. Letting d and n be those two numbers, respectively, then our algorithm enumerates all feasible contact modes in $O(n^d)$ time. By efficient, we mean the first algorithm that is polynomial in the number of contact points. The exponent in d is unavoidable (see Section 4.8 for an example). Our algorithm can enumerate all of the following:

- contact modes that are contacting or separating,
- contact modes that are sticking or sliding in some set of directions,

- contact modes for linear and nonlinear friction cones,
- contact modes involving multiple objects.

In this chapter, we address the contact mode enumeration problem as a combinatorial geometry problem, and the key concern is to compute the combinatorial structure of convex hulls and hyperplane arrangements. Leveraging the rigid body kinematics, our method performs faster enumeration by dividing the problem into two steps: contacting/separating enumeration and sliding/sticking enumeration. The results obtained after the two steps are regions of valid object motions and their corresponding contact modes.

4.2 Related Work

This section discusses related work specific to contact mode enumeration and our algorithm (for related work with respect to application areas refer to Section 4.9). Mason [79] sketched an algorithm for contact mode enumeration in 2D for a single rigid body which intersects the positive (negative) rotation centers on the positive (negative) oriented plane and intersects the rotation centers at infinity on the equator. Though Mason [79] upper-bounded the number of modes at $O(n^2)$, by our analysis, the algorithm's runtime is actually $O(n \log n)$ and the correct number of modes is $\Theta(n)$. Unfortunately, the oriented plane technique does not generalize to contact mode enumeration in 3D. Later, Haas-Heger et al. [41] independently published an algorithm for partial contact mode enumeration in 2D. There, they interpret the feasible modes as the regions of an arrangement of hyperplanes in 3D. However, Haas-Heger et al. [41]'s algorithm is at least $\Omega(n^4)$ and does not enumerate separating modes. Disregarding these issues, Haas-Heger et al. [41]'s work inspired us to investigate hyperplane arrangements in higher dimensions for our algorithm. To the best of our knowledge, our algorithm is the first method for contact mode enumeration in 3D.

The existence of an efficient contact mode enumeration algorithm in 2D does not appear to wide-spread knowledge. For instance, Greenfield et al. [39] used the exponential time algorithm for contact mode enumeration in 2D. Johnson and Koditschek [57] published a optimization-based technique for legged robot leaping which searched along the contacting-separating mode graph for a sequence of foot takeoffs. Their application was sufficiently simple that they could enumerate the modes by hand. Recently, Hou et al. [51] published a controller for robotic manipulation which achieves robustness by analyzing the adjacent contact modes. It used a variety of metrics to

determine the most stable (contact) mode of execution. Recently, Cheng et al. have developed an algorithm for contact-mode guided planning for robotic manipulation. The contact modes serve to enumerate the different possible ways a robot can move the object at a given state, thereby removing the need to manually construct manipulation primitives such as grasping, pushing, pulling, or flipping etc. [22]. The main goal of this work is to enable further development of analysis, planning, and control algorithms based on contact modes.

It is well known that frictional contact problems can be modeled as a complementarity problem or equivalently, a variational inequality [35]. Within that theory, it is known that the normal manifold (which is a linear hyperplane arrangement) divides the solution space of an affine variational inequality [35]. Not surprisingly, we found related papers in other fields containing problems that can be modeled as variational inequalities [37, 94]. For example, in the study of digital controllers and power electronics, Geyer et al. [37] proposed a mode enumeration algorithm for compositional hybrid systems based on the reverse search technique of Avis and Fukuda [7]. The contribution of our work (and theirs) is in presenting the theory in an understandable manner for our field and optimizing the relevant algorithms for our specific problem formulation.

4.3 Background

This chapter establishes the background required to understand contact modes. Section 4.3.1 covers the kinematics of contact. Section 4.3.2 reviews the geometry of hyperplanes. This topic includes convex polyhedra, hyperplane arrangements, face lattices, and other important geometric objects used on our enumeration algorithm. Section 4.3.3 covers basic ideas from matroid theory, which will be used to preprocess the contact constraints.

4.3.1 Contact Kinematics

This section reviews the theory behind contact kinematics. We introduce the normal and tangent velocity constraints generated by rigid contacts and describe how they partition the space of generalized velocities into discrete contact modes. To concisely explain these concepts, we assume the reader is familiar with spatial transforms and robot kinematics [82]. The concepts in this section are illustrated in Figure 4-6.

Normal Velocity Constraints

In a rigid body model of the world, two rigid bodies in collision cannot penetrate one another. To a first-order approximation, this generates a linear constraint on their relative velocities with respect to the contact normal.

This normal velocity constraint can be derived as follows. Let α and β be two rigid bodies in collision at a point $c \in \mathbb{R}^3$. Let $g_{wc} \in SE(3)$ be the contact frame at a contact point c with z-axis pointing along β's surface normal towards α. Let $\phi \in \mathbb{R}_{\geq 0}$ be the contact distance as measured along the contact normal. Taking the derivative of ϕ, we obtain the constraint $\phi + \dot{\phi} \geq 0$. We can rewrite this in terms of the generalized (system) velocity $\dot{q} \in \mathbb{R}^d$

$$\phi + B^T(J^b_{\alpha c} + J^b_{\beta c})\dot{q} \geq 0, \tag{4.1}$$

where B is the wrench basis $[0, 0, 1, 0, 0, 0]^T$, $J^b_{\alpha c} \in \mathbb{R}^{6\times d}$ is the body Jacobian of α to the contact frame, and $J^b_{\beta c} \in \mathbb{R}^{6\times d}$ is the body Jacobian of β to the contact frame. We define a contact to be *separating* if (4.1) is positive and *contacting* if (4.1) is equal to zero. We refer to this classification as the *contacting-separating mode* (cs-mode) at that contact. For a system with n contacts, we can concatenate (4.1) into a set of linear inequalities

$$N\dot{q} \leq \phi, \quad N \in \mathbb{R}^{n\times d}. \tag{4.2}$$

Tangent Velocity Constraints

Coulomb friction is a simple model of dry friction which characterizes frictional forces based on the relative velocity of the contact point. The tangent velocity constraints are a set of hyperplanes which help us classify the relative velocity of the contact point.

We approximate the infinite tangent velocity directions by dividing the tangent plane into sectors of equal angles. Let k be the number of dividing planes (which generate $2k$ sectors). We define a basis matrix D such that its i-th column equals $[\cos\frac{i\pi}{k}, \sin\frac{i\pi}{k}, 0, 0, 0, 0]^T$. Given a generalized velocity $\dot{q}$, we can use the following equation to determine the discretized tangent velocity direction

$$D^T(J^b_{\alpha c} + J^b_{\beta c})\dot{q}, \tag{4.3}$$

where $J^b_{\alpha c}$ and $J^b_{\beta c}$ are the body Jacobians from before. We define a contact to be *right-sliding* with respect to a tangent direction if its row in (4.3) is positive, *left-sliding* if its row is negative, and *sticking* if all rows are zero. We refer to this classification as the *sliding-sticking mode* (ss-mode) at that contact. For a system

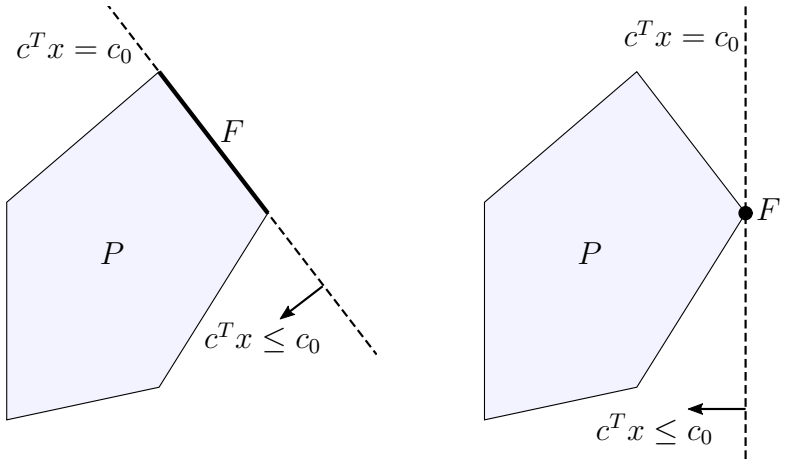


Figure 4-1: Examples of polytope faces.

with n contacts, we can concatenate (4.3) into the tangent velocity constraints

$$T\dot{q}, \quad T \in \mathbb{R}^{(nk)\times d}. \tag{4.4}$$

4.3.2 Hyperplane Geometry

This section covers the background necessary for understanding convex polytopes, hyperplane arrangements, and face lattices. The former two geometric structures describe the topology of the space partitioned by the normal velocity and tangent velocity constraints. The latter describes the combinatorial structure of the contact mode graph. For a more complete introduction, one can refer to [121] and [30]. We highly recommend looking at Figure 4-6 while reading this section.

Convex Polyhedra

Convex polyhedra describe the region of space enclosed by normal velocity constraints. A *hyperplane* $h = (a, z)$ is the set $h = (a, z) = \{x \in \mathbb{R} : a^T x = z\}$. Associated with each hyperplane h are the positive and negative *halfspaces*, h^+ and h^-. An $\mathcal{H}$*-polyhedron* $P \subseteq \mathbb{R}^d$ is the intersection of closed negative halfspaces in the form $P = \mathcal{H}(A, z) = \{x \in \mathbb{R}^d : Ax \leq z\}$, for some $A \in \mathbb{R}^{n\times d}, z \in \mathbb{R}^n$. A $\mathcal{V}$*-polytope* $P \subseteq \mathbb{R}^d$ is a convex combination of points, i.e. $P = \mathcal{V}(A) = \{At : t \geq 0, \sum t = 1\}$ for some $A \in \mathbb{R}^{n\times d}$. A *face* F of a polyhedron P is any set of the form $F = \{x \in P : c^T x = c_0\}$ with $c \in \mathbb{R}^d, c_0 \in \mathbb{R}$, where $c^T x \leq c_0$ is true for all $x \in P$. The *sign* of a

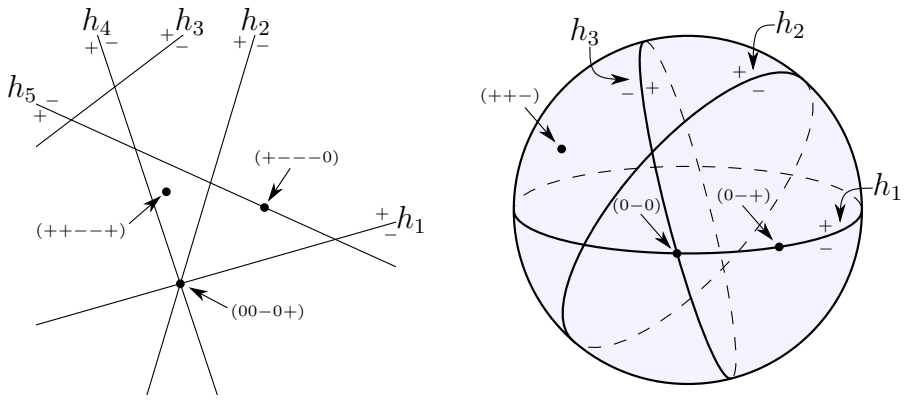


Figure 4-2: Examples of a hyperplane arrangement in $\mathbb{R}^2$ and a linear hyperplane arrangement in $\mathbb{R}^3$ (visualized on a sphere). Some faces are labeled with their signed vectors.

face $F \subseteq \mathbb{R}^d$ with respect to a hyperplane h of P is defined as

$$\text{sign}_h(F) = \begin{cases} + & \text{if} \quad F \subseteq h^+ \\ 0 & \text{if} \quad F \subseteq h \\ - & \text{if} \quad F \subseteq h^- \end{cases} \tag{4.5}$$

and the *signed vector* of a face F with respect to P is the vector

$$\text{sign}(F) = \begin{bmatrix} \text{sign}_{h_1}(F) & \cdots & \text{sign}_{h_n}(F) \end{bmatrix},$$

where h_i is the i-th hyperplane of P.

Hyperplane Arrangements

Hyperplane arrangements describe the regions of space partitioned by tangent velocity constraints. A *hyperplane arrangement* $\mathcal{A}(H)$ is a set of hyperplanes $H = (A, z)$, for some $A \in \mathbb{R}^{n \times d}, z \in \mathbb{R}^n$ which dissect $\mathbb{R}^d$ into different regions of space. The arrangement is *linear* when $z = 0$. The *signed vector* of a point p with respect to a hyperplane arrangement $\mathcal{A}(H)$ is $\text{sign}_{\mathcal{A}}(p) = \begin{bmatrix} \text{sign}_{h_1}(p) & \cdots & \text{sign}_{h_n}(p) \end{bmatrix}$. The signed vectors of the points in $\mathbb{R}^d$ define equivalence classes known as the *faces* of $\mathcal{A}$. That is, given a signed vector $s \in \{+, 0, -\}^n$, the associated face is $F = \{p \in \mathbb{R}^d : \text{sign}_{\mathcal{A}}(p) = s\}$.

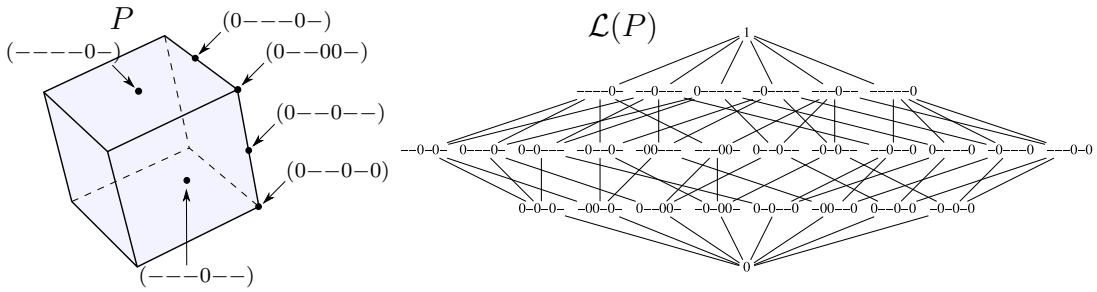


Figure 4-3: Left: Some signed vectors of a cube. Right: The face lattice of the cube.

Face Lattice

The face lattice describes the hierarchical structure of the faces of convex polyhedra and hyperplane arrangements. Consequently, it describes the hierarchical structure of contact modes and their adjacencies, i.e. it is the contact mode graph. A *partially ordered set* is a set L and a binary relation $\leq$ such that for all $u, v, w \in L$

$$\begin{aligned} &u \leq u && \text{(reflexivity)} \\ &u \leq v \wedge v \leq w \Rightarrow u \leq w && \text{(transitivity)} \\ &u \leq v \wedge v \leq w \Rightarrow u \leq w && \text{(anti-symmetry).} \end{aligned}$$

Moreover, a pair of elements u, v are comparable if $u \leq v$ or $v \leq u$; otherwise, they are incomparable. The faces of a polyhedra or arrangement form a partially ordered set over their signed vectors. Signed vectors u, v satisfy $u \leq v$ if and only if $u_i \leq v_i$, for all indices i, where individual signs are compared according to $0 < +$, $0 < -$, and $+$ and $-$ incomparable. A *lattice* is a partially ordered set $\mathcal{L}$ such that for any elements $x, y \in \mathcal{L}$, there exists elements $x \wedge y$ and $x \vee y$ in $\mathcal{L}$ satisfying

$$x \wedge y \geq x, y \text{ and if } z \geq x, y \text{ then } z \geq x \wedge y \tag{4.6}$$
$$x \vee y \leq x, y \text{ and if } z \leq x, y \text{ then } z \leq x \wedge y. \tag{4.7}$$

The faces of a polyhedra or arrangement in $\mathcal{R}^d$ form a lattice $\mathcal{L}$ known as the *face lattice*. The face lattice is bounded by unique minimal and maximal elements which we denote $\{0\}$ and $\{1\}$, respectively. The face lattice $\mathcal{L}$ contains $d+1$ proper *ranks*. The k-th rank contains the faces of dimension k. The dimension of a face is defined as the dimension of its affine hull. The faces of dimensions 0, 1, $d-2$, and $d-1$ are called *vertices*, *edges*, *ridges*, and *facets*, respectively.

Polar Polytope

When designing computational geometry algorithms, it is important to base the algorithm on the appropriate mathematical object. The *polar transformation* allows us to easily convert between combinatorially equivalent $\mathcal{H}$-polytopes and $\mathcal{V}$-polytopes. Without loss of generality, let $P \subseteq \mathbb{R}^d$ be a polytope with $0 \in P$. Its *polar polytope* $P^\triangle \subseteq \mathbb{R}^d$ is the set

$$P^\triangle = \{c \in \mathbb{R}^d : c^T x \leq 1, \forall x \in P\}. \tag{4.8}$$

The polar polytope can be specified in closed form for $\mathcal{H}$ and $\mathcal{V}$ polytopes. If P is a $\mathcal{V}$-polytope with $0 \in \text{int}(P)$ and $P = \mathcal{V}(A)$ then

$$P^\triangle = \mathcal{H}(A, 1) = \{x : Ax \leq 1\}. \tag{4.9}$$

If P is a $\mathcal{H}$-polytope with $0 \in \text{int}(P)$ and $P = \mathcal{H}(A, 1)$ then

$$P^\triangle = \mathcal{V}(A^T) = \{A^T t : t \geq 0, \sum t = 1\}. \tag{4.10}$$

Polar polytopes are useful because P and P^Δ share the same combinatorial structure. Specifically, the face lattice of the polar polytope $P^\triangle$ is the opposite of the face lattice of P:

$$L(P^\triangle) = L(P)^{op} \tag{4.11}$$

and there is a bijection between the faces

$$\begin{aligned} \emptyset &\longleftrightarrow P \\ vertices &\longleftrightarrow facets \\ edges &\longleftrightarrow ridges \\ ... &\longleftrightarrow ... \end{aligned} \tag{4.12}$$

Zonotopes

Zonotopes are a special type of convex polytope which are combinatorially equivalent to linear hyperplane arrangements. Recall that the *Minkowski sum* of sets X and Y is given by $X \oplus Y = \{x + y : x \in X, y \in Y\}$. We can define a zonotope as the Minkowski sum of a set of line segments

$$Z(V) = [-v_1, v_1] \oplus \cdots \oplus [-v_k, v_k], \tag{4.13}$$

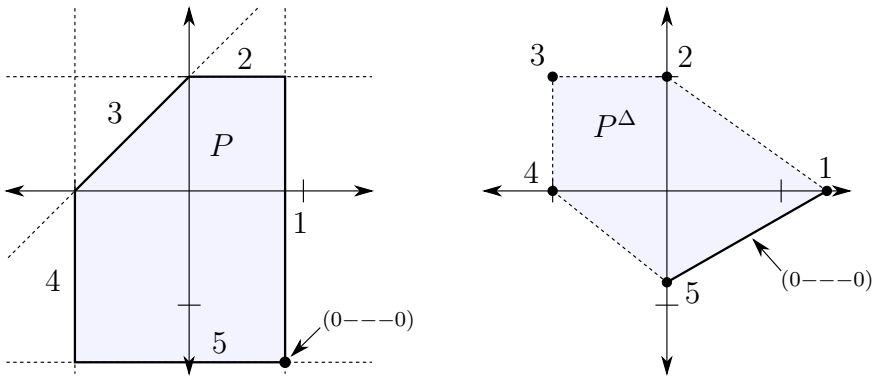


Figure 4-4: Example of an $\mathcal{H}$-polytope P and its polar $\mathcal{V}$-polytope $P^{\triangle}$.

where $V = [v_1, \ldots, v_k] \in \mathbb{R}^{d \times k}$. We can map each face F of a zonotope to an unique signed vector. Let $p = \sum \lambda_i v_i \in \operatorname{int} F$ be an interior point of F. Then the signed vector with respect to v_i is

$$\operatorname{sign}_{v_i}(F) = \begin{cases} +1 & \text{if} \quad \lambda_i = +1 \\ 0 & \text{if} \quad -1 < \lambda_i < 1 \\ -1 & \text{if} \quad \lambda_i = -1, \end{cases} \tag{4.14}$$

and the signed vector is $\operatorname{sign}_Z(F) = \big[\operatorname{sign}_{v_1}(F), \ \ldots, \ \operatorname{sign}_{v_k}(F)\big]$. From Corollary 7.17 in Ziegler [121], there is a bijection between the signed vectors of $\mathcal{A}(V)$ and $Z(V)$. We have the identification of face lattices

$$L(Z(V)) \longleftrightarrow L(Z(V)^{\triangle}) \longleftrightarrow L(\mathcal{A}(V)). \tag{4.15}$$

For example, there is a correspondence between the facets of $Z(V)$, the vertices of $Z(V)^{\triangle}$, and the rays (unbounded edges) of $\mathcal{A}(V)$.

4.3.3 Matroid Theory

This section provides the pertinent details of matroid theory which will help us preprocess the input normal and tangent velocity constraints. Specifically, it will help us remove degenerate hyperplanes and reduce problem dimensionality. A *matroid* is the pair $(E, \mathcal{I})$ of a finite set E and a collection $\mathcal{I}$ of subsets of E such that

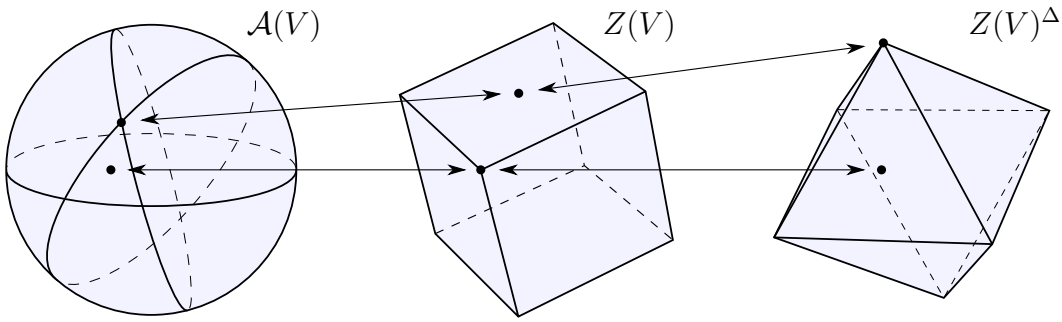


Figure 4-5: Example of a combinatorially equivalent linear hyperplane arrangement, zonotope, and polar zonotope.

(I1) $\emptyset \in \mathcal{I}$.

(I2) If $I \in \mathcal{I}$ and $I' \subseteq I$, then $I' \in \mathcal{I}$.

(I3) If I_1 and I_2 are in $\mathcal{I}$ and $|I_1| < |I_2|$, then there is an element e of $I_2 - I_1$ such that $I_1 \cup e \in \mathcal{I}$.

Let $A \in \mathbb{R}^{d \times n}$ be a matrix. Let E be the set of column vector indices of A and $\mathcal{I}$ the collection of subsets $I \subseteq E$ such that the corresponding vectors in A are linearly independent. This matroid $M[A]$ is known as a *vector matroid*. The *signed covectors* of $M[A]$ are the elements of the set $\mathcal{V}^*(A) = \{\text{sign}(c^T A) : c \in \mathbb{R}^d\}$. The signed covectors are one of several sets of data which uniquely define a matroid. If we interpret c as a point in $\mathbb{R}^d$, then the signed covectors of $M[A]$ are the signed vectors of the arrangement $\mathcal{A}(A^T)$.

The matroid $M[A]$ is unchanged if one performs any of the following operations on A.

(E1) Interchange two rows.

(E2) Multiply a row by a non-zero member of $\mathbb{R}$.

(E3) Replace a row by the sum of that row and another.

(E4) Adjoin or remove a zero row.

(E5) Interchange two columns (moving their labels with their columns).

(E6) Multiply a column by a non-zero member of $\mathbb{R}$.

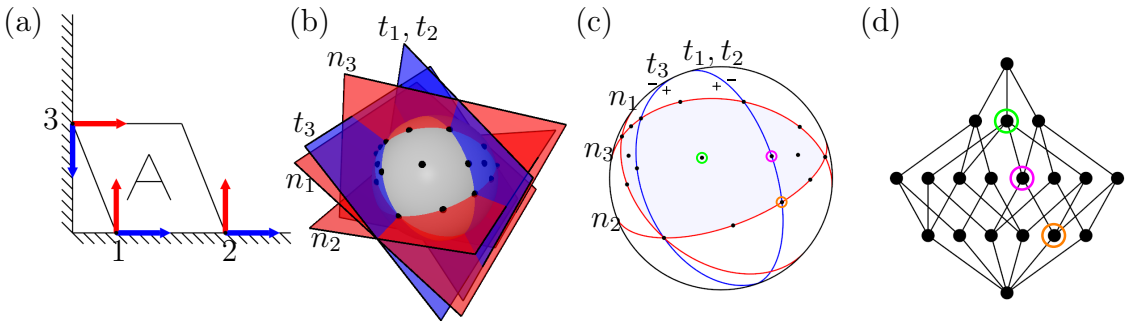


Figure 4-6: From velocity constraints to contact modes (left to right): (a) Input 2D scene with a parallelogram against the wall. The normals and tangents are depicted in red and blue, respectively. (b) The normal and tangent velocity constraints visualized as a hyperplane arrangement $\mathcal{A}([N;T])$ in velocity space. Each feasible face of the arrangement is marked with a black dot. The gray sphere serves as a visual aid. (c) The constraints visualized again as their projected geodesics on the visible hemisphere. The region of feasible velocities $\mathcal{H}(N,0)$ is shaded light gray-blue. (d) The face lattice $\mathcal{L}$ over the feasible faces, i.e. the graph of contact modes.

(E7) Replace each matrix entry by its image under some automorphism of $\mathbb{R}$.

Using operations **(E1)**-**(E5)**, one can always reduce a matrix A into the form $[I|D]$. The only automorphism of the real numbers is the identity map. Therefore, ignoring **(E7)**, we arrive at the following notion of *projective equivalence*. Matrices $A_1, A_2 \in \mathbb{R}^{d\times n}$ are *projectively equivalent* representations of a matroid if and only if there exists a non-singular matrix $X \in \mathbb{R}^{d\times d}$ and non-singular diagonal matrix $Y \in \mathbb{R}^{n\times n}$ such that $A_2 = XA_1Y$. For further reading, the books by Oxley and Bjorner are excellent references [88, 14].

4.4 Geometry of Contact Modes

4.4.1 Contacting-Separating Modes

4.1 Theorem. For a system at a given state, let $N \in \mathbb{R}^{n\times d}$, $\phi \in \mathbb{R}^n$ be its normal velocity constraints. The contacting-separating modes of this system are the faces of the convex polyhedra $\mathcal{H}(N,\phi)$.

Proof. Let m_{cs} be a contacting-separating mode and let $\dot{q}_{cs}$ be a velocity which realizes this mode. Since $\mathcal{H}(N,\phi)$ is the disjoint union of the relative interiors of its

faces, $\dot{q}_{cs}$ must be contained in the relative interior of a single face. The sign vector of this face is unique and it is m_{cs}. □

4.4.2 Sliding-Sticking Modes

4.2 Theorem. For a system at a given state with cs-mode m_{cs}, let $N_c \in \mathbb{R}^{n_c \times d}$, $\phi_c \in \mathbb{R}^{n_c}$ be the normal velocity constraints which are maintaining contact, let $N_s \in \mathbb{R}^{n_s \times d}$, $\phi_s \in \mathbb{R}^{n_s}$ be the normal velocity constraints which are separating, and let $T_c \in \mathbb{R}^{(n_c k) \times d}$ be the active tangent velocity constraints. The sliding-sticking modes are the faces of the hyperplane arrangement $\mathcal{A}(T_c)$ which intersect the relative interior of the convex polyhedra $H_{cs} = \{N_c\dot{q} = \phi_c, N_s\dot{q} \geq \phi_s\}$.

Proof. Let m_{ss} be a sliding-sticking mode for a given contacting-separating mode m_{cs} and let $\dot{q}_{ss}$ be a velocity which realizes these modes. Since $\mathbb{R}^d$ is the disjoint union of the relative interiors of the faces of $\mathcal{A}(T_c)$, we know that the relative interior of H_{cs} can be written as the disjoint union of the non-empty intersections of the faces of $\mathcal{A}(T_c)$ and H_{cs}. As before, $\dot{q}_{ss}$ must be contained in an unique face of $\mathcal{A}(T_c)$ with sign vector m_{ss}. □

4.3 Corollary. For a system at a given state, let $N \in \mathbb{R}^{n \times d}$, $\phi \in \mathbb{R}^n$ be its normal velocity constraints and let $T \in \mathbb{R}^{(nk) \times d}$ be its tangent velocity constraints. The sliding-sticking modes of this system are contained in the faces of the hyperplane arrangement $\mathcal{A}(T)$ which intersect the boundary of the convex polyhedra $\mathcal{H}(N, \phi)$.

4.5 Contacting-Separating Mode Enumeration

The first task in contact mode enumeration is to determine the contacting-separating modes. Once the contacting-separating modes are obtained, the sliding-sticking modes may also be enumerated.

4.5.1 Convex Hull Method

The contacting/separating mode enumeration algorithm, or CS-Enumerate, takes as input the normal velocity constraint equations $A \in \mathbb{R}^{n \times d}$ (see Section 4.3.1) and generates a list of valid contacting/separating sign vectors of the form $m \in \{0, +\}^n$. The algorithm presented in this subsection is based on taking the convex hull in polar form of the polytope associated with the normal velocity constraints. The pseudo-code is listed in Algorithm 4 and we provide explanations for each of the steps below.

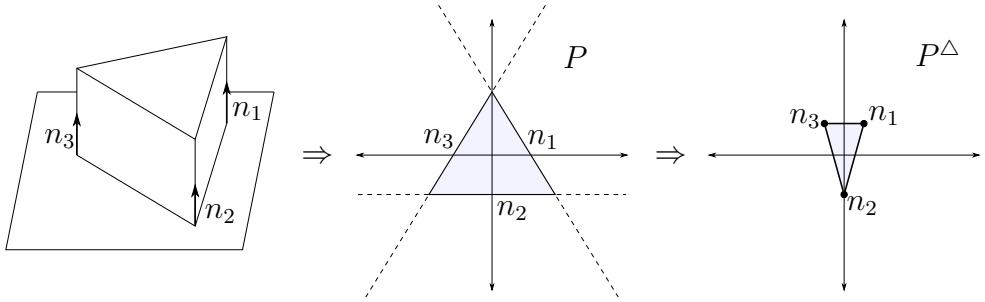


Figure 4-7: Example of contacting-separating mode enumeration using the convex hull method.

Find an interior point: The polar form P^Δ of a polyhedron P is defined only when $0 \in \text{relint}(P)$. However, 0 is on the boundary of the polyhedral cone $\mathcal{H}(A, 0)$ defined by our normal velocity constraints. Therefore, our first step is to find a point $r \in \text{relint}(\mathcal{H}(A, 0))$. This is a classical problem in linear programming, and for our implementation, it amounts to solving the following linear program

$$\min_{r,c} \quad \begin{bmatrix} 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} r \\ c \end{bmatrix} \tag{4.16}$$

$$\text{s.t.} \quad Ar + c \leq 0,\ c \geq 0 \tag{4.17}$$

$$\|r\|_\infty \leq 1, \tag{4.18}$$

where $\|r\|_\infty \leq 1$ constrains r to be within the hypercube in $\mathbb{R}^d$. Note that if the solution to the linear program is $r = 0$, then the only valid mode is all-contacting and the algorithm can terminate early. The above method was adapted from [95] to handle cones.

Project to contact planes: If the interior point is on the boundary for a subset of the normal velocity constraints, then that subset of contact points must always be in contact (for example, a box sandwiched between two walls). Let A_c be the contacting normal velocity constraints and $A_o = A/A_c$. Then we map A_o into the nullspace of A_c, like so $A_o = A_o \cdot \textsc{Null}(A_c)$, to reduce the dimension of the problem. We also express the interior point as coordinates in the null space.

Convert to polar form: Given a strictly interior point r, we translate the origin to r, resulting in the new $\mathcal{H}$-polyhedron $P = \mathcal{H}(A, -Ar)$. Next we normalize the inequalities so that $P = \mathcal{H}(A, 1)$ and obtain the polar polytope $P^\Delta = \mathcal{V}(A^T)$.

Project to affine subspace: The affine dimension of the polar polytope dim aff P^Δ

Algorithm 4 CS Mode Enumeration

Require: Contact points: $p_1, \ldots, p_k$; Contact normals: $n_1, \ldots, n_k$
Ensure: Contacting/Separating Modes: M_{cs}

function CS-ENUM(P, N)
  $A \leftarrow$ ADD-HYPERPLANES($[n_i, -n_i\hat{p}_i]$) for all contacts $i \in [1, \ldots, k]$
  $r \leftarrow$ INTERIOR-POINT($\mathcal{H}(A, 0)$)
  A, $r \leftarrow$ PROJECT-TO-NULLSPACE(A, r)
  $\mathcal{V}(A^T) \leftarrow$ POLAR($\mathcal{H}(A, 0), r$)
  $A^T \leftarrow$ PROJECT-TO-AFFINE-SUBSPACE(A^T)
  $M \leftarrow$ CONV-HULL($\mathcal{V}(A^T)$)
  $L \leftarrow$ FACE-LATTICE(M)
  M_{cs}, $m \leftarrow \varnothing$, []
  for $k \in \{0, \ldots, d-1\}$ **do** ▷ The face lattice has d proper ranks.
    for $f \in L[k]$ **do** ▷ Each face in $L[k]$ as defined by its vertex (= constraint) set.
      $m[f] \leftarrow 0$ ▷ A vertex $v \in f \Rightarrow$ sign of normal vel at that contact is 0
      $m[A^T \setminus f] \leftarrow 1$ ▷ A vertex $v \notin f \Rightarrow$ sign of normal vel at that contact is $+1$
      $M_{cs} \leftarrow M_{cs} \cup \{m\}$
  return M_{cs}

may not necessarily be equal to the dimension of the ambient space $\mathbb{R}^d$. In this situation, we project P^Δ into its affine subspace $\text{aff}\, P^\Delta = \{A^T v : 1^T v = 1\}$ and further reduce the dimensionality of convex hull. Recall that an affine space can also be expressed as a linear space plus a translate, i.e. $\text{aff}\, P^\Delta = \{Vx + z\}$ for some V and z. If $0 \notin \text{aff}\, P^\Delta$, then $z \neq 0$ and we translate the affine space until it contains the origin. Now that $0 \in \text{aff}\, P^\Delta$, $\text{aff}\, P^\Delta$ is a linear subspace and we project each point (column vector) in A^T to coordinates on the column space of A^T.

Get facet-vertex incidence matrix: Next, the algorithm constructs the vertex-facet incidence matrix M of $P^\Delta = \mathcal{V}(A^T)$ by using a convex-hull algorithm. The vertex-facet incidence matrix is a matrix $M \in \{0, 1\}^{n_v \times n_f}$, where n_v and n_f are the number of vertices and facets, respectively. We associate the vertices and facets with the index sets $I_V = \{1, \ldots, n_v\}$ and $I_F = \{1, \ldots, n_f\}$, so that $m_{vf} = 1$ if facet f contains v and $m_{fv} = 0$ otherwise. The vertex-facet incidence matrix is a standard return value from convex hull algorithms such as `qhull` [9].

Build face lattice: Given the facet-vertex incidence matrix M of P^Δ, we can construct the face lattice $L(P^\Delta)$ using the algorithm of Kaibel and Pfetsch [60]. Their method is based on find the closed sets (= faces) with respect to a closure map

defined over vertex sets. Obviously, each face is uniquely represented by its vertex set.

Convert faces to mode strings: Finally, we construct contacting/separating mode strings using the vertex sets associated with each face f in $L(P^\Delta)$. By polarity, each vertex in f corresponds to the hyperplane $\{x : ax = 0\}$ defined by a normal velocity constraint. Therefore, we can read off the mode string by assigning contacting modes to every vertex in f and separating modes to every vertex not in f.

4.4 Theorem. For a set of n contacts in a system of colliding bodies with d degrees of freedom, Algorithm 4 enumerates the possible contacting and separating modes in $O(d \cdot n^{d+1} + l(n,d))$ time.

Proof. We analyze correctness first before complexity. The proof is simple and relies on the combinatorial equivalences between

$$\text{CS-MODES} \leftrightarrow L(\mathcal{H}(A,0)) \leftrightarrow L(\mathcal{H}(A_{\text{int}},1)) \leftrightarrow L(\mathcal{V}(A_{\text{int}}^T)). \tag{4.19}$$

First, we show that $\mathcal{H}(A,0)$ and $\mathcal{H}(A_{\text{int}},1)$ are affinely isomorphic and thus, combinatorially equivalent [121]. Two polytopes P and Q are *affinely isomorphic* if there exists an affine map $f : \mathbb{R}^d \to \mathbb{R}^e$ that is a bijection between the vertices of the two polytopes. By inspection, the operations $P \cap \text{aff}(P)$ and $P + r$ preserve the extremal points (vertices). Finally, re-scaling the inequalities does not affect the underlying polytope.

For this next paragraph, let us define $P = \mathcal{H}(A_{\text{int}},1)$ and $P^\Delta = \mathcal{V}(A_{\text{int}}^T)$. Our aim is to show the first and third bijections in (4.19). Let $F \in L(P^\Delta)$ be identified by its vertex set $V(F) = \{a : a \cap F \neq \varnothing, a \in \text{vert}(P^\Delta)\}$ and recall that $\text{vert}(P^\Delta) \subseteq \text{col}(A_{\text{int}}^T) = \text{row}(A_{\text{int}})$. (That is, each vertex of P^Δ corresponds to a facet of P, i.e. a normal velocity constraint.) Then by Corollary 2.13 of Ziegler [121], there is a bijection $L(P^\Delta) \leftrightarrow L(P)$ from F to $F^\diamond$ such that

$$F^\diamond = \{x : A_{\text{int}}x \leq 1, ax = 1, \forall a \in V(F)\} \tag{4.20}$$

is a non-empty face of P. Because face lattice of a polytope is coatomic, we can uniquely specify its proper elements as meets (intersections) $a_1 \wedge \ldots \wedge a_k$ of its coatoms (facets). Therefore, for each $F \in L(P^\Delta)$, the vertex set $V(F)$ maps bijectively to a valid contacting/separating mode string, and $L(P^\Delta)$ enumerates the set of all valid contacting/separating modes.

The normal velocity constraint matrix A can be constructed in $O(n \cdot d)$ time. The orthonormal basis and null space can be computed in $O(\min\{n \cdot d^2, n^2 \cdot d\})$ using SVD. An interior point can be computed in time $O(l(n,d))$, where $l(n,d)$ is the cost

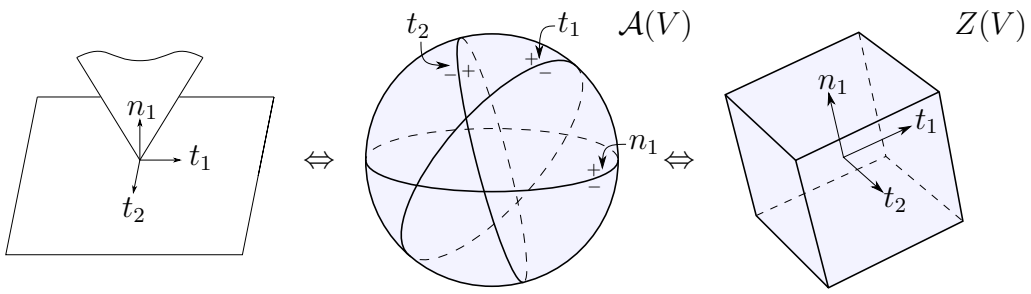


Figure 4-8: Combinatorially equivalent representations of sliding-sticking modes.

of linear programming. For a balanced problem like this one (every input point is extremal), quick hull runs in $O(f_{d-1}) = O(n^{d/2})$. The number of k-faces in $L(P)$ is bound by $O(n^{d/2})$. The combinatorial face enumeration algorithm runs in time $O(n \cdot \sum_{k=0}^{d} f_k^2) = O(d \cdot n^{d+1})$. Therefore, the total runtime is $O(d \cdot n^{d+1} + l(n,d))$. $\square$

4.6 Sliding-Sticking Mode Enumeration

This section derives two algorithms for sliding-sticking mode enumeration. Recall that the sliding-sticking modes are contained in the faces of the arrangement generated by the normal and tangent velocity equations. Therefore, the first algorithmic desigin is the combinatorially equivalent geometric object to base the algorithm on. The two choices, zonotopes or hyperplane arrangements, used in this work are illustrated in Figure 4-8. The zonotope algorithm is based on computing the zonotope using an iterative Minkowski sum. The (partial) hyperplane arrangement algorithm iteratively builds the arrangement directly in the space of hyperplanes. We begin with the former algorithm.

4.6.1 Zonotope/Minkowski Sum Method

The algorithm, SS-Enumerate, generates a list of sliding/sticking sign vectors of the form $m_{ss} \in \{-1, 0, +1\}^{n_t}$, where n_t is the total number of tangent velocity hyperplanes. As before we provide explanations for each of the steps below.

Partition the hyperplanes: The goal of our algorithm is to enumerate sliding/sticking modes for the "contacting" contacts. Given a contacting/separating mode

Algorithm 5 Zonotope/Minkowski Sum Method

Require: Contact points: $p_1, \ldots, p_k$; Contact normals: $n_1, \ldots, n_k$; Contact tangent dividing planes: $T_1, \ldots, T_k$; Contact/Separating Mode: m_{cs}
Ensure: Sliding Modes: M_{ss}

function SS-ENUM-ZONO(P, N, T, m)
  $H \leftarrow \varnothing$
  $H \leftarrow$ ADD-HYPERPLANES($[n_i, -n_i\hat{p_i}]$) for all separating contacts $m_{cs}^{(i)} = 1$
  $H \leftarrow$ ADD-HYPERPLANES($[T_j, -T_j\hat{p_j}]$) for all contacting contacts $m_{cs}^{(j)} = 0$
  $H \leftarrow$ PROJECT-TO-NULLSPACE($[n_j, -n_j\hat{p_j}]$) for all contacting contacts $m_{cs}^{(j)} = 0$
  $V, S \leftarrow$ GET-ZONOTOPE-VERTICES(H)
  $L \leftarrow$ FACE-LATTICE($\mathcal{V}(V)$)
  $\mathcal{F} \leftarrow F \in L$ with positive signs $\{+1\}$ for all separating contact hyperplanes
  $M_{ss} \leftarrow$ GET-SIGN-VECTOR($\mathcal{F}, S$)
  return M_{ss}
function GET-ZONOTOPE-VERTICES(H)
  $V,\ S \leftarrow [0],\ \varnothing$
  for $h \in H$ **do**
    $V',\ S' \leftarrow [\,],\ \varnothing$
    for $v \in V$ **do**
      $V' \leftarrow$ ADD-POINTS($v + h, v - h$)
      $S' \leftarrow$ ADD-SIGN-VECTORS($(S(v), +1), (S(v), -1)$)
    $V \leftarrow$ CONVEX-HULL(V')
    $S \leftarrow S'(V)$
  return V, S

m_{cs}, the normal velocity constraints on the object velocity x are

$$H_c x = 0, \quad H_{sep} x > 0 \tag{4.21}$$

where $H_{sep} = \begin{bmatrix} n_i, & -n_i\widehat{p}_i \end{bmatrix}$ for all separating contacts $m_{cs}^{(i)} = 1$, $H_c = \begin{bmatrix} n_j, & -n_j\widehat{p}_j \end{bmatrix}$ for all contacting contacts $m_{cs}^{(j)} = 0$. We have hyperplanes $H_{ss} = \begin{bmatrix} T_j, -T_j\hat{p_j} \end{bmatrix}$ for all contacting contacts $m_{cs}^{(j)} = 0$. If we let $H_o = [H_{sep}, H_{ss}]$, all valid sliding/sticking modes can be written as

$$M_{ss} = \{\text{sign}(H_o x) : x \in \mathbb{R}^d, H_{sep} x > 0, H_c x = 0\}. \tag{4.22}$$

Our algorithm computes the combinatorial structure of the hyperplane arrangement H_o to get the sliding/sticking modes M_{ss}. To decrease the zonotope construction cost, one may omit some separating hyperplanes from H_o at the expense of allowing some invalid sliding/sticking modes to slip through.

Project to contact planes: We project H_o into halfspaces on the coordinate of the null space H_c to speed up computation. In Algorithm 5 step 3-5, the projected hyperplanes are obtained by $H = \big[H_{sep} \cdot \textsc{Null}(H_c), H_{ss} \cdot \textsc{Null}(H_c)\big]$.

Construct the zonotope: From the set of hyperplanes H, we can identify the vertices of its associated zonotope $Z(H)$. From Equation 4.13, zonotopes can be represented by the minkowski sum of line segments, which in this case are $\big[h_i, -h_i\big]$ for $h_i \in H$. Algorithm 5 Function Get-Zonotope-Vertices obtains zonotope vertices V through computing the minkowski sum iteratively for every $h_i \in H$. We initialize the vertex set $V = [0]$, and at the i-th iteration we update V with the convex hull of $\cup_{v \in V}\{v + h_i, v - h_i\}$. We are also able to get the sign vector for every vertex: $\text{sign}(v + h_i) = [\text{sign}(v), +]$, $\text{sign}(v - h_i) = [\text{sign}(v), -]$.

Build the face lattice Using the same method as describe in Section 4.5.1, we construct the face lattice $L(V)$ from the vertices V. Not every $F \in L(V)$ corresponds to a valid sliding/sticking contact mode. Only the faces that have positive signs $+1$ with respect to normal velocity constraints for all separating contacts are valid sliding/sticking contact modes. After building the face lattice $L(V)$, valid faces $\mathcal{F}$ are selected by ensuring their sign vectors with respect to H_{sep} are all $\{+1\}$s:

$$\mathcal{F} = \{F \in L(V) : \text{sign}_{H_{sep}}(F) = [+1, \ldots, +1]\} \tag{4.23}$$

The sign vectors of all faces in $\mathcal{F}$ with respect to H represent all valid sliding/sticking contact modes for the given contact/separating mode.

4.5 Theorem. For a set of n contacts (modeled with k tangent planes) in a system of colliding bodies with d degrees of freedom, Algorithm 5 enumerates the possible sliding/sticking modes in $O(n^{d^2/2+2d})$ time for a given contacting/separating mode.

Proof. As before, we first proceed with a proof of correctness. For a given contacting/separating mode string m_{cs}, let $H = [h_{s_1}, \cdots, h_{s_k}, h_{t_1}, \cdots, h_{t_m}]$ be the input hyperplanes to our zonotope construction algorithm, where k is the number of separating hyperplanes and m is the number of tangent hyperplanes. We incrementally construct the zonotope by using the fact that the Minkowski sum of two polytopes is the convex hull of the sums of their vertices [26]. By Corollary 7.18 of [121], the face lattice of the zonotope constructed above is the opposite of the face lattice of the hyperplane arrangement.

Next we analyze the complexity of our algorithm. The maximum number of hyperplanes is kn. The number of vertices, i.e. f_0, for a d-zonotope that is the projection of a p-cube is of the order $O(p^{d-1})$ [30]. Therefore, our zonotope construction algorithm takes time

$$O(\sum_{p=1}^{kn}(p^{d-1})^{\frac{d+1}{2}}) \approx O(\sum_{p=1}^{kn} p^{\frac{d^2}{2}}) \approx O((kn)^{\frac{d^2}{2}}). \tag{4.24}$$

We use Kaibel and Pfetsch [60] to construct the face lattice of the resulting zonotope. As before, their algorithm runs in

$$O(kn \cdot d \cdot \alpha \cdot (kn)^{d-1}) \approx O(d(kn)^{2d}), \tag{4.25}$$

where $\alpha = kn \cdot (kn)^{d-1}$ in the worst case (when the zonotope is simple) [121]. The full complexity of SS-ENUMERATE is therefore $O((kn)^{\frac{d^2}{2}+2d})$.

□

The zonotope algorithm's strength is its ease of implementation. However, it has two major drawbacks. First, the iterative Minkoski sum is very inefficient. It requires d orders of magnitude more computation than there are faces in $\mathcal{A}$. Second, the algorithm computes the full arrangement, including invalid modes (faces) that have $-$ signs with respect to the separating normals. In the next section, we will address both those of those issues and derive the partial hyperplane arrangement algorithm that has runtime nearly proportional to the number of valid sliding-sticking modes.

4.6.2 Partial Hyperplane Arrangement Method

The Edelsbrunner's [30] incremental hyperplane arrangement algorithm is a natural choice for this method as it achieves runtime proportional to the number of faces in the arrangement, $O(n^d)$. We further improve the method with a "partial" hyperplane arrangement technique based on the following observation. The normal velocity equations define one-sided constraints, and as such, the algorithm should avoid enumerating modes representing invalid, penetrating velocities. Figure 4-9 illustrates the key idea. The ideal location to enforce one-sided hyperplanes is in initialization phase of Edelsbrunner's algorithm. During initialization, the partial variant of the algorithm either initializes using a convex hull of the one-sided hyperplanes or a mixed initialization of d_{eff} one-sided and two-sided hyperplanes, where d_{eff} is the effective dimension of the arrangement. (The mixed method is used when a convex hull of rank d_{eff} is not possible.) The next paragraphs describes in more

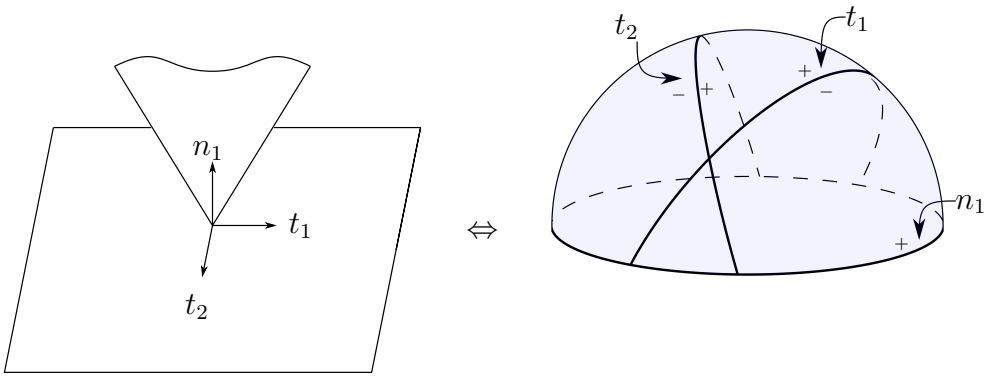


Figure 4-9: Partial hyperplane arrangement representation of sliding-sticking modes.

detail each bold commented step of the pseudo-code, Algorithm 6, for the partial hyperplane arrangement method.

1. Partition hyperplanes based on cs-mode: The normal and tangent velocity equations, $Nx = d$, and $Tx = 0$ are partitioned into contacting and separating sub-matrices and sub-vectors. The c subscript denotes contacting and the s separating.

2. Project into null space of $\mathbf{N_c}$ and row space of $[\mathbf{N_s}; \mathbf{T_c}]$: As before, both these operations reduce the problem dimension without changing the topology of the underlying geometric object. Note, the arrangement will be built from the hyperplanes in $[N_s; T_c]$.

3. Initialize the hyperplane arrangement: The initialization is broken down into two cases. If the degree of the separating hyperplanes, $d_s = \text{Rank}(N_s)$, is equal to the degree of the system $d_{\text{eff}} = \text{Rank}([N_s; T_c])$ and number of separating hyperplanes, n_s, is greater than d_s, then we may initialize the arrangement with a convex hull of separating hyperplanes. Under the hood, the convex hull function computes $\mathcal{H}(N_s, d_s)$ in polar form. Otherwise, an initial arrangement is generated from a set of d_{eff} linearly independent hyperplanes. If all the chosen hyperplanes are two-sided, this initial arrangement has $3^{d_{\text{eff}}}$ proper faces. But, we can choose a set of d_s separating hyperplanes and $d_{\text{eff}} - d_s$ tangent hyperplanes, for a total of $2^{d_s} 3^{d_{\text{eff}} - d_s}$ proper faces. To construct the initial arrangement, the faces are trivially enumerated in lexicographical order and the arcs are determined by finding the covers of each face.

Algorithm 6 Partial Hyperplane Arrangement Method

Require: Normal velocity equations: N, d; Tangent velocity equations: T; CS-mode: m_{cs}; Tolerance: ϵ.

Ensure: Sliding Modes: M_{ss}

function SS-ENUM-PARTIAL($N, d, T, m_{cs}, \epsilon$)
  ▷ **1. Partition hyperplanes based on cs-mode.**
  $N_c, N_s, d_c, d_s \leftarrow$ PARTITION(N, d, m_{cs})
  $T_c, T_s \leftarrow$ PARTITION(T, m_{cs})
  ▷ **2. Project into null space of $\mathbf{N_c}$ and row space of $[\mathbf{N_s}; \mathbf{T_c}]$.**
  $K \leftarrow$ KERNEL(N_c, ϵ)
  $[N_s; T_c] \leftarrow [N_s; T_c]K$
  $R \leftarrow$ ROW-SPACE($[N_s; T_c], \epsilon$)
  $[N_s; T_c] \leftarrow [N_s; T_c]R$
  ▷ **3. Initialize the hyperplane arrangement**
  $d_{\text{eff}} \leftarrow$ RANK($[N_s; T_c], \epsilon$)
  $n_s, d_s \leftarrow$ NUM-ROWS(N_s), RANK(N_s, ϵ)
  if $n_s > d_s$ AND $d_s = d_{\text{eff}}$ **then**
    $\mathcal{A} \leftarrow$ CONVEX-HULL(N_s, d_s, ϵ)
  else
    $\mathcal{A} \leftarrow$ INITIAL-ARRANGEMENT($[N_s; T_c], [d_s; 0], \epsilon$)
  ▷ **4. Incrementally add the remaining hyperplanes.**
  for $h \in [N_s; T_c],\ z \in [d_s; 0], (h, z) \notin \mathcal{I}$ **do**
    $\mathcal{A} \leftarrow$ INCREMENT-ARRANGEMNT($h, z, \mathcal{A}, \epsilon$)
  ▷ **5. Update sign vectors.**
  $N \leftarrow NKR$
  $\mathcal{A} \leftarrow$ SET-HYPERPLANES($\mathcal{A}, [N; T_c], [d; 0]$)
  return $M_{ss} \leftarrow$ SIGN-VECTORS($\mathcal{A}, \epsilon$)

4. Incrementally add the remaining hyperplanes: For each hyperplane not in the arrangement $\mathcal{A}$, incrementally add it to $\mathcal{A}$ using Edelsbrunner's method. This technique is fairly robust as it only uses floating point operations, i.e. dot products, to color vertices and edges with respect to the incoming hyperplane. The rest of the lattice is updated purely based on that coloring.

5. Update sign vectors: Internally, each face in the lattice stores an interior point. To calculate the sign vectors with respect to the original hyperplane ordering of $[N; T_c]$, it is as simple as reading off the signs of a dot product between the interior point and the hyperplanes. This trick makes implementing the rest of the algorithm

Algorithm 7 Preprocess Hyperplanes

```
1: function PREPROCESS(A^T)
2:   C ← COLUMN-SPACE(A)
3:   A ← C^T A
4:   A ← REMOVE-ZEROS(A)
5:   A[:, i] ← A[:, i]/‖A[:, i]‖
6:   A ← REMOVE-PARALLEL(A)
7:   return A^T
```

easier as it removes need for bookkeeping during initialization and incrementation.

4.6 Theorem. The partial hyperplane arrangement computes the sliding-sticking modes in $O(n^d)$ time.

We present a sketch of a proof. To recap, we are using an incremental hyperplane arrangement algorithm on a partial hyperplane arrangement instead of a full hyperplane arrangement. First, we argue the correctness of the algorithm. The key idea is that faces of the partial hyperplane arrangement are faces in the full hyperplane arrangement. The correctness of Edelsbrunner's algorithm is solely based on the properties of faces outlined in Table 7.3, Observation 7.2, Observation 7.3 and Observation 7.4 [30]. It is straightforward to verify each property holds for the faces in the partial hyperplane arrangement using proof by contradiction.

The runtime analysis is based on the Section 7.6 of [30]. The key result is that a new hyperplane h can be inserted into an existing arrangement with time proportional to the number of faces properly intersecting h. However, it is outside the scope of this work to derive bounds on the number of k-faces intersecting h, as is done in Theorem 5.4 of [30]. A conservative bound on the runtime is $O(n^d)$ which is the cost of constructing the full hyperplane arrangement.

4.7 Preprocessing Hyperplanes

The preprocessing routine is an important step which removes degeneracies from the input hyperplanes and reduces problem dimensionality. Recall from Section 4.3.3 that a linear hyperplane arrangement $\mathcal{A}(A^T)$ is combinatorially equivalent to the vector matroid $M[A]$. This section outlines a matroid-based preprocessing routine for reducing the set of normal and tangent velocity hyperplanes into a minimal set of projectively equivalent hyperplanes. For simplicity, we first present our preprocessing

	Convex Hull	Zonotope	Partial
Test Case	cs-modes	ss-modes	ss-modes
Box-1	0.3 ms	18.5 ms	0.6 ms
Box-2	0.7 ms	83.0 ms	4.8 ms
Box-3	2.5 ms	191.8 ms	42.5 ms
Quadraped	0.4 ms	106 582.2 ms	26.0 ms
Arm-Box	0.5 ms	629.1 ms	9.8 ms
Humanoid	0.4 ms	107 764.0 ms	37.4 ms

Table 4.1: Timing benchmarks for contact mode enumeration algorithms.

routine for the zero offset case, i.e. $N\dot{q} \leq 0$. Afterwards, we will describe how to extend this test to non-zero offsets.

Let $A = [N; T]^T \in \mathbb{R}^{d \times m}$ be the input hyperplane normals in column vector form. We want to find invertible matrices X and Y, with Y diagonal, such that $XAY = A' = [N'; T']^T \in \mathbb{R}^{d' \times m'}$ has minimal dimensions. We can accomplish this in two steps.

1. Reduce dimension: To minimize the dimension d, we can choose $X = [C^T; L^T]$ where $C \in \mathbb{R}^{d' \times d}$ is the an orthonormal basis for the column space of A and $L \in \mathbb{R}^{(d-d') \times d}$ is a basis of the left nullspace of A. After multiplying X and A, we can remove the bottom $d - d'$ rows (**E4**). Note that this operation does not change the orientation of the hyperplanes.

2. Remove duplicate/zero hyperplanes: First, we remove any hyperplanes with zero magnitude. Next, we use Y to normalize each hyperplane and remove any hyperplanes that are parallel to a previous column in A. This operation maintains projective equivalence because $M[A]$ is defined over the *set* of column vectors in A.

We can extend the above test to handle non-zero offsets by lifting the hyperplanes into dimension $d + 1$. Given a hyperplane $h = \{x \in \mathbb{R}^d : ax = b\}$, observe that h is the orthogonal projection of a $d + 1$ dimensional linear hyperplane onto the unit vertical plane

$$\{y : [a, -b]y = 0\} \cap \{y : y_{d+1} = 1\},\ y \in \mathbb{R}^{d+1}. \tag{4.26}$$

Therefore, we can extend this test to the non-zero offset case by changing the inputs to $N' = [N, \phi]$ and $T' = [T, 0]$, where ϕ is the offset.

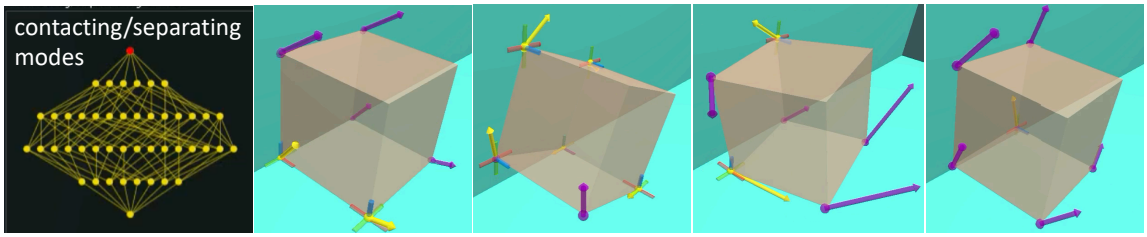


Figure 4-10: Visualization of some contact modes of a box enclosed by 2 walls. Given a contact mode, we sampled object velocity and rendered it in the simulation. Yellow arrows: velocities of contacting contacts. Purple arrows: velocities of separating contacts.

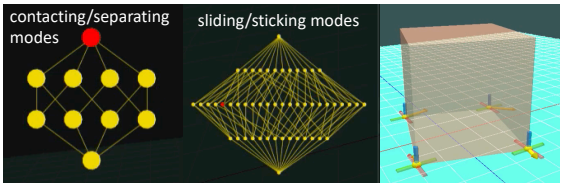


Figure 4-11: The contacting/separating modes and sliding/sticking modes of a box on the table case.

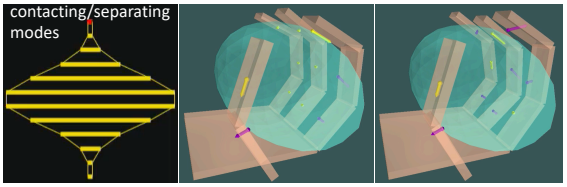


Figure 4-12: A ball in a hand. There are too many contacting/separating modes so we only visualize them by layers.

4.8 Results

This section collects the results of running our algorithms (cs-mode-enum, ss-mode-enum, all-modes) on the example scenarios described below. The examples were run on a computer with an Intel i7-7820x CPU (3.5 MHz, 16 threads).

Box-#: This example simulates a box enclosed by # walls, $1 \leq \# \leq 3$. The face contact between the box and a wall is modeled as four point contacts on the corresponding vertices of the box. Constrained by one wall (the ground), the box is free to separate with or move along the wall. As the constraints grow to 6 walls, the box becomes progressively more constrained. Figure 4-10 and 4-11 visualize some of our results.

Quadraped: This example consists of a MIT Mini-Cheetah standing on the ground. There are three DoFs in each leg for a total of 18 DoFs.

Arm-Box: This example consists of a KUKA arm making a point contact on a box on the ground. There are a total of 13 DoFs.

Humanoid: This example consists of a humanoid figure standing on the ground with two contacts at each foot. There are a total of 24 DoFs.

The above examples can be replicated ad infinitum (by adding more boxes or fingers) to generate contact modes which grow exponentially with the system dimension. Table 4.1 aggregates the results of running our algorithm on the above examples. Selected videos of the examples are also provided as supplementary material. From the results, it is clear that the zonotope algorithm is inferior to the partial hyperplane arrangement algorithm.

4.9 Related Applications & Future Work

Simulation: Friction contact dynamics have been modeled as complementarity problems by many researchers [8, 101]. Approaches for solving complementarity problems may be broken into direct methods, such as Lemke's algorithm, and iterative methods, such as Projected Gauss Siedel [84, 24]. The simulation research community has focused on issues such as proving finite termination [111, 101], satisfying physical realism [100, 109], and improving convergence speed [70]. By design these algorithms find a single solution, when in reality, multiple solutions may exist [101]. Moreover, users are not aware of this additional source of divergence. However, our algorithm can allow the user to enumerate all possible solutions. We believe this capability will be important for robotic systems which use simulation to reason about future actions.

Grasping and Dexterous Manipulation: Most grasping synthesis algorithms [13] [82] [80] are designed to plan force-closure graspings. However, due to the static indeterminacy problem, a force closure grasp does not imply a stable grasp [79]. A typical way to address this problem is to enumerate all adjacent contact modes and make sure they don't have the same solution to the desired contact mode. Our 3D contact enumeration method could provide fast computation tools for stable grasps in 3D, which may help with real-time grasp planning for large scale objects. Similar approach can also be extended to dexterous manipulation tasks, like pushing [73] and grasping using environment contacts [50], where certain contact mode is desired during the task.

4.10 Conclusion

This chapter introduced the first known algorithm for efficient contact mode enumeration in 3D. The algorithm partitions the problem into contacting/separating mode enumeration and sliding/sticking mode enumeration. The algorithms are based on convex hull and hyperplane arrangements, respectively. This paper also presented

results demonstrating real-time enumeration for small problems. Finally, we highlighted related research areas to show that contact mode enumeration can be a useful tool for the simulation, analysis, and control of robotic systems which make and break contact with the environment.

Chapter 5

Autogenerated Manipulation Primitives II

5.1 Introduction

It is hard to estimate the set of manipulation skills used by humans or animals, and even harder to reproduce all of those skills. Byrne [15] documented 72 functionally distinct manipulation primitives used by foraging mountain gorillas. Nakamura [85] studied human grasping behaviors in grocery stores and noticed that existing taxonomies cannot categorize all the observed behaviors. Within robotics, many researchers have explored individual skills at depth, including pushing [77][73], pivoting [3, 47, 49], tumbling [74], whole-body manipulation [98], extrinsic dexterity [20], grasping [33][32], shared grasping [51], etc. Clearly, designing a high-quality robotic manipulation skill for a specific task takes significant human labor. When planning for robotic manipulation tasks, it is therefore desirable to find an approach that reduces the programming effort required and creates skills that generalize to many scenarios.

In this chapter, we propose a novel method for automatically generating robotic manipulation primitives. Given a desired object velocity, the current state, and the dynamic equations of motion, our method generates a partially ordered list of manipulation primitives which would best approximate the object motion while satisfying contact dynamics. The generated manipulation primitives are uniquely specified by their contact mode (see Chapter 4). Our method is based on two key observations. First, specifying the contact mode reduces contact and friction force constraints into equality and inequality constraints. This in turn allows us to solve for the controls using quadratic programming. Second, the contact mode lattice is effectively a graph

data structure. This allows us to use graph search techniques when designing our algorithm. Our algorithm walks along the contact mode lattice and solves a quadratic program at each visited node. In this way, we iteratively build a library of distinct manipulation primitives. A library of manipulation primitives is useful for several reasons.

1. A library of manipulation actions adds robustness to failure. A single type of action may fail, such as grasping in a specific location, however a library gives the robot multiple opportunities to try until success.

2. Automatic primitive generation removes the burden of writing task specific code.

We envision this method having applications in dexterous manipulation ranging from planning to closed-loop control and more. See Figure 5-5 for an example where our method could be applied. For the scope of this project, we focus on solving inverse task mechanics within the quasi-static hybrid dynamical systems introduced in Trinkle et al. [110]. We test our algorithm on 3D examples. We validate the usefulness of autogenerated manipulation primitives on both planar and vertical 2D manipulation problems. These manipulation problems include peg-in-hole and force-limited scenarios.

5.2 Related Work

Past works have modeled contact dynamics as a complementarity problem [101]. Several methods exist for solving complementarity problems, which we categorize as iterative or direct solution methods. A popular iterative method is projected Gauss-Seidel (PGS) [83]. Stewart and Trinkle [101] used a direct pivoting method, Lemke's algorithm, for solving contacts modeled as linear complementarity problems. The mathematical framework used in our work is based on the direct quasi-static implicit time-stepping scheme of Trinkle et al. [110], sometimes with an ellipsoidal limit surface model of sliding friction included from Zhou et al. [119].

This work uses the linear complementarity formulation of frictional contact. Horak and Trinkle [48] compared various popular frictional contact models used in rigid body simulation, including the linear complementarity formulation and the convex relaxation of Todorov [108], and concluded that, apart from some minor artifacts, the qualitative behaviors of the different models were very similar.

5.2.1 Inverse Dynamics

The inverse dynamics problem seeks motor torques given desired joint angle velocities [86]. While this may sound closely related to our work, inverse dynamics is primarily studied and applied in continuous articulated rigid body dynamics and does not tackle the challenges of hybrid dynamics which involves the choice of discrete modes.

5.2.2 Action Models and Search-Based Planning

Researchers combine action models with search-based planning methods to generate manipulation motions for a wide range of tasks. In the early work of in-hand regrasping [69], grasp gaits are obtained from searching on a grasp map developed from the force closure model. For nonprehensible two-palm manipulation, Erdmann [34] proposed get the set of possible motions by partitioning the configuration space through primitive operations under different contact modes, and generate motion plans by searching. More recently, Chavan-Dafle et al. [21] combined high-level sampling based planning with the motion cones to achieve prehensible in-hand manipulation with external pushes. Hou et al. [49] proposed a fast planning framework using two reorientation motion primitives for object reoritation problems. At the higher level of task and motion planning, complicated sequences of tool-use and manipulation motions are planned through path optimization given defined relations and action operators.

All these works and our work adopt the hierarchical structure of combining low-level task models and high-level planning methods to decrease the amount of required search. Previous works require user-defined action models for different tasks. In contrast, our method doesn't need hand-written motion primitives and thus can be used for a wide range of applications.

5.2.3 Contact-Implicit Trajectory Optimization

The goal of contact-implicit trajectory optimization (CITO) is to simultaneously optimize states, controls, and contact modes. Prior work in contact-implicit trajectory optimization (CITO) can be divided into direct and shooting methods. For example, Önol et al. [87] proposed a direct contact-implicit trajectory optimization method based on a variable smooth contact model and successive convexification. Posa et al. [93] proposed a direct method for solving CITO problems leveraging Sequential Quadratic Programming (SQP). An example of a shooting method, Tassa et al. [105] uses differential dynamic programming (DDP) with a smoothing function

on contact constraints to solve CITO problems. Though they share many commonalities, our work and CITO differ in intended purposes. The goal of our work is to automatically generate a library of contact-rich actions. This distinction allows us to explore solution techniques which may not be useful for CITO algorithms.

5.3 Background

This section reviews basic frictional contact models and their relation to the equations of motion. First, we describe the Coulomb friction model and formulate it as a set of nonlinear complementarity constraints. Next, we introduce the polyhedral approximation to Coulomb friction. Lastly, we conclude by showing how these frictional contact models fit into the equations of motion of a system.

5.3.1 Coulomb Friction Model

Coulomb friction is a familiar friction model with the property that the frictional force is both proportional in magnitude to the normal force and anti-parallel to the sliding direction. The Coulomb model of friction can be succinctly represented using the friction cone and the principle of maximal dissipation. Given a contact force, we define the friction cone to be the convex set

$$\mathcal{F} = \{\lambda_{t_x}, \lambda_{t_y} : \mu\lambda_n^2 - \lambda_{t_x}^2 - \lambda_{t_y}^2 \geq 0\}, \tag{5.1}$$

where μ is the coefficient of friction and $\lambda_{t_x}, \lambda_{t_y}$ are the components of the frictional force in the tangent directions t_x, t_y. The principle of maximal dissipation states that for a tangential sliding velocity $v = [v_{t_x}, v_{t_y}]$, the direction of the frictional force maximizes the dissipated energy, i.e.

$$\lambda_{t_x}, \lambda_{t_y} \in \underset{\lambda_{t_x}, \lambda_{t_y} \in \mathcal{F}}{\arg\max} -\lambda_{t_x} v_{t_x} - \lambda_{t_y} v_{t_y}. \tag{5.2}$$

Note when $v = 0$, any $\lambda_{t_x}, \lambda_{t_y} \in \mathcal{F}$ satisfies equation (5.2), but when $v \neq 0$, $[\lambda_{t_x}, \lambda_{t_y}]$ is anti-parallel to v.

We can express Coulomb friction as the solution to a system of equations comprised of (nonlinear) complementarity constraints. This technique is used to encode Coulomb friction into a form suitable for simulation and optimization.

5.1 Definition. A *complementarity constraint* is a constraint on two variables $a, b \in$

$\mathbb{R}$ such that

$$a \geq 0 \tag{5.3}$$
$$b \geq 0 \tag{5.4}$$
$$ab = 0. \tag{5.5}$$

Under this constraint, either $a > 0$ and $b = 0$, or $b > 0$ and $a = 0$, or $a = b = 0$. We also write the constraint as $0 \leq a \perp b \geq 0$.

We describe how to encode Coulomb friction as complementarity constraints. Let ϕ be a function which returns the minimum separating distance between two (potentially) colliding surfaces. The contact normal force $\lambda_n \in \mathbb{R}_{\geq 0}$ is zero if and only if the distance is trending positive $\phi \succeq 0$. We can approximate this as the complementarity constraint

$$0 \leq \lambda_n \perp \phi + \dot{\phi} \geq 0, \tag{5.6}$$

where $\dot{\phi} = \frac{d\phi}{dt}$ is the separating velocity. Next, we show how to extend this complementarity formulation to Coulomb friction. Similar to normal force, we can write Coulomb friction as a set of equality and complementarity constraints

$$0 = \mu\lambda_n v_x + \lambda_x \sigma \tag{5.7}$$
$$0 = \mu\lambda_n v_y + \lambda_y \sigma \tag{5.8}$$
$$0 \leq \sigma \perp \mu\lambda_n^2 - \lambda_x^2 - \lambda_y^2 \geq 0. \tag{5.9}$$

When the sliding velocity is non-zero, σ evaluates to $\sqrt{v_t^2 + v_o^2}$. However, the use of equation (5.9) leads to a nonlinear complementarity problem (NCP).

5.3.2 Polyhedral Friction Model

We can linearize the Coulomb model of friction by approximating the friction cone with the polyhedral convex cone

$$\mathcal{F}_D = \{D\lambda_f : \mu\lambda_n - e^T\lambda_f \geq 0, \lambda_f \geq 0\}, \tag{5.10}$$

where $D = [d_1, \ldots, d_{n_d}] \in \mathbb{R}^{3\times n_d}$ is a matrix whose columns are normalized polygonal basis vectors, $\lambda_f \in \mathbb{R}^{n_d}$ are the components of frictional force along each basis, and $e = [1, \ldots, 1]^T \in \mathbb{R}^{n_d}$ is a vector of ones. The principle of maximal dissipation can

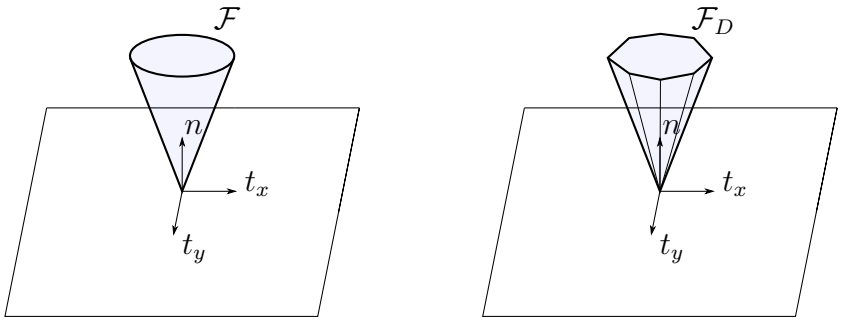


Figure 5-1: (Left) Coulomb friction cone. (Right) Polyhedral friction cone.

be rewritten

$$D\lambda_f = \underset{D\lambda_f \in \mathcal{F}_D}{\arg\max} -v^T D\lambda_f, \tag{5.11}$$

where $v = [v_t, v_o, 0]^T$ is the sliding velocity. When the sliding velocity is non-zero, the resultant frictional force is almost always located at the polygon vertex $\mu d_i \lambda_f$ most anti-parallel to v. We can cast the linearized model as the following linear complementarity problem (LCP)

$$0 \leq \lambda_f \perp D^T v + e\sigma \quad \geq 0 \tag{5.12}$$

$$0 \leq \quad \sigma \perp \mu\lambda_n - e^T\lambda_f \geq 0, \tag{5.13}$$

where the inequalities are evaluated element-wise. This time, $\sigma = -\max -v^T d_i$ when v is non-zero.

5.3.3 Equations of Motion

Consider a system consisting of an object and a manipulator. To properly model this system, we need to define the contact frames and their velocities, and integrate the contact and frictional forces into the dynamic equations of motion. Let the set $\mathcal{K} \subset \mathbb{Z}$ of order $|\mathcal{K}| = k$ enumerate all possible contact pairs between the rigid bodies in our simulation. We can define a minimum seperating distance function $\phi(q_o, q_m) \in \mathbb{R}^k$, where $q_o \in \mathbb{R}^6$ is the object pose, $q_m \in \mathbb{R}^{n_m}$ is the manipulator's state, and the i-th element ϕ_i of returns the distance between the rigid bodies in the i-th contact pair. Let us define a contact frame c_i for each contact pair such that its origin is at the point of contact p_i and its z-axis is pointed along the surface normal $\dot{\phi}_i$. When

a contact pair involves the object, we further require that c_i is rigidly attached to the object and that its z-axis points into the object. Finally, each contact frame c_i has a wrench basis $B_{c_i} = [n_{c_i} D_{c_i}]$ with normal component $n_{c_i} = [0, 0, 1, 0, 0, 0]^T$ and tangential component $D_{c_i} = [D; 0] \in \mathbb{R}^{n_d \times 6}$.

Let $V_o \in \mathbb{R}^6$ be the object's body velocity and $\dot{q}_m \in \mathbb{R}^{n_m}$ be the manipulator's generalized velocity. Let $g_{oc_i} \in SE(3)$ be the transform from the object frame to the i-th contact frame and $g_{sc_i} \in SE(3)$ be the transform from the manipulator's base frame s to the i-th contact frame. We can write the normal and tangential velocities of the contact frame using the following matrices

$$N_o = \begin{bmatrix} n_{c_1}^T \operatorname{Ad}_{g_{oc_1}}^{-1} \\ \vdots \\ n_{c_j}^T \operatorname{Ad}_{g_{oc_j}}^{-1} \\ \mathbf{0} \end{bmatrix}, \qquad N_m = \begin{bmatrix} \mathbf{0} \\ n_{c_1}^T \operatorname{Ad}_{g_{sc_1}}^{-1} J_{sf_1}^s \\ \vdots \\ n_{c_l}^T \operatorname{Ad}_{g_{sc_l}}^{-1} J_{sf_l}^s \end{bmatrix} \tag{5.14}$$

$$T_o = \begin{bmatrix} D_{c_1}^T \operatorname{Ad}_{g_{oc_1}}^{-1} \\ \vdots \\ D_{c_j}^T \operatorname{Ad}_{g_{oc_j}}^{-1} \\ \mathbf{0} \end{bmatrix}, \qquad T_m = \begin{bmatrix} \mathbf{0} \\ D_{c_1}^T \operatorname{Ad}_{g_{sc_1}}^{-1} J_{sf_1}^s \\ \vdots \\ D_{c_l}^T \operatorname{Ad}_{g_{sc_l}}^{-1} J_{sf_l}^s \end{bmatrix}, \tag{5.15}$$

where the matrices N_o, T_o and N_m, T_m map V_o and $\dot{q}_m$, respectively, to the normal and sliding velocities of the contact frame. The matrices have dimensions $N_o \in \mathbb{R}^{k \times 6}$, $N_m \in \mathbb{R}^{k \times n_m}$, $T_o \in \mathbb{R}^{k n_d \times 6}$, $T_m \in \mathbb{R}^{k n_d \times n_m}$.

Following the velocity-impulse formulation of [101], we can write the dynamic equations of motion as

$$\tau = M(\dot{q} - \dot{q}_0) + N^T \lambda_n + T^T \lambda_f + h(C + g) \tag{5.16}$$

$$q = q_0 + h\dot{q}, \tag{5.17}$$

where $N = [N_o, -N_m]$, $T = [T_o, -T_m]$, C are the Coriolis forces, g is the wrench imparted by gravity, and h is the timestep. Combining equations (5.16) with the multi-contact form of the complementarity constraints (5.12-5.13), we arrive at the following mixed LCP (MLCP) for time-stepping dynamical systems with frictional

contact

$$\tau = M(\dot{q} - \dot{q}_0) + N^T\lambda_n + T^T\lambda_f + h(C + g) \tag{5.18}$$

$$q = q_0 + h\dot{q} \tag{5.19}$$

$$0 \leq \lambda_n \perp N(h\dot{q}) - d \quad \geq 0 \tag{5.20}$$

$$0 \leq \lambda_f \perp T(h\dot{q}) + E\sigma \ \geq 0 \tag{5.21}$$

$$0 \leq \sigma \ \perp U\lambda_n - E^T\lambda_f \geq 0 \tag{5.22}$$

Note, this time-stepping program assumes the controls τ are known beforehand.

5.4 Autogenerated Manipulation Primitives

This section describes our approach to autogenerating manipulation primitives. The first step is to establish how to classify manipulation primitives and distinguish them from one another. This is accomplished through the framework of contact modes (Chapter 4). Next we describe how to solve for a manipulation primitive which falls under this classification. We define the manipulation primitive as the solution to an optimization problem. Finally we introduce multiple algorithms for enumerating manipulation primitives, depending on the number of primitives the user desires.

5.4.1 Classification of Primitives

In past research, manipulation primitives are named and defined by the research community studying them. For instance, we can identify papers which study pushing [77][73], pivoting [3, 47, 49], tumbling [74], and whole-body manipulation [98]. However, the labels assigned by researchers, while evocative, are ultimately imprecise. In order to automatically generate manipulation primitives, we first need a mathematically precise method for classifying manipulation primitives. Otherwise, we would not be able to distinguish between different manipulation primitives. This work proposes the use of contact modes (Chapter 4) as programmatic labels for classifying manipulation primitives. Our proposed framework allows us to programmatically generate manipulation primitives by searching the space of contact modes. There are several benefits from the contact mode based classification.

1. Contact modes are partially ordered and have a geometric lattice structure. This allows us to make statements about the primitives (contact modes) such as less than, greater than, equals, or incomparable. If an object is difficult to

$B \in \mathbb{R}^{b \times d}$	– Contact force generators.
$\lambda \in \mathbb{R}^{b}$	– Contact forces.
$\lambda_{z\sigma} \in \mathbb{R}^{t_z + t_\sigma}$	– Active contact forces.
$\lambda_0 \in \mathbb{R}^{b - t_z - t_\sigma}$	– Inactive contact forces.
$N_c \in \mathbb{R}^{n_c \times d}$	– Contacting normal velocity constraints.
$N_s \in \mathbb{R}^{n_s \times d}$	– Separating normal velocity constraints.
$T_z \in \mathbb{R}^{t_z \times d}$	– Sticking tangent velocity constraints.
$T_\sigma \in \mathbb{R}^{t_\sigma \times d}$	– Sliding tangent velocity constraints.

Table 5.1: Additional notation used in Problem 5.3.

manipulation, trying incomparable primitives could be a good way to ensure the robot explores the space of possible actions.

2. Contact modes are complete in the sense that every possible primitive can be described in this framework. Grasping, pushing, throwing, pivoting, peg-in-hole, etc. can all be distinctly labeled according to their contact mode.

3. Contact modes can be programmatically enumerated. In Chapter 4, we introduced an algorithm for efficient contact mode enumeration in 3D.

5.4.2 Optimization of Primitives

We stated before that we classify manipulation primitives based on their contact mode. Given an initial state, target object velocity, and target contact mode, we can define the corresponding primitive as the controls (and trajectory) which best approximate the target object velocity while staying in the target contact mode. Therefore, we need a method to solve for the controls given the target object velocity and target contact mode. We will formulate this as an optimization problem which will be solved at every timestep, i.e. our primitives are generated using a shooting method.

First, we observe that if the contact mode constraint is ignored, then the instantaneous controls which best approximate the target object velocity are the solution to the following mathematical program with complementarity constraints (MPCC), which is in effect, the optimization problem of contact implicit trajectory optimization (CITO) reduced to a single timestep.

5.2 Problem. We can generate a single manipulation primitive by solving the following MPCC

$$\min_x \quad \frac{1}{2}x^T H x + g^T x \tag{5.23}$$
$$\text{s.t.} \qquad q = q_0 + h\dot{q} \tag{5.24}$$
$$0 = h(N^T\lambda_n + T^T\lambda_f + g + \tau) \tag{5.25}$$
$$0 \le \lambda_n \perp hN\dot{q} + d \quad \ge 0 \tag{5.26}$$
$$0 \le \lambda_f \perp hT\dot{q} + E\sigma \quad \ge 0 \tag{5.27}$$
$$0 \le \sigma \perp U\lambda_n - E^T\lambda_f \ge 0, \tag{5.28}$$

where the complementarity constraint $0 \le a \perp b \ge 0$ holds iff a, b are non-negative and $ab = 0$ and equation (5.23) is a quadratic cost on the desired object motion (with additional normalization terms) and equations (5.24-5.28) are the equations of motion under a polyhedral friction cone approximation.

However, our goal is to autogenerate a library of manipulation primitives, labeled by contact mode. When the contact mode is specified, the complementarity constraints in Problem 5.2 are reduced to linear equality and linear inequality constraints. Contact modes partition the nonlinear, discontinuous contact dynamics into regions of smooth dynamical flows. The contact mode determines which contacts are separating, which contacts are maintaining and thus generate normal forces, and which contacts are sliding or sticking and thus the direction of the frictional forces. We use the generator form of the contact dynamics to rewrite Problem 5.2 as a quadratic program.

5.3 Problem. When the contact mode is specified, the MPCC in Problem 5.2 reduces to the following Quadratic Program (QP)

$$\min_x \quad \frac{1}{2}x^T H x + g^T x \tag{5.29}$$
$$\text{s.t.} \qquad q = q_0 + h\dot{q} \tag{5.30}$$
$$0 = h(B^T\lambda + g + \tau) \tag{5.31}$$
$$0 = hN_c\dot{q} + d \tag{5.32}$$
$$0 \le hN_s\dot{q} + d \tag{5.33}$$
$$0 = hT_z\dot{q} \tag{5.34}$$
$$0 \le hT_\sigma\dot{q} \tag{5.35}$$
$$0 \le \lambda_{z\sigma}, \quad 0 = \lambda_0 \tag{5.36}$$

where the contact forces λ_n, λ_f are replaced with contact force generators B, λ to remove the need for the slack variable σ and additional constraints. The new equations state that the

(5.32) contacting normal velocities maintain contact,

(5.33) separating normal velocities do not penetrate,

(5.34) sticking tangent velocities are zero,

(5.35) sliding tangent velocities are in the correct direction,

(5.36) contact force generators are non-negative.

The solution x^* contains the joint torque commands τ^* for achieving the target object motion as closely as possible while staying within the target contact mode. The MPCC in Problem 5.2 and QP in Problem 5.3 are based on linearized frictional dynamics from [101]. There are many other frictional dynamics models in the literature [1]. The method proposed in this paper can be extended to other friction models given an appropriate subsolver.

5.4.3 Full Enumeration Algorithm

Our first proposed algorithm solves a quadratic program for each contact mode. Despite being a brute force approach, this algorithm takes advantage of the lattice structure of the contact modes. A QP solver with good restart capabilities can take advantage of the fact that adjacent contact modes differ by a minimal number of constraints. Rather than solving Problem 5.2, we can reduce the number of variables by solving the QP with the dynamics in contact force generator form.

5.4.4 Adjacent Enumeration Algorithm

Our second proposed algorithm performs a local search on the contact mode lattice to find a locally optimal solution. It traverses the lattice in a depth-first manner using adjacency information until it finds a locally optimal contact mode. In a simulated world, an object can transition instantaneously from one contact mode to another. A viewpoint raised by the adjacent mode algorithm is what if, in reality, an object can only discretely transition through adjacent modes, i.e. transition through the minimal number of constraint changes.

If the adjacent mode algorithm were used to solve for a single optimal contact mode, then we could say that its solution path is most similar to an MPCC active-set

Algorithm 8 The Full Mode Algorithm

```
1: function FULL-MODE-ALG(P, L)
2:   Q ← {ZERO(L)}
3:   while ¬EMPTY(Q) do
4:     u ← POP(Q)
5:     m ← GET-CONTACT-MODE(u)
6:     qp ← GET-QP(P, m)
7:     u.sol ← SOLVE-QP(qp)
8:     for v ∈ SUPER-FACES(u) do
9:       Q ← Q ∪ {v}
10:  return MIN-COST-ELEM(L)
```

method [36]. The lattice structure also suggests that a branch-and-bound technique [12] could be applied to this problem, though we were unable to fully explore this approach. There are some ideas which may be useful in constructing such an algorithm. First, given a child contact mode and its parent contact mode, the feasible velocity space of the parent is strictly larger than that of the child's. Second, the active frictional force bases of the parent are strictly a subset of the child's active frictional force bases. A branch-and-bound approach could start at the top-most mode, explore branches using the velocity cost, and terminate each branch when a dynamically feasible contact mode was found.

5.5 Planning with Autogenerated Manipulation Primitives

We use autogenerated manipulation primitives within the rapidly-exploring random tree (RRT) algorithm to plan manipulation sequences [67]. Autogenerated manipulation primitives are a natural candidate for the connect function within a kinodynamic RRT [68]. We have written pseudo-code describing our RRT planner with autogenerated manipulation primitives (AMP) in Algorithm 10.

The key component of the algorithm is the EXTEND-AMP function. This function takes a target object state, autogenerates a set of manipulation primitives which move the object closer to the target state, and rolls out each primitive towards that state. Each rollout is stored as a new node in the RRT tree. The rollouts are stopped at the nearest state (up to a max distance) which remains statically stable. If multiple hands/fingers are used, then the static stability test can include the hands/fingers.

Algorithm 9 The Adjacent Mode Algorithm

```
1: function ADJ-MODE-ALG(P, 𝓛)
2:   u ← ZERO(𝓛)
3:   while TRUE do
4:     u_prev ← u
5:     for v ∈ SUPER-FACES(u) do
6:       m ← GET-CONTACT-MODE(v)
7:       qp ← GET-QP(P, m)
8:       v.sol ← SOLVE-QP(qp)
9:       if COST(v) ≤ COST(u) then
10:        u ← v
11:    if u = u_prev then
12:      break
13:  return u
```

5.6 Results

5.6.1 Autogenerated Manipulation Primitives in 3D

We tested our algorithms for autogenerating manipulation primitives on various examples in 3D. The results are displayed in Figure 5-2. Similar to the test cases in Chapter 4, we label the test cases as "Box 1" when the target object is a box resting on the ground and we label test cases as "Box 2" when the target object is a box on the ground against the wall. The test cases are labeled as "Point X/Z" if there is a point finger contact on the positive X/Z face. Likewise, the test cases are labeled as "Palm X/Z" if there is a palmar contact (represented as four point contacts) on the positive X/Z face. The case "Palm XX" indicates palmar contacts on both positive and negative X faces.

Our algorithm takes as input a target velocity for the manipulated object. For both "Box 1" and "Box 2", we generated a list of target velocities which represented all the different contact modes of the box. This amounted to 196 target velocities for "Box 1" and 228 target velocities for "Box 2". We then ran our algorithm on each target velocity to autogenerate the manipulation primitives which best approximated the target velocity. We plot the runtimes of each algorithm in Figure 5-2.

From the results, it is clear that the adjacent mode algorithm is around 3x faster than the full mode algorithm. We did not observe a difference in optimal cost of the lowest cost primitive generated by either algorithm. Further improvements in runtime can be easily obtained by using techniques from active set methods. For

Algorithm 10 Rapidly-Exploring Random Tree

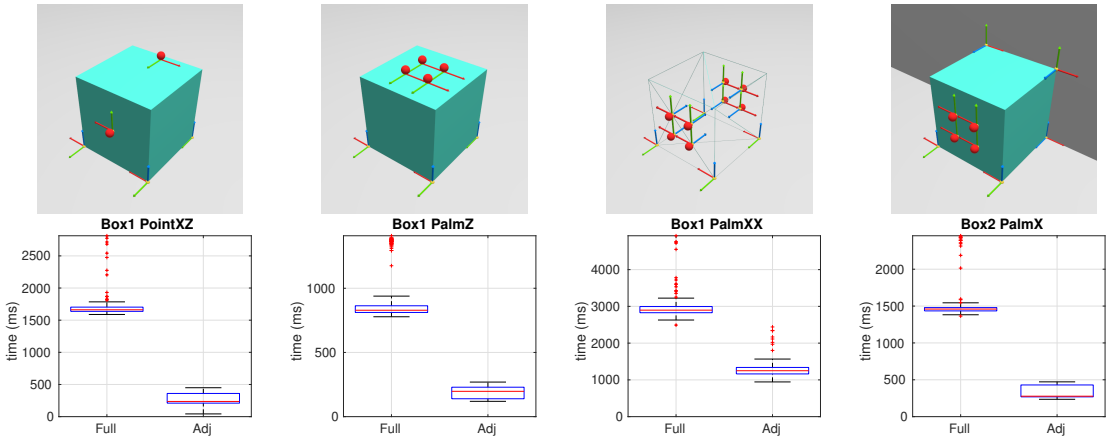
```
1: function RRT-AMP(x, G)
2:   V, E ← {x}, ∅
3:   for i ∈ [0, ..., N] do
4:     x' ← SAMPLE-STATE(G)
5:     X, V, E ← EXTEND-AMP(x', V, E)
6:     if AT-GOAL(X, G) then
7:       break
8:   return PATH(G, V, E)

9: function EXTEND-AMP(x', V, E)
10:  x ← NEAREST(x', V)
11:  h ← SAMPLE-HAND-LOCATION(x)
12:  P ← AUTOGENERATE-MANIPULATION-PRIMITIVES(x, h, x')
13:  X ← ∅
14:  for i ∈ [0, ..., LENGTH(P)] do
15:    x'' ← ROLLOUT(P[i])
16:    X ← X ∪ {x''}
17:    V, E ← ADD-NODE(x'', x, V, E)
18:  return X, V, E
```

instance, the values of the langrange multipliers can be used to select the adjacent mode to expand. This would allow the algorithm to on average only solve 1 QP instead of d when hill climbing to an adjacent mode.

5.6.2 Dexterous Manipulation Planning in 2D

We validate the usefulness of our method in three manipulation problems. For simplicity, we abstract the manipulators as free-floating point contacts in these experiments. For all three problems, we adopt the structure of combining high-level sampling based planning with low-level task models. For high-level planning, we adopt the rapidly exploring random tree (RRT) algorithm [67]. For the problems with top-down 2D (gravity points into the scene), we used the quasi-static motion model from [119].

5.4 Problem. The manipulation primitives in top-down 2D scenes are based on the



(a) Box 1, Point XZ (b) Box 1, Palm Z (c) Box 1, Palm XX (d) Box 2, Palm X

Figure 5-2: Time to enumerate manipulation primitives for various examples.

following MPCC over contact constraints

$$\min_{v} \quad \|v - v_d\|^2 \tag{5.37}$$

$$\text{s.t.} \quad \begin{bmatrix} f_g \\ \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} A & 0 & N_o^T & T_o^T & 0 \\ N_o & -N_m & 0 & 0 & 0 \\ T_o & -T_m & 0 & 0 & E \\ 0 & 0 & U & -E^T & 0 \end{bmatrix} \begin{bmatrix} v \\ \dot{q}_m \\ \lambda_n \\ \lambda_f \\ \sigma \end{bmatrix} \tag{5.38}$$

$$0 \leq \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \perp \begin{bmatrix} \lambda_n \\ \lambda_f \\ \sigma \end{bmatrix} \geq 0. \tag{5.39}$$

The key difference is the inclusion of A in the force-balance. A is the limit-surface model which maps frictional forces to velocities during planar sliding.

This QP uses quasi-static dynamics instead of velocity-impulse dynamics. The connect function samples several finger contact locations and incrementally computes the object motions by iteratively feeding desired object velocity to the QPCC. In the experiements, we used the exponential time version of contact enumeration in 2D.

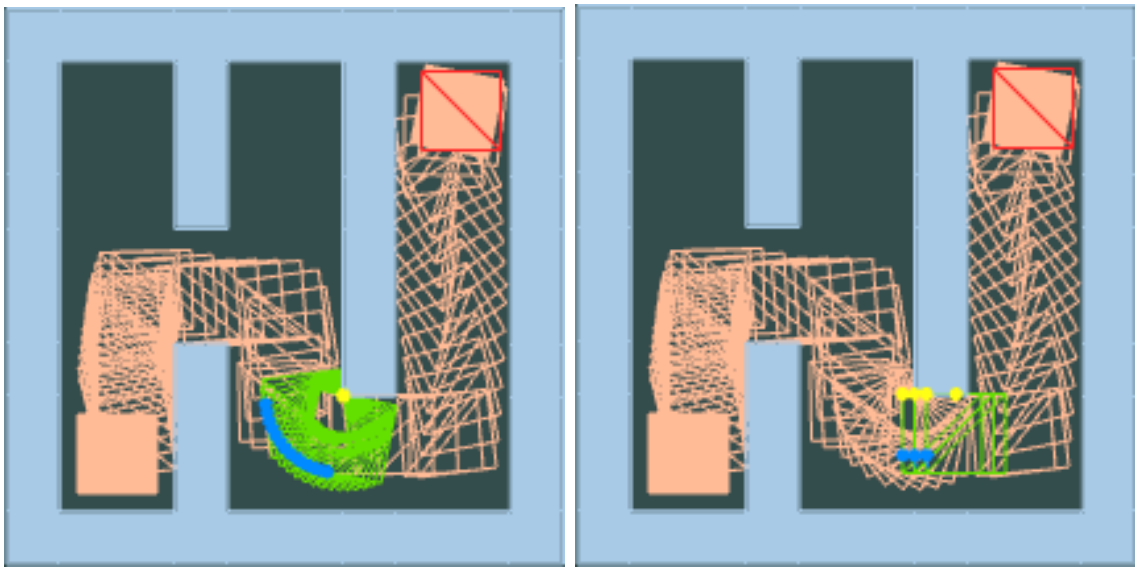


Figure 5-3: The planned trajectory for a maze problem. The red boxes mark the desired goal position; green boxes highlight some actions that exploit external contacts. The blue points are planned finger contact positions; the yellow points are external contacts from collision detection. Right: the pusher uses one external contact to pivot the object; left: after pivoting, the pusher pushes the object along the wall.

5.6.3 Maze

In the Maze example, the goal is to push the block from its start position to the goal position using a single contact pusher. We have no prior knowledge about the shape of the maze. The coefficient of frictions between manipulator/object is 0.8 and wall/object is 0.3. The coefficient matrix for the limit surface is a diagonal matrix $\text{diag}(1, 1, 0.2)$. In Figure 5-3, we show our planned object trajectory which is achievable by a single contact for one maze. As highlighted by the green boxes, when the object is in contact with the environment, our planner exploits the external contacts to get a more stable motion of pivoting-and-pushing to move towards the goal.

We tested our method on randomly initialized 3×3 mazes. Our method solved 27 out of 31 trials where the final positions are within a small threshold of distance to the goal positions. The median of the planning time for all successful trials is 127.15 seconds.

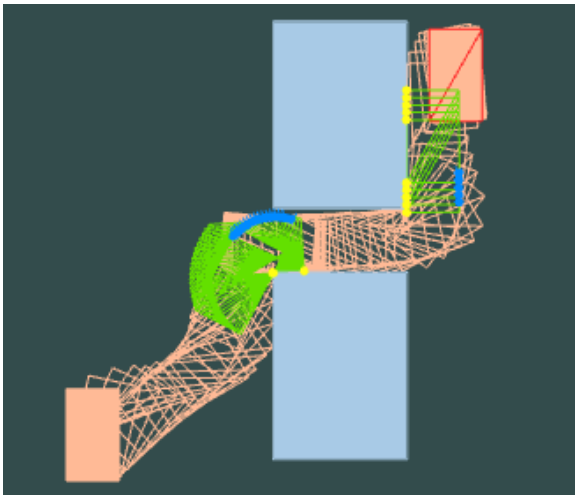


Figure 5-4: The planned trajectory for the narrow passage problem. The red boxes mark the desired goal position; green boxes highlight some actions that exploit external contacts. Blue points are planned finger contact positions; yellow points are external contacts from collision detection.

5.6.4 Narrow Passage

In the Narrow Passage problem, a single contact pusher need to push a rectangle-shaped object to pass a narrow hallway. This problem is like the planar pushing version of peg-in-hole. The shape of the object is 3.5×2 and the width of the hallway is 3. The coefficient of frictions between manipulator/object is 0.8 and wall/object is 0.3. The coefficient matrix for the limit surface is a diagonal matrix $\text{diag}(1, 1, 0.2)$. As highlighted in the green boxes in Figure 5-4, our method generates a pivoting action to reorient the object to get in the narrow gap, and a sliding-along-the wall action to get closer to the goal. The method also generates a wedging motion which is often used in peg-in-hole. The planning time is 14.95 s.

5.6.5 Lift and Flip

The Lift and Flip problem tests the usefulness of our method in scenarios with dynamic (i.e. force) constraints on the capabilities of the robot. In this problem, the objective is to lift a narrow rectangle with unit mass and flip it over 180°. The robot is allowed to make contact at any two collision-free points along the rectangle's long edges. The coefficient of frictions between manipulator/object is 0.5 and ground/object is 1.0. Each manipulator contact can exert a maximum of 4.5 N

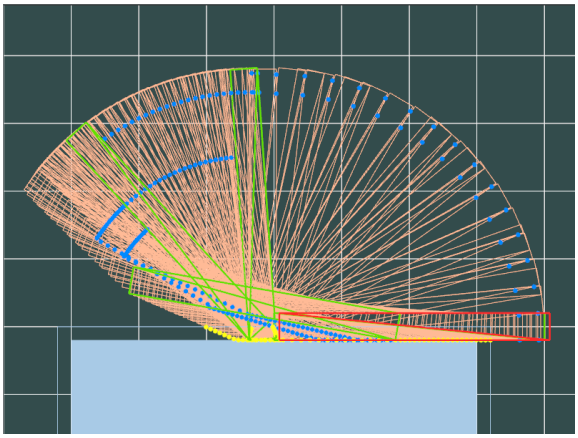


Figure 5-5: Example solution to the Lift and Flip problem generated using RRT + QPCC. Manipulator contact points are blue and environment contacts are yellow. The goal pose (red) is equal to the start pose with a 180° rotation. A few contact transition states are high-lighted in green. The middle, vertical green rectangle is a statically stable state where one manipulator contact switches from the "lifting" edge to the "lowering" edge. In this state, the rectangle is precariously balanced, with the center-of-mass directly above the single ground contact. At the lowest, horizontal green rectangle the planner moves the contacts closer to the rectangles center-of-mass. This enables the manipulator to carry more of the rectangle's weight and reduce the frictional force from the ground, thereby enabling a right sliding motion of the rectangle towards the goal.

in the normal direction. The robot is allowed move contact points only when the rectangle is statically stable (supported by the ground or the other contact). This problem is challenging because the robot must lift using both contact points when the rectangle is nearly horizontal, stably transition contacts from the lifting edge to the lowering edge, and overcome the high ground friction, all while staying within force constraints.

Our method was able to solve 30 out of 30 trials of Lift and Flip, with an average run-time of 5.43 s. Figure 5-5 illustrates an example Lift and Flip solution generated by our method.

5.7 Conclusion

In this chapter, we proposed a method to automatically generate manipulation primitives. Our method focuses on solving for the object velocity, contact modes, contact forces and velocities in a single time-step. Combined with high-leveling sampling based planning algorithms, our method can generate motion plans for different manipulation tasks without any predefined motion primitives. Moreover, in our generated motion plans, several manipulator behaviors are observed: pivoting, sliding, wedging and finger-contact-transition. These kinds of behaviors are often predefined as motion primitives in most manipulation planning work. We believe that our method can be applied to different manipulation tasks and open up opportunities for reactive manipulation in contact-rich environments.

Chapter 6

Conclusion

6.1 Summary

Manipulation primitives have long been studied within the robotics research community. This thesis makes several contributions towards a primitive-centric approach to robotic manipulation.

We studied the quasi-static motion of a planar slider being pushed or pulled through a single contact point assumed not to slip [53]. The main contribution is to derive a method for computing exact bounds on the object's motion for classes of pressure distributions where the center of pressure is known but the distribution of support forces is unknown. The second contribution is to show that the exact motion bounds can be used to plan robotic pulling trajectories that guarantee convergence to the final pose. The planner was tested on the task of pulling an acrylic rectangle to random locations within the robot workspace. The generated plans were accurate to 4.00mm $\pm$ 3.02mm of the target position and 4.35 degrees $\pm$ 3.14 degrees of the target orientation.

We also studied large-scale, multi-object pushing [54]. We introduced a new robotic tabletop rearrangement system, and presented experimental results. The tasks involve rearranging as many as 30 to 100 blocks, sometimes packed with a density of up to 40%. The high packing factor forces the system to push several objects at a time, making accurate simulation difficult, if not impossible. Nonetheless, the system achieves goals specifying the pose of every object, with an average precision of ± 1 mm and $\pm 2^\circ$. The system searches through policy rollouts of simulated pushing actions, using an Iterated Local Search technique to escape local minima. In real world execution, the system executes just one action from a policy, then uses a vision system to update the estimated task state, and replans. The system accepts a

fully general description of task goals, which means it can solve the singulation and separation problems addressed in prior work, but can also solve sorting problems and spell out words, among other things.

These previous attempts demonstrated the value different primitives can bring to different scenarios and motivated the preeminent contribution of this thesis: a general method for autogenerating a manipulation primitive library. Our method is based on the mathematical framework of contact modes. In a hybrid dynamical system with multiple rigid bodies, the relative motions of the contact points on two colliding bodies may be classified as separating, sticking (moving together), or sliding. Given a physical contact model, the active contact modes determine the dynamic equations of motion. Analogously, the set of all possible (valid) contact mode assignments enumerates the set of all possible dynamical flows of the hybrid dynamical system at a given state. Naturally, queries about the kinematics or dynamics of the system can be framed as computations over the set of possible contact modes.

In order to automatically generate manipulation primitives, we developed the theory of contact modes as a mathematically precise method for classifying and distinguishing between different manipulation primitives. This motivated us to investigate efficient ways to compute the set of contact modes. To that end, we developed the first efficient 3D contact mode enumeration algorithm [55]. The algorithm is exponential in the degrees of freedom of the system and polynomial in the number of contacts. The exponential term is unavoidable and an example is provided. Prior work in this area has only demonstrated efficient contact mode enumeration in 2D for a single rigid body [77, 41].

We then used our contact mode enumeration algorithm to create a novel method for automatically generating robotic manipulation primitives. Given a desired object velocity, the current state, and the dynamic equations of motion, our method generates a partially ordered list of manipulation primitives which would best approximate the object motion while satisfying contact dynamics. The generated manipulation primitives are uniquely specified by their contact mode. Our method is based on two key observations. First, specifying the contact mode reduces contact and friction force constraints into equality and inequality constraints. This in turn allows us to solve for the controls using quadratic programming. Second, the contact mode lattice is effectively a graph data structure. This allows us to use graph search techniques when designing our algorithm. Our algorithm walks along the contact mode lattice and solves a quadratic program at each visited node. In this way, we iteratively build a library of distinct manipulation primitives. Finally, we demonstrated our autogenerated manipulation primitives in various manipulation planning problems.

Bibliography

[1] V. Acary, F. Cadoux, C. LemarÃľchal, and J. Malick. A formulation of the linear discrete Coulomb friction problem via convex optimization. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift fÃijr Angewandte Mathematik und Mechanik*, 91(2):155–175, February 2011. ISSN 00442267. doi: 10.1002/zamm.201000073. URL `http://doi.wiley.com/10.1002/zamm.201000073`.

[2] Wisdom C. Agboh and Mehmet R. Dogar. Real-time online re-planning for grasping under clutter and uncertainty. *CoRR*, 2018. URL `https://arxiv.org/abs/1807.09049`.

[3] Y. Aiyama, M. Inaba, and H. Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, volume 1, pages 136–143 vol.1, 1993.

[4] Yasumichi Aiyama, Masayuki Inaba, and Hirochika Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. pages 136–143, Yokohama, Japan, 1993.

[5] JC Alexander and JH Maddocks. Bounds on the friction-dominated motion of a pushed object. *The International Journal of Robotics Research*, 12(3): 231–248, 1993.

[6] Ariel S. Anders, Leslie P. Kaelbling, and Tomas Lozano-Perez. Reliably arranging objects in uncertain domains. In *IEEE Conference on Robotics and Automation (ICRA)*, 2018. URL `http://lis.csail.mit.edu/pubs/anders-icra18.pdf`.

[7] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, March 1996. ISSN 0166-218X.

doi: 10.1016/0166-218X(95)00026-N. URL http://www.sciencedirect.com/science/article/pii/0166218X9500026N.

[8] David Baraff. Coping with friction for non-penetrating rigid body simulation. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 31–41, New York, NY, USA, July 1991. Association for Computing Machinery. ISBN 978-0-89791-436-9. doi: 10.1145/122718.122722. URL https://doi.org/10.1145/122718.122722.

[9] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469âĂŞ483, December 1996. ISSN 0098-3500. doi: 10.1145/235815.235821. URL https://doi.org/10.1145/235815.235821.

[10] Wissam Bejjani, Rafael Papallas, Matteo Leonetti, and Mehmet Remzi Dogar. Planning with a receding horizon for manipulation in clutter using a learned value function. In *CoRR*, 2018. URL https://arxiv.org/abs/1803.08100.

[11] R-P Berretty, Ken Goldberg, Mark H Overmars, and A Frank Van Der Stappen. Orienting parts by inside-out pulling. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1053–1058. IEEE, 2001.

[12] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*, volume 13.

[13] Antonio Bicchi. On the closure properties of robotic grasping. *The International Journal of Robotics Research*, 14(4):319–334, 1995.

[14] Anders Bjorner, Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Gunter M Ziegler. *Oriented matroids*. Number 46. Cambridge University Press, 1999.

[15] Richard W Byrne, Jennifer M Byrne, et al. Manual dexterity in the gorilla: bimanual and digit role differentiation in a natural task. *Animal Cognition*, 4 (3-4):347–361, 2001.

[16] Erin Catto. Box2d, 2013. URL https://box2d.org.

[17] Lillian Chang, Joshua R Smith, and Dieter Fox. Interactive singulation of objects from a pile. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3875–3882. IEEE, 2012.

[18] Lillian Y. Chang, Joshua R. Smith, and Dieter Fox. Interactive singulation of objects from a pile. *2012 IEEE International Conference on Robotics and Automation*, pages 3875–3882, 2012.

[19] Nikhil Chavan-Dafle and Alberto Rodriguez. Prehensile pushing: In-hand manipulation with push-primitives. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6215–6222. IEEE, 2015.

[20] Nikhil Chavan-Dafle, Alberto Rodriguez, Robert Paolini, Bowei Tang, Siddhartha Srinivasa, Michael Erdmann, Matthew T. Mason, Ivan Lundberg, Harald Staab, and Thomas Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, May 2014.

[21] Nikhil Chavan-Dafle, Rachel Holladay, and Alberto Rodriguez. In-hand manipulation via motion cones. *arXiv preprint arXiv:1810.00219*, 2018.

[22] Xianyi Cheng, Eric Huang, Yifan Hou, and Matthew T. Mason. Contact mode guided sampling-based planning for quasistatic dexterousmanipulation in 2d. In *Submitted to Robotics and Automation (ICRA), 2021 IEEE International Conference on*. IEEE, 2021.

[23] Sonia Chernova and Andrea L Thomaz. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121, 2014.

[24] Richard W. Cottle, Jong-Shi. Pang, and Richard E. Stone. *The Linear Complementarity Problem*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, January 2009. ISBN 978-0-89871-686-3. doi: 10.1137/1.9780898719000. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898719000`.

[25] Michael Danielczuk, Jeffrey Mahler, Chris Correa, and Ken Goldberg. Linear push policies to increase grasp access for robot bin picking. *CASE*, 2018.

[26] Vincent Delos and Denis Teissandier. Minkowski Sum of Polytopes Defined by Their Vertices. *Journal of Applied Mathematics and Physics*, 03(01):62, 2015. doi: 10.4236/jamp.2015.31008. URL `http://www.scirp.org/journal/PaperInformation.aspx?PaperID=53577&#abstract`.

[27] Erik D. Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. *CoRR*, abs/1709.01241, 2017. URL `http://arxiv.org/abs/1709.01241`.

[28] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 7, 2011.

[29] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[30] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 2005. ISBN 978-3-642-64873-1 978-3-642-61568-9. OCLC: 890383639.

[31] Andreas Eitel, Nico Hauff, and Wolfram Burgard. Learning to singulate objects using a push proposal network. *"International Symposium on Robotics Research"*, 2017.

[32] Clemens Eppner and Oliver Brock. Planning grasp strategies that exploit environmental constraints. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4947–4952. IEEE, 2015.

[33] Clemens Eppner, Raphael Deimel, José Alvarez-Ruiz, Marianne Maertens, and Oliver Brock. Exploitation of environmental constraints in human and robotic grasping. *The International Journal of Robotics Research*, 34(7):1021–1038, 2015.

[34] Michael Erdmann. An exploration of nonprehensile two-palm manipulation. *The International Journal of Robotics Research*, 1998.

[35] Francisco Facchinei and Jong-Shi Pang. *Finite-dimensional variational inequalities and complementarity problems*. Springer Science & Business Media, 2007.

[36] Masao Fukushima and Paul Tseng. An implementable active-set algorithm for computing a b-stationary point of a mathematical program with linear complementarity constraints. *SIAM J. on Optimization*, 12(3):724âĂŞ739, March 2002. ISSN 1052-6234. doi: 10.1137/S1052623499363232. URL `https://doi.org/10.1137/S1052623499363232`.

[37] Tobias Geyer, Fabio D. Torrisi, and Manfred Morari. Efficient mode enumeration of compositional hybrid systems. *International Journal of Control*, 83(2): 313–329, February 2010. ISSN 0020-7179. doi: 10.1080/00207170903159285. URL https://doi.org/10.1080/00207170903159285.

[38] 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.

[39] Aaron Greenfield, UluÃğ Saranli, and Alfred A. Rizzi. Solving Models of Controlled Dynamic Planar Rigid-Body Systems with Frictional Contact. *The International Journal of Robotics Research*, 24(11):911–931, November 2005. ISSN 0278-3649. doi: 10.1177/0278364905059056. URL https://doi.org/10.1177/0278364905059056.

[40] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016. URL http://www.gurobi.com.

[41] Maximilian Haas-Heger, Christos Papadimitriou, Mihalis Yannakakis, Garud Iyengar, and Matei Ciocarlie. Passive Static Equilibrium with Frictional Contacts and Application to Grasp Stability Analysis. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.064.

[42] Shuai D Han, Nicholas M Stiffler, Athansios Krontiris, Kostas E Bekris, and Jingjin Yu. High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods. In *arXiv preprint arXiv:1705.09180*, 2017.

[43] J.A. Haustein, J. King, S.S. Srinivasa, and T. Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable configurations. In *IEEE International Conference on Robotics and Automation*, 2015.

[44] Joshua A Haustein, Isac Arnekvist, Johannes Stork, Kaiyu Hang, and Danica Kragic. Learning manipulation states and actions for efficient non-prehensile rearrangement planning.

[45] Tucker Hermans, James M Rehg, and Aaron Bobick. Guided pushing for object singulation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4783–4790. IEEE, 2012.

[46] J-B Hiriart-Urruty. Images of connected sets by semicontinuous multifunctions. *Journal of Mathematical Analysis and Applications*, 111(2):407–422, 1985.

[47] Anne Holladay, Robert Paolini, and Matthew T Mason. A general framework for open-loop pivoting. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3675–3681. IEEE, 2015.

[48] P. C. Horak and J. C. Trinkle. On the similarities and differences among contact models in robot simulation. *IEEE Robotics and Automation Letters*, 4 (2):493–499, April 2019. doi: 10.1109/LRA.2019.2891085.

[49] Yifan Hou, Zhenzhong Jia, and Matthew T Mason. Fast planning for 3d any-pose-reorienting using pivoting. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1631–1638. IEEE, 2018.

[50] Yifan Hou, Zhenzhong Jia, and Matthew T Mason. Manipulation with shared grasping. *Under Review*, 2020.

[51] Yifan Hou, Zhenzhong Jia, and Matthew T. Mason. Manipulation with shared grasping, 2020.

[52] Robert D Howe and Mark R Cutkosky. Practical force-motion models for sliding manipulation. *The International Journal of Robotics Research*, 15(6): 557–572, 1996.

[53] Eric Huang, Ankit Bhatia, Byron Boots, and Matt Mason. Exact bounds on the contact driven motion of a sliding object, with applications to robotic pulling. In *Robotics: Science and Systems*, 2017.

[54] Eric Huang, Byron Boots, and Matthew T. Mason. From intractable to tractable: Pushing and sorting clutter using learned sampling heuristics and greedy search. 2018.

[55] Eric Huang, Xianyi Cheng, and Matthew T Mason. Efficient contact mode enumeration in 3d. In *Workshop on the Algorithmic Foundations of Robotics*, 2020.

[56] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[57] Aaron M. Johnson and D. E. Koditschek. Toward a vocabulary of legged leaping. In *2013 IEEE International Conference on Robotics and Automation*, pages 2568–2575, May 2013. doi: 10.1109/ICRA.2013.6630928. ISSN: 1050-4729.

[58] Aaron M. Johnson, Samuel E. Burden, and D. E. Koditschek. A Hybrid Systems Model for Simple Manipulation and Self-Manipulation Systems. *International Journal of Robotics Research*, 35(11):1354–1392, September 2016.

[59] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[60] Volker Kaibel and Marc E. Pfetsch. Computing the Face Lattice of a Polytope from its Vertex-Facet Incidences. *arXiv:math/0106043*, August 2002. URL `http://arxiv.org/abs/math/0106043`. arXiv: math/0106043.

[61] Chung Hee Kim and Jungwon Seo. Shallow-depth insertion: Peg in shallow hole through robotic in-hand manipulation. *IEEE Robotics and Automation Letters*, 4(2):383–390, 2019.

[62] J. King, J.A. Haustein, S.S. Srinivasa, and T. Asfour. Nonprehensile whole arm rearrangement planning with physics manifolds. In *IEEE International Conference on Robotics and Automation*, 2015.

[63] J. King, M. Cognetti, and S.S. Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE International Conference on Robotics and Automation*, 2016.

[64] J. King, V. Ranganeni, and S. S. Srinivasa. Unobservable monte carlo planning for nonprehensile rearrangement tasks. In *IEEE International Conference on Robotics and Automation*, 2017.

[65] Athanasios Krontiris and Kostas E Bekris. Dealing with difficult instances of object rearrangement. In *Robotics: Science and Systems*, 2015.

[66] Michael Laskey, Jonathan Lee, Caleb Chuck, David Gealy, Wesley Hsieh, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, pages 827–834. IEEE, 2016.

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

[68] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. 20(5):378–400, 2001.

[69] Susanna Richmond Leveroni. *Grasp gaits for planar object manipulation.* PhD thesis, Massachusetts Institute of Technology, 1997.

[70] J.E. Lloyd. Fast Implementation of Lemke's Algorithm for Rigid Body Contact Simulation. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4538–4543, April 2005. doi: 10.1109/ROBOT.2005.1570819. ISSN: 1050-4729.

[71] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[72] Kevin M Lynch and Matthew T Mason. Pulling by pushing, slip with infinite friction, and perfectly rough surfaces. *The International journal of robotics research*, 14(2):174–183, 1995.

[73] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6): 533–556, 1996. doi: 10.1177/027836499601500602.

[74] Yusuke Maeda, Tomohisa Nakamura, and Tamio Arai. Motion planning of robot fingertips for graspless manipulation. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2951–2956. IEEE, 2004.

[75] M. T. Mason. On the scope of quasi-static pushing. *ISRR*, 1986.

[76] Matthew T Mason. *Manipulator Grasping and Pushing Operations.* Ph.d. thesis, Massachusetts Institute of Technology, 1982.

[77] Matthew T Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[78] Matthew T. Mason. *Mechanics of Robotic Manipulation.* MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-13396-2.

[79] Matthew T Mason. *Mechanics of robotic manipulation.* MIT press, 2001.

[80] Andrew T. Miller and Peter K. Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11:110–122, 2004.

[81] Muhayyuddin, Mark Moll, Lydia E. Kavraki, and Jan Rosell. Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719, April 2018. doi: 10.1109/LRA.2017.2783445.

[82] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994. ISBN 0849379814.

[83] Katta G Murty and Feng-Tien Yu. *Linear complementarity, linear and nonlinear programming*, volume 3. Citeseer, 1988.

[84] Katta G Murty and Feng-Tien Yu. *Linear complementarity, linear and nonlinear programming*, volume 3. Citeseer, 1988.

[85] Yuzuko C Nakamura, Daniel M Troniak, Alberto Rodriguez, Matthew T Mason, and Nancy S Pollard. The complexities of grasping in the wild. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 233–240. IEEE, 2017.

[86] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

[87] Aykut Özgün Önol, Philip Long, and Taskin Padir. Contact-implicit trajectory optimization based on a variable smooth contact model and successive convexification. *ArXiv*, abs/1810.10462, 2018.

[88] James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

[89] R. Paolini and M. T. Mason. Data-driven statistical modeling of a cube regrasp. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2554–2560, Oct 2016.

[90] Robert Paolini, Alberto Rodriguez, Siddhartha S. Srinivasa, and Matthew T. Mason. A data-driven statistical framework for post-grasp manipulation. 33 (4):600–615, 2014.

[91] Michael A Peshkin and Arthur C Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal on Robotics and Automation*, 4(6):569–598, 1988.

[92] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[93] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[94] BoÅątjan PotoÄĐnik, GaÅąper MuÅąiÄĐ, and Borut ZupanÄĐiÄĐ. A New Technique for Translating Discrete Hybrid Automata into Piecewise Affine Systems. *Mathematical and Computer Modelling of Dynamical Systems*, 10(1):41–57, March 2004. ISSN 1387-3954. doi: 10.1080/13873950412331318062. URL `https://doi.org/10.1080/13873950412331318062`.

[95] QHull Library. QHalf Notes, 2020. URL `http://www.qhull.org/html/qhalf.htm#notes`.

[96] Daniel Ratner and Manfred K Warmuth. Finding a shortest solution for the n× n extension of the 15-puzzle is intractable. In *AAAI*, pages 168–172, 1986.

[97] R Tyrrell Rockafellar and Roger J-B Wets. *Variational Analysis*, volume 317. Springer Science & Business Media, 2009.

[98] Kenneth Salisbury. Whole arm manipulation. In *Proceedings of the 4th international symposium on Robotics Research*, pages 183–189. MIT Press, 1988.

[99] N. Sawasaki, M. Inaba, and H. Inoue. Tumbling objects using a multi-fingered robot. In *Proceedings of the 20th International Symposium on Industrial Robots and Robot Exhibition*, pages 609–616, 1989.

[100] Breannan Smith, Danny M. Kaufman, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Reflections on Simultaneous Impact. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, 31(4):106:1–106:12, 2012.

[101] David E Stewart and Jeffrey 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.

[102] Mike Stilman and James J Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005.

[103] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[104] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012.

[105] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[106] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1168–1175. IEEE, 2014.

[107] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[108] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061. IEEE, 2014.

[109] Richard Tonge, Feodor Benevolenski, and Andrey Voroshilov. Mass splitting for jitter-free parallel rigid body simulation. *ACM Transactions on Graphics (TOG)*, 31(4):105:1–105:8, July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185601. URL `https://doi.org/10.1145/2185520.2185601`.

[110] J. Trinkle, Stephen Berard, and J. S. Pang. A time-stepping scheme for quasistatic multibody systems. *The 6th IEEE International Symposium on Assembly and Task Planning*, pages 174–181, 2005.

[111] Etienne Vouga, Breannan Smith, Danny M. Kaufman, Rasmus Tamstorf, and Eitan Grinspun. All's well that ends well: guaranteed resolution of simultaneous rigid body impact. *ACM Transactions on Graphics (TOG)*, 36(4): 151:1–151:19, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073689. URL `https://doi.org/10.1145/3072959.3073689`.

[112] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *IROS*, pages 4193–4198, 2016.

[113] Gordon T. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3:131–150, 1988.

[114] Kuan-Ting Yu, Maria Bauzá, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. *CoRR*, abs/1604.04038, 2016. URL `http://arxiv.org/abs/1604.04038`.

[115] Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Yu Wang, and Kaiyu Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. *IROS*, abs/1803.05752, 2018.

[116] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *IROS*, abs/1803.09956, 2018.

[117] Jiaji Zhou and Matthew T Mason. Pushing revisited: Differential flatness, trajectory planning and stabilization. *Manuscript in submission*, 2017.

[118] Jiaji Zhou, Robert Paolini, J Andrew Bagnell, and Matthew T Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 372–377. IEEE, 2016.

[119] Jiaji Zhou, J Andrew Bagnell, and Matthew T Mason. A fast stochastic contact model for planar pushing and grasping: Theory and experimental validation. *Robotics: Science and systems XIII*, 2017.

[120] Jiaji Zhou, Robert Paolini, Aaron M Johnson, J Andrew Bagnell, and Matthew T Mason. A probabilistic planning framework for planar grasping under uncertainty. *IEEE Robotics and Automation Letters*, 2(4):2111–2118, 2017.

[121] Gunter M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics. Springer-Verlag, New York, 1995. ISBN 978-0-387-94329-9. doi: 10.1007/978-1-4613-8431-1. URL `https://www.springer.com/gp/book/9780387943299`.