Robust Mobile Manipulation for Robotic Pushing and Nonprehensile Object Transportation

by

Adam William Heins

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Institute for Aerospace Studies
University of Toronto

Robust Mobile Manipulation for Robotic Pushing and Nonprehensile Object Transportation

Adam William Heins
Doctor of Philosophy

Institute for Aerospace Studies
University of Toronto
2026

Abstract

Mobile manipulation is a necessary skill for many tasks ranging from industrial to domestic settings. Humans handle such tasks easily, either using our *prehensile* hands to grasp and carry objects, or using a variety of *nonprehensile* behaviours like pushing, rolling, throwing, and catching. Our goal is to provide robots with similar capabilities, in order to automate the tasks that humans are unwilling or unable to perform. In this thesis, we develop new methods for two nonprehensile manipulation tasks: planar pushing and nonprehensile object transportation. In particular, we develop control and planning algorithms that are robust to uncertain object parameters, including friction, geometry, and mass distribution. We first propose a reactive controller for planar pushing using only force feedback to sense the pushed object, which makes no assumptions about friction, mass distribution, or geometry, except that the object is convex. Next, we develop a model predictive controller for nonprehensile object transportation that is robust to frictional uncertainty and fast enough to avoid moving obstacles, including a thrown ball. Finally, we extend this approach by developing an offline planner that also accounts for inertial parameter uncertainty in the transported objects. All of these approaches are validated through extensive experiments on a real mobile manipulator robot.

Acknowledgements

I would like to thank my supervisor for opportunity and guidance, my thesis committee for patience and feedback, my family for love and support, my friends for humour and diversion, my labmates for camaraderie and curiosity, and my wife for all of the above and more.

Not all those who wander are lost.

J. R. R. Tolkien, *The Fellowship of the Ring*

Engineers like to solve problems. If there are no problems handily available, they will create their own problems.

Scott Adams, *The Dilbert Principle*

Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.

Richard Feynmann

Contents

List of Figures

List of Tables

Acronyms

CoM	center of mass
CW	contact wrench
CWC	contact wrench cone
DOF	degree of freedom
EE	end effector
FK	forward kinematics
FT	force-torque
GIW	gravito-inertial wrench
GN	Gauss-Newton
GP	Gaussian process
IK	inverse kinematics
IL	imitation learning
IP	interior point
KKT	Karush-Kuhn-Tucker
LSTM	long short-term memory
MPC	model predictive control, model predictive controller
NLP	nonlinear program
OCP	optimal control problem
QP	quadratic program, quadratic programming
PCC	polyhedral convex cone
RL	reinforcement learning
ROS	robot operating system
RTI	real-time iteration
SA	support area
SDP	semidefinite program, semidefinite programming
SQP	sequential quadratic programming
TMS	truncated moment sequence
TKMP	truncated K-moment problem
ZMP	zero-moment point

Notation

$\mathbb{R}$	The set of real numbers.
$\mathbb{R}_+$	The set of non-negative real numbers.
$\mathbb{R}^n$	The set of n-dimensional real vectors.
$\mathbb{R}^n_+$	The set of n-dimensional real vectors with non-negative entries.
$\mathbb{N}$	The set of non-negative integers.
$\mathbb{S}^n$	The set of $n \times n$ symmetric matrices.
$\mathbb{S}^n_+$	The set of $n \times n$ symmetric positive semidefinite matrices.
$\mathbb{S}^n_{++}$	The set of $n \times n$ symmetric positive definite matrices.
$SO(n)$	The special orthogonal group.
$SE(n)$	The special Euclidean group.
x	Lower-case symbols in plain font are real scalars.
$\boldsymbol{x}$	Lower-case symbols in bold font are real vectors.
$\boldsymbol{X}$	Upper-case symbols in bold font are real matrices.
$\mathbf{1}_n$	The $n \times n$ identity matrix.
$\leq$	Elementwise inequality: $\boldsymbol{a} \leq \boldsymbol{b}$ means $\boldsymbol{b} - \boldsymbol{a} \in \mathbb{R}^n_+$.
$<$	Strict elementwise inequality.
$\preccurlyeq$	Matrix inequality: $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$ means $\boldsymbol{B} - \boldsymbol{A} \in \mathbb{S}^n_+$.
$\prec$	Strict matrix inequality.
$\dot{(\cdot)}$	Time derivative.

Chapter 1

Introduction

1.1 Motivation

This thesis is fundamentally concerned with the *manipulation* of objects by a robot; that is, we seek to endow robots with the capability to move objects around in a useful way, like humans do. It is particularly appealing to use robots to automate tasks that are undesirable for humans because they are dangerous, difficult, or dull.[1] Examples include moving heavy objects and servicing infrastructure in space, underwater, or in radioactive environments (dangerous, difficult) as well as repetitive tasks like packing containers, stocking shelves, assembling parts, and domestic cleaning (dull). All of these tasks require object manipulation; automating them has the potential to improve overall human quality of life while also reducing labour costs.

Robotic manipulation can be divided into two general categories: prehensile and nonprehensile. *Prehensile* means "capable of grasping or holding" [7]; for example, humans and other primates have prehensile hands, while other animals possess prehensile tails [8], tongues [9], and trunks [10]. Prehensile manipulation thus refers to the case when the manipulated object is securely grasped and has no independent degrees of freedom (DOFs). In contrast, this thesis is focused on *nonprehensile* manipulation [11] (also known as *graspless* manipulation, as the manipulated object is *not* fully or securely grasped), in which the manipulated objects retain some independent DOFs that must be taken into account. The simplest example of nonprehensile manipulation is the act of *pushing*, where the contact between the manipulator and object allows the object to be pushed in one direction but not pulled in the opposite direction. Other examples of nonprehensile manipulation include throwing, catching, rolling, and batting [12]. Nonprehensile manipulation is useful for handling objects that are too heavy, cumbersome, or delicate to easily grasp, or for transporting multiple objects at once by balancing them on a tray (like a restaurant waiter) or stacking them atop one another (see Figure 1.1). However, compared to a prehensile approach, nonprehensile manipulation is challenging due to increased modelling complexity and the hybrid nature of contact mechanics [12].

[1] This is one common variation of the so-called "3Ds" [6].

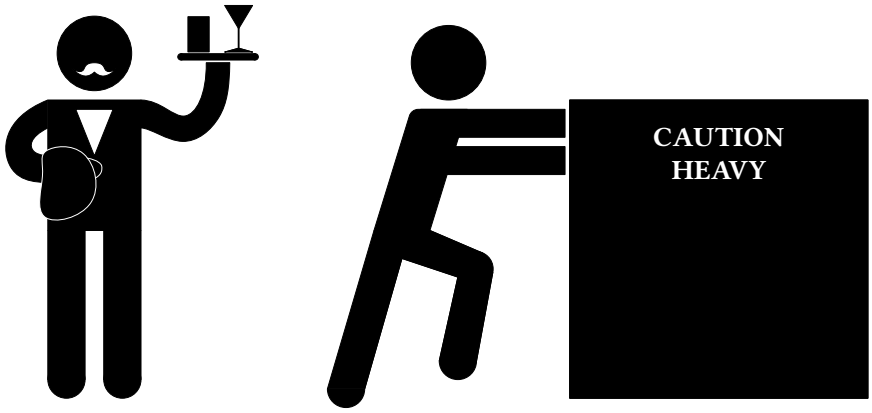

Figure 1.1: Examples of nonprehensile manipulation. *Left:* Waiters use a tray to carry multiple drinks and dishes at once, which is both practical and visually appealing. *Right:* Humans resort to pushing objects that are too large or heavy to pick up.

Research into robotic nonprehensile manipulation began with Mason [13] investigating the mechanics of planar pushing and then continued with Lynch [11], who developed control and planning algorithms for planar pushing as well dynamic manipulation tasks like rolling, throwing, and catching. A rich literature encompassing a variety on nonprehensile manipulation tasks has since developed [12]. Early work on nonprehensile manipulation tended to be restricted to *planar* (i.e., two-dimensional) environments, while more recent work also includes *spatial* (i.e., three-dimensional) environments. In addition, some methods are limited to *quasistatic* motion, in which acceleration and inertial forces are negligible, in contrast to full *dynamic* motion. In this thesis we make novel contributions toward two particular nonprehensile manipulation tasks: quasistatic planar pushing and dynamic nonprehensile object transportation.

This thesis is also focused on *mobile* manipulation. In contrast to traditional static (fixed-base) manipulators, mobile manipulators have no fixed inertial base frame [14]; instead, the robot can move around the environment while manipulating objects within it. Mobile manipulators include underwater [15], wheeled [14], legged (typically humanoid [16] or quadrupedal [17]), aerial [18], and space robots [19]. While our experiments are performed on a wheeled ground robot, the principles are broadly applicable to other types of robots as well. Note also that mobile manipulators need not have the morphology of an arm: for example, a small robot vacuum has no arms but could still usefully manipulate an object by pushing it to a desired location. Mobility greatly expands the workspace of the robot, so that it can perform many more tasks, but introduces new challenges for localization, motion planning complexity, and motion smoothness. In this thesis we focus on the latter two challenges, which motivate two of the main themes of our work: *speed* and *robustness*. Speed refers to both movement speed (of the robot) and computation speed (for control and planning). Industrial manipulators are designed to be able to move at high speed[2], and we want to exploit this potential agility without being limited by slow computational bottlenecks. At

[2]For example, the UR10 arm we use in this thesis has a maximum end effector speed of $5\,\mathrm{m/s}$.

Table 1.1: Assumptions and trade-offs for the different components of the thesis.

Chapter	Motion	Environment	Assumptions	Method
3	Quasistatic (slow)	Planar	Few	Reactive (very fast)
4	Dynamic (fast)	Spatial	Many	Online planning (fast)
5	Dynamic (fast)	Spatial	Some	Offline planning (slower)

the same time, we want to be robust to parameter uncertainty (friction, inertial parameters, etc.) as well as disturbances like vibrations introduced by fast motion. Each component discussed in this thesis—described in more detail below—makes different robustness and speed trade-offs depending on the task and available information, as shown in Table 1.1.

1.2 Outline

This thesis is structured as follows. Chapter 2 provides background information on the physics of robotic manipulation and on the optimization tools that are used in the subsequent chapters. Chapter 3 describes the development of a controller for the task of quasistatic planar pushing, in which a robot pushes an object in a two-dimensional plane with a single point of contact. This relatively simple setting allows us to make few assumptions about the properties of the object being pushed and the available sensing modalities. In particular, we assume that the frictional, inertial, and geometric properties of the object are unknown, except that we assume its shape is convex, and we use only the measured contact force to sense the pushed object—we do not use vision or tactile information. In Chapter 4, we consider a more complex three-dimensional nonprehensile object transportation task known as the *waiter's problem*, which requires the robot to transport objects from one location to another while keeping them balanced on a tray at the end effector (EE) and avoiding static and dynamic obstacles, like a restaurant waiter. We use a whole-body constrained model predictive controller (MPC), which plans using the minimal friction coefficients to provide robustness to frictional uncertainty, but we assume that the geometry and inertial properties of the transported objects are known. In Chapter 5, we consider the case when the inertial properties of the objects are *not* known exactly. We study the set of physically realizable inertial parameters for a given object shape, and extend our MPC to an offline planning method that produces trajectories with verifiable robustness to frictional *and* inertial parameter uncertainty. Some additional results on physically realizable inertial parameters are provided in Appendix A. Finally, Chapter 6 summarizes the thesis and discusses directions for future work.

1.3 Novel Contributions

This thesis makes the following novel contributions:

- **Chapter 3:** We develop the first controller for robotic single-point pushing using *only* force feedback to sense the pushed object, which was published in [1] (see below for the full

citation). We show that the controller successfully pushes objects along both straight and curved paths with single-point contact and no model of the object. We demonstrate the robustness of the controller by simulating pushes using a wide variety of slider parameters and initial states. We also present real hardware experiments in which a mobile manipulator successfully pushes different objects across a room along straight and curved paths, including some with static obstacles. Notably, we do not assume that sufficient friction is available to prevent slip at the contact point; slipping is a natural part of the behaviour of our controller and does not necessarily lead to task failure.

- **Chapter 4:** We propose the first whole-body MPC for a mobile manipulator solving the waiter's problem, which was published in [2]. It is also the first approach to the waiter's problem that handles dynamic obstacles. Compared to existing MPC-based approaches to this problem, which have only been demonstrated on fixed-base arms, our controller optimizes the joint-space trajectory online directly from task-space objectives and constraints, without the use of a higher-level planning step. In addition, the controller uses the minimum statically feasible friction coefficients, which provides robustness to frictional uncertainty, vibration, and other real-world disturbances. When the minimum statically feasible friction coefficients are *zero*, we show that the MPC problem can be solved more efficiently. Furthermore, we present the first demonstrations of the waiter's problem with a real velocity-controlled mobile manipulator transporting up to seven objects; transporting an assembly of stacked objects; and avoiding static and dynamics obstacles, including a thrown volleyball. The EE achieves speeds and accelerations up to $2.0\,\mathrm{m/s}$ and $7.9\,\mathrm{m/s^2}$, respectively. Finally, we also present some further experiments comparing against an additional baseline method, which were not included in [2].

- **Chapter 5:** We develop a planner for nonprehensile object transportation that explicitly handles objects with uncertain centers of mass (CoMs), which extends the framework from Chapter 4 and was published in [3]. Furthermore, we perform a novel theoretical analysis of the constraint satisfaction in the presence of a bounded CoM and any physically realizable inertia matrix, based on moment relaxations [20] (a more computationally efficient version of the analysis is explored in Appendix A). This is the first time that moment relaxations have been used to characterize the set of physically realizable inertial parameters and the first time that this set of parameters has been used to analyze the worst-case constraint satisfaction of a robotic trajectory. Finally, simulations and hardware experiments demonstrating that our proposed robust constraints successfully balance the object—despite using tall objects with high inertial parameter uncertainty—while baseline approaches drop the object.

1.4 Publications

This thesis is based on the following publications:

[1] **A. Heins** and A. P. Schoellig, "Force Push: Robust Single-Point Pushing with Force Feedback," *IEEE Robotics and Automation Letters*, vol. 9, iss. 8, pp. 6856–6863, 2024.

- DOI: 10.1109/LRA.2024.3414180
- Video: http://tiny.cc/force-push
- Software: https://github.com/utiasDSL/force_push

[2] **A. Heins** and A. P. Schoellig, "Keep it Upright: Model Predictive Control for Nonprehensile Object Transportation with Obstacle Avoidance on a Mobile Manipulator," *IEEE Robotics and Automation Letters*, vol. 8, iss. 12, pp. 7986–7993, 2023.

- DOI: 10.1109/LRA.2023.3324520
- Video: http://tiny.cc/keep-it-upright
- Software: https://github.com/utiasDSL/upright

[3] **A. Heins** and A. P. Schoellig, "Robust Nonprehensile Object Transportation with Uncertain Inertial Parameters," *IEEE Robotics and Automation Letters*, vol. 10, iss. 5, 4492–4499, 2025.

- DOI: 10.1109/LRA.2025.3551067
- Video: http://tiny.cc/upright-robust
- Software: https://github.com/utiasDSL/upright

Additional contributions include [4] and [5]. In [4], we provide a system overview of our mobile manipulator using differential inverse kinematics control, including the ability to do force regulation, obstacle avoidance, and manipulability maximization. We also perform an extremely preliminary experiment with force-based pushing, which eventually led to the approach developed in [1] (Chapter 3). In [5], we propose a method for online learning with Gaussian processes that provides robustness guarantees in the face of dynamic model uncertainty; the author of this thesis was specifically responsible for the controller implementation and experiments.

[4] **A. Heins**, M. Jakob, and A. P. Schoellig, "Mobile manipulation in unknown environments with differential inverse kinematics control," in *Proc. Conf. Robots and Vision*, 2021, pp. 64–71.

[5] M. K. Helwa, **A. Heins**, and A. P. Schoellig, "Provably robust learning-based approach for high-accuracy tracking control of Lagrangian systems," *IEEE Robotics and Automation Letters*, vol. 4, iss. 2, pp. 1587–1594, 2019.

Chapter 2

Background

2.1 Rigid Bodies

Our goal is make robots manipulate objects in three-dimensional space. In this thesis, we limit ourselves to the manipulation of *rigid bodies*, which are three-dimensional objects whose shape does not change or deform when subjected to external forces. Many common objects can be well-approximated as rigid bodies.

2.1.1 Geometry and Kinematics

The three-dimensional *pose* of a rigid body consists of its position $\boldsymbol{r} \in \mathbb{R}^3$ and rotation $\boldsymbol{C} \in SO(3)$. We can gather these two quantities together into the *homogeneous transformation matrix*

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{r} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \in SE(3).$$

The spatial velocity of a rigid body is

$$\boldsymbol{\xi} \triangleq \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} \in \mathbb{R}^6,$$

where $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular component and $\boldsymbol{v} \in \mathbb{R}^3$ is the linear component. The spatial acceleration $\dot{\boldsymbol{\xi}}$ is simply the time derivative of the spatial velocity.[1] In the two-dimensional (planar) case, we instead have $\boldsymbol{T} \in SE(2)$ with $\boldsymbol{C} \in SO(2)$ and $\boldsymbol{r} \in \mathbb{R}^2$, and $\boldsymbol{\xi} = [\omega, \boldsymbol{v}^T]^T \in \mathbb{R}^3$ with $\omega \in \mathbb{R}$ and $\boldsymbol{v} \in \mathbb{R}^2$; we will indicate when we are working in a spatial or planar context when needed to avoid confusion.

[1] Note that the linear component of the spatial acceleration is *not* the same as the classical linear acceleration. For more information on spatial vectors and spatial algebra, see Featherstone [21].

2.1.2 The Newton-Euler Equations

A spatial *wrench*

$$\boldsymbol{w} \triangleq \begin{bmatrix} \boldsymbol{\tau} \\ \boldsymbol{f} \end{bmatrix} \in \mathbb{R}^6$$

consists of a torque $\boldsymbol{\tau} \in \mathbb{R}^3$ and a force $\boldsymbol{f} \in \mathbb{R}^3$. The motion of a rigid body is governed by the Newton-Euler equations

$$\boldsymbol{w}_{\mathrm{C}} = \boldsymbol{w}_{\mathrm{GI}}, \tag{2.1}$$

where $\boldsymbol{w}_{\mathrm{C}}$ is the contact wrench (CW) and $\boldsymbol{w}_{\mathrm{GI}}$ is the gravito-inertial wrench (GIW); in this thesis we assume that the only external forces affecting objects are contact forces and gravity. Expressed in a frame that moves with the body (the *body frame*), we have

$$\boldsymbol{w}_{\mathrm{GI}} = \boldsymbol{\Xi}\boldsymbol{\eta} - \mathrm{ad}(\boldsymbol{\xi})^T \boldsymbol{\Xi}\boldsymbol{\xi}, \tag{2.2}$$

where $\boldsymbol{\Xi} \in \mathbb{S}^6_+$ is the object's spatial mass matrix,

$$\mathrm{ad}(\boldsymbol{\xi}) \triangleq \begin{bmatrix} \boldsymbol{\omega}^\times & \boldsymbol{0} \\ \boldsymbol{v}^\times & \boldsymbol{\omega}^\times \end{bmatrix}$$

is the adjoint of $\boldsymbol{\xi}$ with

$$\boldsymbol{a}^\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

forming a skew-symmetric matrix from a vector $\boldsymbol{a} = [a_1, a_2, a_3]^T$ such that $\boldsymbol{a}^\times \boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b}$ for any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$, and $\boldsymbol{\eta} = [\dot{\boldsymbol{\omega}}^T, \dot{\boldsymbol{v}}^T - \boldsymbol{g}^T]^T$ is the difference between the spatial acceleration $\dot{\boldsymbol{\xi}} = [\dot{\boldsymbol{\omega}}^T, \dot{\boldsymbol{v}}^T]^T$ and the body-frame gravitational acceleration $\boldsymbol{g} \in \mathbb{R}^3$. In the two-dimensional case, the planar wrench is $\boldsymbol{w} = [\tau, \boldsymbol{f}^T]^T \in \mathbb{R}^3$ with torque $\tau \in \mathbb{R}$ and force $\boldsymbol{f} \in \mathbb{R}^2$.

2.1.3 Inertial Parameters

The mass $m \in \mathbb{R}_+$, center of mass (CoM) $\boldsymbol{c} \in \mathbb{R}^3$, and inertia matrix

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \in \mathbb{S}^3_+$$

of a rigid body are referred to as its *inertial parameters*. The inertial parameters can be represented in a variety of ways.

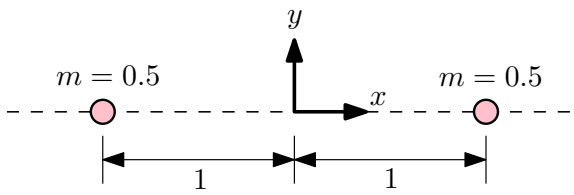

Figure 2.1: Two point masses located along the x-axis. The z-axis comes out of the page.

Spatial Mass Matrix One way of representing the inertial parameters is through the spatial mass matrix we saw in (2.2), defined as

$$\boldsymbol{\Xi} \triangleq \begin{bmatrix} \boldsymbol{I} & m\boldsymbol{c}^{\times} \\ m\boldsymbol{c}^{\times^T} & m\mathbf{1}_3 \end{bmatrix}$$

and expressed with respect to the origin of the body frame.

Pseudo-Inertia Matrix Another way of representing the inertial parameters is using the *pseudo-inertia matrix* [22], defined as

$$\boldsymbol{\Pi} \triangleq \begin{bmatrix} \boldsymbol{S} & m\boldsymbol{c} \\ m\boldsymbol{c}^T & m \end{bmatrix}, \tag{2.3}$$

where $\boldsymbol{S}$ is known as the *second moment matrix* [23]. Given a mass density function $\rho : \mathbb{R}^3 \to \mathbb{R}_+$, we can write $\boldsymbol{S}$ as the integral

$$\boldsymbol{S} = \int_{\mathbb{R}^3} \rho(\boldsymbol{r})\boldsymbol{r}\boldsymbol{r}^T d\boldsymbol{r}. \tag{2.4}$$

In contrast, the inertia matrix $\boldsymbol{I}$ is given by the integral

$$\boldsymbol{I} = -\int_{\mathbb{R}^3} \rho(\boldsymbol{r})\boldsymbol{r}^{\times}\boldsymbol{r}^{\times} d\boldsymbol{r}.$$

The two are related by the (linear) identities

$$\boldsymbol{I} = \operatorname{tr}(\boldsymbol{S})\mathbf{1}_3 - \boldsymbol{S}, \qquad\qquad \boldsymbol{S} = (1/2)\operatorname{tr}(\boldsymbol{I})\mathbf{1}_3 - \boldsymbol{I},$$

where $\operatorname{tr}(\cdot)$ denotes the matrix trace. Qualitatively, $\boldsymbol{S}$ encodes the spread of the mass distribution while $\boldsymbol{I}$ encodes its ability to resist rotation. For example, consider a system of two point masses, each with mass 0.5, placed at ± 1 unit distance from the origin along the x-axis, as shown in Figure 2.1. We have

$$\boldsymbol{I} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad\qquad \boldsymbol{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

where $\boldsymbol{S}$ shows that the mass is spread along the x-axis and $\boldsymbol{I}$ shows that such a spread resists rotation about the y- and z-axes.

The pseudo-inertia matrix representation is convenient because $\boldsymbol{\Pi} \in \mathbb{S}^4_{++}$ is a necessary and sufficient condition for the inertial parameters to be *fully physically consistent* [23]; that is, there exists some mass density function $\rho : \mathbb{R}^3 \to \mathbb{R}_+$ that realizes those inertial parameters, with the relationship

$$\boldsymbol{\Pi} = \int_{\mathbb{R}^3} \rho(\boldsymbol{r}) \tilde{\boldsymbol{r}} \tilde{\boldsymbol{r}}^T d\boldsymbol{r},$$

which is analogous to (2.4) except using the homogeneous representation of the points $\tilde{\boldsymbol{r}} = [\boldsymbol{r}^T, 1]^T$. The strict positive definiteness requirement on $\boldsymbol{\Pi}$ excludes degenerate densities corresponding to point, line, and plane masses. We will return to the idea of physically realizable inertial parameters in Chapter 5 and Appendix A.

Linearity Finally, we note that the Newton-Euler equations (2.1) are linear in the inertial parameters [24], which we can make explicit by writing (2.2) in the form

$$\boldsymbol{w}_{\mathrm{GI}} = \boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta}, \tag{2.5}$$

where $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta}) \in \mathbb{R}^{6\times 10}$ is known as the *regressor matrix* and

$$\boldsymbol{\theta} \triangleq [m, m\boldsymbol{c}^T, I_{xx}, I_{xy}, I_{xz}, I_{yy}, I_{yz}, I_{zz}]^T \in \mathbb{R}^{10} \tag{2.6}$$

is the *inertial parameter vector*, a third representation of the inertial parameters. Note that both $\boldsymbol{\Xi}$ and $\boldsymbol{\Pi}$ are linear functions of $\boldsymbol{\theta}$. Define

$$\boldsymbol{L}(\boldsymbol{\omega}) \triangleq \begin{bmatrix} \omega_x & \omega_y & \omega_z & 0 & 0 & 0 \\ 0 & \omega_x & 0 & \omega_y & \omega_z & 0 \\ 0 & 0 & \omega_x & 0 & \omega_y & \omega_z \end{bmatrix}, \qquad \boldsymbol{\Lambda}(\boldsymbol{\xi}) \triangleq \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{v}^\times & \boldsymbol{L}(\boldsymbol{\omega}) \\ \boldsymbol{v} & \boldsymbol{\omega}^\times & \boldsymbol{0} \end{bmatrix}.$$

Then we have $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta}) = \boldsymbol{\Lambda}(\boldsymbol{\eta}) - \mathrm{ad}(\boldsymbol{\xi})^T \boldsymbol{\Lambda}(\boldsymbol{\xi})$, which is independent of the inertial parameters. This linear relationship also extends to *systems* of rigid bodies like serial manipulators, for which we can find a linear mapping between the joint torques and the link inertial parameters analogous to (2.5).

2.1.4 Zero-Moment Point

In some cases we are interested in keeping robots or objects balanced upright; that is, we do not want them to tip over. Historically, particularly for legged robots, this condition has been encoded by the *zero-moment point (ZMP)* [25], which is the point on the contact surface about which the torques (moments) about the tangential directions are zero. In other words, the ZMP is the point at which the contact torque balances the gravito-inertial torque in the contact plane, which we can

express mathematically as

$$\begin{aligned} \hat{\boldsymbol{n}} \times \boldsymbol{\tau}_{\mathrm{GI}} &= \hat{\boldsymbol{n}} \times (\boldsymbol{r}_{\mathrm{ZMP}} \times \boldsymbol{f}_{\mathrm{C}}), \\ \hat{\boldsymbol{n}}^T (\boldsymbol{r}_{\mathrm{ZMP}} - \boldsymbol{r}_0) &= \mathbf{0}, \end{aligned} \tag{2.7}$$

where $\hat{\boldsymbol{n}}$ is a unit vector along the contact normal, $\boldsymbol{r}_{\mathrm{ZMP}}$ is the position of the ZMP and $\boldsymbol{r}_0$ is an arbitrary point on the contact surface [26]. Rearranging the equations in (2.7), we obtain

$$\boldsymbol{r}_{\mathrm{ZMP}} = \frac{\hat{\boldsymbol{n}} \times \boldsymbol{\tau}_{\mathrm{GI}} + (\hat{\boldsymbol{n}}^T \boldsymbol{r}_0) \boldsymbol{f}_{\mathrm{C}}}{\hat{\boldsymbol{n}}^T \boldsymbol{f}_{\mathrm{C}}}.$$

Recalling from (2.1) that $\boldsymbol{f}_{\mathrm{C}} = \boldsymbol{f}_{\mathrm{GI}}$, we see that the ZMP is entirely a function of the GIW and the contact surface. The support area (SA) of an object with respect to a given contact surface is the convex hull of the contact points on that surface (see Figure 2.3). The object does not tip if the ZMP is inside the SA.

2.2 Robotic Manipulation

2.2.1 Geometry and Kinematics

Robotic manipulation is concerned with making robots (most commonly a robotic arm) move objects in a useful way. Objects are positioned in *task space* $\mathcal{T}$, but the configuration (or *generalized position*) of a robot belongs to its *configuration space* $\mathcal{Q}$, which corresponds to the robot's DOFs [27].[2] We typically use $\mathcal{T} = SE(3)$ for spatial (three-dimensional) problems or $\mathcal{T} = SE(2)$ for planar (two-dimensional) problems.

Forward Kinematics The mapping from configuration space to task space is called *forward kinematics (FK)*, and is typically easy to compute in closed-form. We write the forward kinematic mapping as

$$\boldsymbol{T} = FK(\boldsymbol{q}), \tag{2.8}$$

where $\boldsymbol{T} \in \mathcal{T}$ and $\boldsymbol{q} \in \mathcal{Q}$ is the robot's generalized position. The mapping from joint space to task space on the velocity level is called *differential kinematics*; we have

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{\nu} = \boldsymbol{\xi}, \tag{2.9}$$

where $\boldsymbol{J}$ is the (geometric) Jacobian of the manipulator, $\boldsymbol{\nu} \in \mathbb{R}^{n_\nu}$ is the robot's generalized velocity, and $\boldsymbol{\xi}$ is the spatial velocity of the EE.

Inverse Kinematics For the purposes of planning and control, we are also interested in determining which configuration(s) corresponds to a particular desired EE position or pose. This is called

[2] It is common to refer to the configuration space as the *joint space*, particularly when dealing with robotic arms, because each dimension of $\mathcal{Q}$ typically corresponds to one of the arm's joints.

inverse kinematics (IK), and is typically much harder than forward kinematics—in general, zero, one, or infinite solutions may exist. In this thesis, we typically approach this problem by including the kinematic relationship between task and joint space as a constraint in an optimization problem. We either include the FK equation (2.8) directly as a nonlinear constraint, or use the differential relationship (2.9), which is conveniently linear in $\boldsymbol{\nu}$.

2.2.2 Friction and the Contact Wrench Cone

The interaction between the robot and manipulated objects is governed by frictional contacts, the mechanics of which we describe here.

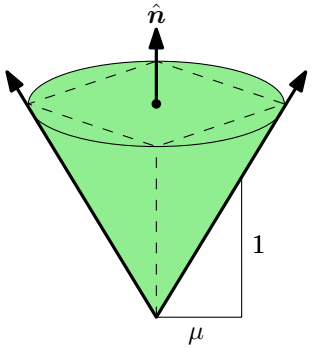

Figure 2.2: The Coulomb friction cone with friction coefficient μ and contact normal $\hat{\boldsymbol{n}}$. The dashed lines show the linearized inner approximation.

Coulomb Friction We assume friction obeys Coulomb's law, which states that the magnitude of the frictional force cannot exceed the magnitude of the normal force scaled by a constant $\mu \in \mathbb{R}_+$, called the *friction coefficient*. We make no distinction between static and kinetic friction. Mathematically, a contact force $\boldsymbol{f} \in \mathbb{R}^3$ must live inside the *friction cone* (see Figure 2.2), satisfying

$$\|\boldsymbol{f}_t\|_2 \leq \mu f_n, \tag{2.10}$$

where $\boldsymbol{f}_t = [f_{t_1}, f_{t_2}]^T$ is the friction force with components $f_{t_1} = \boldsymbol{f}^T\hat{\boldsymbol{t}}_1$ and $f_{t_2} = \boldsymbol{f}^T\hat{\boldsymbol{t}}_2$ along the tangent directions $\hat{\boldsymbol{t}}_1$ and $\hat{\boldsymbol{t}}_2$, and $f_n = \boldsymbol{f}^T\hat{\boldsymbol{n}}$ is the normal force along the contact normal $\hat{\boldsymbol{n}}$. The corresponding feasible pushing velocities are described by the *motion cone* [13]. We typically linearize (2.10) (see e.g. [28]) to obtain the pyramidal approximation

$$\|\boldsymbol{f}_t\|_1 \leq \mu f_n, \tag{2.11}$$

which is an inner approximation of (2.10), meaning that any $\boldsymbol{f}$ satisfying (2.11) also satisfies (2.10).

Limit Surface When two objects are in contact at a *surface* rather than a just a single point, the friction cone can be generalized to the *limit surface* [29]—a closed, convex set that describes the set of feasible friction wrenches that can be supported by the contact. That is, it describes the two frictional forces along the contact's tangential directions as well as their relationship with the frictional torque about the contact normal. The limit surface may be approximated as an ellipsoid [30] or more general models, such the level set of a homogeneous even-degree polynomial [31]. While the limit surface can be useful when modelling a single object, for general *assemblies* of multiple objects, it is easier to reason about the contact wrench cone, which we describe next.

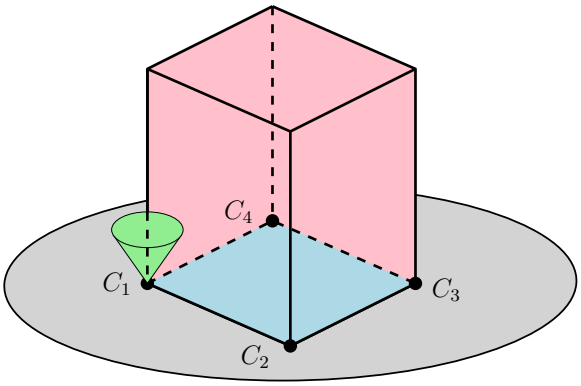

Figure 2.3: A red box in frictional surface contact with a gray tray. The contact *surface* (blue) can be approximated by n_c contact *points* $\{C_i\}_{i=1}^{n_c}$ at its vertices, each with a corresponding contact force $\boldsymbol{f}_i$ that lives inside its friction cone (one friction cone shown in green). The object's support area is the convex hull of its contact points, which in this case coincides with the contact surface.

Contact Wrench Cone Suppose a manipulated object has n_c contact points $\{C_i\}_{i=1}^{n_c}$ with corresponding contact forces $\{\boldsymbol{f}_i\}_{i=1}^{n_c}$ (see Figure 2.3). We can express each friction cone (2.11) as

$$\boldsymbol{F}_i \boldsymbol{f}_i \leq \boldsymbol{0}, \tag{2.12}$$

where

$$\boldsymbol{F}_i = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 1 & -\mu_i \\ 1 & -1 & -\mu_i \\ -1 & 1 & -\mu_i \\ -1 & -1 & -\mu_i \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{t}}_{i_1} & \hat{\boldsymbol{t}}_{i_2} & \hat{\boldsymbol{n}}_i \end{bmatrix}^T.$$

Let $\boldsymbol{\zeta} = [\boldsymbol{f}_1^T, \ldots, \boldsymbol{f}_{n_c}^T]^T$. Then we can write the linearized friction cone for all n_c contact forces in matrix form as

$$\boldsymbol{F}\boldsymbol{\zeta} \leq \boldsymbol{0}, \tag{2.13}$$

where $\boldsymbol{F} = \mathrm{diag}(\boldsymbol{F}_1, \ldots, \boldsymbol{F}_{n_c})$ is a block diagonal matrix. The CW acting on the object is

$$\boldsymbol{w}_{\mathrm{C}} \triangleq \begin{bmatrix} \boldsymbol{\tau}_{\mathrm{C}} \\ \boldsymbol{f}_{\mathrm{C}} \end{bmatrix} = \sum_{i=1}^{n_c} \boldsymbol{G}_i \boldsymbol{f}_i, \tag{2.14}$$

where $\boldsymbol{\tau}_{\mathrm{C}}$ and $\boldsymbol{f}_{\mathrm{C}}$ are the total contact torque and force and

$$\boldsymbol{G}_i = \begin{bmatrix} \boldsymbol{r}_i^{\times} \\ \boldsymbol{1}_3 \end{bmatrix}$$

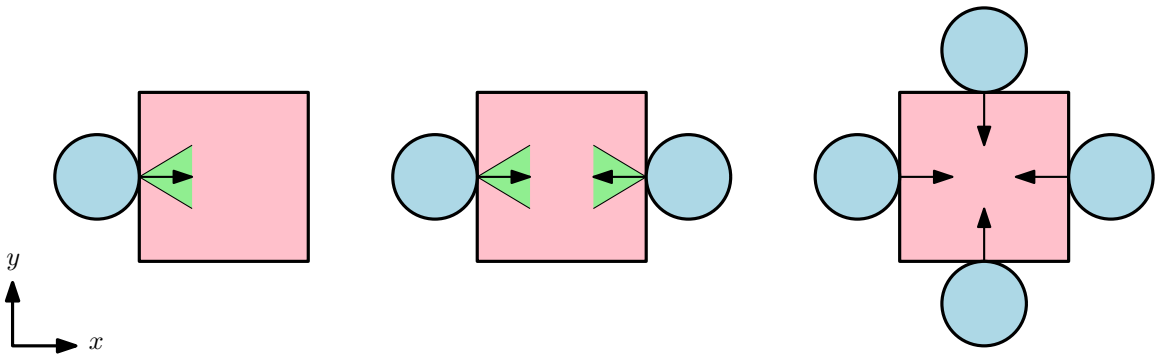

Figure 2.4: Planar examples of frictional contact. *Left:* A single frictional contact can push but not pull: the blue finger can instantaneously apply a force to push the red block in the positive x-direction, but cannot pull it in the negative x-direction, since no forces with negative x-component are contained in the friction cone (green). The block is in neither force nor form closure. *Center:* The red block is held in force closure by the blue fingers. If there was no friction at the contacts, such that the friction cone collapses to a line along the contact normal, then the body would not be in force closure, because the contacts could not resist a force along the y-axis. *Right:* The red block is in form closure, which means the contacts prevent it from moving even without any friction.

is transpose of the ith contact Jacobian[3] with $\boldsymbol{r}_i$ the location of C_i expressed in the body frame. The set $\mathcal{W}_{\mathrm{C}}$ of all possible contact wrenches is known as the *contact wrench cone* (CWC) [32], which is a polyhedral convex cone (PCC) containing all CWs that can be produced by feasible contact forces. We have

$$\mathcal{W}_{\mathrm{C}} = \{\boldsymbol{G}\boldsymbol{\zeta} \mid \boldsymbol{F}\boldsymbol{\zeta} \leq \boldsymbol{0}\} \subseteq \mathbb{R}^6, \tag{2.15}$$

where $\boldsymbol{G} = [\boldsymbol{G}_1, \ldots, \boldsymbol{G}_{n_c}]$ is known as the *grasp matrix*.

2.2.3 Prehensile and Nonprehensile Manipulation

Intuitively, a frictional contact allows the robot to push but not pull. We can think of prehensile manipulation as the case when the manipulated object has a sufficient set of contacts to be "pushed" in any direction. More precisely, prehensile manipulation focuses on generating grasping configurations that ensure a manipulated object is in force or form closure. An object is said to be in *force closure* if the contact forces can resist any applied wrench; that is, if $\mathcal{W}_{\mathrm{C}} = \mathbb{R}^6$ (or $\mathcal{W}_{\mathrm{C}} = \mathbb{R}^3$ in the planar case). An object is said to be in *form closure* if any wrench can be resisted even when the contacts are frictionless (i.e., considering only the contact normals; see Figure 2.4) [27, §12.2]. Form closure implies force closure. In either case, the object does not retain any independent degrees of freedom, so the controller or motion planner can simply treat the grasped object as a rigid attachment to the robot. In contrast, this thesis is concerned with *nonprehensile* manipulation, which refers to the case when the object is *not* in force closure and therefore *retains independent degrees of freedom*. This means that our control and planning algorithms have to model the object

[3] The contact Jacobian maps the spatial velocity of the body to the linear velocity of the contact point.

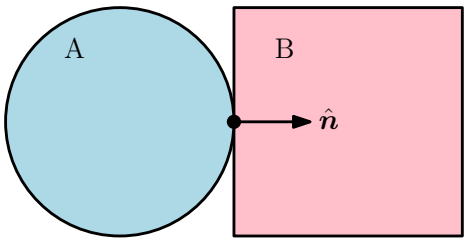

Figure 2.5: Two objects, A and B, in contact at a single point. The contact mode may be sticking, sliding, or separating.

dynamics in addition to those of the robot, while also taking into account the mechanics of contact between them.

Hybrid Dynamics In general, nonprehensile manipulation tasks exhibit *hybrid* dynamics, because the robot-object system dynamics undergo discrete changes depending on the contact *mode*. Each contact point can be in one of three modes: sticking, sliding, or separating. *Sticking* means that the relative velocity between the robot and object at the contact point is zero, so they "stick" together (at that point). *Sliding* means that the robot and object are sliding with respect to each other at that point, and (2.10) is satisfied with equality. *Separating* means that the robot and object are moving away from each other at that point. For example, consider two objects, A and B, in contact at a single point with contact normal $\hat{\boldsymbol{n}}$ pointing out of A and into B, as shown in Figure 2.5. Let $\boldsymbol{v}_a$ be the velocity of the contact point on A, and let $\boldsymbol{v}_b$ be the velocity of the contact point on B. Then we have the following contact mode conditions:

- Sticking: $\boldsymbol{v}_a = \boldsymbol{v}_b$,
- Sliding: $\hat{\boldsymbol{n}}^T(\boldsymbol{v}_a - \boldsymbol{v}_b) = \boldsymbol{0}$, $\hat{\boldsymbol{n}} \times (\boldsymbol{v}_a - \boldsymbol{v}_b) \neq \boldsymbol{0}$,
- Separating: $\hat{\boldsymbol{n}}^T(\boldsymbol{v}_a - \boldsymbol{v}_b) < \boldsymbol{0}$.

Dealing with these hybrid dynamics is one of the main challenges of nonprehensile manipulation. One approach is to use mixed-integer programming (e.g., [33], [34]), but this scales combinatorially in the number of modes.

Force Optimization Problem Instead of reasoning about the full hybrid dynamics, in some applications (such as nonprehensile object transportation in Chapters 4 and 5) we want to constrain the system to stay in a single specific mode. In particular, to stay in the sticking mode, we want to ensure there exist contact forces that can balance a given GIW $\boldsymbol{w}_{\mathrm{GI}}$, such that $\boldsymbol{w}_{\mathrm{GI}} \in \mathcal{W}_{\mathrm{C}}$. This is typically formulated as an optimization problem over the contact forces $\boldsymbol{\zeta}$, and is therefore known as the *force optimization problem* [35]. Contact surfaces are handled by discretizing them into a finite set of contact points at their boundaries. This approach allows us to handle arbitrary contacts between multiple objects, in contrast to constraints based on the limit surface and ZMP, which cannot be applied to closed contact loops (e.g., three objects all in contact with one another).

2.3 Optimization

Each component of this thesis requires the solution of optimization problems to generate control inputs, desired trajectories, or to analyze the effect of parameter choices. We briefly introduce the main problem classes here. For more details, see [36] on convex programming in particular and [37] on numerical optimization algorithms in general.

2.3.1 Quadratic Programming

The first optimization problem class of interest is the *quadratic program (QP)*, which has the form

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \quad & (1/2)\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{p}^T\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}, \end{aligned}$$

where we are trying to minimize a quadratic objective function subject to affine constraints. The values $(\boldsymbol{Q}, \boldsymbol{p}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{G}, \boldsymbol{h})$ are collectively referred to as the problem data, and $(\boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{G})$ are referred to specifically as the *data matrices*. When $\boldsymbol{Q} \in \mathbb{S}^n_+$, the problem is *convex*, which means that any local minimum is also a global minimum. Convex problems can typically be solved very efficiently: intuitively, we can always walk downhill toward the global minimum without worrying about getting stuck in a non-global local minimum along the way. Indeed, QPs can be efficiently solved using a variety of algorithms, including interior point, active set, and first-order methods [38]. In this thesis, we make use of the ProxQP [38] and HPIPM [39] QP solvers.

Differential Inverse Kinematics Control As we will see in Chapter 3, a common use of QPs in robotic manipulation is *differential inverse kinematics control*, a control scheme in which the inverse kinematics problem is solved online on the velocity (differential) level. At each control timestep, we compute the commanded generalized velocity by solving a QP that looks something like

$$\begin{aligned} \underset{\boldsymbol{\nu}}{\text{minimize}} \quad & (1/2)\|\boldsymbol{\nu}\|_2^2 \\ \text{subject to} \quad & \boldsymbol{J}\boldsymbol{\nu} = \boldsymbol{\xi}, \\ & \boldsymbol{\nu}_{\min} \leq \boldsymbol{\nu} \leq \boldsymbol{\nu}_{\max}, \end{aligned} \tag{2.16}$$

where in this case we are trying to find the joint velocities with minimum 2-norm that achieve some given task-space velocity $\boldsymbol{\xi}$ while respecting some limits on the generalized velocity.[4] QPs also form the key building block of a powerful approach for solving nonlinear programs known as *sequential quadratic programming*, which we will discuss below.

[4] Without the limits on $\boldsymbol{\nu}$, (2.16) is an equality-constrained QP and has the closed-form solution $\boldsymbol{\nu}^\star = \boldsymbol{J}^+\boldsymbol{\xi}$, where $\boldsymbol{J}^+ = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}$ is the (right) pseudo-inverse of $\boldsymbol{J}$.

2.3.2 Optimal Control

We will also consider finite-horizon optimal control problems (OCPs) of the form

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{u}}{\text{minimize}} \quad & \ell_f(\boldsymbol{x}_K) + \sum_{k=0}^{K-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0 \\ & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, \ldots, K-1, \\ & \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k) \leq \boldsymbol{0}, \quad k = 0, \ldots, K-1, \\ & \boldsymbol{g}_f(\boldsymbol{x}_K) \leq \boldsymbol{0}, \end{aligned} \tag{2.17}$$

where K is the number of steps in the horizon, $\boldsymbol{x} = [\boldsymbol{x}_0^T, \ldots, \boldsymbol{x}_K^T]^T$ is the state trajectory, $\boldsymbol{u} = [\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{K-1}^T]^T$ is the input trajectory, $\ell(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is the stage cost, $\ell_f(\boldsymbol{x}_K)$ is the terminal cost, $\bar{\boldsymbol{x}}_0$ is the current state, $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is the dynamics equation, $\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k) \leq \boldsymbol{0}$ is the stage constraint, and $\boldsymbol{g}_f(\boldsymbol{x}_K)$ is the terminal constraint. MPC solves OCPs repeatedly online at each control timestep, as we will see in Chapter 4. Alternatively, in Chapter 5, we solve an OCP once offline with a long horizon to generate a full trajectory to subsequently track.

Sequential Quadratic Programming In this thesis, we use sequential quadratic programming (SQP) to solve the OCP (2.17). In general, SQP solves a nonlinear program (NLP) of the form[5]

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & c(\boldsymbol{z}) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z}) \leq \boldsymbol{0}, \end{aligned} \tag{2.18}$$

where we will assume that both c and $\boldsymbol{g}$ are twice-differentiable. The SQP approach is to solve a sequence of QPs approximating (2.18) until the solution converges [37, §18]. At each iterate, we obtain a step $\Delta\boldsymbol{z}$ by solving the QP

$$\begin{aligned} \underset{\Delta\boldsymbol{z}}{\text{minimize}} \quad & (1/2)\Delta\boldsymbol{z}^T \nabla^2 \mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) \Delta\boldsymbol{z} + \nabla c(\boldsymbol{z})^T \Delta\boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{G}(\boldsymbol{z}) \Delta\boldsymbol{z} \leq \boldsymbol{0}, \end{aligned} \tag{2.19}$$

where

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = c(\boldsymbol{z}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{z})$$

is the Lagrangian of (2.18) with Lagrange multipliers $\boldsymbol{\lambda}$, $\nabla^2 \mathcal{L}$ is its Hessian, and

$$\boldsymbol{G}(\boldsymbol{z}) = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}).$$

[5] Equality constraints can be incorporated as double-sided inequalities.

After solving (2.19) for the optimal step $\Delta \boldsymbol{z}^\star$, we update $\boldsymbol{z} \leftarrow \boldsymbol{z} + \alpha \Delta \boldsymbol{z}^\star$ and obtain new Lagrange multipliers by solving the system of Karush-Kuhn-Tucker (KKT) conditions[6] for (2.19). The step size $\alpha \in (0, 1]$ is usually determined using line search or filter methods [37, §18.2].

Gauss-Newton Approximation In this thesis, our OCPs have a least-squares objective function of the form $\boldsymbol{c}(\boldsymbol{z}) = \boldsymbol{e}(\boldsymbol{z})^T \boldsymbol{W} \boldsymbol{e}(\boldsymbol{z})$. We therefore make the *Gauss-Newton (GN) approximation* [40, §8.6]

$$\nabla^2 \mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) \approx \nabla^2 c(\boldsymbol{z}) = \boldsymbol{E}(\boldsymbol{z})^T \boldsymbol{W} \boldsymbol{E}(\boldsymbol{z}),$$

where

$$\boldsymbol{E}(\boldsymbol{z}) = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{z}}(\boldsymbol{z}).$$

The GN approximation ensures the Hessian is positive semidefinite, and therefore (2.19) is convex. Furthermore, we only need first-order differentiability in $\boldsymbol{e}(\boldsymbol{z})$ and $\boldsymbol{g}(\boldsymbol{z})$, and we do not need to worry about the Lagrange multipliers $\boldsymbol{\lambda}$.

Interior Point Methods Another popular class of algorithms for solving NLPs are *interior point (IP)* methods, so-called because they use barrier functions to ensure the current solution iterate is always in the interior of the feasible set. While IP methods are typically faster on larger problems [41], SQP methods are attractive for online control because they can be efficiently warm-started from one control timestep to the next [40, §8.7]. Furthermore, the QP solver we use to solve (2.19) (HPIPM [39]) is designed to exploit the sparse structure of the data matrices arising from the Markov property of the dynamics constraint in (2.17), leading to highly efficient performance.[7]

Real-Time Iteration If we were only solving a single NLP (2.18), it would make sense to iteratively build and solve (2.19) until the solution converges. However, in online settings like MPC, a new but similar problem is solved at each control timestep. Instead of iterating to convergence from the current state at each timestep, it makes more sense to update the state to the most recent estimate at each iteration and solve a new iteration from there. This approach is known as *real-time iteration (RTI)* [42], and it is a key ingredient for fast model predictive control.

Soft Constraints In an online control setting, it is usually unacceptable for the controller to simply fail because the optimal control problem is infeasible. In particular, state constraints often represent desirable conditions that are not always actually achievable, depending on the disturbances to the system [40, §1.2]. In this case, we can *soften* a state constraint by introducing an additional decision variable $s \geq 0$, called a *slack variable*. For example, suppose we have a

[6] The KKT conditions are a set of first-order necessary conditions for optimality; see [36, §5.5] or [37, §12.3] for more information.

[7] This sparse structure is only present when optimizing over both state and input variables, which is known as *multiple shooting*. In contrast, a *single shooting* approach eliminates the state variables using the dynamics constraint and optimizes over only the inputs, resulting in fewer variables but dense data matrices. Multiple shooting typically scales better with longer time horizons; see [40, §8.1].

Figure 2.6: The mobile manipulator used to perform the experiments in this thesis. It consists of a Clearpath Ridgeback mobile base, UR10 arm, Robotiq FT-300 wrist-mounted force-torque sensor, and Robotiq 3-finger gripper. The planar pose of the base is tracked with a Vicon motion capture system.

constraint $g(\boldsymbol{x}) \leq 0$ on the robot state $\boldsymbol{x}$. We soften it to $g(\boldsymbol{x}) \leq s$ and penalize s in the objective using quadratic and/or linear penalties. The slack penalty weight determines the trade-off between feasibility and constraint satisfaction.

2.3.3 Semidefinite Programming

Finally, in Chapter 5 we will also formulate and solve *semidefinite programs (SDPs)*, so-called because their decision variables are semidefinite matrices. A standard form SDP [36, §4.6] is written as

$$\begin{aligned} \underset{\boldsymbol{X} \in \mathbb{S}^n_+}{\text{minimize}} \quad & \operatorname{tr}(\boldsymbol{A}_0 \boldsymbol{X}) \\ \text{subject to} \quad & \operatorname{tr}(\boldsymbol{A}_i \boldsymbol{X}) = b_i, \quad i = 1, \ldots, n_p, \end{aligned}$$

for symmetric data matrices $\boldsymbol{A}_i \in \mathbb{S}^n, i = 0, \ldots, n_p$. SDPs are convex problems that can be solved efficiently using off-the-shelf software such as MOSEK [43], which we use in this thesis via CVXPY [44].

2.4 Experimental Platform

All of our hardware experiments are performed on our mobile manipulator platform shown in Figure 2.6, which consists of a 3-DOF Ridgeback omnidirectional mobile base and and 6-DOF UR10 industrial robot arm. A Robotiq FT300 6-axis force-torque sensor is mounted on the wrist. A Robotiq 3-finger gripper is mounted at the end of the arm. The UR10 provides joint angle measurements from its internal joint encoders at a frequency of 125 Hz. We use a Vicon motion capture system to provide pose measurements of the Ridgeback mobile base (and other objects in the environment)

at 100 Hz. The UR10 accepts joint velocity commands at 125 Hz while the base accepts planar velocity commands, which are converted to wheel velocities by its onboard controller, at 25 Hz. The FT sensor provides wrench measurements at approximately 63 Hz. All of our experiments are run on a standard laptop with eight Intel Xeon CPUs at 3 GHz and 16 GB of RAM running Ubuntu 20.04 with the Preempt_RT real-time patch and communicating with the robot over ethernet with ROS Noetic.

Chapter 3

Robotic Pushing With Force Feedback

In this chapter we investigate one of the most fundamental nonprehensile manipulation tasks: planar pushing. In particular, we are interested in the case when the only measurements of the pushed objects are the contact force at the robot's end effector. It is based on the following publication:

> A. Heins and A. P. Schoellig, "Force Push: Robust Single-Point Pushing with Force Feedback," *IEEE Robotics and Automation Letters*, vol. 9, iss. 8, pp. 6856–6863, 2024.

This is the first single-point pushing scheme that uses *only* force feedback to sense the pushed object.

3.1 Introduction

Pushing is a nonprehensile manipulation primitive that allows robots to move objects without grasping them, which is useful for objects that are too large or heavy to be reliably grasped [45]. In this work we investigate robotic planar pushing with single-point contact using only force feedback to sense the pushed object ("the slider"), in contrast to previous works on single-point pushing, which use visual or tactile sensing; e.g., [30], [46]–[48]. The "pusher" is an omnidirectional mobile robot equipped with a force-torque (FT) sensor to measure the contact force applied by the robot's end effector (EE) on the slider through the contact point. The geometric, inertial (i.e., mass, center of mass (CoM), and inertia), and frictional parameters of the slider are assumed to be unknown. We also assume that online feedback of the slider's pose is not available—the only measurement of the slider is through the contact force. Finally, we assume that the global position of the pusher is known at all times (i.e., the robot can be localized) and that motion is quasistatic.

Single-point pushing is a simple approach that does not require assumptions about the slider's shape. In particular, we envision our force-based approach being useful for pushing unknown objects between distant waypoints, where reliable localization of the object is not available (e.g., due to poor lighting). For example, consider pushing objects through hallways within warehouses or factories. In this case, we are not concerned about tracking a path exactly at all times, but rather

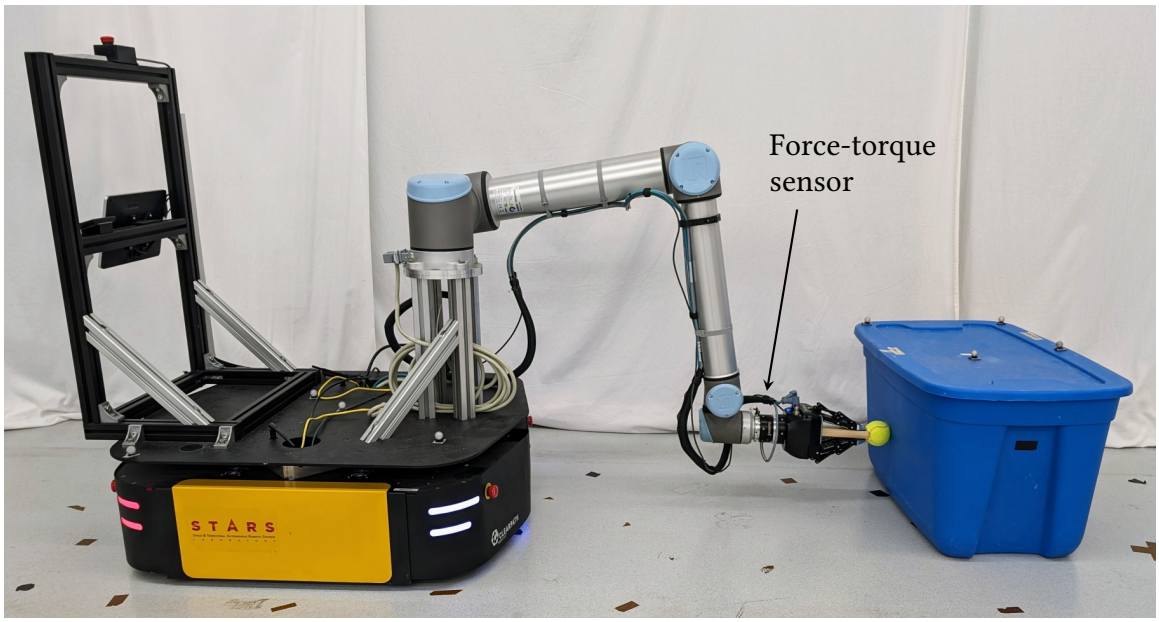

Figure 3.1: Our robot pushing a blue box across the floor using single-point contact. The contact force is measured by a force-torque sensor in the robot's wrist, but no other measurements of the object are provided. A video of our approach is available at http://tiny.cc/force-push.

ultimately pushing the object from one place to another. Our approach can handle collisions with obstacles like walls, using an admittance controller to regulate the contact force. Furthermore, we hope that the insights presented here for force-based pushing can be combined with other sensing modalities to improve overall performance.

The main contribution of this work is a controller for robotic single-point pushing using *only* force feedback to sense the pushed object. We show that it successfully pushes objects along both straight and curved paths with single-point contact and no model of the object. We demonstrate the robustness of the controller by simulating pushes using a wide variety of slider parameters and initial states. We also present real hardware experiments in which a mobile manipulator successfully pushes different objects across a room (see Figure 3.1) along straight and curved paths, including some with static obstacles. Notably, we do not assume that sufficient friction is available to prevent slip at the contact point; slipping is a natural part of the behaviour of our controller and does not necessarily lead to task failure. Our code is available at https://github.com/utiasDSL/force_push.

We first briefly described a controller for single-point pushing based on force feedback in [4], but we have substantially changed and augmented this approach here. In particular, we reformulate the pushing control law, add a term to track a desired path, add admittance control to handle obstacles, provide an analysis of its robustness in simulation, and perform more numerous and challenging real-world experiments.

3.2 Related Work

Research on robotic pushing began with Mason [13], and a recent survey can be found in [45]. Early approaches were typically either open-loop planning methods that rely on line contact for stability [49], [50] or feedback-based approaches using vision [46], [47] or tactile sensing [30], [48]. Tactile sensing is the most similar to our work, though we assume only a single contact force vector is available, rather than the contact angle and normal that a tactile sensor provides. Furthermore, [30] only focuses on stable translational pushes (i.e., where the slider moves in a constant direction) without path-tracking, and [48] assumes a model of the slider is available and that there is sufficient friction to prevent slip.

An FT sensor is used with a fence to orient polygonal parts using a sequence of one-dimensional pushes in [51], which was shown to require less pushes than the best sensorless alternative [52]. Another use of an FT sensor was in [53], where FT measurements are used to detect slip while pushing. In contrast, we do not detect slip, and our controller can still successfully push an object despite (unmeasured) slip.

In [54], a path planning method is proposed for pushing an object while exploiting contact with obstacles in the environment. However, while we rely on an admittance controller to adjust the commanded velocity based on sensed force, the approach in [54] exploits the geometry of the obstacles to provide additional kinematic constraints on the slider's motion. Furthermore, in [54] the obstacles are assumed to be frictionless and the approach is limited to a disk-shaped slider, while we demonstrate multiple slider shapes in contact with obstacles with non-zero friction.

More recent work uses supervised learning to obtain models of the complex pushing dynamics arising from uncertain friction distributions and object parameters. In [55], the pushing dynamics are learned using Gaussian process (GP) regression. In [56] the authors propose Push-Net, an LSTM-based neural network architecture for pushing objects using an image mask representing the slider's pose. In [57], the analytical model of quasistatic pushing from [30] is combined with a learned model, which maps the slider position and depth images to the inputs of the analytical model.

Pushing is also a popular task for reinforcement learning (RL). RL with dynamics randomization is used in [58] to train a pushing policy, which is then transferred to a real robot arm with no fine-tuning. In [59], a multimodal RL policy is trained in simulation, which is hypothesized to better represent the underlying hybrid dynamics of planar pushing compared to previous unimodal policies. These supervised learning and RL works all rely on visual feedback to obtain (some representation of) the object's pose, and are also typically focused on pushing small objects on a tabletop.

Another recent approach is to use MPC for fast online replanning. The GP-based model in [55] is used for MPC in [60]. In [33], hybrid contact dynamics (with hybrid modes corresponding to sticking and sliding of the contact point) are incorporated into the controller using approximate hybrid integer programming. The approach based on mathematical programming with complementary constraints in [34] is more computationally expensive than [33], but can handle complete loss

of contact between objects. In [61], the dynamics of pushing are modelled as a Markov decision process, thus taking stochasticity into account. This allows the controller to adjust its speed based on how much uncertainty can be tolerated for each part of the task. These works all depend on feedback of the slider's pose as well as a (learned or analytical) model of its dynamics. A linear time-varying MPC approach and a nonlinear MPC approach are developed for nonholonomic mobile robots pushing objects with line contact in [62] and [63], respectively. Line contact allows the controller to assume the slider stays rigidly attached to the pusher as long as the friction cone constraints are satisfied, similar to the open-loop planning approaches in [49] and [50]. However, line contact requires the slider to have a flat edge to push against, which excludes, e.g., cylindrical sliders.

Finally, quadruped robots have also been used to push objects across the ground [64] and up slopes [65] while regulating the required pushing force. In [64], friction between the pusher and slider is neglected so that the resulting MPC optimization problem is linear in both contact force and contact location, and it is assumed that the slider's measured pose is available. In [65], an adaptive MPC framework is used to push objects with unknown (and possibly slowly varying) mass and friction parameters; it is the only work that assumes the object model is unknown and only interacts with it through the contact force, like we do. However, in [65] it is assumed that line contact between pusher and slider is present, and curved paths are not considered. In contrast, we consider single-point contact and show that our controller can push the slider along curved paths.

In summary, most methods assume at least visual or tactile feedback of the slider is available or assume line contact. Many also require an a priori model of the slider and may be expensive to evaluate if the hybrid dynamics are taken into account. *None* of these methods perform single-point pushing using *only* force feedback to sense the slider.

3.3 Problem Statement

Our goal is to push an object along a given continuous planar path $\boldsymbol{p}_d(s) : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$, parameterized by the distance $s \geq 0$ from its start. In this work, we use paths made up of straight-line segments and circular arcs. We assume we have a velocity-controlled pusher that is capable of measuring the planar force $\boldsymbol{f} \in \mathbb{R}^2$ it applies on the slider. Our controller must generate a commanded velocity $\boldsymbol{v}_{\text{cmd}} \in \mathbb{R}^2$ for the pusher's EE at each control timestep, which pushes the slider along the desired path.

We assume that the motion of the slider is quasistatic (i.e., inertial forces are negligible), which means that the slider does not move when not in contact with the pusher. We assume that the slider's geometric, inertial, and frictional properties are unknown, except that its shape is convex. We also assume that the slider is a single, non-articulated body (e.g., no wheels or moving joints). Finally, we assume that an approximate initial position of the slider is available, so that the robot can be positioned such that it makes first contact with the slider by moving forward in the direction of the desired path.

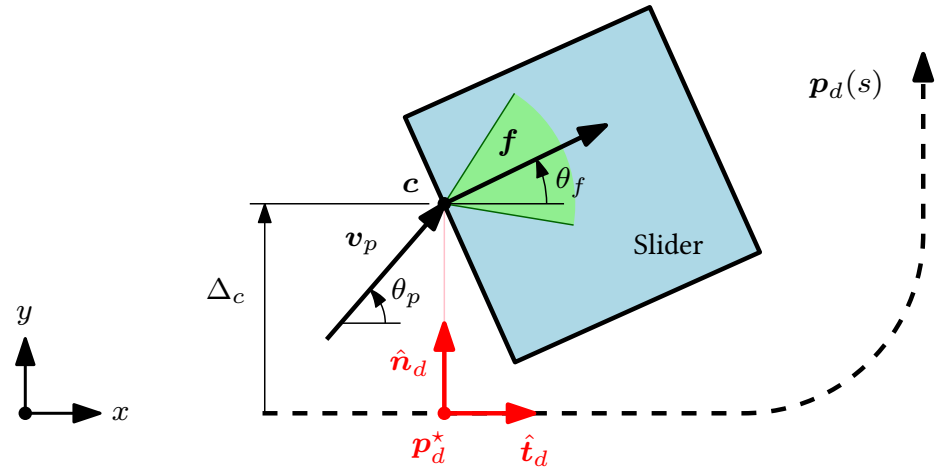

Figure 3.2: Example of our pushing controller with a square slider. The goal is to push the slider along the path $\boldsymbol{p}_d(s)$ by pushing with velocity $\boldsymbol{v}_p$ at the contact point $\boldsymbol{c}$. The resulting contact force $\boldsymbol{f}$ lies inside the friction cone (green). The pushing angle θ_p is proportional to the lateral offset Δ_c and difference between measured force angle and the desired path heading $\Delta_f = \theta_f - \theta_d$; all angles are measured with respect to the global fixed frame. In this example, the pushing velocity $\boldsymbol{v}_p$ will eventually rotate the slider so that the contact force points back toward the desired path. Depending on the contact friction coefficient μ_c, the contact point may slip along the slider's edge over the course of a trajectory.

3.4 Task-Space Pushing Controller

When the properties of the slider are known, we can predict its motion using the equations of motion given in [30]. However, since we do not assume to know the geometry or inertial properties of the slider, we do not rely on a particular mathematical motion model in our controller. Instead, we use the following intuition—similar to Mason's Voting Theorem [13]—to control the system based on contact force measurements. Suppose the object is starting to turn counterclockwise, but we would like the robot to push it straight, as in Figure 3.2. The contact force vector $\boldsymbol{f}$ will also start rotating counterclockwise. To recover, we need to move the robot's EE forward in a direction *even further* counterclockwise to (eventually) push the slider back toward the desired straight-line direction. That is, the EE needs to move so that the line connecting the contact point $\boldsymbol{c}$ and the slider's (unknown) CoM is kept parallel to the desired direction of motion.

This controller is essentially a proportional controller which acts on the pushing angle, except that we actually need to turn further *away* from the desired path in order to ultimately correct the error. This yields a behaviour that trades-off short-term error for long-term performance, which is more typically seen in predictive controllers that consider the effect of their actions over a horizon. A block diagram of the entire controller is shown in Figure 3.3; the different components are described in the remainder of this and the following section.

3.4.1 Stable Pushing and Path-Tracking

Let $\boldsymbol{v}_p \in \mathbb{R}^2$ be the pushing velocity of the contact point, which, together with the contact force $\boldsymbol{f}$, we express in polar coordinates with respect to the global frame as

$$\boldsymbol{v}_p = v \begin{bmatrix} \cos\theta_p \\ \sin\theta_p \end{bmatrix}, \qquad \boldsymbol{f} = \|\boldsymbol{f}\| \begin{bmatrix} \cos\theta_f \\ \sin\theta_f \end{bmatrix},$$

where $v \triangleq \|\boldsymbol{v}_p\|$ is the pushing speed. We will start by taking v to be constant and controlling the pushing angle θ_p. We denote unit vectors with $\hat{(\cdot)}$, such that $\hat{\boldsymbol{f}}$ is the unit vector pointing in the direction of $\boldsymbol{f}$. At the current timestep, let $\boldsymbol{p}_d^\star$ be the closest point on the desired path to the contact point $\boldsymbol{c}$ (in practice we assume $\boldsymbol{c}$ is some fixed point on the pusher's EE), where $\boldsymbol{p}_d^\star \triangleq \boldsymbol{p}_d(s^\star)$ with

$$s^\star = \underset{s \geq \bar{s}}{\operatorname{argmin}} \|\boldsymbol{p}_d(s) - \boldsymbol{c}\|_2. \tag{3.1}$$

The lower bound $\bar{s} \geq 0$ is initialized to zero at the start of the trajectory and then updated at each timestep using $\bar{s} \leftarrow \max(\bar{s}, s^\star)$, which ensures that the values of $s^\star$ obtained from (3.1) are monotonically increasing through time and therefore $\boldsymbol{p}_d^\star$ never moves backward along the path.

Let us attach a Frenet-Serret frame at $\boldsymbol{p}_d^\star$ with direction $\hat{\boldsymbol{t}}_d$ pointing tangent to (along) the path and direction $\hat{\boldsymbol{n}}_d$ orthogonal to the path (see Figure 3.2). We denote the angle from the x-axis to $\hat{\boldsymbol{t}}_d$ as θ_d. Our pushing control law is simply

$$\theta_p = \theta_d + (k_f + 1)\Delta_f + k_c\Delta_c, \tag{3.2}$$

where $k_f, k_c > 0$ are tunable gains, $\Delta_f = \theta_f - \theta_d$ is the signed angle between $\hat{\boldsymbol{f}}$ and $\hat{\boldsymbol{t}}_d$, and $\Delta_c = \hat{\boldsymbol{n}}_d^T(\boldsymbol{c} - \boldsymbol{p}_d^\star)$ is the lateral offset from the path. The Δ_f term steers toward a stable translational pushing direction and the Δ_c term steers toward the desired path. Notice the gain on Δ_f is $(k_f + 1)$; the $+1$ makes the pushing angle θ_p go beyond Δ_f (with respect to θ_d), eventually rotating the object back toward the desired pushing direction. Ultimately, the controller converges to a configuration where the contact force points along the desired path. The controller does not depend explicitly on any slider parameters, and can thus be used to push a variety of unknown objects. Notably, we do not require knowledge of the support friction, pressure distribution, or contact friction, which are often uncertain and subject to change. Furthermore, depending on the contact friction coefficient μ_c, the contact point may slip or stick along the edge of the slider over the course of a successful push.

In many cases we could just take $\boldsymbol{v}_{\text{cmd}} = \boldsymbol{v}_p$ and skip to Section 3.5; however, there are a number of additions we can make to our pushing controller to improve robustness and even handle collisions between the slider and obstacles, which we discuss in the remainder of this section.

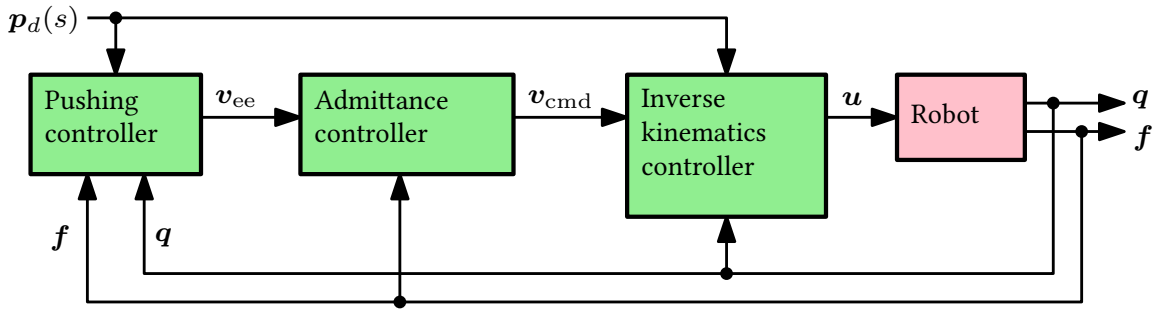

Figure 3.3: Block diagram of the system. The components of our controller (in green) use measurements of the robot's pose $\boldsymbol{q}$ and contact force $\boldsymbol{f}$ to produce joint velocity inputs $\boldsymbol{u}$ that push the slider along a desired path $\boldsymbol{p}_d(s)$.

3.4.2 Contact Recovery

It is possible that following the pushing angle produced by (3.2) will cause the pusher to lose contact with the slider, especially with larger gains k_f and k_c. This can happen if the local curvature of the slider is such that the angle θ_p points away from the current contact edge. Assuming quasistatic motion, we know that the slider does not move after contact is broken. We will say that contact is lost if $\|\boldsymbol{f}\|$ is less than some threshold $f_{\min}$. In the absence of a meaningful force measurement with $\|\boldsymbol{f}\| \geq f_{\min}$, a reasonable approach is to just follow the desired path using the open-loop (with respect to the contact force) control law

$$\theta_o = \theta_d - k_c \Delta_c, \tag{3.3}$$

which just steers the EE toward the desired path. Notice that the sign of $k_c\Delta_c$ is opposite to that in (3.2); this is because here the pusher does not need to move *away* from the path to steer the slider back toward it.

Let us now combine (3.2) and (3.3). Suppose that the pusher loses contact with the slider. Then our approach is to rotate from the current pushing direction θ_p toward the open-loop angle θ_o from (3.3). When contact is made again, such that $\|\boldsymbol{f}\| \geq f_{\min}$, we switch back to (3.2). Thus the combined EE velocity angle θ_{ee} is given by

$$\theta_{\text{ee}} = \begin{cases} \theta_{\text{ee}}^- + \gamma, & \text{if } \|\boldsymbol{f}\| < f_{\min} \\ \theta_p, & \text{otherwise,} \end{cases} \tag{3.4}$$

where θ_{ee}^- is the value of θ_{ee} from the previous control iteration and $\gamma = \theta_o - \theta_{\text{ee}}^-$ with limit $|\gamma| \leq \gamma_{\max}$. The corresponding EE velocity is $\boldsymbol{v}_{\text{ee}} = v\,[\cos\theta_{\text{ee}},\, \sin\theta_{\text{ee}}]^T$.

We have also experimented with contact recovery mechanisms that attempts to "circle back" to the last contact point where $\|\boldsymbol{f}\| \geq f_{\min}$. We found (3.4) to be somewhat more reliable in our experiments, but a more sophisticated contact recovery mechanism is worth further investigation.

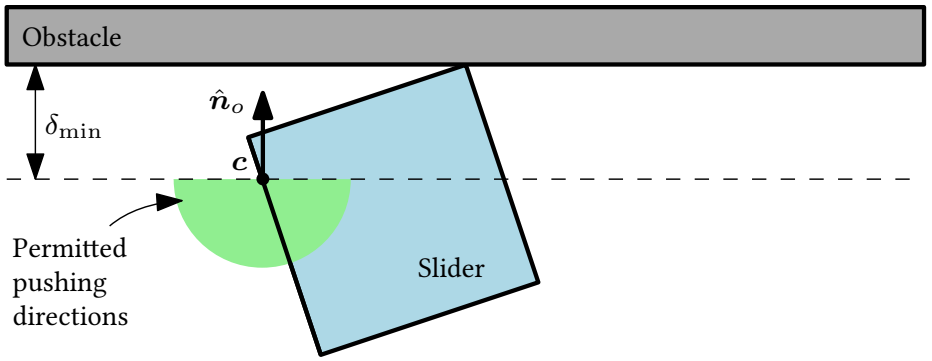

Figure 3.4: Basic obstacle avoidance of the pusher. When the contact point $\boldsymbol{c}$ is within distance $\delta_{\min}$ of an obstacle, the direction of motion is adjusted to not move any closer to it.

3.4.3 Obstacle Avoidance and Admittance Control

So far we have not said anything about obstacle avoidance. Consider the task of pushing an object along a hallway. We will assume that the location of the walls is known to the controller, so the pusher can avoid colliding with them. To do this, if the EE is within some distance $\delta_{\min}$ of an obstacle, then we rotate $\boldsymbol{v}_{\text{ee}}$ by the smallest angle possible such that it no longer points toward the obstacle (i.e., we want $\hat{\boldsymbol{n}}_o^T \boldsymbol{v}_{\text{ee}} \leq 0$, where $\hat{\boldsymbol{n}}_o$ is the unit vector pointing from the EE to the closest point on the obstacle; see Figure 3.4). However, since the geometry and pose of the slider are not known, collisions between the slider and walls cannot be completely avoided, especially if the hallway contains turns, so we need to handle these collisions. It turns out that the pushing angle produced by (3.2) is still useful in many cases when the slider is in contact with obstacles. However, we need to avoid producing excessively large forces by jamming the slider against an obstacle, to prevent damage. We use an admittance controller to adjust the velocity when $\|\boldsymbol{f}\|$ is above a threshold $f_{\max}$. In particular, we compute a velocity offset

$$\boldsymbol{v}_a = \begin{cases} k_a(f_{\max} - \|\boldsymbol{f}\|)\hat{\boldsymbol{f}} & \text{if } \|\boldsymbol{f}\| > f_{\max}, \\ \boldsymbol{0} & \text{otherwise,} \end{cases} \tag{3.5}$$

where $k_a > 0$ is a tunable gain, and finally generate our commanded EE velocity $\boldsymbol{v}_{\text{cmd}} = \boldsymbol{v}_{\text{ee}} + \boldsymbol{v}_a$. The upshot of this admittance control scheme is that the commanded velocity $\boldsymbol{v}_{\text{cmd}}$ is reduced in the direction of $\boldsymbol{f}$ when $\|\boldsymbol{f}\|$ is large, and can even move opposite to $\boldsymbol{f}$. To avoid excessive movement opposite $\boldsymbol{f}$, we found it useful to clamp the magnitude of $\boldsymbol{v}_{\text{cmd}}$ back to at most v.

3.4.4 Force Filtering

The force measurements from our FT sensor are quite noisy, so we employ the exponential smoothing filter

$$\boldsymbol{f}_{\text{filt}} = \beta \boldsymbol{f}_{\text{meas}} + (1 - \beta)\boldsymbol{f}_{\text{filt}}^{-},$$

where $\boldsymbol{f}_{\text{filt}}$ is the filtered force, $\boldsymbol{f}_{\text{meas}}$ is the raw measured force, and $\beta = 1 - \exp(-\delta t/\tau)$ with the δt the time between force measurements and $\tau > 0$ the tunable filter time constant. We actually use this filtering approach in both simulation and experiment; in simulation it helps to smooth out numerical noise in the force values computed by the simulator. It should be assumed that all references to $\boldsymbol{f}$ elsewhere in the paper refer to the filtered value.

3.5 Inverse Kinematics Controller

The pushing controller described in the previous section produces a desired velocity of the EE $\boldsymbol{v}_{\text{cmd}}$ in task-space. We use an inverse kinematics (IK) controller to realize the desired pushing velocity while avoiding collisions between the robot body and known static obstacles. We use a planar omnidirectional mobile robot with motion model $\dot{\boldsymbol{q}} = \boldsymbol{u}$, where $\boldsymbol{q} = [x, y, \theta]^T$ is the robot's configuration, consisting of the position (x, y) and the heading angle θ, and $\boldsymbol{u}$ is the corresponding joint velocity input. We use the quadratic program-based differential inverse kinematics controller

$$\begin{aligned} \boldsymbol{u} = \underset{\boldsymbol{\nu}}{\text{argmin}} \quad & (1/2)\|\boldsymbol{\nu}_d - \boldsymbol{\nu}\|^2 \\ \text{subject to} \quad & \boldsymbol{J}_c(\boldsymbol{q})\boldsymbol{\nu} = \boldsymbol{v}_{\text{cmd}} \\ & -\boldsymbol{\nu}_{\max} \leq \boldsymbol{\nu} \leq \boldsymbol{\nu}_{\max}, \end{aligned} \tag{3.6}$$

where $\boldsymbol{\nu}_d = [0, 0, k_\omega(\theta_d - \theta)]^T$ is designed to minimize the linear velocity and the difference between the robot's heading θ and the path heading θ_d, with $k_\omega > 0$ a tunable gain. The matrix $\boldsymbol{J}_c(\boldsymbol{q})$ is the Jacobian of the contact point, and $\boldsymbol{\nu}_{\max}$ is the joint velocity limit. This IK controller allocates the joint velocities such that the desired EE velocity is achieved exactly while trading off between small linear velocities and rotating to match the path's heading. In the presence of obstacles, we also add constraints to avoid collisions with the robot base. We model the base as a circle with center (x, y). If any part of this circle is within distance $\delta_{\min}$ of an obstacle $\mathcal{O}$, then we add the constraint $[\hat{\boldsymbol{n}}_o^T, 0]\boldsymbol{\nu} \leq 0$ to (3.6), where $\hat{\boldsymbol{n}}_o$ is the unit vector pointing from (x, y) to the closest point on $\mathcal{O}$.

While here we have only considered a mobile robot with three DOFs, which is sufficient to accomplish our pushing task, (3.6) has the structure of a standard IK controller and can be augmented with additional DOFs, objectives, and constraints (see e.g. [4]), as long as the required EE velocity for pushing is achieved.

3.6 Simulation Experiments

We first validate our controller in simulation with Box and Cylinder sliders representing the planar sliders shown in Figure 3.5. We use the PyBullet simulator[1]. The Box has x-y side lengths $\ell = 1\,\text{m}$

[1]We applied a small patch to PyBullet to improve sliding friction behaviour; see https://github.com/bulletphysics/bullet3/pull/4539.

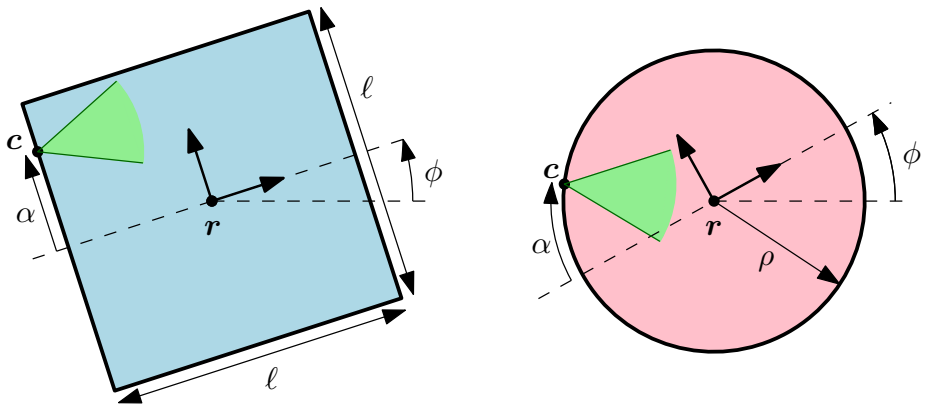

Figure 3.5: Examples of two sliders, located at position $\boldsymbol{r}$ and orientation ϕ in the global frame. The contact with the pusher is located at point $\boldsymbol{c}$, which is distance α along the slider's edge from a reference point. The contact force must lie in the friction cone at the contact point (shown in green).

Table 3.1: Controller parameters for simulation and hardware experiments.

Parameter	Simulation	Hardware	Unit
v	0.1	0.1	$\mathrm{m/s}$
k_f	0.3	0.3	–
k_c	0.1	0.5	$\mathrm{rad/m}$
k_a	0.003	0.003	$\mathrm{s/kg}$
k_ω	–	1	$1/\mathrm{s}$
$f_{\min}$	1	5	N
$f_{\max}$	50	50	N
$\gamma_{\max}$	0.1	0.1	rad
$\delta_{\min}$	0.1	0.1	m
τ	0.05	0.05	s
$\boldsymbol{\nu}_{\max}$	–	$[0.5, 0.5, 0.25]^T$	$[\mathrm{m/s}, \mathrm{m/s}, \mathrm{rad/s}]^T$

and height $12\,\mathrm{cm}$; the Cylinder slider has the same height and radius $\rho = 0.5\,\mathrm{m}$. Each has mass $m = 1\,\mathrm{kg}$ with CoM located at the centroid. The pusher is a sphere of radius $5\,\mathrm{cm}$ and the height of the contact point is $6\,\mathrm{cm}$. For each slider, we assess the robustness of our controller by running simulations in different scenarios, each of which has a different combination of lateral offset, contact offset, slider orientation, contact friction, and slider inertia, as listed in Table 3.2. We use the controller parameters listed in the Simulation column of Table 3.1. The simulation timestep is $1\,\mathrm{ms}$ and the control timestep is $10\,\mathrm{ms}$ (i.e., the controller is run once every 10 simulation steps). The friction coefficient between the slider and floor and obstacles is set to $\mu_o = 0.25$. We also set the contact stiffness and damping of the sliders to 10^4 and 10^2, respectively, for stability during collisions between the slider and wall obstacles.[2]

[2]We also performed simulations with $\mu_o = 0.5$ and with contact stiffness and damping of 10^5 and 10^3, respectively, to ensure we could also handle variation in these parameters. The results are similar to those shown here.

Table 3.2: Initial states and parameters used for simulation. We refer to each combination of states and parameters as a scenario, for a total of $3^5 = 243$ scenarios per slider. The values of the 3×3 inertia matrix $\boldsymbol{I}$ depend on the slider shape, with $\hat{\boldsymbol{I}}$ computed assuming uniform density and $\boldsymbol{I}_{\max}$ computed assuming all mass is concentrated in the outside wall of the Cylinder and in the vertices of the Box.

Parameter	Symbol	Values	Unit
Initial lateral offset	Δ_{c_0}	$-40, 0, 40$	cm
Initial contact offset	α_0	$-40, 0, 40$	cm
Initial orientation	ϕ_0	$-\pi/8, 0, \pi/8$	rad
Contact friction	μ_c	$0, 0.5, 1.0$	$-$
Slider inertia	$\boldsymbol{I}$	$0.5\hat{\boldsymbol{I}}, \hat{\boldsymbol{I}}, \boldsymbol{I}_{\max}$	$\mathrm{kg{\cdot}m^2}$

The position trajectories for each of the $3^5 = 243$ scenarios per slider are shown in Figure 3.6 with straight desired paths. Our controller successfully steers both sliders to the desired path along the positive x-axis for every scenario using the same controller parameters. While k_c could be increased to reduce the deviation from the desired path, we found that a larger k_c did not converge to a stable translational push for all scenarios. The results of the same scenarios are shown in Figure 3.7 with a curved desired path, with and without walls simulating a hallway corner. Without the walls, there is overshoot at the turn before the slider ultimately returns to the desired path. With the walls, the slider collides with the wall and the pushing velocity is adjusted by the admittance controller (3.5) before again eventually returning to the desired path.

Individual sample trajectories are shown in Figure 3.8 and Figure 3.9. In Figure 3.8, we compare two trajectories along the straight-line path to demonstrate how the behaviour of the system changes when sufficient friction to prevent slip at the contact point is not available. The two scenarios are the same except that one has no contact friction ($\mu_c = 0$) and the other has high contact friction ($\mu_c = 1$). With no contact friction, we see that the contact point quickly slides to the middle of the contact edge. In contrast, with high friction, the contact point does not slip and a stable translational push is achieved with a large angle between the contact normal and pushing direction. In both cases, the closed-loop system successfully converges to the desired path along the x-axis. In Figure 3.9, we show a trajectory along the curved path with walls. After colliding with the wall, the pusher actually switches the contact edge[3] for the remainder of the trajectory. The contact point slides along the original edge while attempting to turn the slider, eventually briefly losing contact and circling back toward the path due to (3.4), ultimately making contact again on a different edge of the slider.

[3]Technically it is a contact *face* since the sliders are three-dimensional objects, but we will say edge given our planar context.

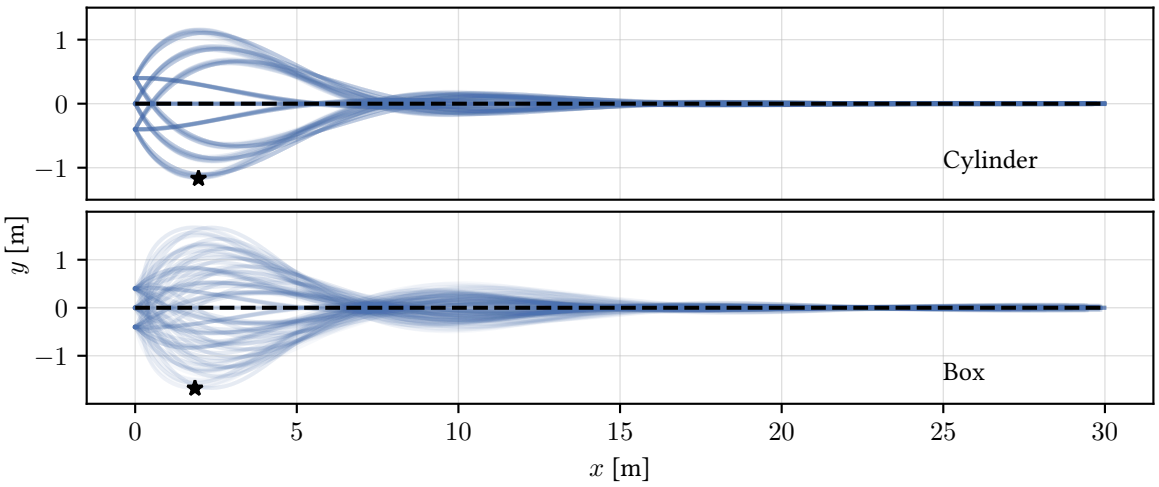

Figure 3.6: Simulated trajectories for all 243 scenarios given in Table 3.2 for the Box and Cylinder sliders shown in Figure 3.5. Each trajectory has a duration of 5 min. All trajectories converge to the desired straight-line path using our control law. The point of maximum deviation from the path for any of the trajectories is marked with a star.

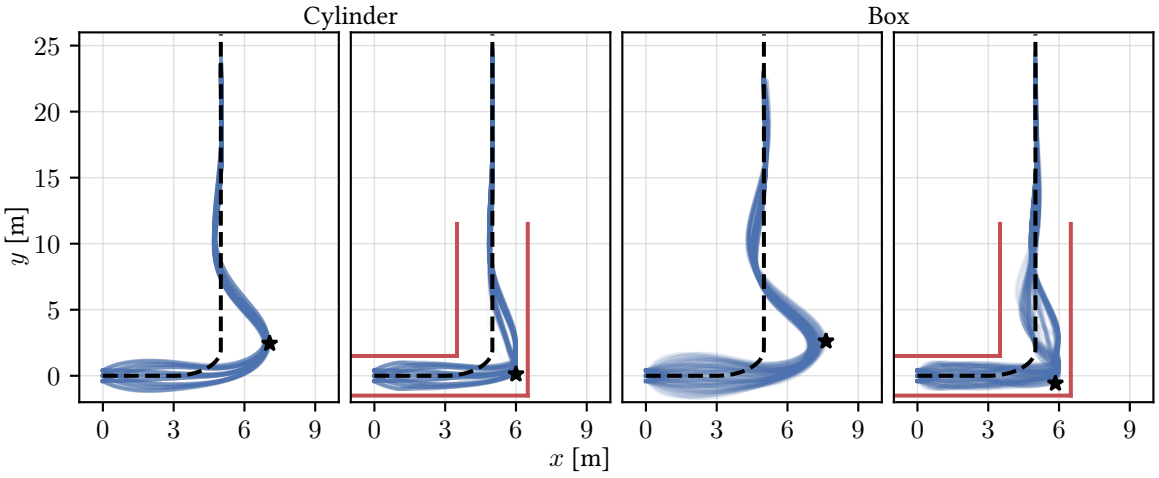

Figure 3.7: Simulated trajectories for the Box and Cylinder sliders along a curved path, with and without walls (in red) simulating a corridor. All trajectories again converge despite collisions with the wall. The point of maximum deviation from the path for any of the trajectories is marked with a star.

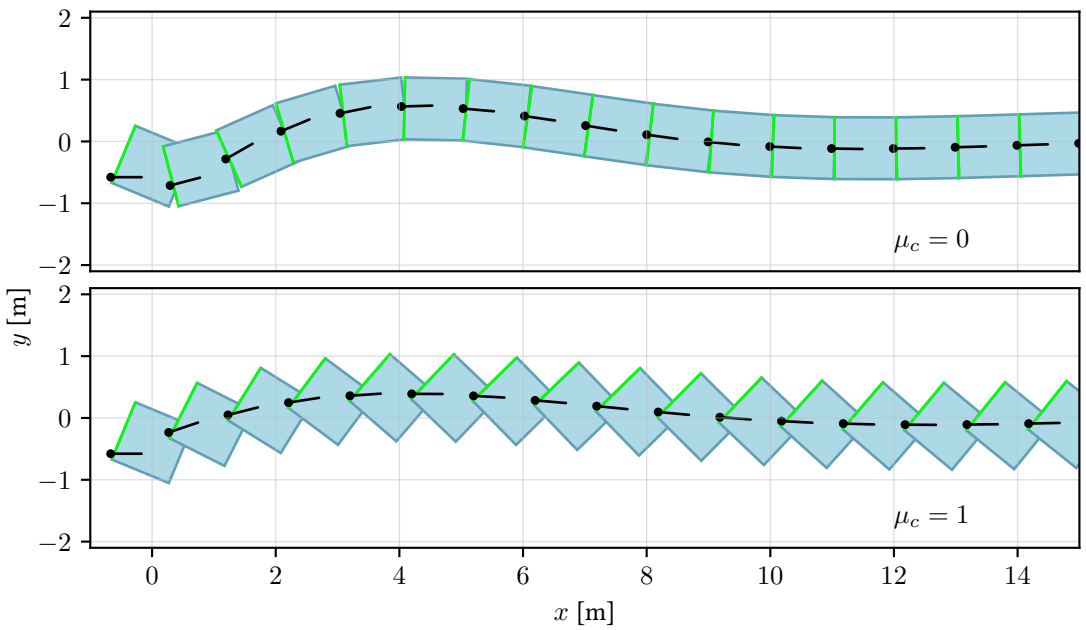

Figure 3.8: Samples of simulated trajectories of the Box slider with the straight desired path along the x-axis. Each image of the slider is taken $10\,\mathrm{s}$ apart. The slider has initial state $(x_0, y_0, s_0, \phi_0) = (0, -40\,\mathrm{cm}, -40\,\mathrm{cm}, -\pi/8)$ and uniform density inertia. Results are shown for low and high contact friction. Pusher and pushing direction are shown in black, initial contact edge is highlighted in green. With $\mu_c = 0$, the contact point ultimately slides to the center of the contact edge; with $\mu_c = 1$, the contact point does not slide.

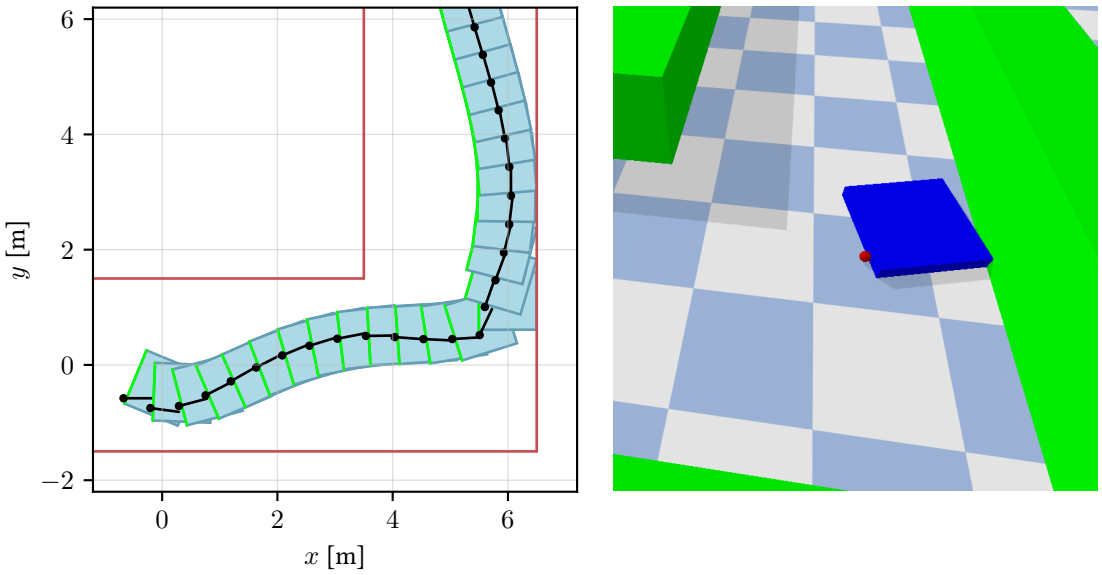

Figure 3.9: *Left:* A sample simulated trajectory of the Box slider along the curved path with walls. Each image of the slider is taken 5 s apart. The slider has initial state $(x_0, y_0, s_0, \phi_0) = (0, -40\,\text{cm}, -40\,\text{cm}, -\pi/8)$, a uniform density inertia, and $\mu_c = 0$. Pusher and pushing direction are shown in black; the walls are red. The initial contact edge of the slider is highlighted in green. After contact with the wall, the pusher switches edges for the remainder of the trajectory. *Right:* An image of the simulation, with red pusher, blue slider, and green walls. Readers are encouraged to watch the video to see this in more detail.

Figure 3.10: *Left:* The "Box" and "Barrel" sliders used for real-world experiments. Each is empty except for $2.25\,\mathrm{kg}$ weights located approximately at the colored circles. The red weights are always present, but we add or remove the green weight to vary the mass and pressure distribution of the box. When both weights are present, we refer to the slider as "Box2"; with a single weight it is called "Box1". The Barrel has radius $20.5\,\mathrm{cm}$, height $56\,\mathrm{cm}$, and total mass $4.5\,\mathrm{kg}$. The Box has width $65\,\mathrm{cm}$, depth $33\,\mathrm{cm}$, and height $43\,\mathrm{cm}$; the total mass of Box1 is $4.8\,\mathrm{kg}$ and of Box2 is $7\,\mathrm{kg}$. The friction coefficients with the ground were estimated to be approximately 0.3–0.4 on average for both objects. *Right:* The robot pushing the Box along the curved trajectory, shortly after the slider first makes contact with the "wall" (we use overturned tables).

3.7 Hardware Experiments

We now demonstrate our controller in real-world experiments. The robot used for pushing is a mobile manipulator consisting of a UR10 arm mounted on a Ridgeback omnidirectional base (see Figure 3.1). The arm's wrist is equipped with a Robotiq FT 300 force-torque sensor. A tennis ball mounted at the EE is used to contact the slider. Since we are only pushing in the x-y plane, we fix the joint angles of the arm and only control the base using (3.6)—the arm is used only for the FT sensor. The base is localized using a Vicon motion capture system that provides pose measurements at $100\,\mathrm{Hz}$, which is also used to record the trajectories of the sliders (but the slider poses are *not* provided to our controller). The FT sensor provides force measurements at approximately $63\,\mathrm{Hz}$, the mobile base accepts commands at $25\,\mathrm{Hz}$, and we run our control loop[4] at $100\,\mathrm{Hz}$. A video of the experiments can be found at http://tiny.cc/force-push.

We test our controller's ability to push three sliders: Barrel, Box1, and Box2 (shown and described in Figure 3.10). The height of the contact point is constant and we assume it is low enough that the sliders do not tip over. For each experiment, the slider starts slightly in front of the EE with various lateral offsets; the robot moves forward until contact is made (i.e., $\|\boldsymbol{f}\| \geq f_{\min}$). For smoothness, the EE accelerates at a constant rate over $1\,\mathrm{s}$ to reach the constant desired pushing speed v. We use ProxQP [38] to solve (3.6). For obstacle avoidance, we model the base as a circle of radius $55\,\mathrm{cm}$. We use the controller parameters in the Hardware column of Table 3.1, which are the same as in simulation except for increased values of k_c and $f_{\min}$. A lower value of k_c was required in simulation so that the trajectory converged to the desired path for *all* combinations of parameters, but for these real-world experiments we found that a higher k_c improves tracking

[4]We could reduce the control frequency to match the slower command frequency of the robot, but our approach *(a)* ensures the most up-to-date command is sent to the robot, and *(b)* demonstrates the efficiency of the controller.

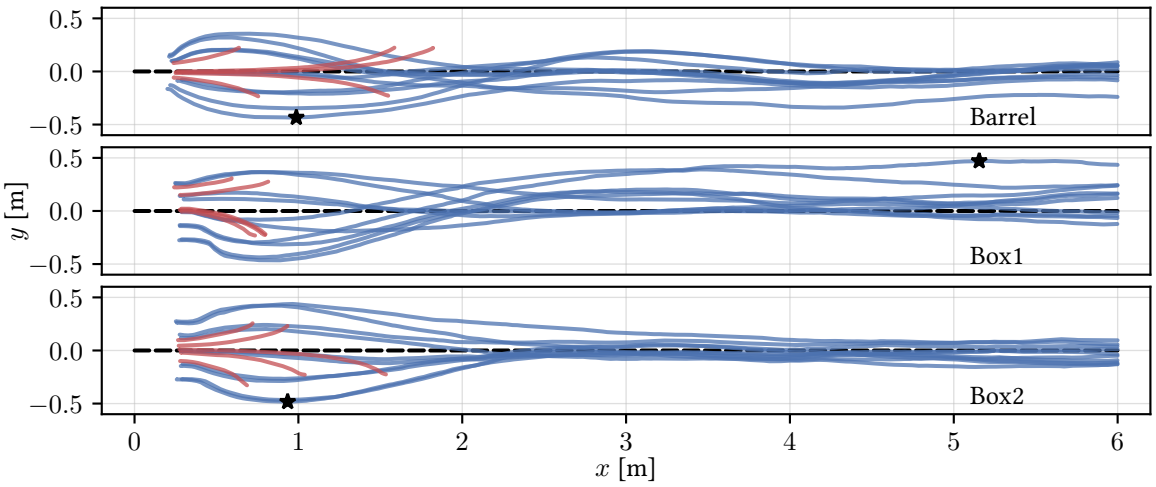

Figure 3.11: Position trajectories for real sliders pushed along a straight path starting from various lateral offsets. Ten trajectories using our pushing control law are shown for each slider (in blue), with the point of maximum deviation of any trajectory from the path marked by a star. We compare against an open-loop controller, which only tracks the path and does not use any slider feedback. Five open-loop trajectories are shown for each slider (in red). The open-loop trajectories end once contact between the EE and slider is lost. All open-loop trajectories fail within about $2\,\mathrm{m}$, whereas our controller is able to push the objects across the full length of the room.

performance. In general, the gains can be tuned to give better tracking performance when the set of possible slider parameters is smaller. We increased $f_{\min}$ to reject noise in the real-world FT sensor.

The results for a straight-line desired path are shown in Figure 3.11. Ten trajectories using our pushing control law are shown for each slider. Here we compare against an open-loop controller, which just follows the path using the open-loop angle (3.3) and constant speed v. Five open-loop trajectories are shown for each slider. Open-loop pushing with single-point contact is not robust to changing friction, misalignment with the slider's CoM, or other disturbances, and indeed we see that the open-loop trajectories all fail within $2\,\mathrm{m}$ of the start of the path, demonstrating the need for a closed-loop controller. In contrast, our controller successfully pushes the sliders across the full $6\,\mathrm{m}$ length of the room.

The trajectories do not converge perfectly to the desired path, at least not within the available $6\,\mathrm{m}$ distance. This is expected in the real world, as the slider is constantly perturbed by imperfections on the surface of the ground as it slides, which must then be corrected by the controller. Indeed, as can be seen in Figure 3.10, the floor of the room has various pieces of tape and other markings which change the surface friction properties as the object slides. For Box1, we actually expect the slider to end up slightly above the path, since its CoM is offset from the measured position. Indeed, our controller only ensures *some* point on the slider (i.e., the contact point) tracks the path, which depends on the slider's frictional and inertial parameters. Regardless, in Figure 3.11 we see that the controller keeps the slider within approximately $0.5\,\mathrm{m}$ of the path at all times, even with different pressure distributions and considerable initial lateral offsets between pusher and slider,

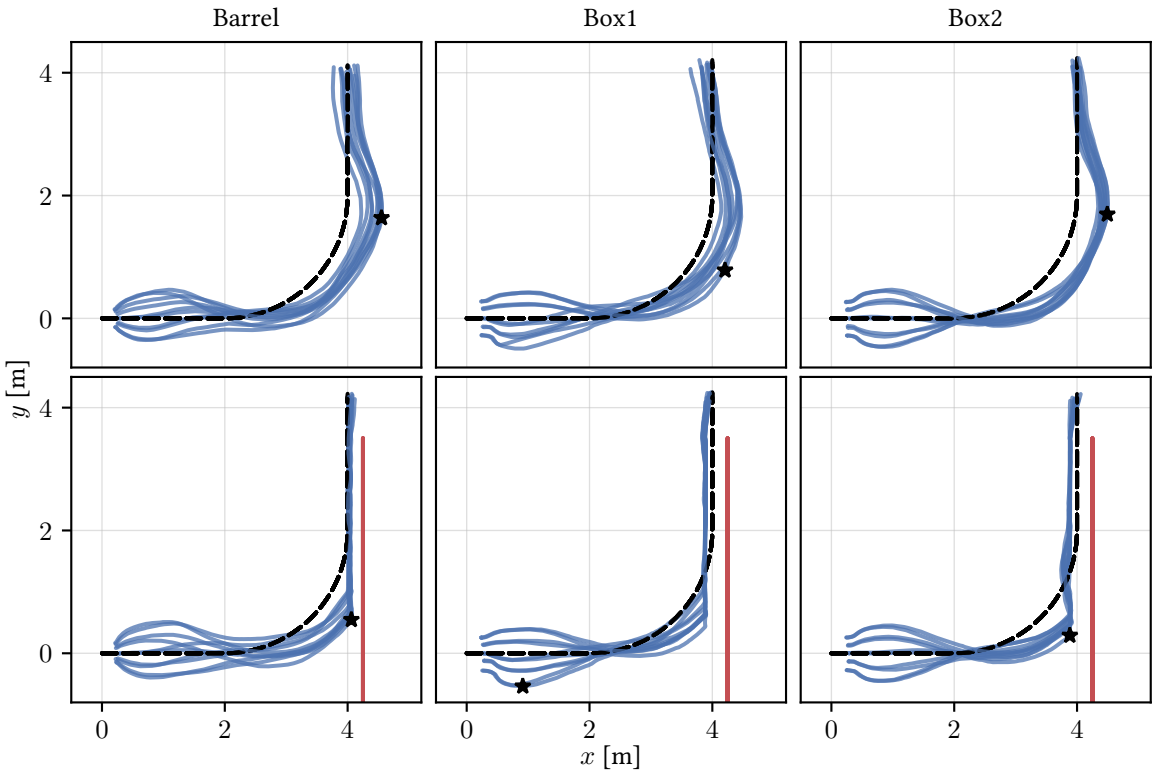

Figure 3.12: Position trajectories (in blue) for real sliders pushed along a curved path, using our pushing controller. In the top row, each slider is pushed through freespace; in the bottom row, an obstacle acting as a wall is introduced (in red), which blocks the motion of the sliders. Ten trajectories are shown for each scenario, with the point of maximum deviation from the path marked by a star. Despite hitting the wall, the controller adjusts the pushing velocity to continue pushing the sliders in the desired direction.

and converges to an even narrower range.

The results for tracking a curved path are shown in Figure 3.12. We show results for freespace as well as with the addition of a wall, which blocks slider motion (see Figure 3.10). The controller knows the location of the wall, so the robot itself can avoid colliding with it. However, the slider does hit the wall, after which the controller adjusts the pushing velocity to continue in the desired direction. This setup represents a simplified version of navigating a turn in a hallway, and demonstrates that we can, in principle, handle contact with obstacles. In this work we assume the space is open enough that the robot can always maneuver to obtain the desired EE velocity using (3.6); future work will investigate narrower hallways and cluttered environments.

Finally, we compare our force-based controller to a vision-based controller (i.e., one that uses measurements of the slider's position, which we obtain using motion capture). We use the "dipole" approach from [47], which generates pushing directions based on the measured angle between the desired goal position and the EE position with respect to the slider. The goal position is the point $1\,\mathrm{m}$ ahead of the current closest point on the desired path. Figure 3.13 presents metrics

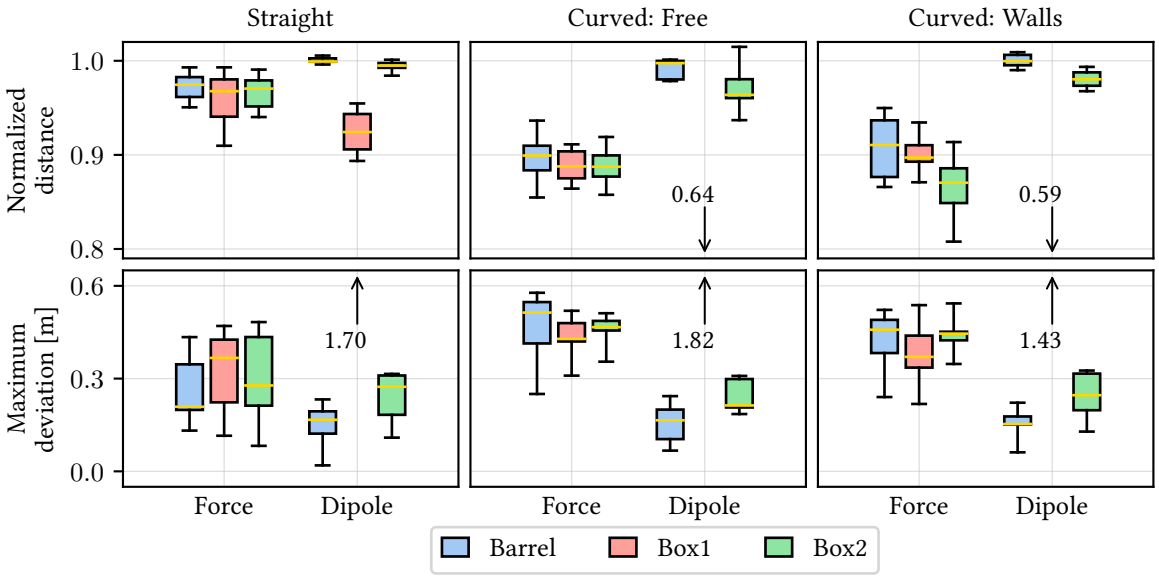

Figure 3.13: Boxplots of metrics for our proposed approach ("Force"; 10 trajectories per boxplot) and the vision-based baseline from [47] ("Dipole"; 5 trajectories per boxplot). The middle yellow line is the median, the box represents the first and third quartiles, and the whiskers represent the minimum and maximum values. The numbered arrows indicate the medians of values outside the axis limits. The normalized distance is the distance actually travelled along the path divided by the distance that would have been travelled if the path were perfectly tracked; the maximum deviation is the farthest point of the slider from the desired path. The normalized distance can exceed 1 if velocity tracking is imperfect or if some of the path is skipped, but the latter necessarily results in some path deviation. Notice that while the dipole approach is effective when the slider's position is well-aligned with its CoM, it deviates substantially if not (e.g., with Box1).

comparing the controllers. The maximum deviation metric is simply the farthest distance between the slider and the desired path, which gives a measure of worst-case path-tracking error. The normalized distance metric is calculated as follows. Let t_0 be the time of first contact, let t_f be the final time, and let $\boldsymbol{p}_{d_0}$ and $\boldsymbol{p}_{d_f}$ be the closest points on the path to the slider's position at t_0 and t_f, respectively. Then we define the ideal distance travelled as $\bar{d} = v(t_f - t_0)$ and the actual distance travelled d as the distance along the path between $\boldsymbol{p}_{d_0}$ and $\boldsymbol{p}_{d_f}$. The normalized distance $d/\bar{d}$ is a measure of how well the task was completed relative to an "ideal" controller that tracked the path perfectly. For simplicity, we neglect the short acceleration phase at the start of each trajectory.

Looking at Figure 3.13, let us first examine our proposed force-based approach. We see that the normalized distance is higher and the maximum deviation is lower for the straight path compared to the curved one—the curved path is in some sense more difficult. The metrics between each of the sliders are fairly similar for a given desired path. The addition of the wall also does not substantially alter the metrics for the curved path, though the normalized distance for Box2 is slightly lowered. The wall briefly slows down the slider after collision, but this removes the overshoot from the path that occurs when the wall is not present (see Figure 3.14 for an example of the change in

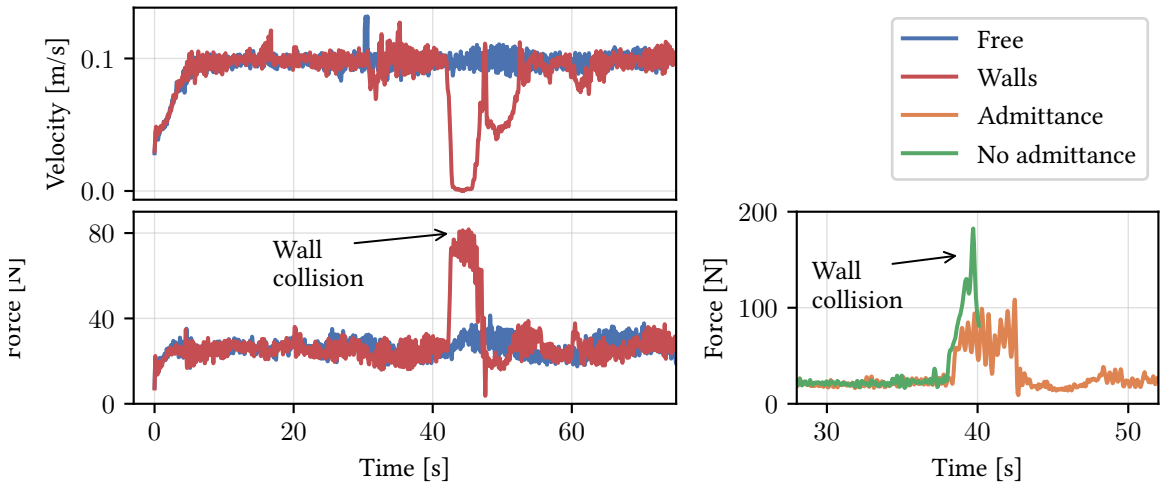

Figure 3.14: *Left:* Slider velocity and contact force magnitudes from two runs of the Box2 slider along the curved path, with and without the wall obstacle. The force increases and the velocity decreases upon collision with the wall, until the controller adjusts the pushing velocity to continue along the path. *Right:* Contact force magnitudes from two different runs of the Box2 slider along the curved path with walls, with and without the admittance controller. Without the admittance controller, the maximum force is well in excess of 150 N, and the experiment was stopped to avoid damage. The other run has the highest maximum force of any runs that used the admittance controller, which is much lower than 150 N.

contact force and slider velocity that occurs when colliding with the wall). Second, consider the dipole approach. We expect a vision-based approach to generally outperform one using only the contact force, since the measured force is noisy and only provides local information at the contact point. Indeed, the dipole approach is effective when the measured position is closely aligned with the slider's CoM, but results in large errors when it is not (e.g., with Box1). This would require extra online adaptation to resolve, something which our force-based controller provides automatically. Ultimately, one must decide which approach (or a combination) makes sense given the available sensing infrastructure—but now force-based pushing is a possible option. Overall, our force-based controller is able to efficiently navigate the path while keeping the deviation reasonable, and we encourage readers to gain more insight into the controller's behaviour by watching the supplemental video.

3.8 Conclusion

We presented the first controller for quasistatic robotic planar pushing with single-point contact using only force feedback to sense the slider. The controller does not require known slider parameters or slider pose feedback. We demonstrated its robustness in simulated and real-world experiments, including collisions with a static wall obstacle, which show that our controller reliably converges to the desired path with reasonable deviation errors. Future work includes a formal

proof of stability, more sophisticated force-based controllers that can improve performance over time through interaction with the slider, and investigation of hybrid approaches that combine force and vision.

Chapter 4

Model Predictive Control for Nonprehensile Object Transportation

In this chapter we develop a solution for another nonprehensile manipulation task—known as the *waiter's problem*—using a mobile manipulator with model predictive control. It is based on the following publication:

> A. Heins and A. P. Schoellig, "Keep it Upright: Model Predictive Control for Nonprehensile Object Transportation with Obstacle Avoidance on a Mobile Manipulator," *IEEE Robotics and Automation Letters*, vol. 8, iss. 12, pp. 7986–7993, 2023.

This is the first whole-body MPC for a mobile manipulator solving the waiter's problem, and the first approach to the waiter's problem that handles dynamic obstacles. Beyond the results presented in the above publication, we also include some further experiments comparing against an additional baseline method in Section 4.8.3.

4.1 Introduction

We consider the nonprehensile object transportation task known as the *waiter's problem* [66], which requires the robot to transport objects from one location to another while keeping them balanced on a tray at the end effector (EE), like a restaurant waiter (see Figure 4.1). In contrast to existing approaches, our focus is on fast online planning in response to new and changing environments. Our main contribution is a whole-body constrained model predictive controller (MPC) for a mobile manipulator that transports objects on a tray to a desired location while avoiding collisions with static and dynamic obstacles, the trajectories of which may not be known a priori.

Specifically, we address the waiter's problem using a velocity-controlled mobile manipulator. Mobile manipulators are capable of performing a wide variety of tasks due to the combination of the large workspace of a mobile base and the manipulation capabilities of robotic arms. We are particularly interested in having the mobile manipulator move and react *quickly*, whether to avoid

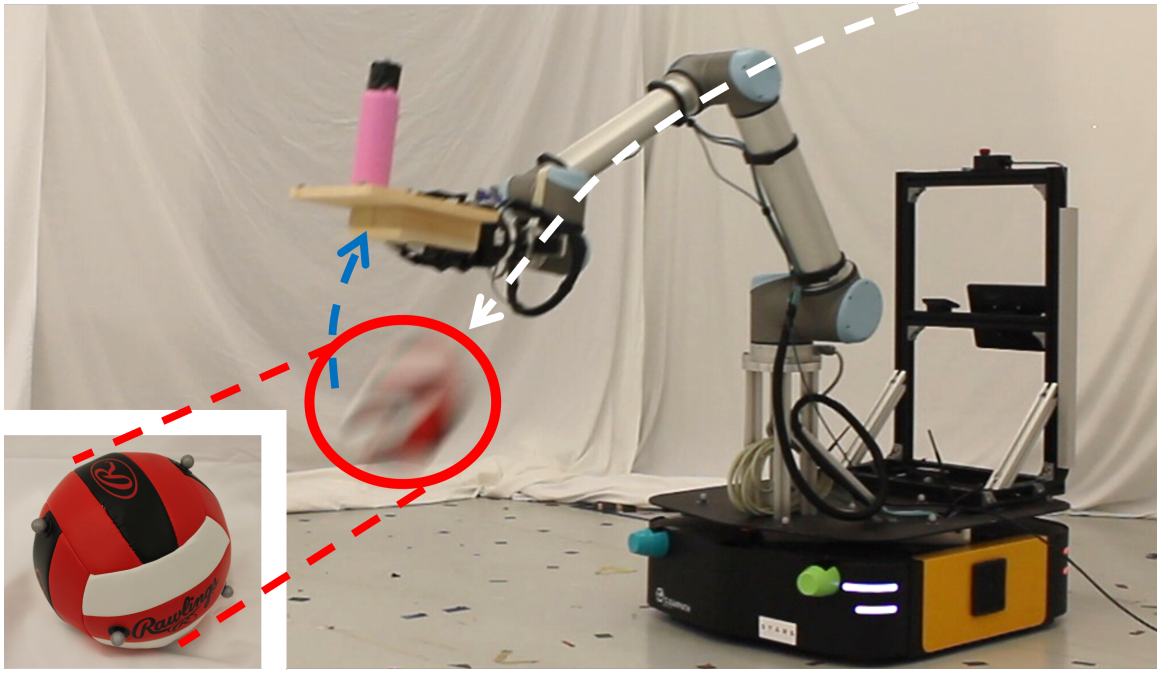

Figure 4.1: Our mobile manipulator balancing a pink bottle while avoiding a thrown volleyball (ball circled in red with approximate trajectory in white; approximate end effector trajectory in blue). The controller has less than $0.75\,\mathrm{s}$ between first observing the ball and a potential collision. A video of our experiments is available at http://tiny.cc/keep-it-upright.

obstacles or simply for efficiency. However, a challenge of *mobile* manipulation is that moving across the ground causes vibration at the EE, which requires our object transportation strategy to be robust to such disturbances.

Objects are held on a tray at the EE under frictional contact (i.e., without the use of grasping or adhesive), and they should neither fall over nor slip off the tray. We assume that the geometry, inertial properties, and initial poses of the objects are known, but we do not assume that feedback of the objects' poses is available online. We assume the robot is velocity-controlled and a kinematic model is available; its dynamic model is not required. Furthermore, we propose planning using the minimum statically feasible friction coefficients, which provides robustness to frictional uncertainty and other force disturbances while also substantially reducing the compute time required to update the MPC policy.

In summary, this work makes the following contributions:

1. **Control:** We propose the first whole-body model predictive controller (MPC) for a mobile manipulator solving the waiter's problem. Compared to existing MPC-based approaches to this problem, which have only been demonstrated on fixed-base arms, our controller optimizes the joint-space trajectory online directly from task-space objectives and constraints, without the use of a higher-level planning step. Furthermore, the controller uses the minimum statically feasible friction coefficients, which provides robustness to frictional uncertainty,

vibration, and other real-world disturbances. When the minimum statically feasible friction coefficients are *zero*, we show that the MPC problem can be solved more efficiently.

2. **Experiments:** We present the first demonstrations of the waiter's problem with a real velocity-controlled mobile manipulator transporting up to seven objects; transporting an assembly of stacked objects; and avoiding static and dynamic obstacles, including a thrown volleyball (see Figure 4.1). The EE achieves speeds and accelerations up to $2.0\,\mathrm{m/s}$ and $7.9\,\mathrm{m/s^2}$, respectively.

3. **Code:** Our code is available as an open-source library at https://github.com/utiasDSL/upright.

After discussing related work in Section 4.2 and background information in Section 4.3 and 4.4, we present our robust sticking constraints (so-called because they ensure the objects remain stationary with respect to the tray; that is, they "stick" to it) in Section 4.5 and our controller in Section 4.6. Simulations and hardware experiments follow in Section 4.7 and 4.8, and Section 4.9 concludes the chapter.

4.2 Related Work

Prior examples of robots directly inspired by waiters in a restaurant include [67]–[69], but these are mobile robots without manipulator arms. In contrast, a mobile *manipulator* has additional DOFs that provide redundancy and a larger workspace, at the cost of requiring a larger and more computationally demanding control problem.

One approach for transporting objects is to use some manner of sensor feedback to infer the object states. In [70], a manipulator performs the classic inverted pendulum task. In [71], a controller is developed to stabilize a tray based on data from an attached accelerometer and gyroscope. In [72], an object is balanced on a tray by a humanoid robot based on force-torque measurements from the robot's wrists. While the focus of [72] is correcting for an object's loss of balance, we focus on generating fast motions that *maintain* open-loop balance without object feedback.

A two-dimensional version of the waiter's problem is addressed in [73], in which a parallel manipulator is mounted on a mobile robot to compensate for the sensed acceleration of the system. The manipulator is controlled to act like a pendulum to minimize the tangential forces acting on a transported object. Simulation of pendular motion has also been used for the slosh-free transport of liquids [74], [75], though these works focus on imposing particular dynamics on the EE rather than directly constraining its motion. EE acceleration constraints are imposed in [76] to avoid dropping grasped objects or spilling liquids, but nonprehensile object transportation is not addressed.

The waiter's problem has also been addressed using offline motion planning. Time-optimal path planning (TOPP) approaches minimize the time required to traverse a provided path subject to the constraint that the transported objects remain balanced. In [77], convex programming is used to solve the TOPP problem. In [78], a robust time-scaling approach is used to handle confidence

bounds on model parameters like friction, which is combined with iterative learning to learn the bounds. Other planning-based approaches do not assume a path is provided. A kinodynamic RRT-based planner is applied to the nonprehensile transportation task in [79], which demonstrates solving a task where no quasistatic solution exists. An optimization-based planner is applied to the task in [66]. In contrast to these offline planning approaches, our method runs online to react quickly to changes in the environment.

In [80] and [81], a reactive controller automatically regulates the commanded motion to ensure the object remains balanced. A similar approach is applied to legged robots in [82], where the desired trajectory is generated by a spline-based planner. This is one of the only works to use a full mobile manipulator (a quadruped) for the waiter's problem, but it is demonstrated only in simulation and does not consider dynamic obstacles. To our knowledge, the only physical mobile manipulator experiments for the waiter's problem have been performed on a humanoid in [83], but similar to [72] they focus on the detection and rejection of disturbances to the object's stability rather than fast object transportation.

Finally, like us, some recent works use MPC to address the waiter's problem, but only on fixed-based arms. In [28], a dual-arm approach is proposed in which a time-optimal trajectory is planned and MPC is used to compute the applied wrench required to realize the object's trajectory. Another MPC approach is described in [84], which is designed to track a manipulator's joint-space reference trajectory. In contrast, our MPC approach optimizes the joint-space trajectory online while considering task-space objectives and constraints, which allows us to respond quickly to changes in the environment like dynamic obstacles. We also show how reducing the friction coefficients in the controller constraints can provide robustness and computational savings.

4.3 System Model

In this section we present the models of the robot and balanced objects.

4.3.1 Robot Model

We consider a velocity-controlled mobile manipulator with state $\boldsymbol{x} = [\boldsymbol{q}^T, \boldsymbol{\nu}^T, \dot{\boldsymbol{\nu}}^T]^T$, where $\boldsymbol{q}$ is the generalized position, which includes the planar pose of the mobile base and the arm's joint angles, and $\boldsymbol{\nu}$ is the generalized velocity. We include acceleration in the state and take the input $\boldsymbol{u}$ to be jerk, which ensures a continuous acceleration profile [84]. The input is double-integrated to obtain the velocity commands sent to the actual robot. We require only a kinematic model, which we represent generically as

$$\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u},$$

with $\boldsymbol{a}(\boldsymbol{x}) \in \mathbb{R}^{\dim(\boldsymbol{x})}$ and $\boldsymbol{B}(\boldsymbol{x}) \in \mathbb{R}^{\dim(\boldsymbol{x})\times\dim(\boldsymbol{u})}$. The actual kinematic model for our robot is

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{4.1}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{0}_9 & \mathbf{1}_9 & \mathbf{0}_9 \\ \mathbf{0}_9 & \mathbf{0}_9 & \mathbf{1}_9 \\ \mathbf{0}_9 & \mathbf{0}_9 & \mathbf{0}_9 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \mathbf{0}_9 \\ \mathbf{0}_9 \\ \mathbf{1}_9 \end{bmatrix},$$

with $\mathbf{0}_n$ denoting a $n \times n$ matrix of zeros. The fact that our mobile base is omnidirectional gives us a linear model, but the nonlinear equations of motion arising from a nonholonomic base, for example, can also be handled.

4.3.2 Object Model

We model each object O as a rigid body subject to the Newton-Euler equations (2.1) expressed in the EE frame. Recall that the GIW acting on the object is

$$\boldsymbol{w}^o_{\mathrm{GI}} \triangleq \boldsymbol{\Xi}^o \boldsymbol{\eta} - \mathrm{ad}(\boldsymbol{\xi})^T \boldsymbol{\Xi}^o \boldsymbol{\xi}, \tag{4.2}$$

where $\boldsymbol{\Xi}^o$ is the object's spatial mass matrix expressed with respect to the EE frame, which we assume is known, $\boldsymbol{\eta}$ is the difference between the EE's spatial acceleration and gravity, and $\boldsymbol{\xi}$ is the EE's spatial velocity.

4.4 Sticking Constraints

To control the interaction between the EE and transported objects in the most general case, we would need to reason about the hybrid dynamics resulting from different contact modes (sticking, sliding, no contact, etc.), as discussed in Chapter 2. Instead, our approach is to enforce constraints that keep the system in a single mode: sticking. That is, we constrain the robot's motion so that the transported objects do not move with respect to the EE, which is known as a *dynamic grasp* [11]. Let us define the EE state as the tuple

$$\boldsymbol{\varepsilon} = (\boldsymbol{C}_e, \boldsymbol{r}_e, \boldsymbol{\xi}, \dot{\boldsymbol{\xi}}),$$

where $\boldsymbol{C}_e \in SO(3)$ is the EE's orientation and $\boldsymbol{r}_e \in \mathbb{R}^3$ is its position in the global frame. We can compute $\boldsymbol{\varepsilon}$ from the robot state $\boldsymbol{x}$ via forward kinematics (2.8), in which case we may explicitly write $\boldsymbol{\varepsilon}(\boldsymbol{x})$. When in the sticking mode, the the object's motion is completely determined by $\boldsymbol{\varepsilon}$; the remainder of this section describes the constraints required to maintain the sticking mode. We do not use online feedback of the object state—given the initial object poses with respect to the EE, the controller generates trajectories to keep those poses constant in an open-loop manner.

A general approach for ensuring an object sticks to the EE can be obtained by including all contact forces directly into the optimal control problem and constraining the solution to be consistent with the desired (sticking) dynamics, which has been previously applied to the waiter's problem in, e.g., [80] and [84]. Consider an arrangement of objects with N total contact points $\{C_i\}_{i \in \mathcal{I}}$ and

corresponding contact forces $\{\boldsymbol{f}_i\}_{i\in\mathcal{I}}$, where $\mathcal{I} = \{1, \ldots, N\}$ (see Figure 4.2). By Coulomb's law, each $\boldsymbol{f}_i$ must be inside its friction cone, for which we use the inner pyramidal approximation (2.11). The total contact wrench acting on an individual object O is

$$\boldsymbol{w}_{\mathrm{C}}^{o} = \sum_{j\in\mathcal{J}} \begin{bmatrix} \boldsymbol{r}_j \times \boldsymbol{f}_j \\ \boldsymbol{f}_j \end{bmatrix}, \tag{4.3}$$

where $\mathcal{J} \subseteq \mathcal{I}$ is the subset of contact indices for O and $\boldsymbol{r}_j$ is the position of C_j. The object remains stationary with respect to the tray for a given $\boldsymbol{e}$ if a set of contact forces can be found each satisfying the fricton cone constraint (2.11) and consistent with the Newton-Euler equations (2.1), (4.2), and (4.3). We assume that contact patches can be represented as polygons with a contact point at each vertex; as in [80] we always use four points with equal friction coefficients.

However, we need an extra constraint for each contact point shared between two objects (as opposed to contact points between an object and the tray; again refer to Figure 4.2): per Newton's third law, the contact force acting on each object must be equal and opposite. Let A and B be two objects in contact at some point C_i, and denote $\boldsymbol{f}_i^a$ and $\boldsymbol{f}_i^b$ the corresponding contact forces acting on A and B, respectively. Then we have the constraint

$$\boldsymbol{f}_i^a = -\boldsymbol{f}_i^b. \tag{4.4}$$

To lighten the notation going forward, we gather all contact forces into the vector $\boldsymbol{\zeta} = [\boldsymbol{f}_1^T, \ldots, \boldsymbol{f}_N^T]^T$, and write

$$(\boldsymbol{\varepsilon}, \boldsymbol{\zeta}) \in \mathcal{S} \tag{4.5}$$

to indicate that the EE state $\boldsymbol{\varepsilon}$ and contact forces $\boldsymbol{\zeta}$ together satisfy the sticking constraints (2.11), (2.1), (4.2), (4.3), and (4.4) for all objects.

4.5 Robust Sticking Constraints

The friction cone constraints (2.11) ensure all contact forces are inside their respective friction cones. However, this assumes accurate knowledge of the friction coefficients, and the constraints may also be violated by unmodelled force disturbances like vibrations and air resistance. To improve the controller's robustness, it is thus desirable for the tangential contact forces to be small, keeping the forces away from the friction cone boundaries [80]. We propose to plan trajectories using the minimum statically feasible values of the friction coefficients; that is, the smallest coefficients for which there exists an EE orientation $\boldsymbol{C}_e$ and contact forces $\boldsymbol{\zeta}$ satisfying the sticking constraints with zero EE velocity and acceleration. This ensures that the controller can always converge to a stationary position. Again considering an arbitrary arrangement of objects, we obtain the minimum

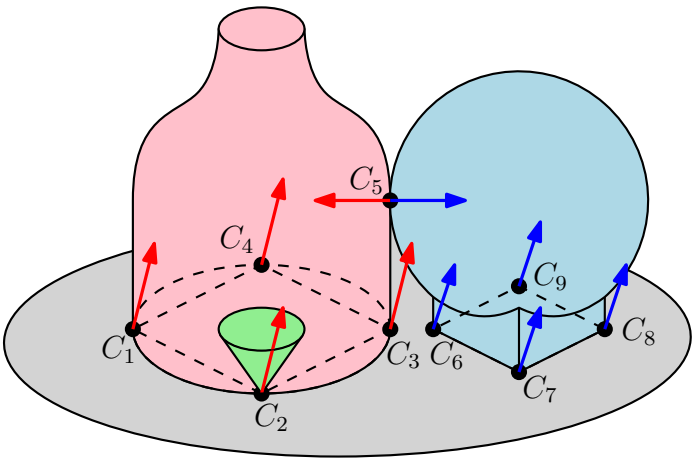

Figure 4.2: A bottle (red) and globe (blue) balanced on a tray. This arrangement has a total of $N = 9$ contact points (black dots), with each object having $n = 5$ (C_5 is shared). Contact forces (arrows) at each contact point must belong to their friction cones (one shown in green). The circular contact patch of the bottle is approximated by a quadrilateral. The contact force acting on each object at the shared contact point C_5 must be equal and opposite. If $\mu_i = 0$, the friction cone at C_i collapses to the line along the normal $\hat{\boldsymbol{n}}_i$.

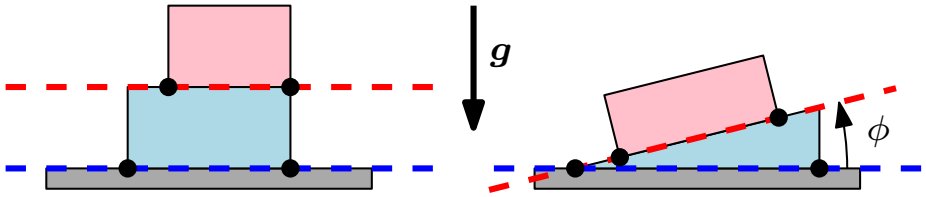

Figure 4.3: Planar view of two arrangements of objects, each with two objects balanced on a tray and a total of four contact points (black dots). *Left:* the support planes (dashed lines) of each object are parallel, so the orientation shown is feasible in the presence of gravity with no friction forces (i.e., we can take $\mu_i = 0$ for all $i \in \mathcal{I}$). *Right:* the support planes are *not* parallel, so some friction is *always* required to balance this arrangement.

statically feasible friction coefficients by solving the optimization problem

$$\begin{aligned}
\underset{\boldsymbol{\phi},\boldsymbol{\zeta},\{\mu_i\}_{i\in\mathcal{I}}}{\operatorname{argmin}} \quad & \frac{1}{2}\sum_{i\in\mathcal{I}} \alpha_i \mu_i^2 && \text{(4.6a)}\\
\text{subject to} \quad & \mu_i \geq 0, \quad i \in \mathcal{I} && \text{(4.6b)}\\
& \boldsymbol{F}_i \boldsymbol{f}_i \leq \boldsymbol{0}, \quad i \in \mathcal{I} && \text{(4.6c)}\\
& \boldsymbol{w}_{\mathrm{C}}^o = -\boldsymbol{\Xi}^o \begin{bmatrix} \boldsymbol{C}_e(\boldsymbol{\phi})\boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix}, \quad \forall O, && \text{(4.6d)}
\end{aligned}$$

where $\{\alpha_i\}_{i\in\mathcal{I}}$ are a set of weights and $\{\mu_i\}_{i\in\mathcal{I}}$ are the friction coefficients of the contact points. The constraint (4.6c) is the linearized friction cone from (2.12) and (4.6d) are the Newton-Euler equations for each object when $\boldsymbol{\xi} = \dot{\boldsymbol{\xi}} = \boldsymbol{0}$, with $\boldsymbol{C}_e(\boldsymbol{\phi})$ parameterized by the roll-pitch-yaw Euler

angles $\boldsymbol{\phi}$ and $\boldsymbol{g} = [0, 0, -9.81]^T$ the gravitational acceleration in the global frame.

If we have nominal estimates of the friction coefficients $\{\bar{\mu}_i\}_{i\in\mathcal{I}}$, we set each weight $\alpha_i = 1/\bar{\mu}_i$ to lower each coefficient proportionally; otherwise we set $\alpha_i = 1$ for all $i \in \mathcal{I}$. In the common case when the support planes of each object are parallel to each other (see Figure 4.3), the solution to (4.6) is simply $\mu_i = 0$ for all $i \in \mathcal{I}$ with $\boldsymbol{C}_e$ such that the support planes are orthogonal to gravity. An example when the solution of (4.6) is not $\mu_i = 0$ for all $i \in \mathcal{I}$ is discussed in Section 4.7.2. The problem (4.6) need only be solved once for a given arrangement of objects. It is always non-convex due to the product of decision variables μ_i and $\boldsymbol{f}_i$ in the friction cone constraint and the nonlinear mapping $\boldsymbol{C}_e(\boldsymbol{\phi})$, but we did not have a problem solving it with the SLSQP solver [85] from scipy [86].

While choosing the minimum friction coefficients may at first appear overly conservative, this approach has a number of benefits. First, it removes the need for accurate friction coefficient estimates, which requires time-consuming physical manipulation of the objects to estimate. Second, as we mentioned in Chapter 1, *mobile* manipulation can produce significant EE vibration, requiring robust motions to ensure objects do not move with respect to the tray. Third, in the common case when $\mu_i = 0$ for all $i \in \mathcal{I}$, the optimal control problem can be simplified as follows. In general we require one contact force variable $\boldsymbol{f}_i \in \mathbb{R}^3$ per contact point, each constrained to satisfy (2.11). However, when $\mu_i = 0$, we can parameterize the force with a single scalar $f_i \geq 0$ such that $\boldsymbol{f}_i = f_i\hat{\boldsymbol{n}}_i$. This reduces the number of force decision variables by two thirds and replaces (2.11) with a simple bound, making the optimization problem faster to solve. However, as we discuss in more detail below, this choice may make the optimization problem infeasible. Our solution is to soften the constraints using slack variables, which ensures feasibility while retaining improved computation speed and motion robustness.

We solved (4.6) assuming the EE was *stationary*, since we do not assume to know the full EE trajectories a priori. However, in general it is not possible to accelerate multiple objects while assuming *zero* friction, even when there is a feasible stationary solution. To see this, first consider a single object on a tray with its support plane orthogonal to gravity and with $\mu_i = 0$ for all $i \in \mathcal{I}$. From (2.14) we have $\boldsymbol{f}_{\mathrm{C}_{xy}} = \boldsymbol{0}$ and $\tau_{\mathrm{C}_z} = 0$, where the subscript $(\cdot)_{xy}$ denotes the tangential component and $(\cdot)_z$ denotes the normal component. Define the selector matrix $\boldsymbol{S}$ such that $\boldsymbol{S}\boldsymbol{w} = [\tau_z, f_x, f_y]^T$ for a given wrench $\boldsymbol{w} = [\boldsymbol{\tau}^T, \boldsymbol{f}^T]^T$. Substituting (4.2) into (2.1) with $\boldsymbol{S}\boldsymbol{w}_{\mathrm{C}} = \boldsymbol{0}$ gives us

$$\boldsymbol{S}(\boldsymbol{\Xi}^o\boldsymbol{\eta} - \mathrm{ad}(\boldsymbol{\xi})^T\boldsymbol{\Xi}^o\boldsymbol{\xi}) = \boldsymbol{0}.$$

So far this is fine: we can plan trajectories that always satisfy this equation. However, if we have two objects A and B with mass matrices $\boldsymbol{\Xi}^a$ and $\boldsymbol{\Xi}^b$, respectively (e.g., the left arrangement in Figure 4.3), then the EE trajectory needs to satisfy *both*

$$\boldsymbol{S}(\boldsymbol{\Xi}^a\boldsymbol{\eta} - \mathrm{ad}(\boldsymbol{\xi})^T\boldsymbol{\Xi}^a\boldsymbol{\xi}) = \boldsymbol{0}, \tag{4.7}$$

$$\boldsymbol{S}(\boldsymbol{\Xi}^b\boldsymbol{\eta} - \mathrm{ad}(\boldsymbol{\xi})^T\boldsymbol{\Xi}^b\boldsymbol{\xi}) = \boldsymbol{0}, \tag{4.8}$$

at all times, where the only difference between (4.7) and (4.8) is the mass matrix. In general, we cannot find an EE trajectory with non-zero accelerations that always satisfies both equations. However, if there is *some* friction force, the right-hand sides of (4.7) and (4.8) are no longer restricted to be identically zero and also need not be equal to each other. Thus we choose to soften the object dynamics constraints (see the next section), which allows tangential contact force to be used when needed, but with a penalty. This approach still requires tuning: instead of tuning friction coefficients, we now must tune the penalty weights. The benefit is that we obtain computational savings when each force can be represented by a non-negative scalar.

4.6 Constrained Model Predictive Controller

We now formulate a model predictive controller to solve the waiter's problem. The controller optimizes trajectories $\boldsymbol{x}(\tau)$, $\boldsymbol{u}(\tau)$, and $\boldsymbol{\zeta}(\tau)$ over a time horizon $\tau \in [t, t+T]$ by solving a nonlinear optimization problem at each control timestep t. Suppressing the time dependencies, the problem is

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{u},\boldsymbol{\zeta}}{\operatorname{argmin}} \quad & \frac{1}{2}\int_{\tau=t}^{t+T} L(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\zeta})\, d\tau \\ \text{subject to} \quad & \dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u} && \text{(system model)} \\ & (\boldsymbol{\varepsilon}(\boldsymbol{x}), \boldsymbol{\zeta}) \in \mathcal{S} && \text{(sticking)} \\ & \mathbf{0} \leq \boldsymbol{d}(\boldsymbol{x}) && \text{(collision)} \\ & \underline{\boldsymbol{x}} \leq \boldsymbol{x} \leq \bar{\boldsymbol{x}} && \text{(state limits)} \\ & \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \bar{\boldsymbol{u}} && \text{(input limits)} \end{aligned} \tag{4.9}$$

where the stage cost is

$$L(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\zeta}) = \|\Delta\boldsymbol{r}(\boldsymbol{x})\|^2_{\boldsymbol{W}_r} + \|\boldsymbol{x}\|^2_{\boldsymbol{W}_x} + \|\boldsymbol{u}\|^2_{\boldsymbol{W}_u} + \|\boldsymbol{\zeta}\|^2_{\boldsymbol{W}_f},$$

with $\|\cdot\|^2_{\boldsymbol{W}} = (\cdot)^T\boldsymbol{W}(\cdot)$ for weight matrix $\boldsymbol{W}$. The EE position error is $\Delta\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{r}_d - \boldsymbol{r}_e(\boldsymbol{x})$. We focus on the case where the desired position $\boldsymbol{r}_d$ is constant, to assess the ability of our controller to rapidly move to a new position without a priori trajectory information. The matrices $\boldsymbol{W}_r$ and $\boldsymbol{W}_x$ are positive semidefinite; $\boldsymbol{W}_u$ and $\boldsymbol{W}_f$ are positive definite. Notice that we do not include a desired orientation: we allow the sticking constraints to handle orientation as needed. If $\mu_i = 0$, then only a scalar f_i is included as a decision variable for each contact force (contained in $\boldsymbol{\zeta}$) and (2.11) is replaced by the constraint $f_i \geq 0$. The vector $\boldsymbol{d}(\boldsymbol{x})$ contains the distances between all pairs of collision spheres representing obstacles and the robot body, which must be non-negative to avoid collisions. When *dynamic* obstacles are used, then we also augment the state $\boldsymbol{x}$ to predict their motion (see Section 4.8.2). We assume that the system can always reach a feasible state that achieves the desired EE position. We discretize the prediction horizon of (4.9) with a fixed timestep Δt and solve it online using SQP (see Chapter 2) via the open-source framework OCS2 [87] and the QP solver HPIPM [39], with the Jacobians required to linearize (4.9) computed using automatic

differentiation via CppAD [88]. We assume that T can be chosen sufficiently long to obtain stability. We use the Gauss-Newton approximation for the Hessian of the cost and we soften the constraints with L_2 penalties [39]. The optimal state trajectory produced by (4.9) is tracked by a low-level joint controller at the robot's control frequency. The remainder of this section provides additional implementation details.

4.6.1 Soft Constraints

We soften all of the constraints in (4.9) except for the system model constraints $\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}$. Consider a general inequality constraint $g(\boldsymbol{x}, \boldsymbol{u}) \leq 0$ (equality constraints are just treated as two-sided inequalities with equal lower and upper limits). We *soften* the constraint by introducing a slack variable $s \geq 0$ as another decision variable in the optimization problem and relaxing the inequality constraint to $g(\boldsymbol{x}, \boldsymbol{u}) \leq s$. The optimizer is encouraged to make s small (and thus reduce constraint violation) by adding a term penalizing s to the objective function. In this work we use an L_2 penalty of the form $w_s s^2$ for each slack variable, where $w_s > 0$ is a tunable weight. We use $w_s = 100$ for all slacks except for the projectile avoidance constraint, which uses $w_s = 4/d^2$, where $d = 0.35\,\mathrm{m}$ is the specified minimum distance between the end effector and the predicted projectile trajectory. We found that the relative weight between the slack penalties for the sticking constraints and the projectile avoidance constraints was the most difficult part of the controller to tune, hence the different slack weight for the projectile avoidance constraint.

When the constraints are soft, the relative magnitudes of the constraint violations must also be considered (which are weighed against each other in the problem's objective function). In particular, we adjust the Newton-Euler dynamics constraints (2.1) for each object to

$$m^{-1}\left(\boldsymbol{w}_{\mathrm{C}} + \boldsymbol{w}_{\mathrm{GI}}/\sqrt{N}\right) = \boldsymbol{0}.$$

Dividing by the object's mass m ensures that transporting heavier objects is not prioritized over lighter objects. Dividing the gravito-inertial wrench by $\sqrt{N}$ reduces the magnitude of the contact force variables in the optimization problem as the number of contact points N increases. The idea is that we do not want the penalties on (2.1) to dominate the other objectives and penalties in the problem (4.9) just because more objects and contact points have been added to the problem.

4.6.2 Low-level Joint Controller

The MPC problem (4.9) typically cannot be solved at the same frequency that the robot accepts commands, so we need a strategy to compute inputs between solutions of (4.9). Suppose we compute a new MPC policy using (4.9) at time t, which is valid until time $t + T$. Then at each control time $\tau \in [t, t + T]$, we can compute the jerk input $\boldsymbol{u}(\tau)$ using an affine state feedback controller of the form

$$\boldsymbol{u}(\tau) = \boldsymbol{K}(\tau)(\boldsymbol{x}^\star(\tau) - \boldsymbol{x}(\tau)) + \boldsymbol{k}(\tau), \tag{4.10}$$

where $\boldsymbol{x}^\star$ is the optimal state trajectory, $\boldsymbol{K}$ is the feedback gain matrix, and $\boldsymbol{k}$ is the feedforward input, all obtained from the most recent policy. In particular, at each control timestep t, (4.9) is discretized and linearized to form a quadratic program (QP), which is solved using an IP method [39]. The terms $\boldsymbol{K}$ and $\boldsymbol{k}$ are obtained from the Riccati recursion used to solve the linear system arising from the Karush-Kuhn-Tucker conditions in the final iteration of the IP method used to solve the QP (see [39] as well as [89] and [90] for more details). The upshot of (4.10) is that we can cheaply generate inputs $\boldsymbol{u}$ based on the most recent MPC solution, unless more time than the horizon T has elapsed since the solution, which never occurred during our experiments.

In simulation we do not run in real time, which allows us to recompute the policy every $10\,\mathrm{ms}$ of simulation time, regardless of the actual required compute time. We use (4.10) to generate the input at every step of the simulation, which has a timestep of $1\,\mathrm{ms}$. In our hardware experiments, the MPC policy (4.9) is solved in a separate process. We limit policy updates to at most once every $10\,\mathrm{ms}$ and we use (4.10) to generate commands at the robot's control frequency of $125\,\mathrm{Hz}$.

4.6.3 State Estimation

A Kalman filter is used to estimate the state of the robot $\boldsymbol{x}$ in our hardware experiments. The model is linear, allowing us to use the standard linear Kalman filter (see e.g. [91]). Measurements of the pose of the mobile base are provided by a Vicon motion capture system and measurements of the joint angles of the arm are provided by its joint encoders. The position of the projectile is also obtained from the Vicon system. Discretizing (4.1) gives us the discrete-time model

$$\boldsymbol{x}^+ = \bar{\boldsymbol{A}}\boldsymbol{x} + \bar{\boldsymbol{B}}\boldsymbol{u},$$

where

$$\bar{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{I}_9 & \delta t \mathbf{1}_9 & (1/2)\delta t^2 \mathbf{1}_9 \\ \mathbf{0}_9 & \mathbf{1}_9 & \delta t \mathbf{1}_9 \\ \mathbf{0}_9 & \mathbf{0}_9 & \mathbf{1}_9 \end{bmatrix}, \qquad \bar{\boldsymbol{B}} = \begin{bmatrix} (1/6)\delta t^3 \mathbf{1}_9 \\ (1/2)\delta t^2 \mathbf{1}_9 \\ \delta t \mathbf{1}_9 \end{bmatrix},$$

are obtained from Taylor series expansions of $\boldsymbol{x}$ and we have used $\bar{(\cdot)}$ to denote the discrete-time system matrices. The sampling time is $\delta t = 8\,\mathrm{ms}$, which is the duration of each iteration of the robot control loop. We measure generalized positions $\boldsymbol{q}$, and so our measurement model is $\boldsymbol{q} = \bar{\boldsymbol{C}}\boldsymbol{x}$, where

$$\bar{\boldsymbol{C}} = \begin{bmatrix} \mathbf{1}_9 & \mathbf{0}_9 & \mathbf{0}_9 \end{bmatrix}.$$

The other ingredients we need for the Kalman filter are the process covariance $\bar{\boldsymbol{Q}}$, the measurement covariance $\bar{\boldsymbol{R}}$, and the initial state covariance $\boldsymbol{P}_0$. In experiment we use $\bar{\boldsymbol{Q}} = \bar{\boldsymbol{B}}\boldsymbol{Q}\bar{\boldsymbol{B}}^T$ with $\boldsymbol{Q} = 10\mathbf{1}_9$, $\bar{\boldsymbol{R}} = 0.001\mathbf{1}_9$, and $\boldsymbol{P}_0 = 0.1\mathbf{1}_{27}$.

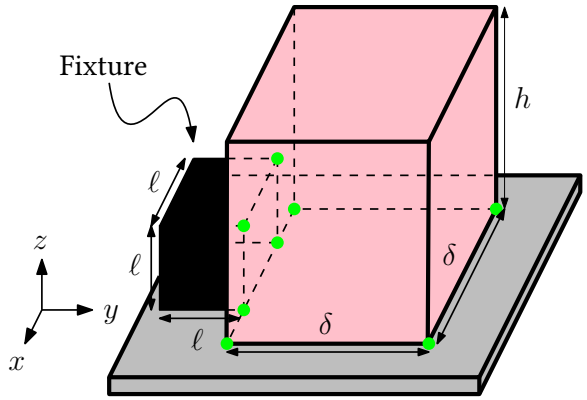

Figure 4.4: An arrangement consisting of a red box balanced on a tray along with a black *fixture*, which is rigidly attached to the tray. The fixture adds contact points (shown in green) up the side of the box, which our controller can exploit to accelerate faster.

4.7 Simulation Experiments

We begin with simulations to gain insight into the performance of our controller in an idealized environment. We use a simulated version of our experimental platform, a 9-DOF mobile manipulator consisting of a Ridgeback mobile base and UR10 arm, depicted in Figure 4.8. In all experiments (simulated and real) we use $\Delta t = 0.1\,\mathrm{s}$, $T = 2\,\mathrm{s}$, and weights

$$\boldsymbol{W}_r = \mathbf{1}_3, \qquad \boldsymbol{W}_x = \mathrm{diag}(0\mathbf{1}_9, 0.1\mathbf{1}_9, 0.01\mathbf{1}_9),$$
$$\boldsymbol{W}_u = 0.001\mathbf{1}_9, \qquad \boldsymbol{W}_f = 0.001\mathbf{1}_{\dim(\boldsymbol{\zeta})}.$$

The state and input constraints used for the robot are

$$\bar{\boldsymbol{q}} = \begin{bmatrix} 10\boldsymbol{e}_3 \\ 2\pi\boldsymbol{e}_6 \end{bmatrix}, \bar{\boldsymbol{\nu}} = \begin{bmatrix} 1.1\boldsymbol{e}_2 \\ 2\boldsymbol{e}_3 \\ 3\boldsymbol{e}_4 \end{bmatrix}, \dot{\bar{\boldsymbol{\nu}}} = \begin{bmatrix} 2.5\boldsymbol{e}_2 \\ 1 \\ 10\boldsymbol{e}_6 \end{bmatrix}, \bar{\boldsymbol{u}} = \begin{bmatrix} 20\boldsymbol{e}_3 \\ 80\boldsymbol{e}_6 \end{bmatrix},$$

where $\bar{\boldsymbol{x}} = [\bar{\boldsymbol{q}}^T, \bar{\boldsymbol{\nu}}^T, \dot{\bar{\boldsymbol{\nu}}}^T]^T$, $\underline{\boldsymbol{x}} = -\bar{\boldsymbol{x}}$, $\underline{\boldsymbol{u}} = -\bar{\boldsymbol{u}}$, and $\boldsymbol{e}_n$ denotes an n-dimensional vector of ones. We use a single SQP iteration per control policy update.

4.7.1 Sticking Constraint Comparison

We first consider the example shown in Figure 4.4, consisting of a box balanced on a tray and in contact with a *fixture*, which is rigidly attached to the tray. We perform experiments with and without the fixture, which is a cube of side length $\ell = 5\,\mathrm{cm}$. The box has mass $m = 0.5\,\mathrm{kg}$, height $h = 20\,\mathrm{cm}$, and a square base with side length $\delta = 6\,\mathrm{cm}$. The CoM is located at the centroid, the mass distribution is uniform, and $\mu_i = 0.2$ for all $i \in \mathcal{I}$. The task is to move the EE to a desired goal point $\boldsymbol{r}_d = [-2, 1, 0]^T$ (all desired positions are given in meters relative to the initial EE position) without dropping the box. We compare the trajectories that result from imposing four

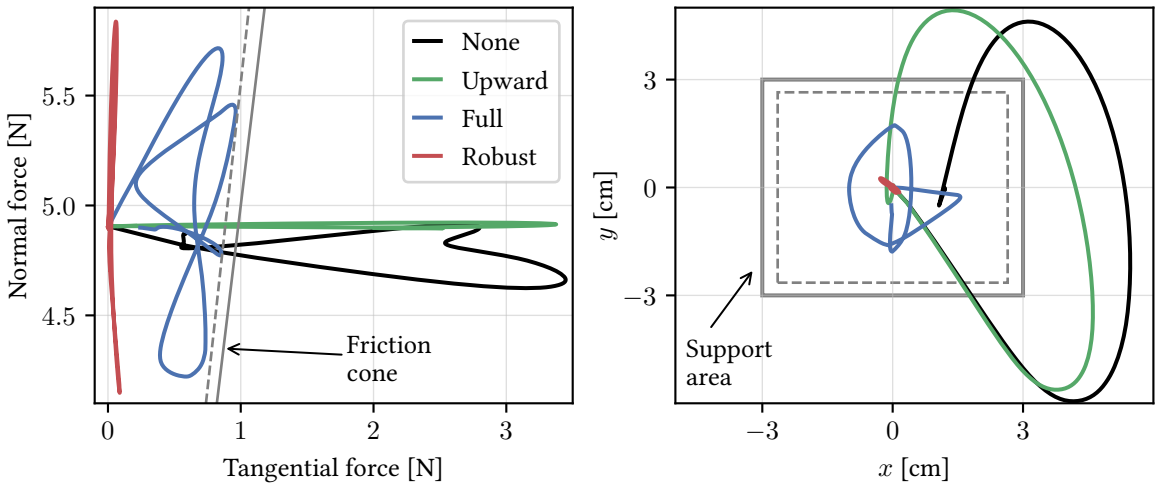

Figure 4.5: *Left:* Force applied to the simulated box during motion. *Right:* Corresponding ZMP trajectories. With no constraints (None) or the Upward constraint, the force leaves the friction cone and the ZMP leaves the support area (safety margins in dashed lines), so the box slides and tips over (and is dropped). The Full constraints touch but do not pass the boundaries; the Robust constraints stay far from the boundaries in both cases.

different sets of constraints:

- **None:** No constraints.
- **Upward:** A constraint to keep the tray oriented upward.
- **Full:** The full set of sticking constraints $(\varepsilon, \zeta) \in \mathcal{S}$ with each μ_i set to 90% of the true value.[1]
- **Robust:** The full set of constraints $(\varepsilon, \zeta) \in \mathcal{S}$ with $\{\mu_i\}_{i \in \mathcal{I}}$ computed using (4.6). Unless otherwise stated, the solution is $\mu_i = 0$ for all $i \in \mathcal{I}$.

In ideal conditions, the Full and Robust constraints should both keep the objects stationary with respect to the tray, but the Full constraints provide less of a safety margin. The Upward approach would work if the motion were quasistatic (i.e., with negligible accelerations), but that would not be fast or reactive.

In Figure 4.5, the force acting on the object and the zero-moment point (ZMP) are shown relative to the friction cone and support area, respectively. The ZMP is the point about which horizontal moments are zero; if it is outside of the support area, then the object tips. Unsurprisingly, the None and Upward approaches significantly violate both the friction cone and ZMP constraints, resulting in the box being dropped. The Full approach produces motion at the boundary of the constraints but does not violate them, while the Robust approach stays away from the boundaries. In Figure 4.6,

[1]We only use 90% of the true (measured or simulated) value to provide some robustness to small constraint violations arising from discretization errors and other numerical disturbances. We subtract a small margin from the support area for the same reason. This is more important in the hardware experiments, where there are more sources of noise and disturbances.

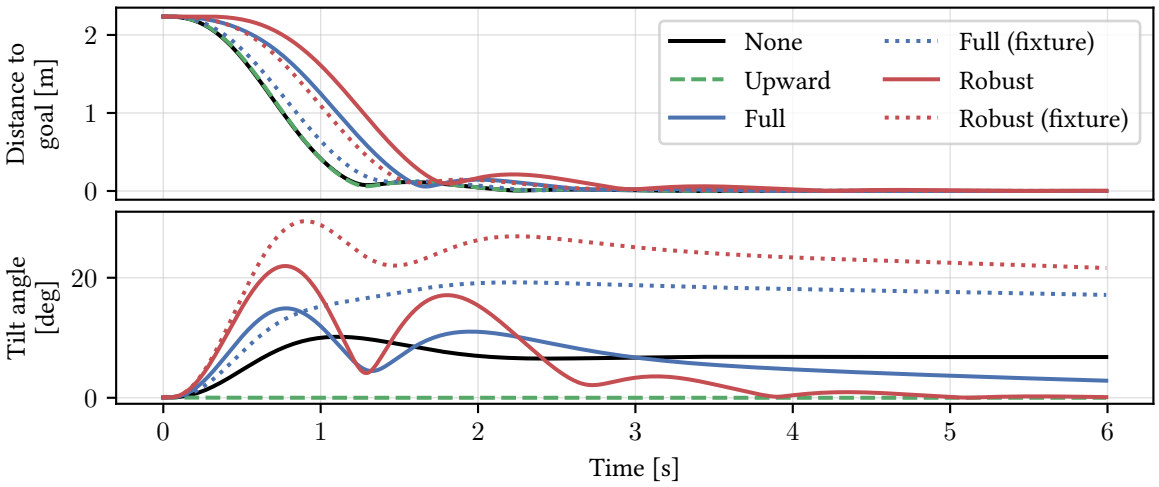

Figure 4.6: *Top:* Distance of EE to goal location. *Bottom:* Tilt angle with respect to the upward-pointing (i.e. gravity-aligned) orientation. The Full and Robust constraints limit acceleration to keep the box stationary with respect to the tray; the Robust approach also uses higher tilt angles. The None and Upward approaches accelerate faster—and drop the box. When the fixture is added, the Full and Robust constraints can exploit it to achieve convergence speeds more similar to the None and Upward cases. Notice that, except for the Upward constraint, there is no need for the tilt angle to be near zero.

we see that the robustness of the Robust approach comes at the cost of slower convergence and higher tilt angles compared to the Full approach. When the fixture is added, the Full and Robust approaches can both exploit it to converge nearly as fast as when no constraints are used at all. Notice that the tilt angle for the None and Full approaches need not converge to zero: the None approach does not consider the EE orientation at all, while the Full approach may converge to any orientation that satisfies the sticking constraints.

4.7.2 Non-Parallel Support Planes

Next we show an example when the solution to (4.6) is not simply $\mu_i = 0$ for all $i \in \mathcal{I}$. The setup consists of a wedge supporting a box at an incline of $\phi = 15°$, similar to the right side of Figure 4.3. For simplicity we assume that μ is constant between each pair of objects, so we need only solve (4.6) for the friction coefficient between the tray and wedge μ_{tw} and between the wedge and box μ_{wb}. We obtain $\mu_{tw} = \mu_{wb} = 0.132$, which corresponds to a tilt angle of $\theta = \arctan(0.132) \approx 7.5°$ relative to the ground for each object, meaning the wedge and box can be oriented so as to split the angle ϕ between them. Using $\mu_{tw} = \mu_{wb} = 0.132$ for the controller and $\mu_{tw} = \mu_{wb} = 0.2$ for the simulator, we run the simulation with the same goal $\boldsymbol{r}_d = [-2, 1, 0]^T$. The initial and final configurations are shown in Figure 4.7. Note that we need not start in a configuration which the controller thinks is feasible (since the constraints are soft), but the controller will steer toward one over the course of the trajectory. If we were to dispense with (4.6) and simply try to enforce $\mu_i = 0$

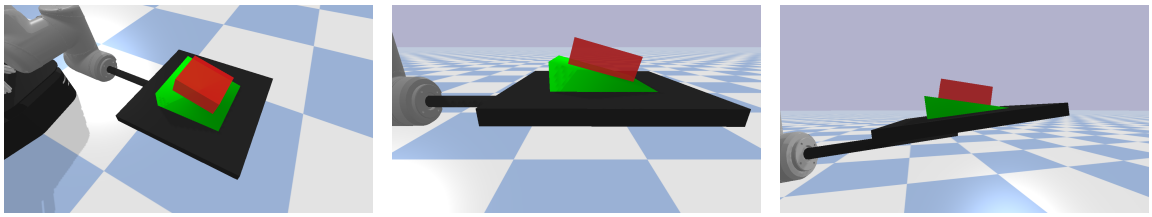

Figure 4.7: *Left:* Initial position of wedge (green) and box (red) arrangement. *Middle:* Initial side view. The box is tilted $15°$ relative to the ground due to the slope of the wedge. *Right:* The position at $t = 6\,\mathrm{s}$. The controller has oriented the tray so that both the wedge and box are tilted $7.5°$ relative to the ground, such that each requires as small a μ as possible.

Table 4.1: Approximate parameters for transported objects shown in Figure 4.8. CoM and inertia are estimated from mass and geometry.

Arrangement	# of objects	# of contacts	Mass per object [g]	Friction coefficients
Bottle	1	4	827	*tray-bottle:* 0.26
Arch	3	16	180	*tray-block:* 0.30 *block-block:* 0.42
Cups	7	28	200	*tray-cup:* 0.28

for all $i \in \mathcal{I}$, the controller fails to converge because no feasible stationary solution exists.

4.8 Hardware Experiments

In simulation we gained insight into the behaviour of the controller without the influence of real-world effects like sensor noise or EE vibrations. We now perform experiments on our real mobile manipulator to assess our approach in more realistic scenarios. Position feedback is provided for the arm by joint encoders and for the base by a Vicon motion capture system, which is used in a Kalman filter to estimate the full robot state as described in Section 4.6.3. We also use motion capture to track the position of the transported objects, which is only used for error reporting. The controller parameters and weights are the same as in the previous section. The robot and transported objects are shown in Figure 4.8; the corresponding object parameters are given in Table 4.1. A video of the experiments can be found at http://tiny.cc/keep-it-upright.

4.8.1 Static Environments

We perform a large set of experiments with different combinations of objects and desired EE positions, each using the None, Upward, Full, and Robust constraint methods described above. The desired positions are $\boldsymbol{r}_{d_1} = [-2, 1, 0]^T$, $\boldsymbol{r}_{d_2} = [2, 0, -0.25]^T$, and $\boldsymbol{r}_{d_3} = [0, 2, 0.25]^T$. The object error and controller compute time in an obstacle-free environment are shown in Figure 4.9; results for an environment with static obstacles are shown in Figure 4.10. We model obstacles as

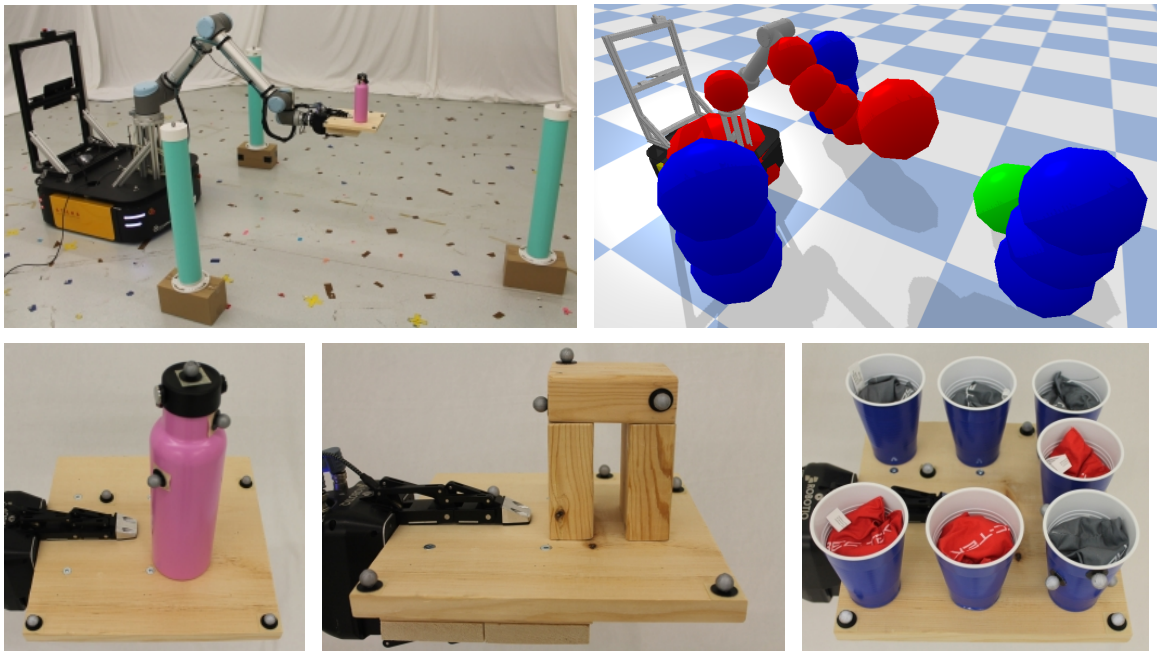

Figure 4.8: *Top left:* Real experimental setup. Robot is shown holding the Bottle object. Obstacle locations marked with pylons. *Top right:* Corresponding simulated experimental setup with collision spheres on the robot in red and on the obstacles in blue. *Bottom row:* Bottle, Arch, and Cups object arrangements used for experiments. The arch is an example of non-coplanar contact (the three blocks composing the arch are not attached together). The bottle is filled with sugar and the cups each contain bean bags instead of liquid to avoid spills in the lab.

collections of spheres; spheres also surround parts of the robot body for collision checking (see top right of Figure 4.8). As expected, the None and Upward approaches almost always fail—the notable exception is for goal $\boldsymbol{r}_{d_2}$, which requires more base motion and is thus slower than the other trajectories. The Robust constraints typically produce the lowest object error or are close to it. In general we expect the Robust constraints to have the lowest error, given that they reduce the tangential contact forces and can thus resist unmodelled force disturbances. However, we noticed that the larger tilt angles required by the Robust constraints can occasionally result in some sliding of the objects.

Computationally the Robust constraints scale *much* better with the number of contacts than the Full constraints, since they require less decision variables and use simpler constraints. The Full constraints also require reasonably accurate friction coefficient estimates; the effectiveness of the Robust constraints show that we need not fear frictional uncertainty and (when statically feasible) can set $\mu_i = 0$ for all $i \in \mathcal{I}$ to reduce compute time. The static obstacle results in Figure 4.10 are similar to those for free space except for a modest increase in compute time. Sample trajectories are shown in Figure 4.11.

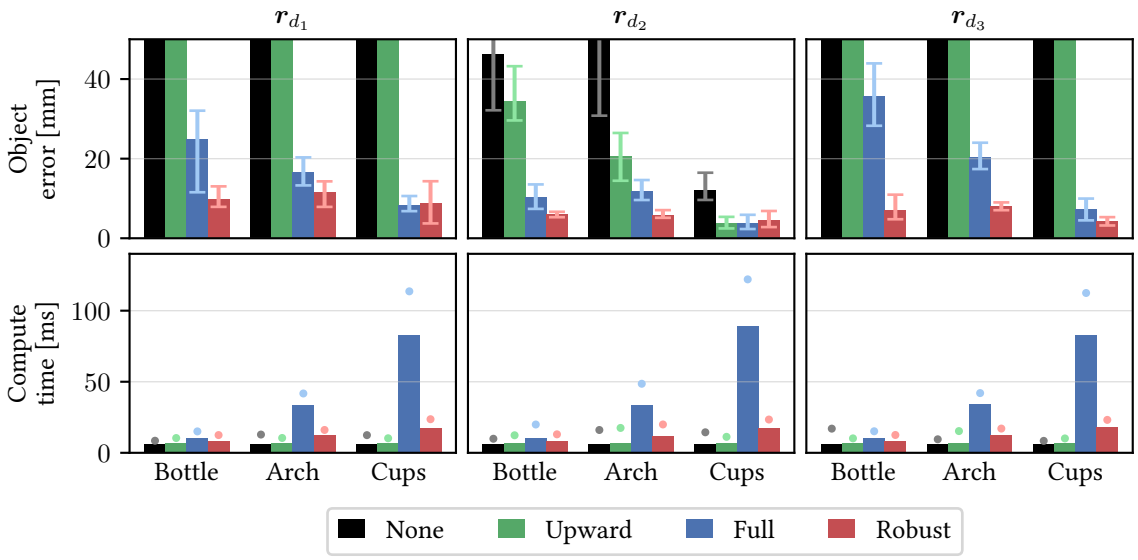

Figure 4.9: Object error (top row) and policy compute time (bottom row) for different combinations of objects, goal positions, and constraints in free space. The object error is the maximum distance the object moves from its initial position relative to the tray. In arrangements with multiple objects, only a single one is tracked. The bar shows the average of three runs; the error bars show the minimum and maximum values. One or more objects were completely dropped in all cases where the minimum error is beyond the axis limits. The compute time is the average time required to compute an updated MPC policy (i.e., one iteration of (4.9)). The bar shows the average across the three runs (up to the first 6 s of the trajectory); the dot shows the average maximum value across the three runs.

4.8.2 Dynamic Environments

We now consider environments that change over time due to dynamic obstacles. Dynamic obstacles are modelled as spheres with known radii, but the controller does not know their trajectories a priori.

An Unexpected Obstacle

Here we test the controller's ability to react to unexpected events. We make the controller aware of a new obstacle at varying times t, and the policy must be quickly updated to avoid a collision. The setup is simple: we use the static obstacle environment and goal $\boldsymbol{r}_{d_2}$ with the Bottle arrangement and a new "virtual" obstacle (the obstacle does not physically exist, but the controller thinks it is present). At time t the new obstacle instantly appears in front of the robot (represented by the green sphere in Figure 4.8)—imagine a restaurant customer suddenly backing out their chair. The results for different t are shown in Figure 4.12. The appearance of the obstacle causes significant changes in the trajectory of both the EE and base, but the object is not dropped despite the sudden change, even when the collision constraint is violated by the obstacle's appearance. The maximum

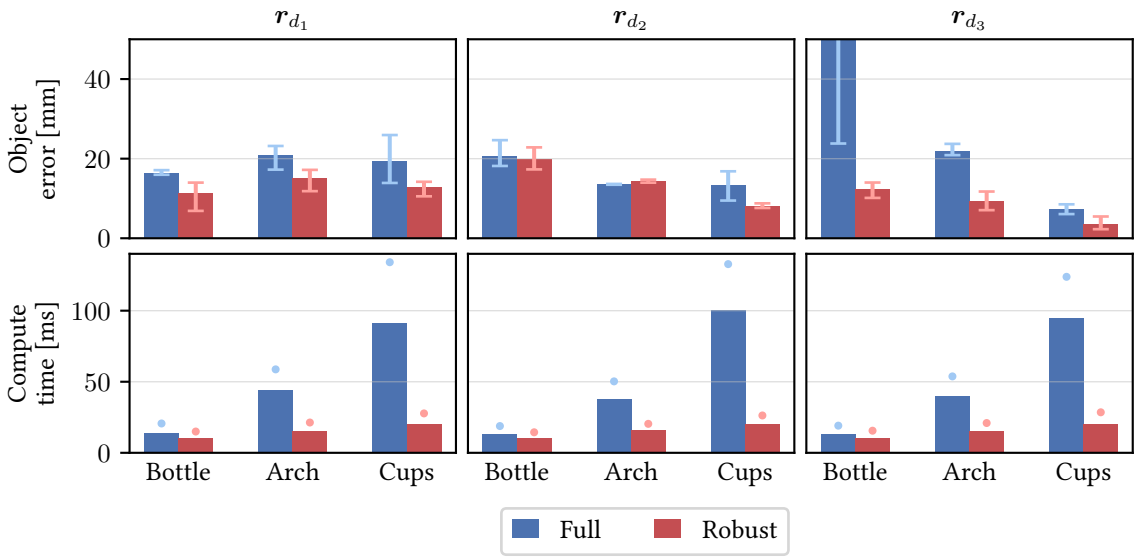

Figure 4.10: The same results as shown in Figure 4.9 but in an environment with static obstacles and only showing the Full and Robust approaches. Compared to Figure 4.9, the errors are similar while the compute times are slightly higher.

object error and policy compute time were $18\,\mathrm{mm}$ and $23\,\mathrm{ms}$, respectively, across three runs of each of the four obstacle appearance times t. The trajectory with $t = 1\,\mathrm{s}$ achieved the highest EE velocity and acceleration of all our experiments, at $2.0\,\mathrm{m/s}$ and $7.9\,\mathrm{m/s^2}$, respectively.

Projectile Avoidance

Finally, we consider a ball with position $\boldsymbol{r}_b$ and state $\boldsymbol{b} = [\boldsymbol{r}_b^T, \dot{\boldsymbol{r}}_b^T]^T$ modelled as a simple projectile with $\ddot{\boldsymbol{r}}_b = \boldsymbol{g}$. We neglect drag and other possible nonlinear terms, because *avoiding* an object requires a less accurate model than when catching [92] or batting it [93]. The ball is thrown toward the EE, and the robot must move to avoid the objects being hit while also keeping them balanced. For these experiments we use the Bottle arrangement and the Robust constraint method. The controller is provided with feedback of $\boldsymbol{b}$ once the ball exceeds the height of $1\,\mathrm{m}$; the state is estimated using a Kalman filter with measurements of the ball's position from the motion capture system. The discrete-time equations of motion for the projectile are

$$\boldsymbol{b}^+ = \bar{\boldsymbol{A}}_b \boldsymbol{b} + \bar{\boldsymbol{B}}_b \boldsymbol{g},$$

where

$$\bar{\boldsymbol{A}}_b = \begin{bmatrix} \mathbf{1}_3 & \delta t \mathbf{1}_3 \\ \mathbf{0}_3 & \mathbf{1}_3 \end{bmatrix}, \qquad \bar{\boldsymbol{B}}_b = \begin{bmatrix} (1/2)\delta t^2 \mathbf{1}_3 \\ \delta t \mathbf{1}_3 \end{bmatrix},$$

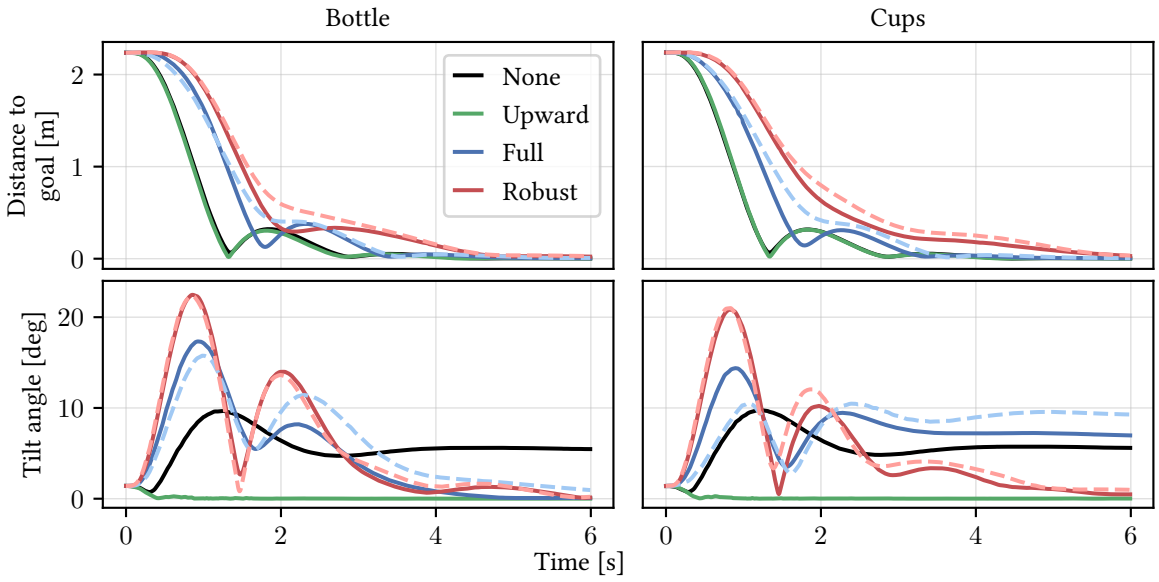

Figure 4.11: Samples of trajectories to goal $\boldsymbol{r}_{d_1}$ for the Bottle and Cup arrangements with different constraints. Free space results are solid lines; results with static obstacles (shown only for Full and Robust) are dashed. The addition of static obstacles modestly changes the shape of the trajectories. The Full and Robust trajectories differ between the two object arrangements; in particular, notice that Full constraints converge to a much smaller tilt angle with the Bottle compared to the Cups. The Bottle's higher CoM makes it easier to tip, so it requires a smaller tilt angle when stationary.

and the measurement model is $\boldsymbol{r}_b = \bar{\boldsymbol{C}}_b \boldsymbol{b}$, with

$$\bar{\boldsymbol{C}}_b = \begin{bmatrix} \mathbf{1}_3 & \mathbf{0}_3 \end{bmatrix}.$$

Here we use a sampling time of $\delta t = 10\,\mathrm{ms}$, which is the rate at which measurements are received from the motion capture system. We use process covariance $\bar{\boldsymbol{Q}}_b = \bar{\boldsymbol{B}}_b \boldsymbol{Q}_b \bar{\boldsymbol{B}}_b^T$ with $\boldsymbol{Q}_b = 1000\mathbf{1}_3$, measurement covariance $\bar{\boldsymbol{R}}_b = 0.001\mathbf{1}_3$, and initial state covariance $\boldsymbol{P}_{b_0} = \mathbf{1}_6$. The state $\boldsymbol{b}$ and the projectile dynamics are added to (4.9) to predict the ball's motion. We found it most effective to use a form of continuous collision checking in which the controller tries to avoid a tube around the entire future trajectory of the ball. Once the ball has passed the EE, the constraint is removed.

The results for 20 throws are shown in Figure 4.14 and images from one throw are shown in Figure 4.13. Throws are split evenly between two directions: toward the front of the EE and toward its side. In all cases, the controller has less than $0.75\,\mathrm{s}$ to react and avoid the ball. Out of the 20 trials, there is one in which the ball would not have penetrated the collision sphere even if the robot did not move, and another where the bottle was actually dropped. This failure was not due to a collision, but because the bottle tipped over due to the aggressive motion used to avoid the ball. Also notice that the controller does not always completely pull the robot out of collision: there is a trade-off between keeping the object balanced on the tray and avoiding collision. However, since the collision spheres are conservatively large, we did not experience any failures due to collisions.

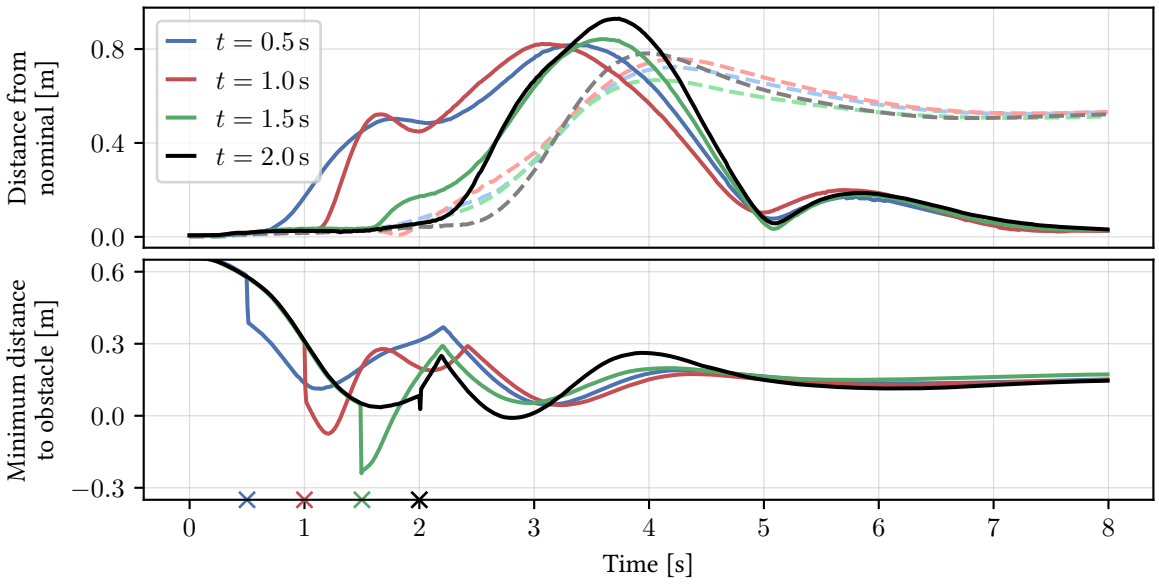

Figure 4.12: *Top:* Distance of EE (solid) and base (dashed) positions from a nominal trajectory with a virtual obstacle suddenly appearing at different times t (also marked with crosses on x-axis). The nominal trajectory has no dynamic obstacle. *Bottom:* Minimum distance between any collision sphere on the robot and the dynamic obstacle. Notice that in some cases the appearance of the obstacle actually violates the collision constraints, which could also happen with a physical obstacle if the collision sphere was conservatively large. Regardless, the Bottle was never dropped.

In these experiments the controller only tries to avoid collisions between the ball and EE; collisions with the rest of the robot's body are not avoided. The maximum object error and policy compute time were $32\,\mathrm{mm}$ (ignoring the single failure) and $20\,\mathrm{ms}$, respectively, across the 20 trials.

4.8.3 Comparison with Aligned Approach

We also subsequently compared our proposed method with an alternative baseline method, which we call "Aligned". The Aligned method seeks to keep the tray's normal $\hat{\boldsymbol{n}}$ aligned with the linear acceleration $\boldsymbol{a} \triangleq \dot{\boldsymbol{v}} - \boldsymbol{g}$ of the EE, without considering the transported object. Instead of the sticking constraints (4.5), the controller enforces the constraints

$$\hat{\boldsymbol{t}}_1^T \boldsymbol{a} = \boldsymbol{0}, \qquad \hat{\boldsymbol{t}}_2^T \boldsymbol{a} = \boldsymbol{0}, \qquad \hat{\boldsymbol{n}}^T \boldsymbol{a} \geq 0,$$

where $\hat{\boldsymbol{t}}_1$ and $\hat{\boldsymbol{t}}_2$ span the tray's surface plane (orthogonal to $\hat{\boldsymbol{n}}$). In other words, the controller always tries to orient the tray normal to the direction of acceleration, such that the total acceleration in the normal direction is non-negative. This is equivalent to enforcing (4.5) for a frictionless particle located at the origin of the EE frame, but becomes less effective as the CoM of the actual transported objects is located farther from the EE frame origin. We test this approach with the original Bottle arrangement as well as a new "Stack" arrangement, shown in Figure 4.15, which has the bottle

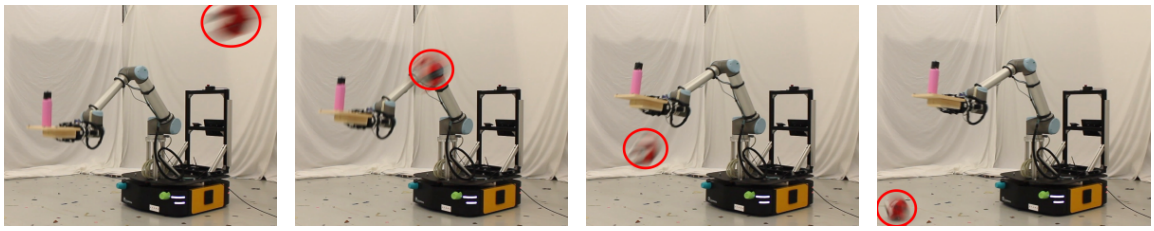

Figure 4.13: Example of the robot dodging the volleyball (circled red) while balancing the bottle, with frames spaced by $\Delta = 0.15\,\mathrm{s}$. Once the ball has passed, the EE moves back to the initial position.

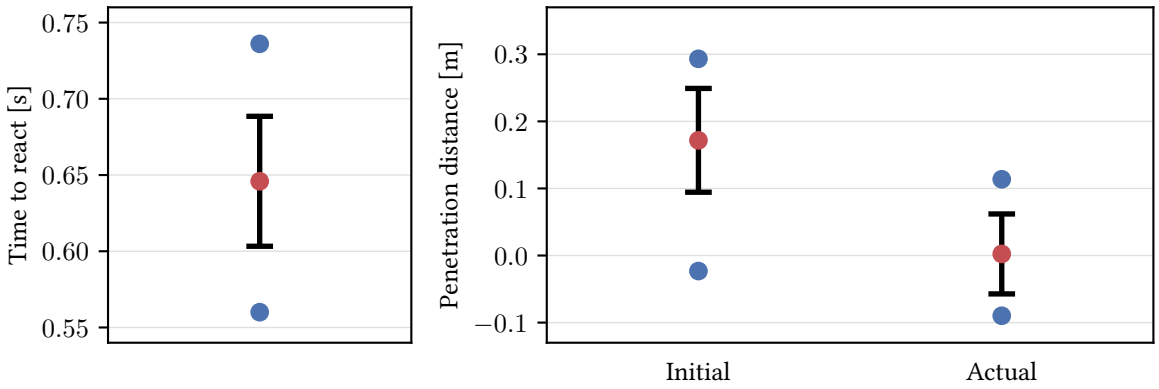

Figure 4.14: The projectile avoidance results over 20 trials. In each plot the red dot is the mean, the error bars represent the standard deviation, and the blue dots are the minimum and maximum values. *Left:* The time at which collision would first occur if the robot did not move. In all cases the controller has less than $0.75\,\mathrm{s}$ to react. *Right:* Maximum penetration distance between the (virtual) collision spheres around the ball and EE. The "Initial" values represent the maximum penetration distances that would have occurred if the robot had not moved. The "Actual" values are what really happened given that the robot did move.

placed atop a tall cardboard box.

We perform experiments in freespace with the same waypoints as the previous experiments. However, we do not test $\boldsymbol{r}_{d_2}$ with the Stack arrangement, because in this case if the objects fall, they fall onto the robot itself, possibly resulting in damage. The results of the experiments are shown in Figure 4.16. The Aligned approach always drops the Bottle for $\boldsymbol{r}_{d_1}$ but it is successful for $\boldsymbol{r}_{d_2}$ and $\boldsymbol{r}_{d_3}$. It always drops the Stack for both $\boldsymbol{r}_{d_1}$ and $\boldsymbol{r}_{d_3}$; we do not test $\boldsymbol{r}_{d_2}$ because in this case if the objects fall, they fall onto the robot itself, possibly resulting in damage. The Aligned approach fails because angular motion of EE results in higher inertial forces acting at CoMs that are farther away, so the taller Stack falls over more easily. Our proposed Robust approach is able to successfully handle both the Bottle and the Stack in all cases.

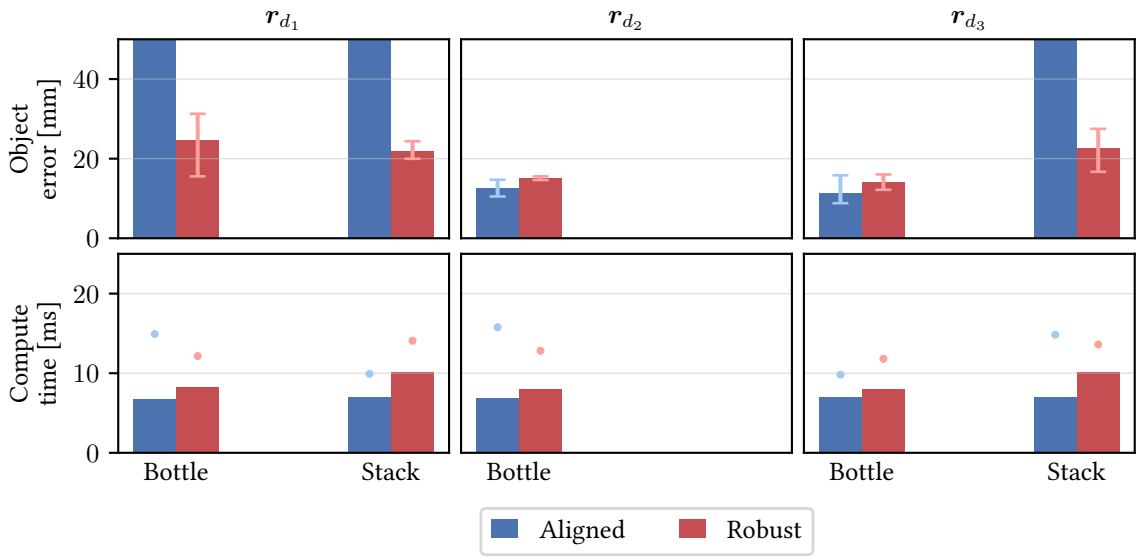

Figure 4.16: The same results as shown in Figure 4.9 but comparing the Robust and Aligned approaches with the Bottle and Stack arrangements. The Aligned approach always drops the Bottle for r_{d_1} but it is successful for r_{d_2} and r_{d_3}. It always drops the Stack for both r_{d_1} and r_{d_3}; we do not test r_{d_2} because in this case if the objects fall, they fall onto the robot itself, possibly resulting in damage. Our proposed Robust approach is successful in all cases.

4.9 Conclusion

We presented the first MPC-based approach for nonprehensile object transportion with a mobile manipulator and demonstrated its performance in simulated and real experiments in a variety of static and dynamic scenarios. Notably, our method is also the first to handle moving obstacles. In addition, we proposed using minimal values of the friction coefficients to add robustness to frictional uncertainty and other force disturbances, and demonstrated that this approach is effective and computationally efficient in real-world experiments.

Figure 4.15: The "Stack" arrangement, which consists of the bottle stacked atop an empty cardboard box.

Chapter 5

Nonprehensile Object Transportation with Uncertain Inertial Parameters

In this chapter we build upon our approach for the waiter's problem from Chapter 4 to consider uncertainty in the transported object's inertial parameters. It is based on the following publication:

> A. Heins and A. P. Schoellig, "Robust Nonprehensile Object Transportation with Uncertain Inertial Parameters," *IEEE Robotics and Automation Letters*, vol. 10, iss. 5, 4492–4499, 2025.

This is the first time that moment relaxations have been used to characterize the set of physically realizable inertial parameters and the first time that this set of parameters has been used to analyze the worst-case constraint satisfaction of a robotic trajectory.

5.1 Introduction

We build on our previous work on the waiter's problem for mobile manipulators from Chapter 4. In contrast to the approach from Chapter 4, which focused on fast online replanning to react to dynamic obstacles while balancing objects with *known* properties, here we focus on offline planning for a transported object with *unknown* inertial parameters—that is, the values of the mass, center of mass (CoM), and inertia matrix are not known exactly but rather lie in some set. Our approach is to plan trajectories to reach a desired EE position while satisfying constraints that ensure the transported object does not move with respect to the tray (see Figure 5.1). These *sticking constraints* (so-called because they ensure the object "sticks" to the tray) depend on the geometric, frictional, and inertial properties of the object. The geometry of the object can in principle be estimated visually (e.g., using a camera), while frictional uncertainty can be reduced by using a high-friction material for the tray surface or by using a low friction coefficient in the planner. However, the inertial properties can only be identified by moving the object around (see e.g. [23], [94]), which is time-consuming and could result in the object being dropped and damaged. Instead, we propose

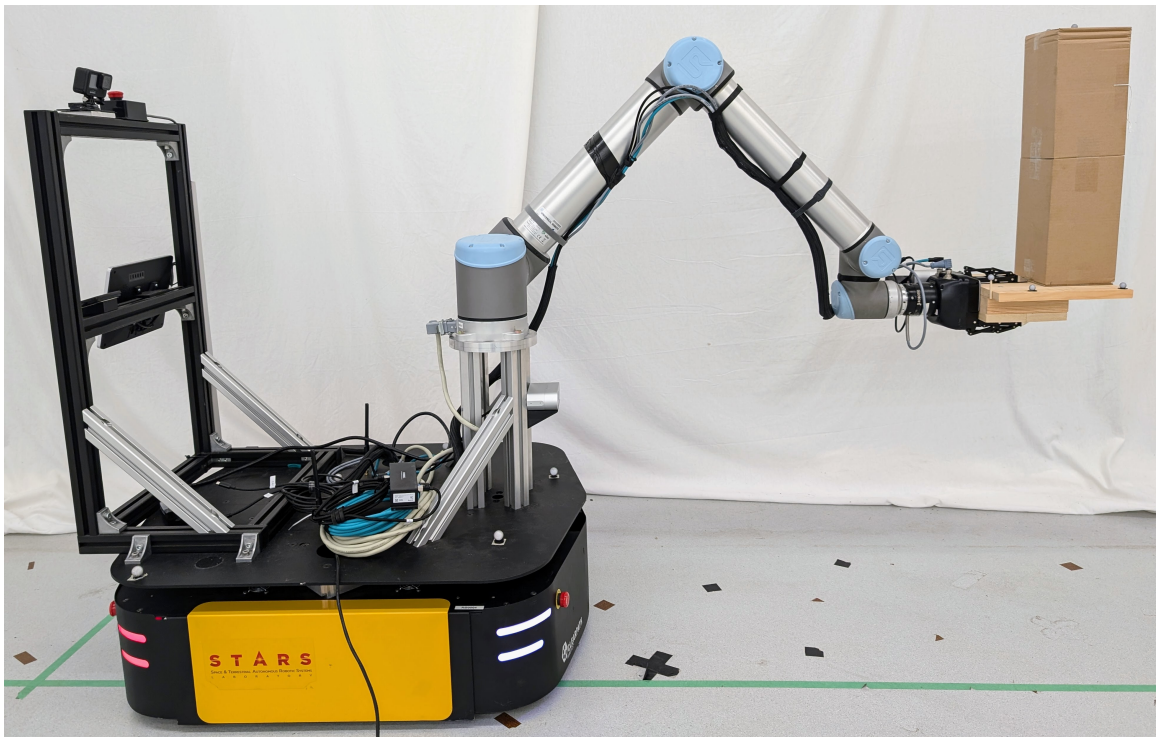

Figure 5.1: The goal of this work is to move an object on a tray to a desired position without dropping it, despite the inertial parameters of the object being uncertain. Here our mobile manipulator is transporting a tall box with uncertain contents. A video of our experiments is available at http://tiny.cc/upright-robust.

using *robust* constraints that successfully transport the object despite the presence of substantial inertial parameter uncertainty. Notably, we assume the CoM can be located at any height within the object, and that the inertia matrix can take any physically realizable value.

We focus on balancing a single tall object with known geometry but unknown mass and inertia matrix, and where the CoM is assumed to lie in a known polyhedral convex set. We use the object's known geometry to constrain the set of possible inertial parameters. A set of inertial parameters can only be physically realized on a given shape if there exists a corresponding mass density function which is zero everywhere outside that shape [23]. We develop necessary conditions for the inertial parameters to be physically realizable on a bounding shape based on moment relaxations [20]. These realizability conditions allow us to verify that a planned trajectory does not violate the sticking constraints for *any* physically realizable value of the inertial parameters, providing theoretical guarantees for the robustness of our planned trajectories. In summary, the contributions of this work are:

- a planner for nonprehensile object transportation that explicitly handles objects with uncertain CoMs, extending the framework from Chapter 4;
- a theoretical analysis of the sticking constraint satisfaction in the presence of a bounded

CoM and any physically realizable inertia matrix, based on moment relaxations [20]—this is the first time that moment relaxations have been used to characterize the set of physically realizable inertial parameters and the first time that this set has been used to analyze the worst-case constraint satisfaction of a robotic trajectory;

- simulations and hardware experiments demonstrating that our proposed robust constraints successfully transport the object—despite using tall objects with high inertial parameter uncertainty—while baseline approaches drop the object;
- an open-source implementation of our planner, available at https://github.com/utiasDSL/upright.

5.2 Related Work

In Chapter 4, we proposed the first whole-body MPC for a mobile manipulator solving the waiter's problem [2]. In contrast to other MPC approaches like [28] and [84], which are only applied to fixed-based arms, our MPC approach optimizes the joint-space trajectory online while considering task-space objectives and constraints, which allows us to respond quickly to changes in the environment like dynamic obstacles. Other approaches to the waiter's problem include offline planning [28], [79], [95], [96] and reactive control [74], [75], [80], [81]. However, most of these approaches (including ours from Chapter 4) assume that the inertial parameters of the transported objects are known. In this work, we are interested in solving the waiter's problem despite inertial parameter uncertainty.

One possible approach to handling inertial parameter uncertainty is to simulate the motion of a pendulum with the EE, which naturally minimizes lateral forces acting on the transported object without explicitly modelling it. However, so far this approach has only been used to minimize slosh when transporting liquids [74], [75] rather than (uncertain) rigid bodies. Another approach is [96], which develops a robust planner for the waiter's problem based on reachability analysis, with parameter uncertainty represented using polynomial zonotopes. In contrast to our work, [96] focuses on small amounts of uncertainty (e.g., a 5% mass and inertia reduction) in both the transported objects and the links of the robot. The resulting trajectories are also quite slow, with negligible inertial acceleration (i.e., quasistatic). Instead, we achieve fast and dynamic motion with tall objects under high parameter uncertainty (i.e., CoMs located at any height in the object and *any* realizable inertia matrix), but we assume that uncertainty in the robot model is negligible (i.e., we use a well-calibrated industrial robot).

Our formulation of robust sticking constraints draws inspiration from legged robot balance control [32], [97]–[99]. Indeed, the task of balancing an object on a tray is quite similar to balancing a legged robot represented with a reduced-order model, which uses the centroidal dynamics of the robot (i.e., the dynamics of a rigid body). In [97]–[99], the CoM of the robot is assumed to be uncertain and motions are generated that keep the robot balanced for any possible CoM value in a polyhedral set. In particular, we follow a similar approach to [99] for handling polyhedral CoM

uncertainty by enforcing sticking constraints corresponding to a CoM located at each vertex of the set, and demonstrate its effectiveness for the waiter's problem. In contrast to these approaches, however, we are also interested in modelling and handling uncertainty in the inertia matrix. While the CoM can reasonably and intuitively be assumed to live in some convex polyhedral set, the inertia matrix is more complicated. When considering the inertia matrix, the set of physically realizable inertial parameters is *spectrahedral* (i.e., it can be represented using linear matrix inequalities (LMIs)) rather than polyhedral, as discussed in [23]. We develop necessary conditions for the inertial parameters to be physically realizable on a bounding shape, based on moment relaxations [20], which we use as constraints in a semidefinite program (SDP) to verify that our planned trajectories do not violate any sticking constraints despite inertial parameter uncertainty. Moment relaxations (i.e., Lasserre's hierarchy) have previously been applied in robotics for tasks like certifiable localization [100] and trajectory planning [101], but not to bounds on the inertial parameters of a rigid body.

5.3 Background

We begin by introducing some mathematical preliminaries, which we will use in Section 5.7.3 to verify that our trajectories do not violate any sticking constraints for any physically realizable value of the transported object's inertial parameters.

5.3.1 Polyhedron Double Description

Any convex polyhedron $\mathcal{P}$ can be described as either the convex hull of its vertices or as a finite intersection of half spaces; this is called the *double description* of $\mathcal{P}$. When $\mathcal{P}$ is also a cone, it is called a polyhedral convex cone (PCC) and can be described using either the *face* or *span* form [32]:

$$\begin{aligned}\mathcal{P} &= \mathrm{face}(\boldsymbol{U}) = \{\boldsymbol{y} \mid \boldsymbol{U}\boldsymbol{y} \leq \boldsymbol{0}\} \\ &= \mathrm{span}(\boldsymbol{V}) = \{\boldsymbol{V}\boldsymbol{z} \mid \boldsymbol{z} \geq \boldsymbol{0}\}.\end{aligned}$$

Following [102], we use the superscript $(\cdot)^F$ to denote conversion from span to face form, such that $\mathrm{face}(\boldsymbol{U}^F) = \mathrm{span}(\boldsymbol{U})$, and we use $(\cdot)^S$ to denote the conversion from face to span form. We perform the conversions using the cdd library [103].

5.3.2 Moment Relaxations

The *moment problem* asks when a sequence corresponds to the moments of some Borel measure. We will make use of SDP relaxations for the moment problem (*moment relaxations*), which we briefly summarize here from [20]. Define $\mathbb{N}^n_d = \{\boldsymbol{\alpha} \in \mathbb{N}^n \mid \sum_{i=1}^n \alpha_i \leq d\}$. Let $\boldsymbol{r} \in \mathbb{R}^n$ be a point and let $\boldsymbol{\alpha} \in \mathbb{N}^n_d$ be a vector of exponents applied elementwise, such that $\boldsymbol{r}^{\boldsymbol{\alpha}} = r_1^{\alpha_1} \dots r_n^{\alpha_n}$. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a polynomial of degree at most d. Then we can write $f(\boldsymbol{r}) = \sum_{\boldsymbol{\alpha} \in \mathbb{N}^n_d} f_{\boldsymbol{\alpha}} \boldsymbol{r}^{\boldsymbol{\alpha}} = \boldsymbol{f}^T \boldsymbol{b}_d(\boldsymbol{r})$, where $\boldsymbol{f} = \{f_{\boldsymbol{\alpha}}\} \in \mathbb{R}^{s(d)}$ is the vector of the polynomial's coefficients with size $s(d) \triangleq \binom{n+d}{d}$

and $\boldsymbol{b}_d(\boldsymbol{r}) = [1, r_1, \ldots, r_n, r_1^2, r_1 r_2, \ldots, r_n^d] \in \mathbb{R}^{s(d)}$ is the basis vector for polynomials of degree at most d in graded lexicographical order. Given a vector $\boldsymbol{z} = \{z_{\boldsymbol{\alpha}}\} \in \mathbb{R}^{s(d)}$, the *Riesz functional* associated with $\boldsymbol{z}$ is $L_{\boldsymbol{z}}(f) = \sum_{\boldsymbol{\alpha} \in \mathbb{N}_d^n} f_{\boldsymbol{\alpha}} z_{\boldsymbol{\alpha}}$, which maps a polynomial $f : \mathbb{R}^n \to \mathbb{R}$ of degree d to a scalar value. Given a vector $\boldsymbol{y} \in \mathbb{R}^{s(2d)}$, the dth-order *moment matrix* associated with $\boldsymbol{y}$ is

$$\boldsymbol{M}_d(\boldsymbol{y}) = L_{\boldsymbol{y}}(\boldsymbol{b}_d(\boldsymbol{r})\boldsymbol{b}_d(\boldsymbol{r})^T) \in \mathbb{R}^{s(d) \times s(d)},$$

where $L_{\boldsymbol{y}}$ is applied elementwise to each element of the matrix $\boldsymbol{b}_d(\boldsymbol{r})\boldsymbol{b}_d(\boldsymbol{r})^T$, such that the element $\boldsymbol{r}^{\boldsymbol{\alpha}}\boldsymbol{r}^{\boldsymbol{\beta}} = \boldsymbol{r}^{\boldsymbol{\alpha}+\boldsymbol{\beta}}$, with $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{N}_d^n$, is mapped to the value $y_{\boldsymbol{\alpha}+\boldsymbol{\beta}}$. In addition, given a polynomial $p : \mathbb{R}^n \to \mathbb{R}$ of degree $2d$, the *localizing matrix* associated with p and $\boldsymbol{y}$ is $\boldsymbol{M}_d(p\boldsymbol{y}) = L_{\boldsymbol{y}}(p(\boldsymbol{r})\boldsymbol{b}_d(\boldsymbol{r})\boldsymbol{b}_d(\boldsymbol{r})^T)$.

Suppose we want to determine if a given sequence $\boldsymbol{y} \in \mathbb{R}^{s(2d)}$, known as a *truncated moment sequence (TMS)*, represents the moments of some Borel measure γ supported in a compact semi-algebraic set $\mathcal{K} = \{\boldsymbol{r} \in \mathbb{R}^n \mid p_j(\boldsymbol{r}) \geq 0, j = 1, \ldots, n_p\}$, where each $p_j(\boldsymbol{r})$ is a polynomial with degree $2v_j$ (even) or $2v_j - 1$ (odd). That is, we want to know if there exists $\gamma : \mathbb{R}^n \to \mathbb{R}_+$ such that

$$\boldsymbol{M}_d(\boldsymbol{y}) = \int_{\mathcal{K}} \boldsymbol{b}_d(\boldsymbol{r})\boldsymbol{b}_d(\boldsymbol{r})^T \, d\gamma(\boldsymbol{r}),$$

which is known as the *truncated K-moment problem (TKMP)*. If such a γ exists, we say that $\boldsymbol{y}$ is *realizable* on $\mathcal{K}$. A necessary condition for γ to exist (see Theorem 3.8 of [20]) is that for any $r \geq d$, we can extend $\boldsymbol{y} \in \mathbb{R}^{s(2d)}$ to some $\boldsymbol{y}' = [\boldsymbol{y}^T, \ldots]^T \in \mathbb{R}^{s(2r)}$ that satisfies

$$\boldsymbol{M}_{r-v_j}(p_j \boldsymbol{y}') \succcurlyeq \boldsymbol{0}, \quad j = 0, \ldots, n_p, \tag{5.1}$$

where we have also defined $p_0(\boldsymbol{r}) = 1$ with $v_0 = 0$. The moment constraints (5.1) become tighter as r increases, forming a hierarchy of SDP relaxations for the TKMP.

Example

For example, suppose $n = 3$ and let

$$\boldsymbol{y} = \begin{bmatrix} y_{000} & y_{100} & y_{010} & y_{001} & y_{200} & y_{110} & y_{101} & y_{020} & y_{011} & y_{002} \end{bmatrix}^T \in \mathbb{R}^{10}$$

be a TMS. Then $\boldsymbol{b}_1(\boldsymbol{r}) = [1, r_1, r_2, r_3]^T$ and the first-order moment matrix is

$$\begin{aligned}\boldsymbol{M}_1(\boldsymbol{y}) &= L_{\boldsymbol{y}}(\boldsymbol{b}_1(\boldsymbol{r})\boldsymbol{b}_1(\boldsymbol{r})^T) \\ &= L_{\boldsymbol{y}}\left(\begin{bmatrix} 1 & r_1 & r_2 & r_3 \\ r_1 & r_1^2 & r_1r_2 & r_1r_3 \\ r_2 & r_1r_2 & r_2^2 & r_2r_3 \\ r_3 & r_1r_3 & r_2r_3 & r_3^2 \end{bmatrix}\right) \\ &= \begin{bmatrix} y_{000} & y_{100} & y_{010} & y_{001} \\ y_{100} & y_{200} & y_{110} & y_{101} \\ y_{010} & y_{110} & y_{020} & y_{011} \\ y_{001} & y_{101} & y_{011} & y_{002} \end{bmatrix}\end{aligned}$$

Next, suppose $\mathcal{K} \subset \mathbb{R}^3$ is an axis-aligned cube of side length $2a$ centered at the origin. Then we can write

$$\mathcal{K} = \{\boldsymbol{r} \in \mathbb{R}^3 \mid p_j(\boldsymbol{r}) \geq 0, j = 1, \ldots, 6\},$$

where

$$\begin{aligned} p_1(\boldsymbol{r}) &= a - r_1, & p_2(\boldsymbol{r}) &= a - r_2, & p_3(\boldsymbol{r}) &= a - r_3, \\ p_4(\boldsymbol{r}) &= a + r_1, & p_5(\boldsymbol{r}) &= a + r_2, & p_6(\boldsymbol{r}) &= a + r_3, \end{aligned}$$

are (affine) polynomials that each define a face of the cube and $v_j = 1$ for each $j = 1, \ldots, 6$. Suppose we want to find an upper bound on some linear function of $\boldsymbol{y}$ subject to the constraint that $\boldsymbol{y}$ is realizable on $\mathcal{K}$. We need to enforce constraints of the form (5.1) on the localizing matrices corresponding to each polynomial p_j. Letting $r = 2$, the localizing matrix corresponding to p_1 is

$$\begin{aligned}\boldsymbol{M}_{r-v_1}(p_1\boldsymbol{y}') &= \boldsymbol{M}_1(p_1\boldsymbol{y}') \\ &= L_{\boldsymbol{y}'}(p_1(\boldsymbol{r})\boldsymbol{b}_1(\boldsymbol{r})\boldsymbol{b}_1(\boldsymbol{r})^T) \\ &= L_{\boldsymbol{y}'}\left((a - r_1)\begin{bmatrix} 1 & r_1 & r_2 & r_3 \\ r_1 & r_1^2 & r_1r_2 & r_1r_3 \\ r_2 & r_1r_2 & r_2^2 & r_2r_3 \\ r_3 & r_1r_3 & r_2r_3 & r_3^2 \end{bmatrix}\right) \\ &= \begin{bmatrix} ay_{000} - y_{100} & ay_{100} - y_{200} & ay_{010} - y_{110} & ay_{001} - y_{101} \\ ay_{100} - y_{200} & ay_{200} - y_{300} & ay_{110} - y_{210} & ay_{101} - y_{201} \\ ay_{010} - y_{110} & ay_{110} - y_{210} & ay_{020} - y_{120} & ay_{011} - y_{111} \\ ay_{001} - y_{101} & ay_{101} - y_{201} & ay_{011} - y_{111} & ay_{002} - y_{102} \end{bmatrix},\end{aligned}$$

where

$$\boldsymbol{y}' = \begin{bmatrix} \boldsymbol{y}^T & y_{300} & y_{210} & y_{201} & y_{120} & y_{111} & y_{102} & y_{030} & \cdots & y_{003} & \cdots & y_{004} \end{bmatrix}^T \in \mathbb{R}^{35}$$

is an extended TMS with dimension $s(2r) = 35$. The localizing matrices for the remaining polynomials are constructed similarly. If we were to increase r, $\boldsymbol{y}'$ and the localizing matrices would increase in size and include higher-order terms.

Finally, suppose $f(\boldsymbol{y}) = \boldsymbol{d}^T\boldsymbol{y}$ is the linear function for which we want to compute an upper bound, where $\boldsymbol{d} \in \mathbb{R}^{10}$ is a known constant. The upper bound is given by the solution to the SDP

$$\begin{aligned} \underset{\tilde{\boldsymbol{y}}}{\text{maximize}} \quad & \boldsymbol{d}'^T\boldsymbol{y}' \\ \text{subject to} \quad & \boldsymbol{M}_{2-v_j}(\tilde{\boldsymbol{y}}) \succcurlyeq \boldsymbol{0}, \quad j = 0, \ldots, 6, \end{aligned}$$

where $\boldsymbol{d}' = [\boldsymbol{d}^T, \boldsymbol{0}^T]^T$ such that $\boldsymbol{d}'^T\boldsymbol{y}' = f(\boldsymbol{y})$.

5.4 Modelling

Next we present the models of the robot and object.

5.4.1 Robot Model

As in Chapter 4, we consider a velocity-controlled mobile manipulator with state $\boldsymbol{x} = [\boldsymbol{q}^T, \boldsymbol{\nu}^T, \dot{\boldsymbol{\nu}}^T]^T \in \mathbb{R}^{n_x}$, where $\boldsymbol{q}$ is the generalized position, which includes the planar pose of the mobile base and the arm's joint angles, and $\boldsymbol{\nu}$ is the generalized velocity. The input $\boldsymbol{u} \in \mathbb{R}^{n_u}$ is the generalized jerk. We use a kinematic model, which we represent generically as $\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}$, with $\boldsymbol{a}(\boldsymbol{x}) \in \mathbb{R}^{n_x}$ and $\boldsymbol{B}(\boldsymbol{x}) \in \mathbb{R}^{n_x \times n_u}$. Though the actual commands sent to the robot are velocities, including acceleration and jerk in the model allows us to reason about the sticking constraints and encourage smoothness.

5.4.2 Object Model

We model the transported object as a rigid body subject to the Newton-Euler equations (2.1), expressed in a frame attached to the EE, where the GIW acting on the object is

$$\boldsymbol{w}_{\mathrm{GI}} \triangleq \boldsymbol{\Xi}\boldsymbol{\eta} - \mathrm{ad}(\boldsymbol{\xi})^T\boldsymbol{\Xi}\boldsymbol{\xi}, \tag{5.2}$$

with $\boldsymbol{\eta}$ the difference between the spatial acceleration and gravity. Recall that the object's spatial mass matrix is defined as

$$\boldsymbol{\Xi} \triangleq \begin{bmatrix} \boldsymbol{I} & m\boldsymbol{c}^\times \\ -m\boldsymbol{c}^\times & m\boldsymbol{1}_3 \end{bmatrix},$$

where m is the object's mass, $\boldsymbol{c}$ is the position of the object's CoM, and $\boldsymbol{I}$ is its inertia matrix.

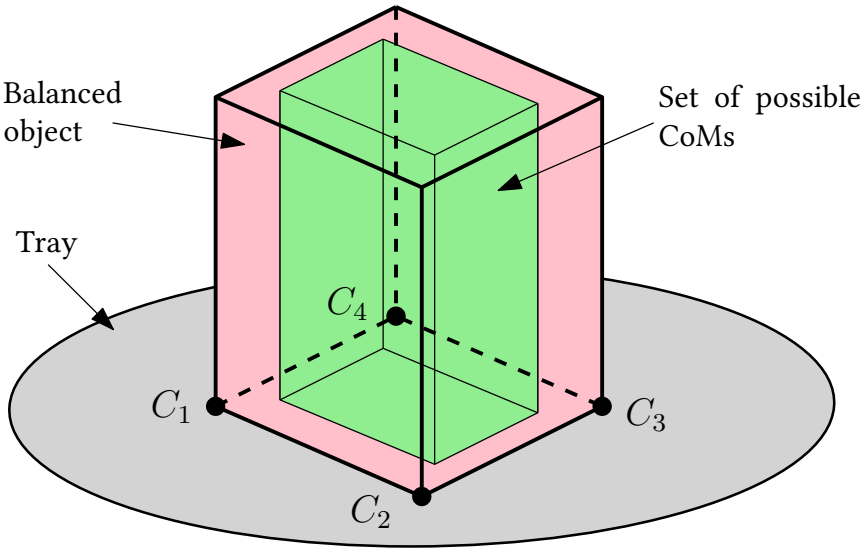

Figure 5.2: A box (red) on a tray, with four contact points C_1–C_4 located at the vertices of the base. We assume that the box's center of mass (CoM) is not known exactly, but rather only known to lie inside some polyhedral set (green).

5.5 Sticking Constraints with Uncertain Inertial Parameters

We want to enforce constraints on the robot's motion so that the transported object does not move relative to the EE (i.e., it "sticks" to the EE). These sticking constraints prevent the object from sliding, tipping, or breaking contact.

5.5.1 Contact Force Constraints

We can ensure an object sticks to the EE by including all contact forces directly into the motion planner and constraining the solution to be consistent with the desired (sticking) dynamics, as we did in Chapter 4. We assume there are n_c contact points $\{C_i\}_{i=1}^{n_c}$ between the object and tray (see Figure 5.2), with corresponding contact forces $\{\boldsymbol{f}_i\}_{i=1}^{n_c}$. By Coulomb's law, each contact force $\boldsymbol{f}_i \in \mathbb{R}^3$ must lie inside its friction cone. Following Section 2.2.2, we gather the contact forces into the vector $\boldsymbol{\zeta} = [\boldsymbol{f}_1^T, \ldots, \boldsymbol{f}_{n_c}^T]^T$. The set of all feasible contact forces lies in the CWC defined in (2.15) as

$$\mathcal{W}_{\mathrm{C}} = \{\boldsymbol{G}\boldsymbol{\zeta} \mid \boldsymbol{F}\boldsymbol{\zeta} \leq \boldsymbol{0}\} \subseteq \mathbb{R}^6,$$

where $\boldsymbol{G}$ is known as the *grasp matrix*. From (2.1), we must have $\boldsymbol{w}_{\mathrm{GI}} \in \mathcal{W}_{\mathrm{C}}$ for the object to remain stationary relative to the EE.

5.5.2 Robustness to Inertial Parameter Uncertainty

Let $\boldsymbol{\theta} \in \mathbb{R}^{10}$ be the inertial parameter vector for the object as defined in (2.6). Let us assume that the exact value of $\boldsymbol{\theta}$ is unknown, but that it lies inside a set Θ. Since $\boldsymbol{w}_{\mathrm{GI}}$ is linear in $\boldsymbol{\theta}$ [94], we can

write the set of possible GIWs under inertial parameter uncertainty as

$$\mathcal{W}_{\mathrm{GI}}(\boldsymbol{\xi}, \boldsymbol{\eta}) = \{\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \mid \boldsymbol{\theta} \in \Theta\}$$

where $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta}) \in \mathbb{R}^{6\times 10}$ is known as the *regressor matrix*. To ensure the sticking constraints are satisfied for any $\boldsymbol{\theta} \in \Theta$, we want to generate EE motions $(\boldsymbol{\xi}, \boldsymbol{\eta})$ that satisfy

$$\mathcal{W}_{\mathrm{GI}}(\boldsymbol{\xi}, \boldsymbol{\eta}) \subseteq \mathcal{W}_{\mathrm{C}} \tag{5.3}$$

at all timesteps. More concretely, (5.3) is satisfied if and only if, for each $\boldsymbol{\theta} \in \Theta$, there exists a set of contact forces $\boldsymbol{\zeta}$ satisfying $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} = \boldsymbol{G}\boldsymbol{\zeta}$ and $\boldsymbol{F}\boldsymbol{\zeta} \leq \mathbf{0}$.

We can enforce (5.3) using only constraints on the extreme points $\mathrm{ex}(\Theta)$ of Θ. To see this, observe that since $\mathcal{W}_{\mathrm{C}}$ is convex, any convex combination of points in $\mathcal{W}_{\mathrm{C}}$ also lies in $\mathcal{W}_{\mathrm{C}}$. And since any $\boldsymbol{\theta} \in \Theta$ is a convex combination of points in $\mathrm{ex}(\Theta)$, it follows that $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \in \mathcal{W}_{\mathrm{C}}$ for any $\boldsymbol{\theta} \in \Theta$ as long as $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \in \mathcal{W}_{\mathrm{C}}$ for all $\boldsymbol{\theta} \in \mathrm{ex}(\Theta)$.

Furthermore, for lightweight objects, we can ignore the value of the object's mass when transporting a single object.[1] To see this, suppose the true inertial parameter vector is $\boldsymbol{\theta} = [m, m\boldsymbol{c}^T, \mathrm{vech}(\boldsymbol{I})^T]^T \in \Theta$. Since $\mathcal{W}_{\mathrm{C}}$ is a convex cone and $m > 0$, it follows that $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \in \mathcal{W}_{\mathrm{C}}$ if and only if $\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\hat{\boldsymbol{\theta}} \in \mathcal{W}_{\mathrm{C}}$, where $\hat{\boldsymbol{\theta}} \triangleq \boldsymbol{\theta}/m$ is the *mass-normalized* parameter vector. In other words, for any $\boldsymbol{\theta} \in \Theta$, we can always instead use $\hat{\boldsymbol{\theta}}$ to enforce the sticking constraints, which is independent of the true mass. This result holds for any object that is not in contact with any others (except of course the tray). However, when multiple objects are in contact with each other, the force transmitted between them depends on their relative masses, so we can no longer ignore them.

Finally, we will also ignore uncertainty in the inertia matrix *while planning* trajectories, as it would be expensive to enforce the LMI constraint required for physical realizability [23]. Instead, we will check our trajectories *after planning* to verify that the sticking constraints are satisfied for any physically realizable inertial parameters—our experiments suggest that handling large uncertainty in the CoM gives us enough robustness to handle any physically realizable inertia matrix as well. We will return to the analysis of physically realizable inertial parameters in Section 5.7. For now, since we are ignoring uncertainty in the mass and inertia matrix, we are left with uncertainty in the CoM. Similar to [99], our approach therefore is to assume that $\boldsymbol{c}$ belongs to a known polyhedral set $\mathcal{C}$ with n_v vertices (see Figure 5.2) and enforcing sticking constraints for n_v objects, which differ only in that the ith object has CoM located at the ith vertex of the CoM uncertainty set. This is equivalent to enforcing (5.3).

[1] We are assuming that the object has a small enough mass for the robot's internal position controller to accurately track the desired trajectory despite the uncertain payload.

5.6 Robust Planning

Our goal is to generate a state-input trajectory for our robot that satisfies the robust sticking constraints developed in the previous section. We formulate the motion planning problem as a constrained optimal control problem (OCP), similar to the formulation in Chapter 4. In contrast to that formulation, however, which solves the OCP online in a model predictive control framework, we solve it once offline with a longer time horizon and then track the resulting optimal trajectory; we found this approach to be more reliable in our experiments. In particular, we found that a longer time horizon converged to the desired position faster with fewer oscillations, and that updating the desired trajectory online could cause additional EE vibration that made it more likely for tall, uncertain objects to be dropped. The trajectories $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$, and $\boldsymbol{\zeta}(t)$ are optimized over a duration T by solving a nonlinear optimization problem. Suppressing the time dependencies, the problem is

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{u},\boldsymbol{\zeta}}{\operatorname{argmin}} \quad & \frac{1}{2}\int_0^T \ell(\boldsymbol{x},\boldsymbol{u})\,dt \\ \text{subject to} \quad & \dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u} && \text{(robot model)} \\ & \mathcal{W}_{\mathrm{GI}}(\boldsymbol{x}) \subseteq \mathcal{W}_{\mathrm{C}} && \text{(sticking)} \\ & \underline{\boldsymbol{x}} \le \boldsymbol{x} \le \bar{\boldsymbol{x}} && \text{(state limits)} \\ & \underline{\boldsymbol{u}} \le \boldsymbol{u} \le \bar{\boldsymbol{u}} && \text{(input limits)} \\ & \boldsymbol{\varphi}(\boldsymbol{x}_f) = \boldsymbol{0} && \text{(terminal constraint)} \end{aligned} \tag{5.4}$$

where the stage cost is

$$\ell(\boldsymbol{x},\boldsymbol{u}) = \|\Delta\boldsymbol{r}(\boldsymbol{x})\|^2_{\boldsymbol{W}_r} + \|\boldsymbol{x}\|^2_{\boldsymbol{W}_x} + \|\boldsymbol{u}\|^2_{\boldsymbol{W}_u},$$

with $\|\cdot\|^2_{\boldsymbol{W}} = (\cdot)^T\boldsymbol{W}(\cdot)$ for weight matrix $\boldsymbol{W}$, and $\Delta\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{r}_d - \boldsymbol{r}_e(\boldsymbol{x})$ is the EE position error between the desired position $\boldsymbol{r}_d$ and the current position $\boldsymbol{r}_e(\boldsymbol{x})$. The sticking constraints implicitly depend on the contact forces $\boldsymbol{\zeta}$, and we have expressed $\mathcal{W}_{\mathrm{GI}}$ as a function of $\boldsymbol{x}$ via forward kinematics. The terminal constraint $\boldsymbol{\varphi}(\boldsymbol{x}_f) = [\Delta\boldsymbol{r}(\boldsymbol{x}_f)^T, \boldsymbol{\nu}_f^T, \dot{\boldsymbol{\nu}}_f^T]^T = \boldsymbol{0}$ acts only on the final state $\boldsymbol{x}_f \triangleq \boldsymbol{x}(T)$ and steers it toward a stationary state with no position error.

We solve (5.4) by discretizing the planning horizon T with a fixed timestep Δt and using sequential quadratic programming (SQP) via the open-source framework OCS2 [87] and quadratic programming solver HPIPM [39], with required Jacobians computed using automatic differentiation. We use the Gauss-Newton approximation for the Hessian of the cost and we soften all state limits and sticking constraints: given a generic state constraint $g(\boldsymbol{x}) \ge 0$, we add a slack variable $s \ge 0$ to relax the constraint to $g(\boldsymbol{x}) + s \ge 0$, and the L_2 penalty $w_s s^2$ is added to the cost with weight $w_s > 0$. As in Chapter 4, we also plan while assuming that there is *zero* contact friction between the tray and transported object (i.e., tangential forces are to be avoided, but small ones are allowed because the constraints are soft). This provides robustness to uncertain friction and other disturbances while also reducing the computational cost of solving (5.4), since each contact force variable need

only be represented by a single non-negative scalar representing the normal force (see the previous chapter for more details). Given the soft constraints and time discretization of (5.4), robust sticking is not *guaranteed* but is heavily *encouraged.*

Once we have solved (5.4) to obtain the planned optimal trajectory $\boldsymbol{x}_d(t) = [\boldsymbol{q}_d^T(t), \boldsymbol{\nu}_d^T(t), \dot{\boldsymbol{\nu}}_d^T(t)]^T$, we need to track it online. At each control timestep, we generate the commanded velocity $\boldsymbol{\nu}_{\text{cmd}}$ using the simple affine control law $\boldsymbol{\nu}_{\text{cmd}} = \boldsymbol{K}_p(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{\nu}_d$, where $\boldsymbol{K}_p \in \mathbb{S}^9_{++}$ is a gain matrix.

5.7 Verifying Sticking Constraint Satisfaction

Let us now consider uncertainty in the inertia matrix. We want a way to show that our choice to ignore inertia matrix uncertainty in the planner is justified; that is, given a trajectory, we want to verify that the sticking constraints are not violated for *any* realizable value of the inertia matrix. In this section we develop an SDP to determine an upper bound on the maximum constraint violation given the uncertain inertial parameters and bounding shape for the object.

5.7.1 Double Description of the Contact Wrench Cone

Following [32], we build the face form of the CWC. First, notice that (2.13) describes a PCC $\text{face}(\boldsymbol{F})$. Converting (2.13) to span form and substituting into (2.15), we get $\mathcal{W}_{\text{C}} = \{\boldsymbol{G}\boldsymbol{F}^S\boldsymbol{z} \mid \boldsymbol{z} \geq \boldsymbol{0}\}$. Next, letting $\boldsymbol{H} = (\boldsymbol{G}\boldsymbol{F}^S)^F \in \mathbb{R}^{n_h \times 6}$, we have the face form of the CWC $\mathcal{W}_{\text{C}} = \{\boldsymbol{w} \in \mathbb{R}^6 \mid \boldsymbol{H}\boldsymbol{w} \leq \boldsymbol{0}\}$. This form allows us to write the robust sticking constraints (5.3) as

$$\boldsymbol{H}\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \leq \boldsymbol{0} \;\; \forall \boldsymbol{\theta} \in \Theta. \tag{5.5}$$

Let $\boldsymbol{h}_i^T$ be the ith row of $\boldsymbol{H}$. Then we can rewrite the constraint (5.5) as a set of inner optimization problems

$$\left(\underset{\boldsymbol{\theta} \in \Theta}{\text{maximize}} \;\; \boldsymbol{h}_i^T \boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \right) \leq 0, \tag{5.6}$$

with one problem for each of the n_h rows. This inequality form of the sticking constraints will allow us to solve for the worst-case value of each constraint. However, we first need to determine appropriate bounds on the set Θ.

5.7.2 Physically Realizable Inertial Parameters

We want to constrain the inertial parameters to correspond to some mass density $\rho : \mathbb{R}^3 \to \mathbb{R}_+$ supported entirely in a compact bounding shape $\mathcal{K} \subset \mathbb{R}^3$ containing the transported object, such that

$$\Theta = \left\{ \boldsymbol{\theta} \in \mathbb{R}^{10} \mid \boldsymbol{\Pi}(\boldsymbol{\theta}) = \int_{\mathcal{K}} \tilde{\boldsymbol{r}}\tilde{\boldsymbol{r}}^T \, d\rho(\boldsymbol{r}) \right\},$$

where $\boldsymbol{\Pi}(\boldsymbol{\theta})$ is the pseudo-inertia matrix defined in (2.3), which depends linearly on $\boldsymbol{\theta}$. That is, we want $\boldsymbol{\theta}$ to be physically realizable on $\mathcal{K}$.

Let us apply the moment relaxation machinery from Section 5.3.2 to the problem of physically realizable inertial parameters. We have dimension $n = 3$ and degree $d = 1$. The key insight is that the pseudo-inertia matrix $\boldsymbol{\Pi}(\boldsymbol{\theta})$ is just a permutation[2] of the first-order moment matrix $\boldsymbol{M}_1(\boldsymbol{y})$, where $\boldsymbol{y} \in \mathbb{R}^{10}$ is a TMS. In particular, we have

$$\boldsymbol{M}_1(\boldsymbol{y}) = \boldsymbol{P}\boldsymbol{\Pi}(\boldsymbol{\theta})\boldsymbol{P}^T \tag{5.7}$$

where

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{0}^T & 1 \\ \boldsymbol{1}_3 & \boldsymbol{0} \end{bmatrix}$$

is a permutation matrix. Furthermore, since $\boldsymbol{M}_1$ and $\boldsymbol{\Pi}$ are linear functions of $\boldsymbol{y}$ and $\boldsymbol{\theta}$, respectively, we can include (5.7) as a constraint in a convex program to map between $\boldsymbol{y}$ and $\boldsymbol{\theta}$. Thus, instead of optimizing over $\boldsymbol{\theta}$ constrained to lie in Θ, we will jointly optimize over $\boldsymbol{\theta}$ and $\boldsymbol{y}$ with the constraint (5.7) relating the two variables and additional constraints to ensure $\boldsymbol{y}$ is realizable on $\mathcal{K}$. In particular, suppose $\mathcal{K} = \{\boldsymbol{r} \in \mathbb{R}^3 \mid p_j(\boldsymbol{r}) \geq 0, j = 1, \ldots, n_p\}$, where each $p_j(\boldsymbol{r})$ is a polynomial with degree $2v_j$ (even) or $2v_j - 1$ (odd), and define $p_0(\boldsymbol{r}) = 1$ with $v_0 = 0$. Then (5.1) gives us necessary conditions for the TMS $\boldsymbol{y}$ to be realizable on $\mathcal{K}$ using localizing matrices. Putting it all together, suppose we want to find an upper bound on a linear function $f(\boldsymbol{\theta}) = \boldsymbol{d}^T\boldsymbol{\theta}$ subject to $\boldsymbol{\theta} \in \Theta$, where $\boldsymbol{d} \in \mathbb{R}^{10}$ is a known constant. The upper bound is given by the solution to the SDP

$$\begin{aligned} \underset{\boldsymbol{\theta}, \boldsymbol{y}'}{\text{maximize}} \quad & \boldsymbol{d}^T\boldsymbol{\theta} \\ \text{subject to} \quad & \boldsymbol{M}_1(\boldsymbol{y}') = \boldsymbol{P}\boldsymbol{\Pi}(\boldsymbol{\theta})\boldsymbol{P}^T \\ & \boldsymbol{M}_{r-v_j}(\boldsymbol{y}') \succcurlyeq \boldsymbol{0}, \quad j = 0, \ldots, n_p, \end{aligned} \tag{5.8}$$

where the two constraints come from (5.7) and (5.1), and $\boldsymbol{y}' \in \mathbb{R}^{s(2r)}$ is an extended TMS. Next we will see how to use an SDP similar to (5.8) to check for constraint violations along our planned trajectories.

5.7.3 Worst-Case Sticking Constraints

Given an EE trajectory, we want to determine if any realizable value of the inertial parameters would violate the sticking constraints (5.6) at any time. That is, we would like to know if the optimal value of

$$\underset{\boldsymbol{\theta} \in \Theta}{\text{maximize}} \quad \boldsymbol{h}_i^T\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \tag{5.9}$$

is positive for any row $\boldsymbol{h}_i^T$ of $\boldsymbol{H}$ at any time instant of the trajectory. Letting $\boldsymbol{d}^T = \boldsymbol{h}_i^T\boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})$, the problem (5.9) can be relaxed to the SDP (5.8). Note that (5.8) is a relaxation of (5.9) because the constraints in (5.8) are only *necessary* for physical realizability, not sufficient, and therefore the solution of (5.8) provides an upper bound to the solution of (5.9). We also add constraints to set the

[2]The need for the permutation matrix simply arises from different conventions in the robot dynamics literature [23] and the TKMP literature [20].

mass $m(\boldsymbol{\theta}) = 1$, since the sticking constraints are independent of mass for a single object, and to constrain the CoM $\boldsymbol{c}(\boldsymbol{\theta})$ to be located within some smaller convex polyhedron $\mathcal{C} \subset \mathcal{K}$ (we typically assume the CoM is not located right at the boundary of $\mathcal{K}$). This yields the SDP

$$\begin{aligned} \underset{\boldsymbol{\theta}, \boldsymbol{y}'}{\text{maximize}} \quad & \boldsymbol{h}_i^T \boldsymbol{Y}(\boldsymbol{\xi}, \boldsymbol{\eta})\boldsymbol{\theta} \\ \text{subject to} \quad & \boldsymbol{M}_1(\boldsymbol{y}') = \boldsymbol{P}\boldsymbol{\Pi}(\boldsymbol{\theta})\boldsymbol{P}^T, \\ & \boldsymbol{M}_{2-v_j}(p_j \boldsymbol{y}') \succcurlyeq \boldsymbol{0}, \quad j = 0, \ldots, n_p, \\ & m(\boldsymbol{\theta}) = 1, \\ & \boldsymbol{c}(\boldsymbol{\theta}) \in \mathcal{C}, \end{aligned} \tag{5.10}$$

where we have used order $r = 2$. For simplicity, we assume our bounding shape $\mathcal{K}$ is a convex polyhedron, so each bounding polynomial has the affine form $p_j(\boldsymbol{r}) = \boldsymbol{\alpha}_j^T \boldsymbol{r} + \boldsymbol{\beta}_j$ for some constants $\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j \in \mathbb{R}^3$, and therefore $v_j = 1$ for each $j = 1, \ldots, n_p$.

We verify that a planned trajectory is robust to inertial parameter uncertainty by solving (5.10) pointwise at a fixed frequency along the trajectory. If the optimal value of (5.10) is always nonpositive, then the constraint is not violated; we assume that the time discretization is fine enough that the constraints are not violated between timesteps. Furthermore, we can verify robustness to uncertain friction coefficients at the same time by constructing $\mathcal{W}_{\mathrm{C}}$ and thus $\boldsymbol{H}$ with low friction coefficients: if the constraints are never violated, then any combination of higher friction coefficients and realizable inertial parameters will also satisfy the constraints.

One could also consider enforcing full realizability constraints directly in the planning problem (5.4) to ensure the planned trajectories are robust a priori. However, this would be computationally expensive and numerically challenging because of the LMI constraints required for physical realizability combined with the nonlinearity of the problem. We leave this for future work.

5.8 Simulation Experiments

We begin the evaluation of our proposed robust planning approach in simulation using the PyBullet simulator and a simulated version of our experimental platform, a 9-DOF mobile manipulator consisting of a Ridgeback mobile base and UR10 arm, shown in Figure 5.1. In all experiments (simulated and real) we use $\Delta t = 0.1\,\mathrm{s}$, $T = 10\,\mathrm{s}$, and weights

$$\begin{aligned} \boldsymbol{W}_r &= \mathbf{1}_3, & \boldsymbol{W}_x &= \mathrm{diag}(0\mathbf{1}_9, 10^{-1}\mathbf{1}_9, 10^{-2}\mathbf{1}_9), \\ \boldsymbol{W}_u &= 10^{-3}\mathbf{1}_9, & w_s &= 100. \end{aligned}$$

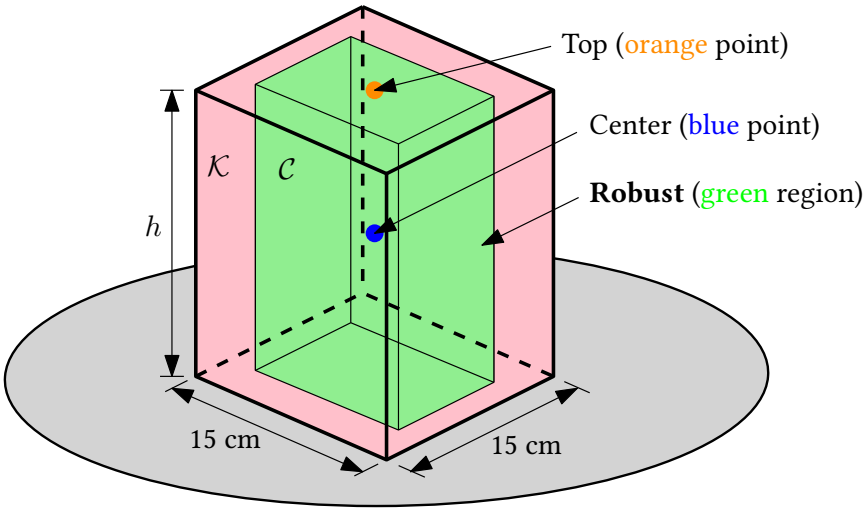

Figure 5.3: We transport a cuboid-shaped object $\mathcal{K}$ (red) with an uncertain CoM contained in $\mathcal{C}$ (green). We test three variations of the sticking constraints: assume the CoM is at the object's centroid (Center), assume it is centered at the top of the object (Top), and Robust, in which the controller enforces sticking constraints for eight different objects, where each has its CoM at one of the vertices of $\mathcal{C}$.

The state and input limits are

$$\bar{\boldsymbol{q}} = \begin{bmatrix} 10\boldsymbol{e}_3 \\ 2\pi\boldsymbol{e}_6 \end{bmatrix}, \bar{\boldsymbol{\nu}} = \begin{bmatrix} 1.1\boldsymbol{e}_2 \\ 2\boldsymbol{e}_3 \\ 3\boldsymbol{e}_4 \end{bmatrix}, \dot{\bar{\boldsymbol{\nu}}} = \begin{bmatrix} 2.5\boldsymbol{e}_2 \\ 1 \\ 10\boldsymbol{e}_6 \end{bmatrix}, \bar{\boldsymbol{u}} = \begin{bmatrix} 20\boldsymbol{e}_3 \\ 80\boldsymbol{e}_6 \end{bmatrix},$$

where $\bar{\boldsymbol{x}} = [\bar{\boldsymbol{q}}^T, \bar{\boldsymbol{\nu}}^T, \dot{\bar{\boldsymbol{\nu}}}^T]^T$, $\underline{\boldsymbol{x}} = -\bar{\boldsymbol{x}}$, $\underline{\boldsymbol{u}} = -\bar{\boldsymbol{u}}$, and $\boldsymbol{e}_n$ denotes an n-dimensional vector of ones. The control gain is $\boldsymbol{K}_p = \mathbf{1}_9$ and the simulation timestep is $0.1\,\mathrm{ms}$.

We are interested in cases where the inertial parameters of the transported object are uncertain and this uncertainty can result in task failures (i.e., the object is dropped) if the uncertainty is ignored. We use a tall box $\mathcal{K}$ with a $15\,\mathrm{cm} \times 15\,\mathrm{cm}$ base and height h (see Figure 5.3) as the transported object. This object is tall relative to its support area and therefore prone to tipping over, particularly if the inertial parameters are not known exactly. Suppose we assume that the CoM can lie anywhere in a box $\mathcal{C}$ with dimensions $12\,\mathrm{cm} \times 12\,\mathrm{cm} \times h$, centered within $\mathcal{K}$; that is, the CoM can be located *anywhere* in the object as long as it is at least $1.5\,\mathrm{cm}$ from the sides. The simulated friction coefficient between the box and tray is $\mu = 0.2$. We compare three sets of sticking constraints (again, see Figure 5.3):

- **Center:** The CoM is located at the center of $\mathcal{C}$;
- **Top:** The CoM is centered on the top face of $\mathcal{C}$;
- **Robust:** A set of eight sticking constraints are used, corresponding to a CoM at each vertex of $\mathcal{C}$.

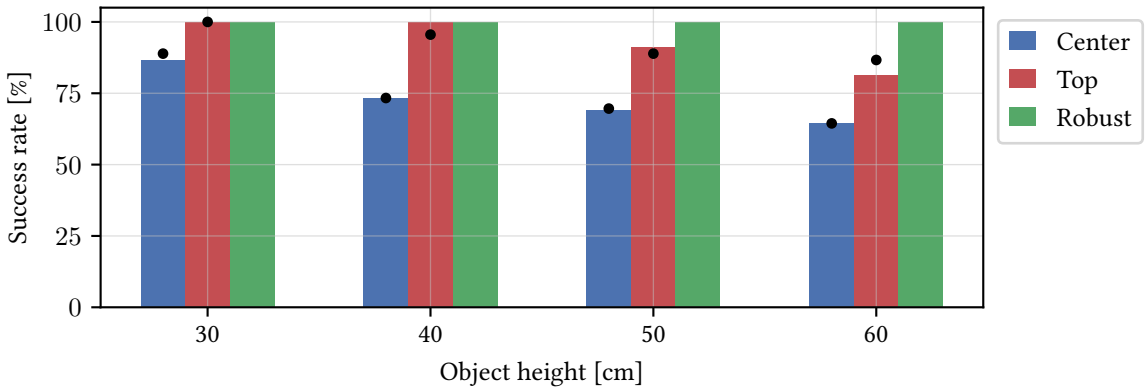

Figure 5.4: Success rate of the different types of sticking constraints for all 135 combinations of desired positions, CoMs, and inertias for each object height and constraint method (1620 total runs). The Robust constraints always successfully transport the object, while the other constraint types result in an increasing number of failures as the object height increases; the rate varies slightly with the number of SQP iterations n_s. Here the bar shows $n_s = 3$ and the black dot $n_s = 10$.

In all cases, the inertia matrix used in the planner is set to correspond to a uniform mass density. The first two constraint methods are baselines where we are not explicitly accounting for the uncertainty in the parameters. Intuitively, it is more difficult to transport an object with a higher CoM, so we may expect the Top constraints to be more successful than the Center constraints. In contrast to these baselines, our proposed Robust constraints explicitly handle uncertainty in the CoM.

We test all combinations of the following: three desired positions $\boldsymbol{r}_{d_1} = [-2, 1, 0]^T$, $\boldsymbol{r}_{d_2} = [0, 2, 0.25]^T$, and $\boldsymbol{r}_{d_3} = [2, 0, -0.25]^T$ (in meters), 15 different simulated CoM positions (one at the center of $\mathcal{C}$, eight at the vertices of $\mathcal{C}$, and six at the centers of the faces of $\mathcal{C}$), and three different values for the inertia matrix, computed as follows. We solve a convex optimization problem to find the diagonal inertia matrix about the CoM which corresponds to a set of point masses at the vertices of $\mathcal{K}$ with the maximum smallest eigenvalue. This gives us a large realizable inertia matrix value $\boldsymbol{I}$; we then also test with the smaller inertia values $0.5\boldsymbol{I}$ and $0.1\boldsymbol{I}$. The simulated mass is fixed to $m = 1\,\mathrm{kg}$. We test each of the 135 total combinations of trajectory, CoM, and inertia matrix for different object heights h and constraint methods.

The success rates for the simulations are shown in Figure 5.4. The success rate is the percentage of runs (out of the 135 total per object) that successfully deliver the object to the goal without it being dropped from the tray. The Robust constraints are always successful (with a maximum object displacement of only $2\,\mathrm{mm}$). The Top and Center constraints produce fewer successful runs as the object height increases, which suggests that the sticking constraints are more sensitive to parameter error for taller objects. The success rate of the Top and Center constraints varies slightly when using more SQP iterations n_s to solve (5.4), at the cost of a longer solve time. We report the results for $n_s = 3$ and $n_s = 10$ (the Robust constraints are completely successful for both values of n_s, so we only report $n_s = 3$); increasing to $n_s = 20$ did not produce a higher success rate. The average

Table 5.1: Maximum constraint violation for each object height and sticking constraint method using the moment conditions for physical realizability. A negative value means that the constraints are never violated for any realizable inertial parameters, which is only the case using our proposed Robust constraints. These results are for $n_s = 3$; the values with $n_s = 10$ are very similar.

Height [cm]	Center	Top	Robust
30	2.48	4.51	−1.15
40	5.03	7.19	−0.97
50	8.21	9.09	−0.77
60	11.96	10.50	−0.56

solve times for the Center constraints are $110\,\mathrm{ms}$ ($n_s = 3$) and $310\,\mathrm{ms}$ ($n_s = 10$); for the Top constraints $111\,\mathrm{ms}$ ($n_s = 3$) and $324\,\mathrm{ms}$ ($n_s = 10$); and for the Robust constraints $324\,\mathrm{ms}$. Even when more compute time is used to refine the trajectory, the baselines are still not as successful as the Robust constraints.

While the results in Figure 5.4 show that our proposed constraints are robust for *particular* combinations of CoM positions and inertia matrices, Table 5.1 shows the maximum possible sticking constraint violations for *any* realizable inertial parameter value, obtained by solving (5.10) at each point along the planned trajectory (discretized with a $10\,\mathrm{ms}$ interval). Solving (5.10) for all $i = 1, \ldots, n_h$ at a single timestep took about $2.2\,\mathrm{s}$, with $n_h = 26$ for our experimental setup. The Robust constraints have no violation for any possible value of the inertia matrix (while the Center and Top constraints always do), justifying our decision to ignore uncertainty in the inertia matrix within the planner. While negative violation implies failure should not occur; positive violation does not mean that it *will* occur.

Finally, recall that the simulated value of the friction coefficient between the tray and object is $\mu = 0.2$. This value does not impact the behavior of the planner because the planner assumes $\mu = 0$ and then tries to satisfy the softened sticking constraints approximately. However, the underlying value of μ can potentially affect the amount of constraint violation, since $\boldsymbol{h}_i$ in (5.10) depends on the friction coefficient. Interestingly, we evaluated the constraint violation for the robust constraints with $h = 60\,\mathrm{cm}$ and $\mu = 0.1$ and found the maximum constraint violation to be the same as with $\mu = 0.2$, which suggests that the friction coefficient is not the limiting factor when transporting tall objects like those used here.

5.9 Hardware Experiments

We also perform experiments on our real mobile manipulator balancing a cardboard box, as shown in Figure 5.1. We test two heights of box, Box1 and Box2, each containing a bottle filled with sugar in one corner to offset the CoM (see Figure 5.5). Box1 has a height of $h_1 = 28\,\mathrm{cm}$ and a square base with side length $15\,\mathrm{cm}$. Its total mass is $933\,\mathrm{g}$, with the bottle contributing $722\,\mathrm{g}$. Box2 is made of two stacked boxes attached together, with the top one containing the bottle. Its total mass is $1046\,\mathrm{g}$ and its height is $h_2 = 56\,\mathrm{cm}$; its base dimensions are the same as Box1. A rigid

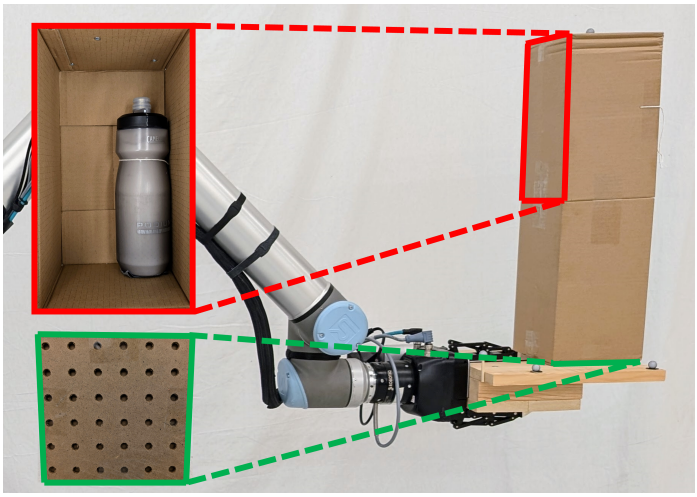

Figure 5.5: Our transported objects are boxes containing a bottle filled with sugar to offset the CoM and to make the task of balancing more difficult. One cannot tell how the box is packed (and therefore what its mass distribution is) just by looking at it. Box2 (shown on the right) consists of two boxes attached together; Box1 is a single box. A firm base board (green) is attached to the bottom box to provide a consistent contact area with the tray.

board is attached to the base of the boxes to ensure consistent contact with the tray (again, refer to Figure 5.5). The friction coefficient between the boxes (with the attached base board) and the tray was experimentally measured to be $\mu = 0.29$. Position feedback is provided for the arm by joint encoders at $125\,\text{Hz}$ and for the base by a Vicon motion capture system at $100\,\text{Hz}$. The laptop specifications and planner parameters are the same as in simulation. The control loop is run at the arm's control frequency of $125\,\text{Hz}$.

We consider the scenario when the planner does not know how the box is packed, and therefore its inertial parameters are not known exactly. We again test the Center, Top, and Robust constraints using the desired positions $\boldsymbol{r}_{d_1}$ and $\boldsymbol{r}_{d_2}$.[3] The Robust constraints assume the CoM lies in the cuboid $\mathcal{C}$ with dimensions $12\,\text{cm} \times 12\,\text{cm} \times h_i$, where $i = 1, 2$ corresponds to Box1 or Box2, which is large enough to contain any possible centroid of the bottle no matter its position in the box. We perform up to three runs of each combination of desired position and constraint method; if a given combination method fails before completing three runs, we stop to avoid extra damage to the boxes. Using $n_s = 3$, each of the constraint methods successfully transported Box1 for three runs, but only the Robust constraints were able to do so with Box2 (with either $n_s = 3$ and $n_s = 10$). With both values of n_s, the Center constraints failed immediately with $\boldsymbol{r}_{d_1}$, and completed one run of $\boldsymbol{r}_{d_2}$ before dropping the box on the second run; the Top constraints failed immediately for both desired positions. The maximum object displacement errors (for $n_s = 3$) are shown in Figure 5.6. The Robust constraints produce smaller errors with both Box1 and Box2 (there is still *some* error, due to unmodelled effects like vibrations and inevitable model inaccuracies); the Center and Top

[3] We did not use the desired position $\boldsymbol{r}_{d_3}$ because in that case if the box falls, it falls onto the robot, possibly causing damage.

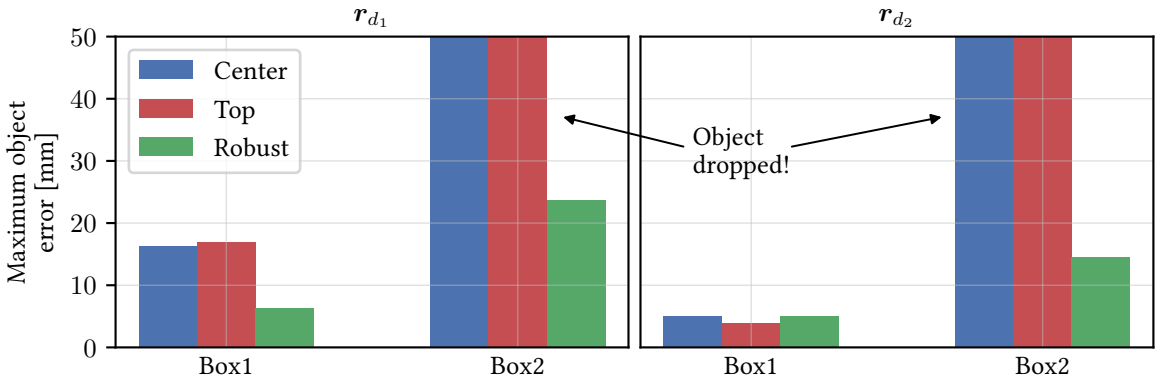

Figure 5.6: Maximum object displacement error for the different constraint methods, desired positions, and box objects. The object displacement error is the maximum distance the object moves from its initial position relative to the tray. The errors are similar between each method with Box1 (the shorter box), but the Center and Top baselines fail with Box2 (the taller box), while the proposed Robust constraints successfully transport it.

Table 5.2: The maximum planned EE velocity and acceleration as well as the root-mean-square tracking error of the arm's joint angles and the base's position and yaw angle in the hardware experiments.

	Box1			Box2		
	Center	Top	Robust	Center	Top	Robust
Max. vel. [m/s]	2.32	2.36	1.27	2.36	2.43	1.08
Max. acc. [m/s^2]	4.24	4.46	1.17	4.46	4.88	0.87
Arm err. [deg]	0.38	0.37	0.22	0.39	0.33	0.21
Base pos. err. [cm]	2.67	2.82	1.96	2.56	3.16	1.48
Base yaw err. [deg]	1.80	1.90	0.62	1.68	2.25	0.31

baselines obviously produce large errors with Box2 since the box was dropped. The planning times were similar to those in simulation.

The convergence of the EE to the desired position for r_{d_1} is shown in Figure 5.7 and additional metrics are in Table 5.2. While the Center and Top constraint methods converge faster, this comes with the risk of dropping uncertain objects, especially taller ones like Box2. While the robust constraints still produce fast motion, the maximum velocity and acceleration is reduced compared to the other baselines. The root-mean-square tracking error (RMSE) for the arm is quite low, which suggests that neglecting the object mass was reasonable. The RMSE for the base is higher, suggesting that adding mobility to the waiter's problem requires additional robustness to error.

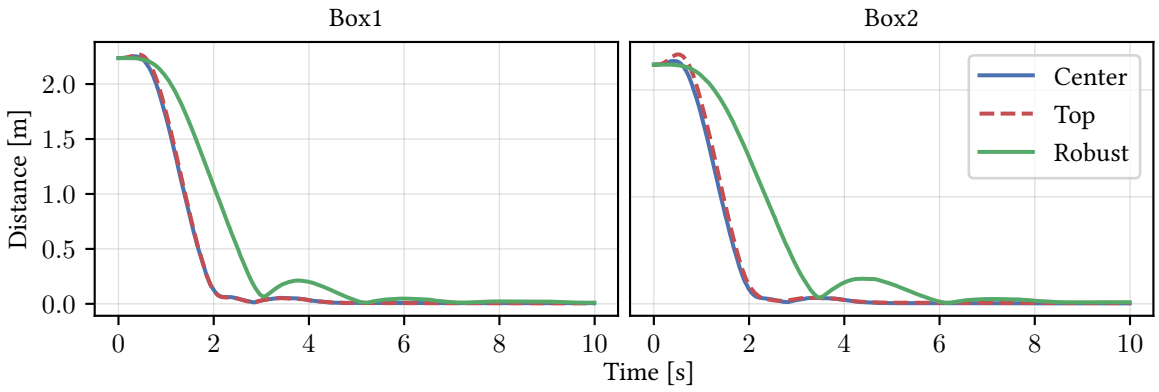

Figure 5.7: Distance between the EE position and the desired position over time for desired position $\boldsymbol{r}_{d_1}$ and each sticking constraint method. The trajectories with the Center and Top constraints, in which only a single CoM value is considered, are nearly the same. The Robust constraints result in slower convergence, but always successfully transport the object.

5.10 Conclusion

We present a planning framework for nonprehensile object transportation that is robust to uncertainty in the transported object's inertial parameters. In particular, we explicitly model and design robust constraints for uncertainty in the object's CoM, and demonstrate successful transportation of tall objects in simulation and on real hardware. We also demonstrate a novel use of moment relaxations to develop conditions for the inertial parameters of the transported object to be physically realizable, which allows us to determine if the constraints would be violated for *any* possible value of the inertial parameters, including the inertia matrix, along the planned trajectories.

Chapter 6

Conclusion

6.1 Summary

This thesis has described new methods for two nonprehensile manipulation tasks: planar pushing and nonprehensile object manipulation. In Chapter 3, we develop the first controller for robotic single-point pushing using *only* force feedback to sense the pushed object. We show that it successfully pushes objects along both straight and curved paths with single-point contact and no model of the object. We demonstrate the robustness of the controller by simulating pushes using a wide variety of slider parameters and initial states. We also present real hardware experiments in which a mobile manipulator successfully pushes different objects across a room along straight and curved paths, including some with static obstacles. Notably, we do not assume that sufficient friction is available to prevent slip at the contact point; slipping is a natural part of the behaviour of our controller and does not necessarily lead to task failure.

In Chapter 4, we propose the first whole-body MPC for a mobile manipulator solving the waiter's problem. It is also the first approach to the waiter's problem that handles dynamic obstacles. Compared to existing MPC-based approaches to this problem, which have only been demonstrated on fixed-base arms, our controller optimizes the joint-space trajectory online directly from task-space objectives and constraints, without the use of a higher-level planning step. In addition, the controller uses the minimum statically feasible friction coefficients, which provides robustness to frictional uncertainty, vibration, and other real-world disturbances. When the minimum statically feasible friction coefficients are *zero*, we show that the MPC problem can be solved more efficiently. Furthermore, we present the first demonstrations of the waiter's problem with a real velocity-controlled mobile manipulator transporting up to seven objects; transporting an assembly of stacked objects; and avoiding static and dynamics obstacles, including a thrown volleyball.

Finally, in Chapter 5 we develop a planner for nonprehensile object transportation that explicitly handles objects with uncertain CoMs, extending the framework from Chapter 4. Furthermore, we perform a theoretical analysis of the sticking constraint satisfaction in the presence of a bounded CoM and any physically realizable inertia matrix, based on moment relaxations [20]. This is the first time that moment relaxations have been used to characterize the set of physically realizable

inertial parameters and the first time that this set of parameters has been used to analyze the worst-case constraint satisfaction of a robotic trajectory. Finally, simulations and hardware experiments demonstrating that our proposed robust constraints successfully transport the object—despite using tall objects with high inertial parameter uncertainty—while baseline approaches drop the object.

6.2 Future Work

One open question in mobile manipulation research, and robotics more generally, is which robot morphologies—that is, hardware configurations—are best for a given set of tasks. An attractive choice of robot morphology is the humanoid, since a sufficiently capable humanoid robot could conceivably handle any task that a real human can. On the other hand, humanoids have legs and dexterous hands, but other morphologies with wheels or simpler end effectors may be cheaper yet still highly effective for many tasks. As we have shown in this thesis, wheeled platforms with static grippers in the form of a tray or a push point can certainly handle some tasks. Indeed, we may think of different end effectors and even wheels as *tools* that an intelligent robot may choose to make use of, much as humans do. Our task as roboticists is to investigate which tools are effective for which tasks and to enable robots to use them (see e.g. [104] for some recent work in this direction).

Another open question is the role of machine learning for generating robot motion policies. Machine learning has a clear place in many robotics perception pipelines given its widespread success in visual object detection [105], but learned motion policies are not yet as mature. Common approaches for policy learning are reinforcement learning (RL) and imitation learning (IL). RL tries to maximize the cumulative reward obtained by interacting with the environment over time, which requires a large amount of training data. While this data can be collected in simulation to reduce costs compared to real-world data collection, the mismatch between simulation and reality may limit the subsequent real-world performance [106]. In contrast, IL tries to learn a policy that *imitates* provided expert demonstrations, which may be produced by a human or a different algorithm. Recent IL approaches use generative models to produce a sequence of robot actions to accomplish a desired task; for example, using diffusion models [107] or vision-language-action models [108]. While these methods have appealing generalization capabilities, we still need to understand the fundamental mechanics of the tasks being solved in order to design reward functions, build expert policies for automatic data collection, and to interpret the learned policies.

We discuss some specific ideas for extensions to the work presented in the thesis below, building on the themes of morphology and machine learning, as well as some other directions.

6.2.1 Planar Pushing With Force Feedback

In Chapter 3, we demonstrated that a robot can perform stable single-point pushing along planar paths using only force measurements to sense the pushed object. One direction for future work is to improve the force-based policy using machine learning, which could be used to train a policy that automatically exploits additional features of the environment. For example, a policy could

learn to estimate the slider's parameters over time or to make intelligent use of learned obstacle positions. One option is to use IL to learn to replicate the performance of expert policies that make use of additional sensor information, improving the policy in our force-based setting. Alternatively, RL can be used to train a policy in simulation with dynamics randomization before being deployed on real hardware.

Another direction for future work is to combine our forced-based pushing approach with other sensor modalities, such as vision. One possibility is to combine the all of the different sensor inputs in some way, possibly using them to estimate inertial, frictional, and geometric properties of the object. Another approach is to automatically switch between a vision-based policy and a force-based one depending on the conditions. For example, if the uncertainty in the visual pose measurement becomes too high due to occlusions or poor lighting conditions, the controller can switch to our proposed force-based controller until visual conditions improve.

6.2.2 Nonprehensile Object Transportation

In Chapters 4 and 5, we demonstrated our solution for the nonprehensile object transportation task known as the waiter's problem. Robustness to parameter uncertainty in general is particularly important when solving the waiter's problem with a mobile manipulator, because movement of the mobile base causes vibration and noise at the EE. In Chapter 4, we introduced robustness by planning with the minimum statically feasible friction coefficients; that is, the solver tries to only rely on small friction forces, resulting in plans with a higher tolerance for frictional uncertainty. In Chapter 5, we additionally consider uncertainty in the inertial parameters. Both approaches to robustness rely on tightening constraints to produce robust motions.

Alternatively, we could change the robot's morphology by changing the design of the EE itself. In [75], the EE *simulates* the motion of a pendulum, which moves to minimize the required friction forces on the transported objects. Instead, an actual *physical* tray could be suspended from the EE, the dynamics of which naturally reduce the required friction forces. On the one hand, the contact dynamics of the objects themselves could be neglected, enabling us to make fewer assumptions about the objects' shape, friction, and inertial parameters. However, the underactuated dynamics of the three-dimensional pendulum would need to be included in the robot model, making the trajectory optimization problem more challenging to solve.

Another alternative is to include feedback of the object poses themselves into the controller. In our work, we enforce constraints such that the objects do not move with respect to the tray, and assume this condition holds true. Instead, one could use a vision system to produce estimates of the object poses online, which could then be fed back into the controller. The transported objects can be modelled as inverted pendulums. With online feedback, the controller can potentially recover once one of the objects starts to tip over. In addition, online feedback may also facilitate online estimation of the frictional and inertial parameters of the objects, improving motion speed and efficiency over time.

6.3 Closing Remarks

Each component of this thesis has been at times challenging, frustrating, enlightening, and rewarding. This work has a strong focus on real hardware experiments and classical algorithms based on optimization and Newtonian physics. For who is a roboticist without robots? Who is an engineer without physics?

Our overall goal, as roboticists, should be to make the world better for people—*all* people, not just a few. Robotics has the potential to transfer the burden of a vast amount of labour from humans to machines, making more time for socialization, recreation, athletics, and creative pursuits. We must not lose sight of this vision, nor forget to stop working ourselves once in a while.

Appendix A

Additional Results on Physically Realizable Inertial Parameters

In Chapter 5 we described necessary conditions for a set of rigid body inertial parameters to be physically realizable on a given shape $\mathcal{K} \subset \mathbb{R}^3$ based on moment relaxations, which consist of LMIs and can therefore be used as constraints in an SDP. However, there exist specialized physical realizability conditions for particular shapes that are tighter or faster than the general SDP conditions.

A.1 Existing Ellipsoid Condition

When $\mathcal{K}$ is an ellipsoid, there exists a simple necessary *and sufficient* condition for physical realizability. Let $\mathcal{K} = \{\boldsymbol{r} \in \mathbb{R}^3 \mid (\boldsymbol{r} - \boldsymbol{e})^T \boldsymbol{A} (\boldsymbol{r} - \boldsymbol{e}) \leq 1\} \subset \mathbb{R}^3$ be an ellipsoid, where $\boldsymbol{A} \in \mathbb{S}^3_{++}$ defines the ellipsoid's shape and $\boldsymbol{e} \in \mathbb{R}^3$ is its center. We can equivalently express the ellipsoid as $\mathcal{K} = \{\boldsymbol{r} \in \mathbb{R}^3 \mid \tilde{\boldsymbol{r}}^T \boldsymbol{Q} \tilde{\boldsymbol{r}} \geq 0\}$, where

$$\boldsymbol{Q} = \begin{bmatrix} -\boldsymbol{A} & \boldsymbol{A}\boldsymbol{e} \\ \boldsymbol{e}^T \boldsymbol{A} & 1 - \boldsymbol{e}^T \boldsymbol{A} \boldsymbol{e} \end{bmatrix} \in \mathbb{S}^4. \tag{A.1}$$

Theorem 4 of [23], building on [109], tells us that the pseudo-inertia matrix $\boldsymbol{\Pi} \in \mathbb{S}^4_{++}$ is physically realizable on $\mathcal{K}$ if and only if $\operatorname{tr}(\boldsymbol{Q}\boldsymbol{\Pi}) \geq 0$, which is a simple affine inequality. An equivalent result was derived in [110].

A.2 Novel Box Conditions

It turns out that there also exist specialized physical realizability conditions for boxes (i.e., rectangular prisms), which to the best of our knowledge have not been presented before in the literature. Let $\mathcal{K} = \{\boldsymbol{r} \in \mathbb{R}^3 \mid -\boldsymbol{\varepsilon} \leq \boldsymbol{r} \leq \boldsymbol{\varepsilon}\} \subset \mathbb{R}^3$ be an axis-aligned box centered at the origin, where $\boldsymbol{\varepsilon} = [\varepsilon_x, \varepsilon_y, \varepsilon_z]^T$ is the vector of *half extents*. Define $\boldsymbol{\pi}(\boldsymbol{\Pi}) \triangleq [m, mc_x, mc_y, mc_z, S_{xy}, S_{xz}, S_{yz}]^T$ as a

subset of the inertial parameters represented by the pseudo-inertia matrix

$$\boldsymbol{\Pi} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} & mc_x \\ S_{xy} & S_{yy} & S_{yz} & mc_y \\ S_{xz} & S_{yz} & S_{zz} & mc_z \\ mc_x & mc_y & mc_z & m \end{bmatrix} \in \mathbb{S}^4_{++}.$$

Then a necessary condition for $\boldsymbol{\Pi}$ to be physically realizable on $\mathcal{K}$ is that there exists $\boldsymbol{\mu} \in \mathbb{R}^8_+$ such that

$$\begin{bmatrix} S_{xx} \\ S_{yy} \\ S_{zz} \end{bmatrix} \leq m \begin{bmatrix} \varepsilon_x^2 \\ \varepsilon_y^2 \\ \varepsilon_z^2 \end{bmatrix}, \tag{A.2}$$

$$\boldsymbol{\pi}(\boldsymbol{\Pi}) = \sum_{i=1}^{8} \mu_i \boldsymbol{\pi}(\tilde{\boldsymbol{v}}_i \tilde{\boldsymbol{v}}_i^T), \tag{A.3}$$

where $\boldsymbol{v}_i \in \mathbb{R}^3$, $i = 1, \ldots, 8$, are the vertices of $\mathcal{K}$. These conditions essentially say that we must be able to exactly realize all of the elements of $\boldsymbol{\Pi}$ except for the diagonal of $\boldsymbol{S}$ using point masses $\boldsymbol{\mu}$ located at the vertices, while the diagonal of $\boldsymbol{S}$ must be bounded. Note that the requirement that $\mathcal{K}$ is axis-aligned and centered at the origin is not limiting, because if $\mathcal{K}$ is not, we can simply transform both it and $\boldsymbol{\Pi}$ to this pose using the homogeneous transformation matrix.

To prove this result, we need to show that any

$$\boldsymbol{\Pi} = \int_{\mathcal{K}} \rho(\boldsymbol{r}) \tilde{\boldsymbol{r}} \tilde{\boldsymbol{r}}^T d\boldsymbol{r} \tag{A.4}$$

satisfies (A.2) and (A.3), where $\rho : \mathbb{R}^3 \to \mathbb{R}_+$ is a mass density. The condition (A.2) follows immediately from the fact that the mass cannot be located farther than the half extent along each axis. The condition (A.3) claims that we can always exactly realize $\boldsymbol{\pi}(\boldsymbol{\Pi})$ just by placing mass at the vertices of $\mathcal{K}$. Let $\boldsymbol{s}_i = [s_{i_x}, s_{i_y}, s_{i_z}] = [(-1)^{\lfloor i/4 \rfloor}, (-1)^{\lfloor i/2 \rfloor}, (-1)^i]$, where $\lfloor \cdot \rfloor$ rounds down to the nearest integer. We can then write the vertices of $\mathcal{K}$ as $\boldsymbol{v}_i = [s_{i_x}\varepsilon_x, s_{i_y}\varepsilon_y, s_{i_z}\varepsilon_z]^T$, $i = 1, \ldots, 8$. If we let $\varphi_i(\boldsymbol{r}) = (\varepsilon_x + s_{i_x} r_x)(\varepsilon_y + s_{i_y} r_y)(\varepsilon_z + s_{i_z} r_z)$ for each $i = 1, \ldots, 8$, then the system of point masses located at the vertices with masses

$$\mu_i = (1/V) \int_{\mathcal{K}} \rho(\boldsymbol{r}) \varphi_i(\boldsymbol{r}) d\boldsymbol{r}, \quad i = 1, \ldots, 8,$$

realizes the parameters $\boldsymbol{\pi}(\boldsymbol{\Pi})$, where $V = 8\varepsilon_x\varepsilon_y\varepsilon_z$ is the volume of $\mathcal{K}$. To see this, first observe that $\varphi_i(\boldsymbol{r}) \geq 0$ for all $\boldsymbol{r} \in \mathcal{K}$ since $-\boldsymbol{\varepsilon} \leq \boldsymbol{r} \leq \boldsymbol{\varepsilon}$ for all $\boldsymbol{r} \in \mathcal{K}$, and it follows that $\mu_i \geq 0$ for all $i = 1, \ldots, 8$. Next, define

$$\boldsymbol{\nu}(\boldsymbol{r}) \triangleq \boldsymbol{\pi}(\tilde{\boldsymbol{r}} \tilde{\boldsymbol{r}}^T)^T, \qquad \tilde{\boldsymbol{\nu}}(\boldsymbol{r}) \triangleq \begin{bmatrix} \boldsymbol{\nu}(\boldsymbol{r})^T & r_x r_y r_z \end{bmatrix}^T, \qquad \boldsymbol{D}(\boldsymbol{r}) \triangleq \operatorname{diag}(\boldsymbol{\nu}(\boldsymbol{r})).$$

From (A.4), we know that

$$\boldsymbol{\pi}(\boldsymbol{\Pi}) = \int_{\mathcal{K}} \rho(\boldsymbol{r})\boldsymbol{\nu}(\boldsymbol{r})\, d\boldsymbol{r} \tag{A.5}$$

and we want to show that

$$\boldsymbol{\pi}(\boldsymbol{\Pi}) = \sum_{i=1}^{8} \mu_i \boldsymbol{\nu}(\boldsymbol{v}_i) = \sum_{i=1}^{8} \mu_i \boldsymbol{D}(\boldsymbol{\varepsilon})\boldsymbol{\nu}(\boldsymbol{s}_i). \tag{A.6}$$

We can write $\varphi_i(\boldsymbol{p}) = \varepsilon_x\varepsilon_y\varepsilon_z \tilde{\boldsymbol{\nu}}(\boldsymbol{s}_i)^T \tilde{\boldsymbol{\nu}}(\hat{\boldsymbol{r}})$, where $\hat{\boldsymbol{r}} = [r_x/\varepsilon_x, r_y/\varepsilon_y, r_z/\varepsilon_z]^T$, which means

$$\mu_i = (1/8) \int_{\mathcal{K}} \rho(\boldsymbol{r}) \tilde{\boldsymbol{\nu}}(\boldsymbol{s}_i)^T \tilde{\boldsymbol{\nu}}(\hat{\boldsymbol{r}})\, d\boldsymbol{r}. \tag{A.7}$$

Substituting (A.7) into the right-hand side (A.6), we get

$$\begin{aligned}
\sum_{i=1}^{8} \mu_i \boldsymbol{D}(\boldsymbol{\varepsilon})\boldsymbol{\nu}(\boldsymbol{s}_i) &= (1/8) \sum_{i=1}^{8} \int_{\mathcal{K}} \rho(\boldsymbol{r}) \tilde{\boldsymbol{\nu}}(\boldsymbol{s}_i)^T \tilde{\boldsymbol{\nu}}(\hat{\boldsymbol{r}}) \boldsymbol{D}(\boldsymbol{\varepsilon})\boldsymbol{\nu}(\boldsymbol{s}_i)\, d\boldsymbol{r} \\
&= \int_{\mathcal{K}} \rho(\boldsymbol{r}) \boldsymbol{D}(\boldsymbol{\varepsilon}) \underbrace{\left((1/8) \sum_{i=1}^{8} \boldsymbol{\nu}(\boldsymbol{s}_i) \tilde{\boldsymbol{\nu}}(\boldsymbol{s}_i)^T \right)}_{\begin{bmatrix} \mathbf{1}_7 & \mathbf{0} \end{bmatrix}} \tilde{\boldsymbol{\nu}}(\hat{\boldsymbol{r}})\, d\boldsymbol{r} \\
&= \int_{\mathcal{K}} \rho(\boldsymbol{r}) \boldsymbol{D}(\boldsymbol{\varepsilon}) \boldsymbol{\nu}(\hat{\boldsymbol{r}})\, d\boldsymbol{r} \\
&= \int_{\mathcal{K}} \rho(\boldsymbol{r}) \boldsymbol{\nu}(\boldsymbol{r})\, d\boldsymbol{r}
\end{aligned}$$

which is equal to (A.5), as desired.

We have only shown that the conditions (A.2) and (A.3) are necessary for physically realizability; it is still an open question if they are also sufficient. However, in practice they appear to be at least as tight as the moment relaxation conditions from Chapter 5, while being much faster to enforce as constraints in SDPs, as we will see next.

A.3 Faster Sticking Constraint Verification

In Chapter 5, we used moment relaxations to verify the worst-case sticking constraint violation given uncertain inertial parameters by solving the SDP (5.10) for each $i = 1, \ldots, n_h$ at each timestep along a discretized trajectory. Since the shape $\mathcal{K}$ of the transported object is actually a box, we can

Table A.1: Comparison of physical realizability conditions for verifying trajectories from Section 5.8.

SDP	Average solve time [ms]	Maximum value		
		Center	Top	Robust
(5.10)	79	11.96	10.50	−0.56
(A.8)	13	11.96	10.50	−0.56
(A.9)	12	21.31	12.36	0.11

instead use our specialized box realizability conditions from the previous section, yielding the SDP

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\mu}}{\text{maximize}} \quad & \boldsymbol{h}_i^T \boldsymbol{Y}(\boldsymbol{\xi},\boldsymbol{\eta})\boldsymbol{\theta} \\ \text{subject to} \quad & \boldsymbol{\Pi}(\boldsymbol{\theta}) \succcurlyeq \mathbf{0}, \\ & \begin{bmatrix} S_{xx}(\boldsymbol{\theta}) \\ S_{yy}(\boldsymbol{\theta}) \\ S_{zz}(\boldsymbol{\theta}) \end{bmatrix} \leq m(\boldsymbol{\theta}) \begin{bmatrix} \varepsilon_x^2 \\ \varepsilon_y^2 \\ \varepsilon_z^2 \end{bmatrix}, \\ & \boldsymbol{\pi}(\boldsymbol{\theta}) = \textstyle\sum_{j=1}^{8} \mu_j \boldsymbol{\pi}(\tilde{\boldsymbol{v}}_j \tilde{\boldsymbol{v}}_j^T), \\ & \boldsymbol{c}(\boldsymbol{\theta}) \in \mathcal{C}, \end{aligned} \tag{A.8}$$

where we have written all representations of the inertial parameters as functions of the parameter vector $\boldsymbol{\theta}$, and we have relaxed the strict positive definiteness of $\boldsymbol{\Pi}$ to positive semidefiniteness.

Alternatively, suppose $\mathcal{E} = \{\boldsymbol{r} \in \mathbb{R}^3 \mid \tilde{\boldsymbol{r}}^T \boldsymbol{Q} \tilde{\boldsymbol{r}} \geq 0\}$ is the minimum-volume bounding ellipsoid of $\mathcal{K}$, with $\boldsymbol{Q}$ as defined in (A.1). Then another SDP that provides an upper-bound on the worst-case constraint violation is

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{maximize}} \quad & \boldsymbol{h}_i^T \boldsymbol{Y}(\boldsymbol{\xi},\boldsymbol{\eta})\boldsymbol{\theta} \\ \text{subject to} \quad & \boldsymbol{\Pi}(\boldsymbol{\theta}) \succcurlyeq \mathbf{0}, \\ & \operatorname{tr}(\boldsymbol{Q}\boldsymbol{\Pi}(\boldsymbol{\theta})) \geq 0, \\ & \boldsymbol{c}(\boldsymbol{\theta}) \in \mathcal{C}, \end{aligned} \tag{A.9}$$

where we have used the physical realizability conditions for ellipsoids described in Section A.1.

To compare the different SDPs (5.10), (A.8), and (A.9), we solved them at each timestep along the discretized trajectories obtained using the Center, Top, and Robust constraint methods for the simulated box with height $60\,\mathrm{cm}$ from Section 5.8. The results are shown in Table A.1. The maximum constraint violations produced by the original SDP (5.10) using moment relaxations and the proposed SDP (A.8) using our specialized constraints for boxes are the same (and of course match those reported in Table 5.1). The SDP (A.8) is also considerably faster to solve: approximately six times faster than (5.10). The SDP (A.9), which uses constraints corresponding to the bounding ellipsoid, is similarly fast to solve, but the optimal values (i.e., the worst-case sticking constraint violations) are much higher, because the ellipsoid is not a tight bounding shape for the true object's shape. Indeed, using the loose ellipsoid constraints would indicate that the Robust trajectory

may have constraint violations, while the other (tighter) constraints confirm that it does not. The proposed specialized physical realizability conditions for boxes from Section A.2 therefore provide both tight constraints and an efficient compute time compared to existing methods. Aside from constraint verification, they could also be used for inertial parameter identification, similar to [23].

Bibliography

[1] A. Heins and A. P. Schoellig, "Force push: Robust single-point pushing with force feedback," *IEEE Robotics and Automation Letters*, vol. 9, no. 8, pp. 6856–6863, 2024.

[2] A. Heins and A. P. Schoellig, "Keep it upright: Model predictive control for nonprehensile object transportation with obstacle avoidance on a mobile manipulator," *IEEE Robotics and Automation Letters*, vol. 8, no. 12, pp. 7986–7993, 2023.

[3] A. Heins and A. P. Schoellig, "Robust nonprehensile object transportation with uncertain inertial parameters," *IEEE Robotics and Automation Letters*, vol. 10, no. 5, pp. 4492–4499, 2025.

[4] A. Heins, M. Jakob, and A. P. Schoellig, "Mobile manipulation in unknown environments with differential inverse kinematics control," in *Proc. Conf. Robots and Vision*, 2021, pp. 64–71.

[5] M. K. Helwa, A. Heins, and A. P. Schoellig, "Provably robust learning-based approach for high-accuracy tracking control of Lagrangian systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1587–1594, 2019.

[6] L. Takayama, W. Ju, and C. Nass, "Beyond dirty, dangerous and dull: What everyday people think robots should do," in *Proc. ACM/IEEE Int. Conf. Human-Robot Interaction*, 2008, pp. 25–32.

[7] *Oxford English Dictionary*. Oxford University Press, 2023.

[8] A. L. Rosenberger, "Tale of tails: Parallelism and prehensility," *American J. Physical Anthropology*, vol. 60, no. 1, pp. 103–107, 1983.

[9] A. C. Noel and D. L. Hu, "The tongue as a gripper," *J. Experimental Biology*, vol. 221, no. 7, jeb176289, 2018.

[10] S. Chevalier-Skolnikoff and J. Liska, "Tool use by wild and captive elephants," *Animal Behaviour*, vol. 46, no. 2, pp. 209–219, 1993.

[11] K. M. Lynch, "Nonprehensile robotic manipulation: Controllability and planning," Ph.D. Carnegie Mellon University, 1996.

[12] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile dynamic manipulation: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1711–1718, 2018.

[13] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *Int. J. Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[14] N. A. M. Hootsmans, "The motion control of manipulators on mobile vehicles," Thesis, Massachusetts Institute of Technology, 1992.

[15] S. Sivčev, J. Coleman, E. Omerdić, G. Dooly, and D. Toal, "Underwater manipulators: A review," *Ocean Engineering*, vol. 163, pp. 431–450, 2018.

[16] K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, and K. Akachi, "Humanoid robot HRP-3," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2008, pp. 2471–2478.

[17] H. Ferrolho, V. Ivan, W. Merkt, I. Havoutis, and S. Vijayakumar, "Roloma: Robust loco-manipulation for quadruped robots with arms," *Autonomous Robots*, vol. 47, no. 8, pp. 1463–1481, 2023.

[18] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi, "Past, present, and future of aerial robotic manipulators," *IEEE Trans. Robotics*, vol. 38, no. 1, pp. 626–645, 2022.

[19] E. Papadopoulos, F. Aghili, O. Ma, and R. Lampariello, "Robotic manipulation and capture in space: A survey," *Frontiers in Robotics and AI*, vol. 8, 2021.

[20] J. B. Lasserre, *Moments, positive polynomials and their applications*. World Scientific, 2009.

[21] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2007.

[22] T. Yoshikawa, *Foundations of Robotics—Analysis and Control*. The MIT Press, 2003.

[23] P. M. Wensing, S. Kim, and J.-J. E. Slotine, "Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 60–67, 2018.

[24] C. G. Atkeson, C. H. An, and J. M. Hollerbach, "Estimation of inertial parameters of manipulator loads and links," *Int. J. Robotics Research*, vol. 5, no. 3, pp. 101–119, 1986.

[25] M. Vukobratović and B. Borovac, "Zero-moment point—thirty five years of its life," *Int. J. Humanoid Robotics*, vol. 01, no. 01, pp. 157–173, 2004.

[26] S. Caron, Q.-C. Pham, and Y. Nakamura, "ZMP support areas for multicontact mobility under frictional constraints," *IEEE Trans. on Robotics*, vol. 33, no. 1, pp. 67–80, 2017.

[27] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[28] C. Zhou, M. Lei, L. Zhao, Z. Wang, and Y. Zheng, "TOPP-MPC-based dual-arm dynamic collaborative manipulation for multi-object nonprehensile transportation," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2022, pp. 999–1005.

[29] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction Part 1. Limit surface and moment function," *Wear*, vol. 143, no. 2, pp. 307–330, 1991.

[30] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1992, pp. 416–421.

[31] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2016, pp. 372–377.

[32] S. Caron, Q. C. Pham, and Y. Nakamura, "Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics," in *Proc. Robotics: Science and Systems*, 2015, ISBN: 978-0-9923747-1-6.

[33] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," in *Proc. Workshop Algorithmic Foundations of Robotics*, 2020, pp. 800–815.

[34] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-prehensile planar manipulation via trajectory optimization with complementarity constraints," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2022, pp. 970–976.

[35] S. P. Boyd and B. Wegbreit, "Fast computation of optimal contact forces," *IEEE Trans. on Robotics*, vol. 23, no. 6, pp. 1117–1132, 2007.

[36] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2009.

[37] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.

[38] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "PROX-QP: yet another quadratic programming solver for robotics and beyond," in *Proc. Robotics: Science and Systems*, 2022, ISBN: 978-0-9923747-8-5.

[39] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.

[40] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. 2017, vol. 2.

[41] P. E. Gill and E. Wong, "Sequential quadratic programming methods," in *Proc. Mixed Integer Nonlinear Programming*, 2012, pp. 147–224.

[42] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM J. Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[43] MOSEK ApS, *MOSEK documentation and API reference (version 10.2)*, 2024. [Online]. Available: https://www.mosek.com/documentation/.

[44] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *J. Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[45] J. Stüber, C. Zito, and R. Stolkin, "Let's push things forward: A survey on robot pushing," *Frontiers in Robotics and AI*, vol. 7, 2020.

[46] R. Emery and T. Balch, "Behavior-based control of a non-holonomic robot in pushing tasks," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2001, pp. 2381–2388.

[47] T. Igarashi, Y. Kamiyama, and M. Inami, "A dipole field for object delivery by pushing on a flat surface," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2010, pp. 5114–5119.

[48] Y. Okawa and K. Yokoyama, "Control of a mobile robot for the push-a-box operation," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1992, pp. 761–766.

[49] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *Int. J. Robotics Research*, vol. 15, no. 6, pp. 533–556, 1996.

[50] S. Akella and M. T. Mason, "Posing polygonal objects in the plane by pushing," *Int. J. Robotics Research*, vol. 17, no. 1, pp. 70–88, 1998.

[51] S. Rusaw, K. Gupta, and S. Payandeh, "Part orienting with a force/torque sensor," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1999, pp. 2545–2550.

[52] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason, "Parts feeding on a conveyor with a one joint robot," *Algorithmica*, vol. 26, no. 3, pp. 313–344, 2000.

[53] F. Ruiz-Ugalde, G. Cheng, and M. Beetz, "Fast adaptation for effect-aware pushing," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2011, pp. 614–621.

[54] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars, "Path planning for pushing a disk using compliance," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2005, pp. 714–720.

[55] M. Bauza and A. Rodriguez, "A probabilistic data-driven model for planar pushing," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2017, pp. 3008–3015.

[56] J. Li, W. S. Lee, and D. Hsu, "Push-net: Deep planar pushing for objects with unknown physical properties," in *Proc. Robotics: Science and Systems*, 2018.

[57] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects from sensory data," *Int. J. Robotics Research*, vol. 41, no. 8, pp. 778–797, 2022.

[58] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2018, pp. 3803–3810.

[59] J. D. A. Ferrandis, J. Moura, and S. Vijayakumar, "Nonprehensile planar manipulation through reinforcement learning with multimodal categorical exploration," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2023, pp. 5606–5613.

[60] M. Bauza, F. R. Hogan, and A. Rodriguez, "A data-efficient approach to precise and controlled pushing," in *Proc. Conf. Robot Learning*, 2018, pp. 336–345.

[61] W. C. Agboh and M. R. Dogar, "Pushing fast and slow: Task-adaptive planning for non-prehensile manipulation under uncertainty," in *Proc. Workshop Algorithmic Foundations of Robotics*, 2020, pp. 160–176.

[62] F. Bertoncelli, F. Ruggiero, and L. Sabattini, "Linear time-varying MPC for nonprehensile object manipulation with a nonholonomic mobile robot," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2020, pp. 11 032–11 038.

[63] Y. Tang, H. Zhu, S. Potters, M. Wisse, and W. Pan, "Unwieldy object delivery with nonholonomic mobile base: A stable pushing approach," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7727–7734, 2023.

[64] A. Rigo, Y. Chen, S. K. Gupta, and Q. Nguyen, "Contact optimization for non-prehensile loco-manipulation via hierarchical model predictive control," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2023, pp. 9945–9951.

[65] M. Sombolestan and Q. Nguyen, "Hierarchical adaptive loco-manipulation control for quadruped robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2023, pp. 12 156–12 162.

[66] F. G. Flores and A. Kecskeméthy, "Time-optimal path planning for the general waiter motion problem," in *Advances in Mechanisms, Robotics and Design Education and Research*, 2013, pp. 189–203.

[67] B. A. Maxwell *et al.*, "Alfred: The robot waiter who remembers you," in *Proc. AAAI Workshop on Robotics*, 1999, pp. 1–12.

[68] A. Cheong, M. Lau, E. Foo, J. Hedley, and J. W. Bo, "Development of a robotic waiter system," *IFAC-PapersOnLine*, vol. 49, no. 21, pp. 681–686, 2016.

[69] A. Y. S. Wan, Y. D. Soong, E. Foo, W. L. E. Wong, and W. S. M. Lau, "Waiter robots conveying drinks," *Technologies*, vol. 8, no. 3, p. 44, 2020.

[70] B. Sprenger, L. Kucera, and S. Mourad, "Balancing of an inverted pendulum with a SCARA robot," *IEEE/ASME Trans. Mechatronics*, vol. 3, no. 2, pp. 91–97, 1998.

[71] T. A. Permadi, J. Halomoan, and S. Hadiyoso, "Balancing system of tray on waiter robot using complementary filter and fuzzy logic," in *Proc. Int. Conf. Industrial Automation, Information and Communications Technology*, 2014, pp. 15–21.

[72] J. M. Garcia-Haro, S. Martinez, and C. Balaguer, "Balance computation of objects transported on a tray by a humanoid robot based on 3D dynamic slopes," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2018, pp. 1–6.

[73] A. Dang and I. Ebert-Uphoff, "Active acceleration compensation for transport vehicles carrying delicate objects," *IEEE Trans. on Robotics*, vol. 20, no. 5, pp. 830–839, 2004.

[74] L. Moriello, L. Biagiotti, C. Melchiorri, and A. Paoli, "Manipulating liquids with robots: A sloshing-free solution," *Control Engineering Practice*, vol. 78, pp. 129–141, 2018.

[75] R. I. C. Muchacho, R. Laha, L. F. C. Figueredo, and S. Haddadin, "A solution to slosh-free robot trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2022, pp. 223–230.

[76] J. Ichnowski, Y. Avigal, Y. Liu, and K. Goldberg, "GOMP-FIT: grasp-optimized motion planning for fast inertial transport," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2022, pp. 5255–5261.

[77] G. Csorvási, Á. Nagy, and I. Vajk, "Near time-optimal path tracking method for waiter motion problem," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4929–4934, 2017.

[78] J. Luo and K. Hauser, "Robust trajectory optimization under frictional contact with iterative learning," *Autonomous Robots*, vol. 41, no. 6, pp. 1447–1461, 2017.

[79] Q.-C. Pham, S. Caron, P. Lertkultanon, and Y. Nakamura, "Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots," *Int. J. Robotics Research*, vol. 36, no. 1, pp. 44–67, 2017.

[80] M. Selvaggio, J. Cacace, C. Pacchierotti, F. Ruggiero, and P. R. Giordano, "A shared-control teleoperation architecture for nonprehensile object transportation," *IEEE Trans. on Robotics*, vol. 38, no. 1, pp. 569–583, 2022.

[81] R. Subburaman, M. Selvaggio, and F. Ruggiero, "A non-prehensile object transportation framework with adaptive tilting based on quadratic programming," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3581–3588, 2023.

[82] V. Morlando, M. Selvaggio, and F. Ruggiero, "Nonprehensile object transportation with a legged manipulator," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2022, pp. 6628–6634.

[83] J. M. Garcia-Haro, "Object oriented control system in humanoid robots for transport tasks," Ph.D. Universidad Carlos III de Madrid, 2019.

[84] M. Selvaggio, A. Garg, F. Ruggiero, G. Oriolo, and B. Siciliano, "Non-prehensile object transportation via model predictive non-sliding manipulation control," *IEEE Trans. Control Systems Technology*, vol. 31, no. 5, pp. 2231–2244, 2023.

[85] D. Kraft, "A software package for sequential quadratic programming," DLR German Aerospace Center – Institute for Flight Mechanics, Tech. Rep. DFVLR-FB 88-28, 1988.

[86] P. Virtanen, R. Gommers, M. T. E., *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[87] *OCS2: An open source library for optimal control of switched systems.* [Online]. Available: https://github.com/leggedrobotics/ocs2.

[88] B. M. Bell, *CppAD: A package for C++ algorithmic differentiation (20190200)*, 2019. [Online]. Available: http://www.coin-or.org/CppAD.

[89] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *Proc. European Control Conf.*, 2014, pp. 128–133.

[90] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *J. Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998.

[91] T. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[92] K. Dong, K. Pereida, F. Shkurti, and A. P. Schoellig, "Catch the ball: Accurate high-speed motions for mobile manipulators via inverse dynamics learning," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2020, pp. 6718–6725.

[93] K. Zhang, Z. Cao, J. Liu, Z. Fang, and M. Tan, "Real-time visual measurement with opponent hitting behavior for table tennis robot," *IEEE Trans. Instrumentation and Measurement*, vol. 67, no. 4, pp. 811–820, 2018.

[94] S. Traversaro, S. Brossette, A. Escande, and F. Nori, "Identification of fully physical consistent inertial parameters using optimization on manifolds," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2016, pp. 5446–5451.

[95] H. Gattringer, A. Müller, S. Weitzhofer, and M. Schörgenhumer, "Point to point time optimal handling of unmounted rigid objects and liquid-filled containers," *Mechanism and Machine Theory*, vol. 184, p. 105 286, 2023.

[96] Z. Brei, J. Michaux, B. Zhang, P. Holmes, and R. Vasudevan, "Serving time: Real-time, safe motion planning and control for manipulation of unsecured objects," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2383–2390, 2024.

[97] S. Caron and A. Kheddar, "Multi-contact walking pattern generation based on model preview control of 3D COM accelerations," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2016, pp. 550–557.

[98] N. Giftsun, A. D. Prete, and F. Lamiraux, "Robustness to inertial parameter errors for legged robots balancing on level ground," in *Proc. Int. Conf. Informatics in Control, Automation and Robotics*, 2017.

[99] X. Jiang, W. Chi, Y. Zheng, *et al.*, "Locomotion generation for quadruped robots on challenging terrains via quadratic programming," *Autonomous Robots*, vol. 47, no. 1, pp. 51–76, 2023.

[100] H. Yang and L. Carlone, "Certifiably optimal outlier-robust geometric perception: Semidefinite relaxations and scalable global optimization," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 2816–2834, 2023.

[101] S. Kang, X. Xu, J. Sarva, L. Liang, and H. Yang, "Fast and certifiable trajectory optimization," in *Proc. Workshop Algorithmic Foundations of Robotics*, 2024.

[102] D. J. Balkcom and J. C. Trinkle, "Computing wrench cones for planar rigid body contact tasks," *Int. J. Robotics Research*, vol. 21, no. 12, pp. 1053–1066, 2002.

[103] K. Fukuda and A. Prodon, "Double description method revisited," in *Proc. Combinatorics and Computer Science*, 1996, pp. 91–111.

[104] L. Wang, J. Zhao, Y. Du, E. Adelson, and R. Tedrake, "PoCo: Policy composition from and for heterogeneous robot learning," in *Proc. Robotics: Science and Systems*, 2024.

[105] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proc. IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[106] S. Höfer, K. Bekris, A. Handa, *et al.*, "Sim2real in robotics and automation: Applications and challenges," *IEEE Trans. Automation Science and Engineering*, vol. 18, no. 2, pp. 398–400, 2021.

[107] C. Chi, Z. Xu, S. Feng, *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," *Int. J. Robotics Research*, 2024.

[108] K. Black, N. Brown, D. Driess, *et al.*, π_0*: a vision-language-action flow model for general robot control*, arXiv: 2410.24164 [cs.LG], 2024.

[109] L. Fialkow and J. Nie, "Positivity of Riesz functionals and solutions of quadratic and quartic moment problems," *J. Functional Analysis*, vol. 258, no. 1, pp. 328–356, 2010.

[110] G. M. Rozenblat, "On the choice of physically realizable parameters when studying the dynamics of spherical and ellipsoidal rigid bodies," *Mechanics of Solids*, vol. 51, no. 4, pp. 415–423, 2016.