Force Push: Robust Single-Point Pushing with Force Feedback

Adam Heins and Angela P. Schoellig

***Abstract*—We present a controller for quasistatic robotic planar pushing with single-point contact using *only* force feedback to sense the pushed object. We consider an omnidirectional mobile robot pushing an object (the "slider") along a given path, where the robot is equipped with a force-torque sensor to measure the force at the contact point with the slider. The geometric, inertial, and frictional parameters of the slider are not known to the controller, nor are measurements of the slider's pose. We assume that the robot can be localized so that the global position of the contact point is always known and that the approximate initial position of the slider is provided. Simulations and real-world experiments show that our controller yields pushes that are robust to a wide range of slider parameters and state perturbations along both straight and curved paths. Furthermore, we use an admittance controller to adjust the pushing velocity based on the measured force when the slider contacts obstacles like walls.**

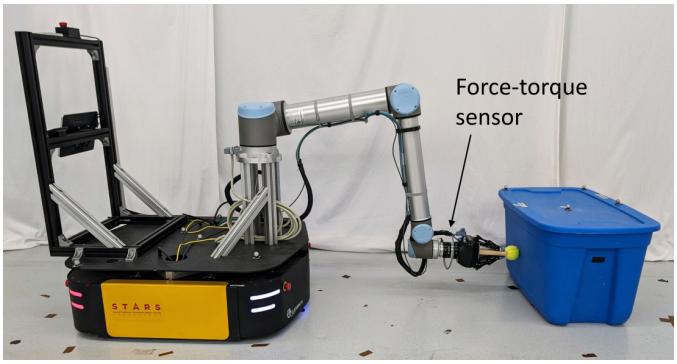

Figure 1: Our robot pushing a blue box across the floor using single-point contact. The contact force is measured by a force-torque sensor in the robot's wrist, but no other measurements of the object are provided. A video of our approach is available at **`http://tiny.cc/force-push`**.

I. 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 [1]. 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., [2]–[5]. 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.

This article was published in IEEE Robotics and Automation Letters. Digital Object Identifier (DOI): 10.1109/LRA.2024.3414180

This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Canadian Institute for Advanced Research.

The authors are with the Learning Systems and Robotics Lab (www.learnsyslab.org) at the Technical University of Munich, Germany, and the University of Toronto Institute for Aerospace Studies, Canada. They are also affiliated with the University of Toronto Robotics Institute, the Munich Institute of Robotics and Machine Intelligence (MIRMI), and the Vector Institute for Artificial Intelligence. E-mail: adam.heins@robotics.utias.utoronto.ca, angela.schoellig@tum.de

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 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 Fig. 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 [6], but we have substantially changed and augmented this approach in the current work. 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.

II. Related Work

Research on robotic pushing began with Mason [7], and a recent survey can be found in [1]. Early approaches were typically either open-loop planning methods that rely on line contact for stability [8], [9] or feedback-based approaches using vision [2], [3] or tactile sensing [4], [5]. 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, [4] only focuses on stable translational pushes (i.e., where the slider moves in a constant direction) without path-tracking, and [5] 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 [10], which was shown to require less pushes than the best sensorless alternative [11]. Another use of an FT sensor was in [12], 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 [13], 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 [13] exploits the geometry of the obstacles to provide additional kinematic constraints on the slider's motion. Furthermore, in [13] 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 [14], the pushing dynamics are learned using Gaussian process (GP) regression. In [15] the authors propose Push-Net, an LSTM-based neural network architecture for pushing objects using an image mask representing the slider's pose. In [16], the analytical model of quasistatic pushing from [4] 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 [17] to train a pushing policy, which is then transferred to a real robot arm with no fine-tuning. In [18], 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 line of work uses model predictive control (MPC) for fast online replanning. The GP-based model in [14] is used for MPC in [19]. In [20], 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 [21] is more computationally expensive than [20] but can handle complete loss of contact between objects. In [22], 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 [23] and [24], 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 [8] and [9]. 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 [25] and up slopes [26] while regulating the required pushing force. In [25], 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 [26], 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 [26] 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.

III. Problem Statement

Our goal is to push an object along a given continuous planar path $\boldsymbol{p}_d(s)\colon \mathbb{R}_{\geq 0} \to \mathbb{R}^2$, 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 $\boldsymbol{v}_{\mathrm{cmd}} \in \mathbb{R}^2$ 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

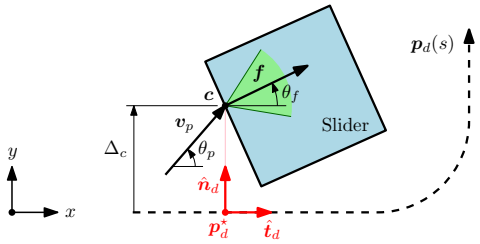

Figure 2: Example of our pushing controller with a square slider. The goal is to push the slider along the path $\boldsymbol{p}_d(s)$ by pushing with velocity $\boldsymbol{v}_p$ at the contact point $\boldsymbol{c}$. The resulting contact force $\boldsymbol{f}$ lies inside the friction cone (green). 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.

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.

IV. Task-Space Pushing Controller

When the properties of the slider are known, we can predict its motion using the equations of motion given in [4]. 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 [7]—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 Fig. 2. The contact force vector $\boldsymbol{f}$ will also start rotating counterclockwise. Then we need to push in a direction *even further* in the counterclockwise direction to eventually rotate the object back clockwise and toward the straight-line direction. 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 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 Fig. 3; the different components are described in the remainder of this and the following section.

A. 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

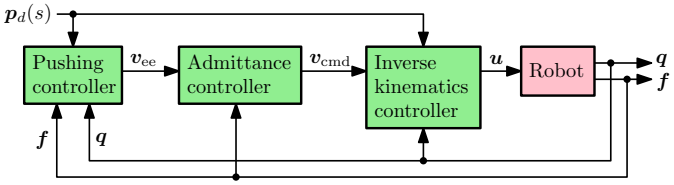

Figure 3: Block diagram of the system. The components of our controller (in green) use measurements of the robot's pose $\boldsymbol{q}$ and contact force $\boldsymbol{f}$ to produce joint velocity inputs $\boldsymbol{u}$ that push the slider along a desired path $\boldsymbol{p}_d(s)$.

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) and let us attach a Frenet-Serret frame at that point, with directions $\hat{\boldsymbol{t}}_d$ pointing tangent to (along) the path and $\hat{\boldsymbol{n}}_d$ orthogonal to the path (see Fig. 2). We denote the angle from the x-axis to $\hat{\boldsymbol{t}}_d$ as θ_d. Our pushing control law is simply

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

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 Sec. V; 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.

B. Contact Recovery

It is possible that following the pushing angle produced by (1) 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

$$\theta_o = \theta_d - k_c\Delta_c, \tag{2}$$

which just steers the EE toward the desired path. Notice that the sign of $k_c\Delta_c$ is opposite to that in (1); this is because here

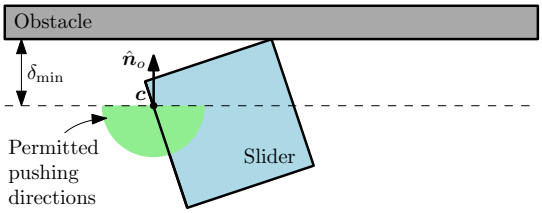

Figure 4: Basic obstacle avoidance of the pusher. When the contact point $\boldsymbol{c}$ is within distance $\delta_{\min}$ of an obstacle, the direction of motion is adjusted to not move any closer to it.

the pusher does not need to move *away* from the path to steer the slider back toward it.

Let's now combine (1) and (2). 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 (2). When contact is made again, such that $\|\boldsymbol{f}\| \geq f_{\min}$, we switch back to (1). Thus the combined EE velocity angle θ_{ee} is given by

$$\theta_{\text{ee}} = \begin{cases} \theta_{\text{ee}}^- + \gamma, & \text{if } \|\boldsymbol{f}\| < f_{\min} \\ \theta_p, & \text{otherwise,} \end{cases} \tag{3}$$

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) to be somewhat more reliable in our experiments, but a more sophisticated contact recovery mechanism is worth investigating in future work.

C. 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 Fig. 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 (1) 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

$$\boldsymbol{v}_a = \begin{cases} k_a(f_{\max} - \|\boldsymbol{f}\|)\hat{\boldsymbol{f}} & \text{if } \|\boldsymbol{f}\| > f_{\max}, \\ \boldsymbol{0} & \text{otherwise,} \end{cases} \tag{4}$$

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.

D. 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.

V. 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 a quadratic program-based inverse kinematics controller

$$\begin{aligned} \boldsymbol{u} = \text{argmin}_{\boldsymbol{\xi}} \quad & (1/2)\|\boldsymbol{\xi}_d - \boldsymbol{\xi}\|^2 \\ \text{subject to} \quad & \boldsymbol{J}_c(\boldsymbol{q})\boldsymbol{\xi} = \boldsymbol{v}_{\text{cmd}} \\ & -\boldsymbol{\xi}_{\max} \leq \boldsymbol{\xi} \leq \boldsymbol{\xi}_{\max}, \end{aligned} \tag{5}$$

where $\boldsymbol{\xi}_d = [0, 0, k_\omega(\theta_d - \theta)]^T$ 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 $\boldsymbol{\xi}_{\max}$ 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 $[\hat{\boldsymbol{n}}_o^T, 0]\boldsymbol{\xi} \leq 0$ to (5), 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 3 degrees of freedom (DOFs), which is sufficient to accomplish our pushing task, (5) has the structure of a standard IK controller and can be augmented with additional DOFs, objectives, and constraints (see e.g. [6]), as long as the required EE velocity for pushing is achieved.

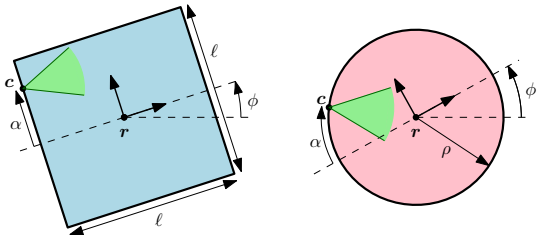

Figure 5: Examples of two sliders, located at position $\boldsymbol{r}$ and orientation ϕ in the global frame. The contact with the pusher is located at point $\boldsymbol{c}$, which is distance α along the slider's edge from a reference point. The contact force must lie in the friction cone at the contact point (shown in green).

Table I: Controller parameters for simulation and hardware experiments.

Parameter	Simulation	Hardware	Unit
v	0.1	0.1	m/s
k_f	0.3	0.3	–
k_c	0.1	0.5	rad/m
k_a	0.003	0.003	s/kg
k_ω	–	1	1/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
$\boldsymbol{\xi}_{\max}$	–	$[0.5, 0.5, 0.25]^T$	$[\text{m/s}, \text{m/s}, \text{rad/s}]^T$

VI. Simulation Experiments

We first validate our controller in simulation with Box and Cylinder sliders representing the planar sliders shown in Fig. 5. We use the PyBullet simulator[1]. The Box has x-y side lengths $\ell = 1\,\text{m}$ and height $12\,\text{cm}$; the Cylinder slider has the same height and radius $\rho = 0.5\,\text{m}$. Each has mass $m = 1\,\text{kg}$ with CoM located at the centroid. The pusher is a sphere of radius $5\,\text{cm}$ and the height of the contact point is $6\,\text{cm}$. 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 II. We use the controller parameters listed in the Simulation column of Table I. The simulation timestep is $1\,\text{ms}$ and the control timestep is $10\,\text{ms}$ (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]

The position trajectories for each of the $3^5 = 243$ scenarios per slider are shown in Fig. 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 Fig. 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 (4) before again eventually returning to the desired path.

Individual sample trajectories are shown in Fig. 8 and Fig. 9. In Fig. 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 Fig. 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), ultimately making contact again on a different edge of the slider.

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

[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 II: 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	$\boldsymbol{I}$	$0.5\hat{\boldsymbol{I}}$, $\hat{\boldsymbol{I}}$, $\boldsymbol{I}_{\max}$	$\text{kg}\cdot\text{m}^2$

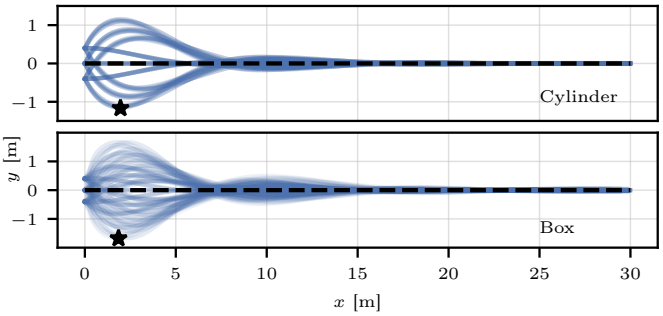

Figure 6: Simulated trajectories for all 243 scenarios given in Table II for the Box and Cylinder sliders shown in Fig. 5. Each trajectory has a duration of 5 min. All trajectories converge to the desired straight-line path using our control law. The point of maximum deviation from the path for any of the trajectories is marked with a star.

VII. Hardware Experiments

We now demonstrate our controller in real-world experiments. The robot used for pushing is a mobile manipulator consisting

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

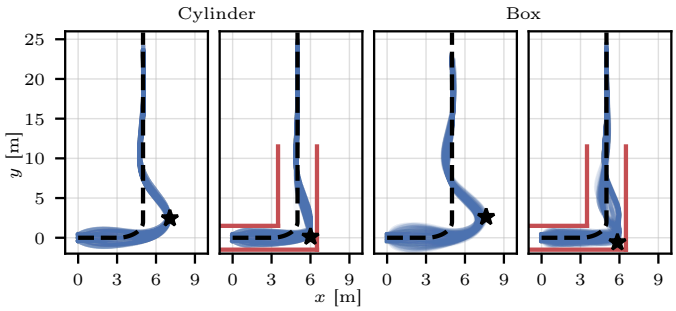

Figure 7: Simulated trajectories for the Box and Cylinder sliders along a curved path, with and without walls (in red) simulating a corridor. All trajectories again converge despite collisions with the wall. The point of maximum deviation from the path for any of the trajectories is marked with a star.

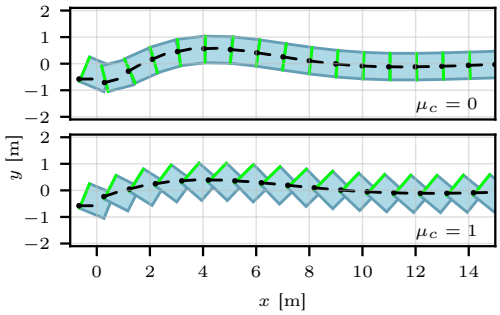

Figure 8: Samples of simulated trajectories of the Box slider with the straight desired path along the x-axis. The slider has initial state $(x_0, y_0, s_0, \phi_0) = (0, -40\,\text{cm}, -40\,\text{cm}, -\pi/8)$ and uniform density inertia. Results are shown for low and high contact friction. Pusher 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.

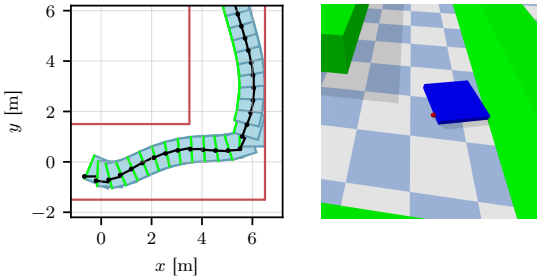

Figure 9: *Left:* A sample simulated trajectory of the Box slider along the curved path with walls. The slider has initial state $(x_0, y_0, s_0, \phi_0) = (0, -40\,\text{cm}, -40\,\text{cm}, -\pi/8)$, a uniform density inertia, and $\mu_c = 0$. Pusher and pushing direction are shown in black; the walls 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 10: *Left:* The "Box" and "Barrel" sliders used for real-world experiments. Each is empty except for 2.25 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 cm, height 56 cm, and total mass 4.5 kg. The Box has width 65 cm, depth 33 cm, and height 43 cm; the total mass of Box1 is 4.8 kg and of Box2 is 7 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).

of a UR10 arm mounted on a Ridgeback omnidirectional base (see Fig. 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 (5)—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 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 Hz, the mobile base accepts commands at 25 Hz, and we run our control loop[4] at 100 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 Fig. 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 s to reach the constant desired pushing speed v. We use ProxQP [27] to solve (5). For obstacle avoidance, we model the base as a circle of radius 55 cm. We use the controller parameters in the Hardware column of Table I, 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 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 Fig. 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 (2) 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 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 m length of the room.

The trajectories do not converge perfectly to the desired path, at least not within the available 6 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

[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.

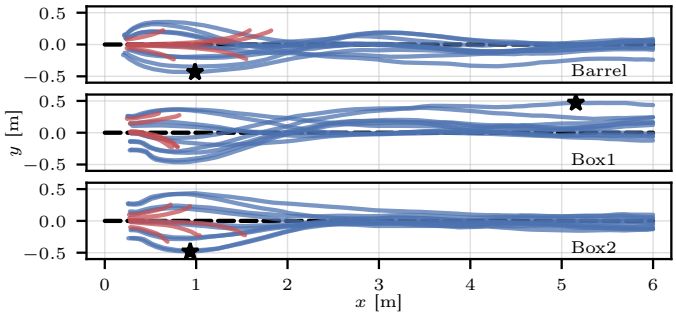

Figure 11: Position trajectories for real sliders pushed along a straight path starting from various lateral offsets. Ten trajectories using our pushing control law are shown for each slider (in blue), with the point of maximum deviation of any trajectory from the path marked by a star. We compare against 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 m, whereas our controller is able to push the objects across the full length of the room.

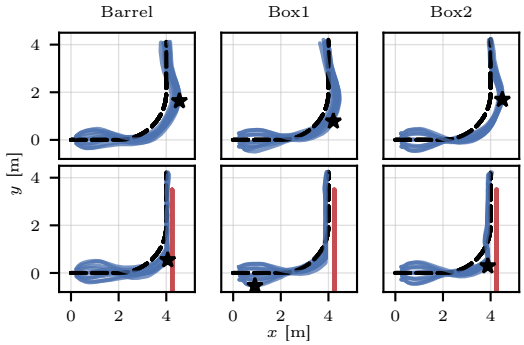

Figure 12: Position trajectories (in blue) for real sliders pushed along a curved path, using our pushing controller. In the top row, each slider is pushed through freespace; in the bottom row, an obstacle acting as a wall is introduced (in red), which blocks the motion of the sliders. Ten trajectories 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.

be seen in Fig. 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 Fig. 11 we see that the controller keeps the slider within approximately 0.5 m of the path at all times, even with different pressure distributions and considerable initial lateral offsets between pusher and slider, and converges to an even narrower range.

The results for tracking a curved path are shown in Fig. 12. We show results for freespace as well as with the addition of a wall, which blocks slider motion (see Fig. 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

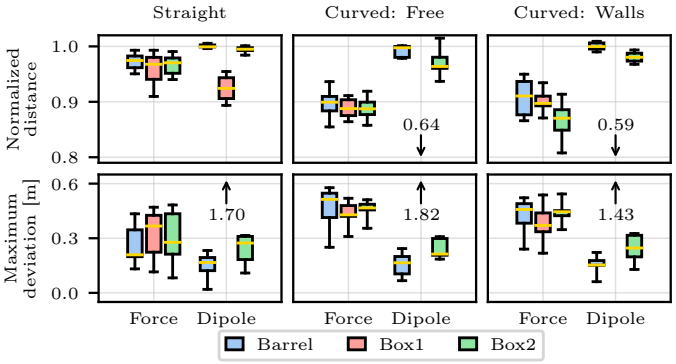

Figure 13: Boxplots of metrics for our proposed approach ("Force"; 10 trajectories per boxplot) and the vision-based baseline from [3] ("Dipole"; 5 trajectories per boxplot). The middle yellow line is the median, the box represents the first and third quartiles, and the whiskers represent the minimum 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).

always maneuver to obtain the desired EE velocity using (5); 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 [3], 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 m ahead of the current closest point on the desired path. Fig. 13 presents metrics 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 Fig. 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 Fig. 14 for an example of the

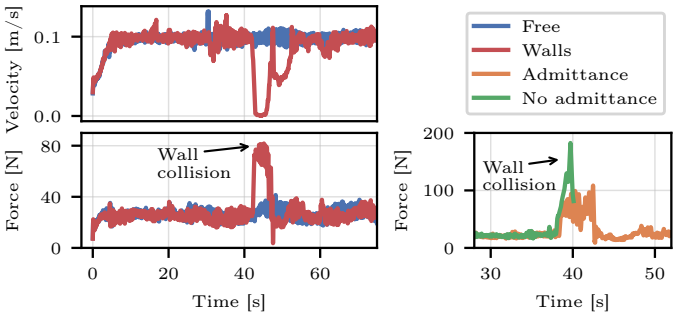

Figure 14: *Left:* Slider velocity and contact force magnitudes from two runs of the Box2 slider along the curved path, with and without the wall obstacle. The force increases and the velocity decreases upon collision with the wall, until the controller adjusts the pushing velocity to continue along 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.

change in 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.

VIII. Conclusion

We presented a control law for quasistatic robotic planar pushing with single-point contact using force feedback, which 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.