Keep it Upright: Model Predictive Control for Nonprehensile Object Transportation with Obstacle Avoidance on a Mobile Manipulator

Adam Heins and Angela P. Schoellig

Abstract—**We consider a nonprehensile manipulation task in which a mobile manipulator must balance objects on its end effector without grasping them—known as the *waiter's problem*—and move to a desired location while avoiding static and dynamic obstacles. 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 balances objects and avoids collisions. 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. Simulations and hardware experiments on a velocity-controlled mobile manipulator with up to seven balanced objects, stacked objects, and various obstacles show that our approach can handle a variety of conditions that have not been previously demonstrated, with end effector speeds and accelerations up to 2.0 m/s and 7.9 m/s^2, respectively. Notably, we demonstrate a projectile avoidance task in which the robot avoids a thrown ball while balancing a tall bottle.**

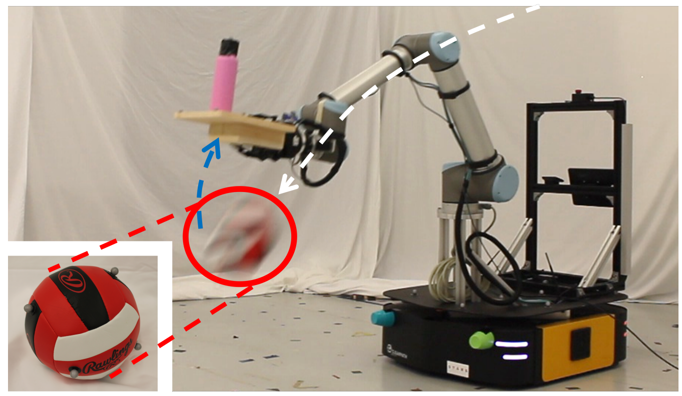

Figure 1: Our mobile manipulator balancing a pink bottle while avoiding a thrown volleyball (ball circled in red with approximate trajectory in white; approximate end effector trajectory in blue). The controller has less than $0.75\,\mathrm{s}$ between first observing the ball and a potential collision. A video of our experiments is available at `http://tiny.cc/keep-it-upright`.

I. Introduction

We consider the nonprehensile object transportation task known as the *waiter's problem* [1], 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. *Nonprehensile* manipulation [2] refers to the case when the manipulated objects are subject only to unilateral constraints [3] and thus retain some degrees of freedom (DOFs); that is, they are not fully grasped. In contrast to *prehensile* manipulation, a nonprehensile approach allows the robot to carry many objects at once with a simple, non-articulated EE (e.g., a tray; see Fig. 1 and 8). Furthermore, a nonprehensile approach skips the potentially slow grasping and ungrasping processes, and can handle small or delicate objects which cannot be adequately grasped [4]. Beyond food service, efficient object transportation is useful across many industries, such as warehouse fulfillment and manufacturing.

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 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 balancing strategy to be robust to such disturbances.

The goal of this work is to develop a controller for a mobile manipulator to quickly transport objects to a desired location without dropping them or colliding with any static or dynamic obstacles, the trajectories of which may not be known a priori. 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.

This work makes the following contributions:

1) **Control:** We propose the first whole-body model predictive controller (MPC) for a mobile manipulator solv-

This is the long version of an article published in IEEE Robotics and Automation Letters. Digital Object Identifier (DOI): 10.1109/LRA.2023.3324520

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

ing 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 balancing up to seven objects; balancing an assembly of stacked objects; and avoiding static and dynamics obstacles, including a thrown volleyball (see Fig. 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 Sec. II and background information in Sec. III and IV, we present our robust balancing constraints in Sec. V and our controller in Sec. VI. Simulations and hardware experiments follow in Sec. VII and VIII, and Sec. IX concludes the paper.

II. Related Work

Prior examples of robots directly inspired by waiters in a restaurant include [5]–[7], 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 balancing objects is to use some manner of sensor feedback to infer the object states. In [8], a manipulator performs the classic inverted pendulum task. In [9], a controller is developed to stabilize a tray based on data from an attached accelerometer and gyroscope. In [10], 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 [10] 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 [11], 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 [12], [13], though these works focus on imposing particular dynamics on the EE rather than directly constraining its motion. EE acceleration constraints are imposed in [14] 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 [15], convex programming is used to solve the TOPP problem. In [16], 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 [4], which demonstrates solving a task where no quasistatic solution exists. An optimization-based planner is applied to the task in [1]. In contrast to these offline planning approaches, our method runs online to react quickly to changes in the environment.

In [17] and [18], a reactive controller automatically regulates the commanded motion to ensure the object remains balanced. A similar approach is applied to legged robots in [19], 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 [20], but similar to [10] 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. In [21], 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 [22], 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, and we also show how reducing the friction coefficients in the controller constraints can provide robustness and computational savings.

III. System Model

We start with the models of the robot and balanced objects.

A. Robot Model

We consider a velocity-controlled mobile manipulator with state $\boldsymbol{x} = [\boldsymbol{q}^T, \boldsymbol{v}^T, \dot{\boldsymbol{v}}^T]^T$, where $\boldsymbol{q}$ is the generalized position, which includes the planar pose of the mobile base and the arm's joint angles, and $\boldsymbol{v}$ 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 [22]. 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 $\boldsymbol{B}(\boldsymbol{x}) \in \mathbb{R}^{\dim(\boldsymbol{x})\times\dim(\boldsymbol{u})}$.

B. Object Model

We model each object $\mathcal{O}$ as a rigid body subject to the Newton-Euler equations

$$\boldsymbol{w}_{\mathrm{C}} + \boldsymbol{w}_{\mathrm{GI}} = \boldsymbol{0}, \tag{1}$$

where $\boldsymbol{w}_\mathrm{C}$ is the total contact wrench and $\boldsymbol{w}_\mathrm{GI}$ is the gravito-inertial wrench, expressed in the body frame as

$$\boldsymbol{w}_\mathrm{GI} \triangleq \begin{bmatrix} \boldsymbol{f}_\mathrm{GI} \\ \boldsymbol{\tau}_\mathrm{GI} \end{bmatrix} = -\begin{bmatrix} m(\dot{\boldsymbol{v}}_o - \boldsymbol{R}_o \boldsymbol{g}) \\ \boldsymbol{J}\dot{\boldsymbol{\omega}}_o + \boldsymbol{\omega}_o \times \boldsymbol{J}\boldsymbol{\omega}_o \end{bmatrix},$$

where $\boldsymbol{f}_\mathrm{GI}$ and $\boldsymbol{\tau}_\mathrm{GI}$ are the gravito-inertial force and torque, m is the object's mass, $\boldsymbol{v}_o$ and $\boldsymbol{\omega}_o$ are the body-frame linear and angular velocity of the object's CoM, $\boldsymbol{g}$ is the gravitational acceleration, and $\boldsymbol{J}$ is the object's inertia matrix taken about the CoM. The rotation matrix $\boldsymbol{R}_o$ represents the object's orientation with respect to the world and is used to rotate gravity into the body frame.

IV. Balancing Constraints

To control the interaction between the EE and balanced 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.). 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 balanced objects do not move with respect to the EE. This is known as a *dynamic grasp* [2]. Now assuming the object is in the sticking mode, we define the object's orientation as $\boldsymbol{R}_o = \boldsymbol{R}_e$, such that it is aligned with the EE's orientation $\boldsymbol{R}_e$. Furthermore, we have $\boldsymbol{v}_o = \boldsymbol{v}_e + \boldsymbol{\omega}_e \times \boldsymbol{c}$ and $\boldsymbol{\omega}_o = \boldsymbol{\omega}_e$, where $\boldsymbol{v}_e$ and $\boldsymbol{\omega}_e$ are the EE's linear and angular velocity in the body frame and $\boldsymbol{c}$ is the position of the object's CoM with respect to the EE. Thus we can write the object's gravito-inertial wrench as

$$\boldsymbol{w}_\mathrm{GI} = -\begin{bmatrix} m(\dot{\boldsymbol{v}}_e - \boldsymbol{R}_e\boldsymbol{g}) + m(\dot{\boldsymbol{\omega}}_e^\times + \boldsymbol{\omega}_e^\times\boldsymbol{\omega}_e^\times)\boldsymbol{c} \\ \boldsymbol{J}\dot{\boldsymbol{\omega}}_e + \boldsymbol{\omega}_e^\times \boldsymbol{J}\boldsymbol{\omega}_e \end{bmatrix}, \quad (2)$$

where $(\cdot)^\times$ converts a vector to a skew-symmetric matrix such that $\boldsymbol{a}^\times\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b}$ for any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$. We assume that the inertial parameters m, $\boldsymbol{c}$, and $\boldsymbol{J}$ are known. Let us group the remaining variables, along with the EE position $\boldsymbol{r}_e$, into the tuple $\boldsymbol{e} = (\boldsymbol{R}_e, \boldsymbol{r}_e, \boldsymbol{\varpi}_e, \dot{\boldsymbol{\varpi}}_e)$, where $\boldsymbol{\varpi}_e = [\boldsymbol{v}_e^T, \boldsymbol{\omega}_e^T]^T$ is the EE's generalized velocity, which we refer to as the EE state. We can compute $\boldsymbol{e}$ from the robot state $\boldsymbol{x}$ via forward kinematics, in which case we may explicitly write $\boldsymbol{e}(\boldsymbol{x})$. As can be seen in (2), the object's motion is completely determined by $\boldsymbol{e}$ when sticking; 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., [17] and [22]. 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 $\mathcal{I} = \{1, \dots, N\}$ (see Fig. 2). By Coulomb's law, each $\boldsymbol{f}_i$ must be inside its friction cone. We use an inner pyramidal approximation of the friction cone

$$\|\boldsymbol{f}_i^t\|_1 \leq \mu_i f_i^n, \quad (3)$$

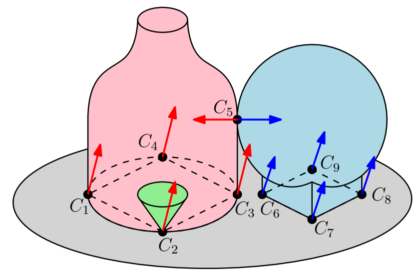

Figure 2: A bottle (red) and globe (blue) balanced on a tray. This arrangement has a total of $N = 9$ contact points (black dots), with each object having $n = 5$ (C_5 is shared). Contact forces (arrows) at each contact point must belong to their friction cones (one shown in green). The circular contact 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 $\hat{\boldsymbol{n}}_i$.

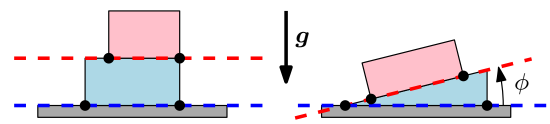

Figure 3: Planar view of two arrangements of objects, each with two objects balanced on a tray and a total of four contact points (black dots). *Left:* the support planes (dashed lines) of each object are parallel, so the orientation shown is feasible in the presence of gravity with no friction forces (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.

where $f_i^n \triangleq \hat{\boldsymbol{n}}_i^T \boldsymbol{f}_i$ is the force along the contact normal $\hat{\boldsymbol{n}}_i$, $\boldsymbol{f}_i^t$ is the force tangent to $\hat{\boldsymbol{n}}_i$, and μ_i is the friction coefficient.

The total contact wrench acting on an individual object is

$$\boldsymbol{w}_\mathrm{C} \triangleq \begin{bmatrix} \boldsymbol{f}_\mathrm{C} \\ \boldsymbol{\tau}_\mathrm{C} \end{bmatrix} = \sum_{j\in\mathcal{J}} \begin{bmatrix} \boldsymbol{f}_j \\ \boldsymbol{r}_j \times \boldsymbol{f}_j \end{bmatrix}, \quad (4)$$

where $\boldsymbol{f}_\mathrm{C}$ and $\boldsymbol{\tau}_\mathrm{C}$ are the total contact force and torque, $\mathcal{J} \subseteq \mathcal{I}$ is the subset of contact indices for this particular object, and $\boldsymbol{r}_j$ is the location of C_j with respect to the object's CoM. The object is successfully balanced for a given $\boldsymbol{e}$ if a set of contact forces can be found each satisfying (3) and consistent with (1), (2), and (4). We assume that contact patches can be represented as polygons with a contact point at each vertex; as in [17] we always use four points with equal μ.

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 Fig. 2): per Newton's third law, the contact force acting on each object must be equal and opposite. Let $\mathcal{O}^a$ and $\mathcal{O}^b$ 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 $\mathcal{O}^a$ and $\mathcal{O}^b$, respectively. Then we have the constraint

$$\boldsymbol{f}_i^a = -\boldsymbol{f}_i^b. \quad (5)$$

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

$$(\boldsymbol{e}, \boldsymbol{\xi}) \in \mathcal{B}$$

to indicate that the EE state $\boldsymbol{e}$ and contact forces $\boldsymbol{\xi}$ together satisfy the balancing constraints (1)–(5) for all objects.

V. Robust Contact Force Constraints

The constraint (3) ensures all contact forces are inside their respective friction cones. However, this assumes accurate knowledge of the friction coefficients, and the constraint 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 [17]. 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 $\boldsymbol{R}_e$ and contact forces $\boldsymbol{\xi}$ satisfying the balancing 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 statically-feasible friction coefficients by solving the optimization problem[1]

$$\begin{aligned} \underset{\boldsymbol{R}_e,\boldsymbol{\xi},\{\mu_i\}_{i\in\mathcal{I}}}{\operatorname{argmin}} \quad & \frac{1}{2}\sum_{i\in\mathcal{I}}\alpha_i\mu_i^2 \\ \text{subject to} \quad & \mu_i \geq 0, \quad i\in\mathcal{I} \\ & (\boldsymbol{e},\boldsymbol{\xi})\in\mathcal{B}, \\ & \boldsymbol{e} = (\boldsymbol{R}_e,\boldsymbol{0},\boldsymbol{0},\boldsymbol{0}), \end{aligned} \tag{6}$$

where $\{\alpha_i\}_{i\in\mathcal{I}}$ are a set of weights. 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 Fig. 3), the solution to (6) is simply $\mu_i = 0$ for all $i\in\mathcal{I}$ with $\boldsymbol{R}_e$ such that the support planes are orthogonal to gravity. An example when the solution of (6) is not $\mu_i = 0$ for all $i\in\mathcal{I}$ is discussed in Sec. VII-B. The problem (6) need only be solved once for a given arrangement of objects.

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, *mobile* manipulation can produce significant EE vibration, requiring robust motions to ensure objects are balanced. 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 (3). 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 (3) with a simple bound, making the optimization problem faster to solve.

We solved (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 (4) we have $\boldsymbol{f}_{\mathrm{C}_{xy}} = \boldsymbol{0}$, where the subscript $(\cdot)_{xy}$ denotes the tangential component. Substituting (2) into (1) with $\boldsymbol{f}_{\mathrm{C}_{xy}} = \boldsymbol{0}$ and dividing out m gives us $[\dot{\boldsymbol{v}}_e - \boldsymbol{R}_e\boldsymbol{g} + (\dot{\boldsymbol{\omega}}_e^\times + \boldsymbol{\omega}_e^\times\boldsymbol{\omega}_e^\times)\boldsymbol{c}]_{xy} = \boldsymbol{0}$. So far this is fine: we can plan trajectories that always satisfy this equation. However, if we have two objects $\mathcal{O}^a$ and $\mathcal{O}^b$ with CoMs $\boldsymbol{c}^a$ and $\boldsymbol{c}^b$, respectively (e.g., the left arrangement in Fig. 3), then the EE trajectory needs to satisfy *both*

$$[\dot{\boldsymbol{v}}_e - \boldsymbol{R}_e\boldsymbol{g} + (\dot{\boldsymbol{\omega}}_e^\times + \boldsymbol{\omega}_e^\times\boldsymbol{\omega}_e^\times)\boldsymbol{c}^a]_{xy} = \boldsymbol{0}, \tag{7}$$

$$[\dot{\boldsymbol{v}}_e - \boldsymbol{R}_e\boldsymbol{g} + (\dot{\boldsymbol{\omega}}_e^\times + \boldsymbol{\omega}_e^\times\boldsymbol{\omega}_e^\times)\boldsymbol{c}^b]_{xy} = \boldsymbol{0}, \tag{8}$$

at all times, where the only difference between (7) and (8) is the CoM $\boldsymbol{c}$. 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 (7) and (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.

VI. 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 $\boldsymbol{\xi}(\tau)$ 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

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{u},\boldsymbol{\xi}}{\operatorname{argmin}} \quad & \frac{1}{2}\int_{\tau=t}^{t+T} L(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\xi})\; d\tau \\ \text{subject to} \quad & \dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u} && \text{(system model)} \\ & (\boldsymbol{e}(\boldsymbol{x}),\boldsymbol{\xi})\in\mathcal{B} && \text{(balancing)} \\ & \boldsymbol{0} \leq \boldsymbol{d}(\boldsymbol{x}) && \text{(collision)} \\ & \underline{\boldsymbol{x}} \leq \boldsymbol{x} \leq \bar{\boldsymbol{x}} && \text{(state limits)} \\ & \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \bar{\boldsymbol{u}} && \text{(input limits)} \end{aligned} \tag{9}$$

where the stage cost is

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

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 balancing 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 $\boldsymbol{\xi}$) and (3) 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 Sec. VIII-B). We assume that the system can always reach a feasible state that achieves the desired EE position. We discretize the prediction horizon

[1] It is not necessary to explicitly use a rotation matrix to parameterize orientation in (6): any representation of $SO(3)$ can be used. We use Euler angles. Note also that the balancing constraints are independent of the EE's position, so we set $\boldsymbol{r}_e = \boldsymbol{0}$. See the appendix for more details.

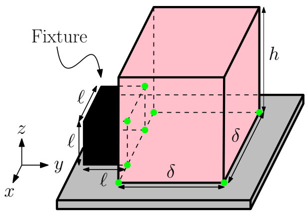

Figure 4: An arrangement consisting of a red box balanced on a tray along with a black *fixture*, which is rigidly attached to the tray. The fixture adds contact points (shown in green) up the side of the box, which our controller can exploit to accelerate faster.

of (9) with a fixed timestep Δt and solve it online using sequential quadratic programming (SQP) via the open-source framework OCS2 [23], with required Jacobians computed using automatic differentiation. 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 [24]. The optimal state trajectory produced by (9) is tracked by a low-level joint controller at the robot's control frequency. More details can be found in the appendix.

VII. 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 Fig. 8. In all experiments (simulated and real) we use $\Delta t = 0.1\,\mathrm{s}$, $T = 2\,\mathrm{s}$, and weights

$$\begin{aligned} \boldsymbol{W}_r &= \boldsymbol{I}_3, & \boldsymbol{W}_x &= \mathrm{diag}(0\boldsymbol{I}_9, 0.1\boldsymbol{I}_9, 0.01\boldsymbol{I}_9), \\ \boldsymbol{W}_u &= 0.001\boldsymbol{I}_9, & \boldsymbol{W}_f &= 0.001\boldsymbol{I}_{\dim(\boldsymbol{\xi})}, \end{aligned}$$

where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix. We use a single SQP iteration per control policy update.

A. Balancing Constraint Comparison

We first consider the example shown in Fig. 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 different sets of balancing constraints:

- **None:** No constraints.
- **Upward:** A constraint to keep the tray oriented upward.
- **Full:** The full set of balancing constraints $(\boldsymbol{e}, \boldsymbol{\xi}) \in \mathcal{B}$ with each μ_i set to 90% of the true value.[2]
- **Robust:** The full set of constraints $(\boldsymbol{e}, \boldsymbol{\xi}) \in \mathcal{B}$ with $\{\mu_i\}_{i \in \mathcal{I}}$ computed using (6). Unless otherwise stated, the solution is $\mu_i = 0$ for all $i \in \mathcal{I}$.

[2]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.

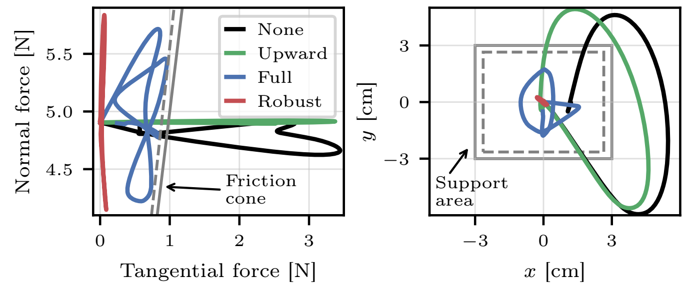

Figure 5: *Left:* Force applied to the simulated box during motion. *Right:* Corresponding ZMP trajectories. With no constraints (None) or the Upward constraint, the force leaves the friction cone and the ZMP leaves the support area (safety margins in dashed lines), so the box slides and tips over (and is dropped). The Full constraints touch but do not pass the boundaries; the Robust constraints stay far from the boundaries in both cases.

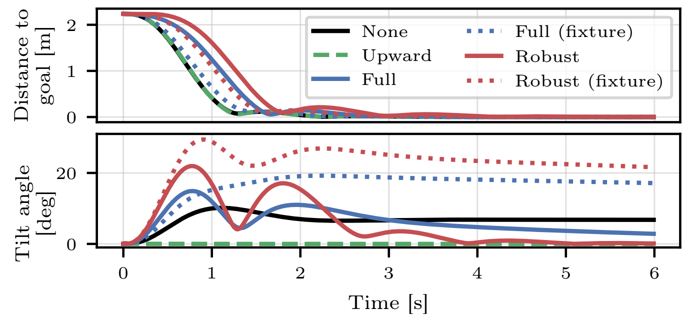

Figure 6: *Top:* Distance of EE to goal location. *Bottom:* Tilt angle with respect to the upward-pointing (i.e. gravity-aligned) orientation. The Full and Robust constraints limit acceleration to keep the box balanced; the Robust approach also uses higher tilt angles. The None and Upward approaches 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.

In ideal conditions, the Full and Robust constraints should both keep the objects balanced, 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 Fig. 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 Fig. 6, 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 balancing

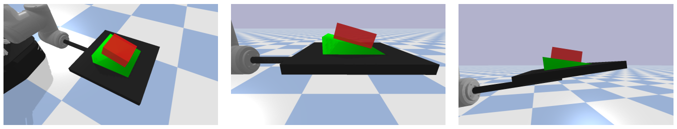

Figure 7: *Left:* Initial position of wedge (green) and box (red) arrangement. *Middle:* Initial side view. The box is tilted $15°$ relative to the ground due to the slope of the wedge. *Right:* The position at $t = 6\,\mathrm{s}$. The controller has oriented the tray so that both the wedge and box are tilted $7.5°$ relative to the ground, such that each requires as small a μ as possible.

constraints.

B. Non-Parallel Support Planes

Next we show an example when the solution to (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 Fig. 3. For simplicity we assume that μ is constant between each pair of objects, so we need only solve (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 Fig. 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 (6) and simply try to enforce $\mu_i = 0$ for all $i \in \mathcal{I}$, the controller fails to converge because no feasible stationary solution exists.

VIII. 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. We also use motion capture to track the position of the balanced objects, which is only used for error reporting. The controller parameters and weights are the same as in the previous section. The controller is run on a standard laptop with eight Intel Xeon CPUs at $3\,\mathrm{GHz}$ and $16\,\mathrm{GB}$ of RAM. The robot and balanced objects are shown in Fig. 8; the corresponding object parameters are given in Table I. A video of the experiments can be found at `http://tiny.cc/keep-it-upright`.

A. 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

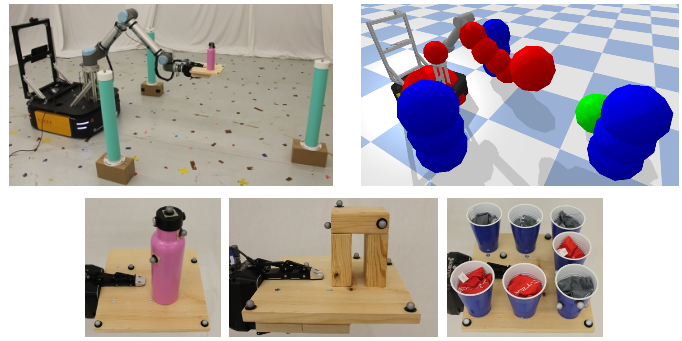

Figure 8: *Top left:* Real experimental setup. Robot is shown holding the Bottle object. Obstacle locations marked with pylons. *Top right:* Corresponding simulated experimental setup with collision spheres on the robot in red and on the obstacles in blue. *Bottom row:* Bottle, Arch, and Cups object 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.

Table I: Approximate parameters for balanced objects shown in Fig. 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

and controller compute time in an obstacle-free environment are shown in Fig. 9; results for an environment with static obstacles are shown in Fig. 10. We model obstacles as collections of spheres; spheres also surround parts of the robot body for collision checking (see top right of Fig. 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 Fig. 10 are similar to those for free space except for a modest increase in compute time. Sample trajectories are shown in Fig. 11.

B. Dynamic Environments

We now consider environments that change over time due to dynamic obstacles. Dynamic obstacles are modelled as

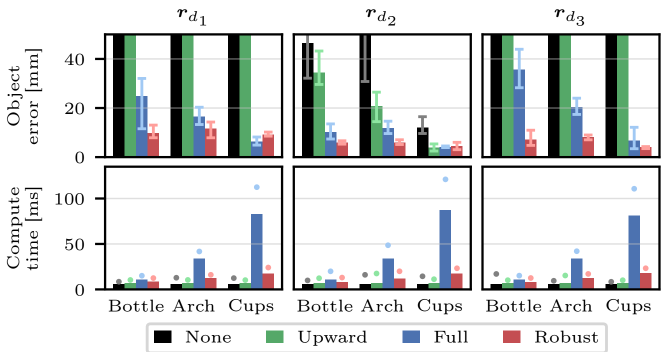

Figure 9: Object error (top row) and policy compute time (bottom row) for different combinations of objects, goal positions, and constraints in free space. The object error is the maximum distance the object moves from its initial position relative to the tray. In arrangements with multiple objects, 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 (9)). The bar shows the average across the three runs (up to the first 6 s of the trajectory); the dot shows the maximum value from any of the runs.

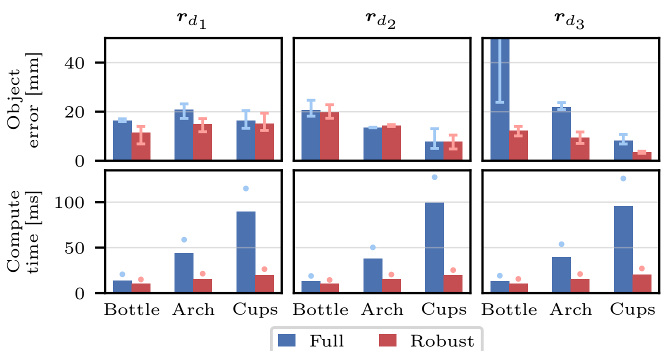

Figure 10: The same results as shown in Fig. 9 but in an environment with static obstacles and only showing the Full and Robust approaches. Compared to Fig. 9, the errors are similar while the compute times are slightly higher.

spheres with known radii, but the controller does not know their trajectories a priori.

1) 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 Fig. 8)—imagine a restaurant customer suddenly backing out their chair. The results for different t are shown in Fig. 12. The appearance of the obstacle causes significant changes in the trajectory of both the EE and base, but the object is continually balanced despite the sudden change, even when the collision constraint is violated by the obstacle's appearance. The maximum 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.

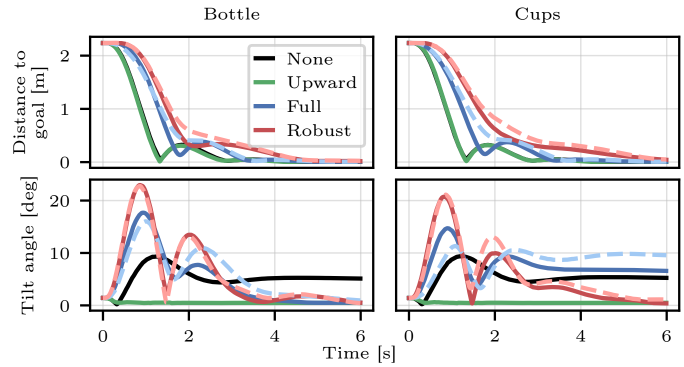

Figure 11: Samples of trajectories to goal $\boldsymbol{r}_{d_1}$ for the Bottle and Cup arrangements with different constraints. Free space results are solid lines; results with static obstacles (shown only for Full and Robust) are dashed. The addition of static obstacles modestly changes the shape 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.

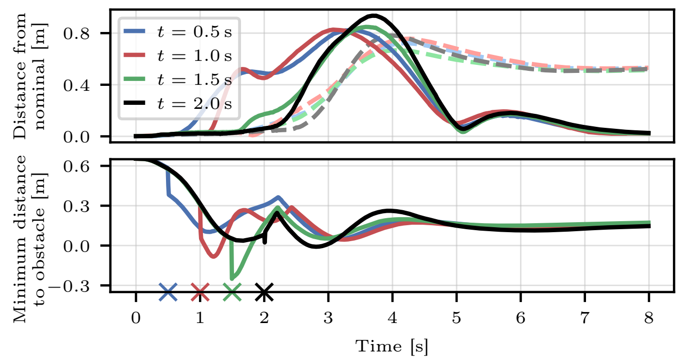

Figure 12: *Top:* Distance of EE (solid) and base (dashed) positions from a nominal trajectory with a virtual obstacle suddenly appearing at different times t (also marked with crosses on x-axis). The nominal trajectory has no dynamic obstacle. *Bottom:* Minimum distance between any collision sphere 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.

2) 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 [25] or batting it [26]. 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 the motion capture system. The state $\boldsymbol{b}$ and the projectile dynamics are added to (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 Fig. 14 and images from one throw are shown in Fig. 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

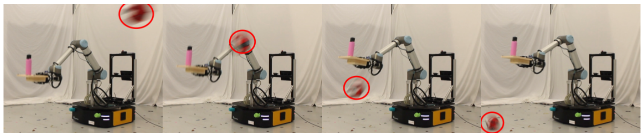

Figure 13: Example of the robot dodging the volleyball (circled red) while balancing the bottle, with frames spaced by $\Delta = 0.15\,\mathrm{s}$. Once the ball has passed, the EE moves back to the initial position.

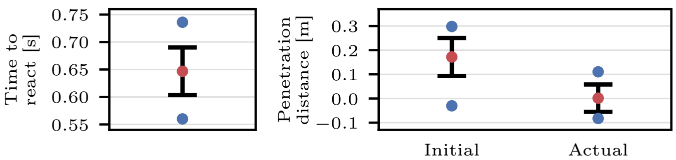

Figure 14: The projectile avoidance results over 20 trials. In each plot the red dot is the mean, the error bars represent the standard deviation, and the blue dots are the minimum and maximum values. *Left:* The time at which collision would first occur if the robot did not move. In all cases the controller 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.

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 balancing the object and avoiding collision. However, since the collision spheres are conservatively large, we did not experience any failures due to collisions. 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.

IX. Conclusion

We presented an MPC-based approach for balancing objects with a velocity-controlled mobile manipulator and demonstrated its performance in simulated and real experiments in a variety of static and dynamic scenarios. In particular, our method is able to react quickly to moving obstacles. We also proposed using minimal values of μ to add robustness to frictional uncertainty and other force disturbances, and demonstrated that this approach is effective and computationally efficient in real-world experiments. Future work will explore the effect of uncertainty in the objects' inertial parameters and the use of object state feedback in the controller.

Appendix

This appendix provides additional implementation details to complement the main body of the manuscript.

A. Robot Kinematic Model

In the main body of the manuscript we leave the robot kinematic model general to accommodate different systems. The actual kinematic model for the robot used in our experiments is

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{10}$$

where

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

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, since the MPC problem (9) is already nonlinear.

B. Minimum Statically-Feasible Friction Coefficients

Here we provide more details on the structure of the optimization problem (6), used to obtain the minimum statically-feasible friction coefficients. For simplicity we assume that a single object is balanced. We use roll-pitch-yaw Euler angles $\boldsymbol{\theta}$ to parameterize rotation. In this case (6) has the form

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\xi},\{\mu_i\}_{i\in\mathcal{I}}}{\operatorname{argmin}} \quad & \frac{1}{2}\sum_{i\in\mathcal{I}} \alpha_i \mu_i^2 & \text{(11a)} \\ \text{subject to} \quad & \mu_i \geq 0, \quad i \in \mathcal{I} & \text{(11b)} \\ & \boldsymbol{F}_i \boldsymbol{f}_i \geq \mathbf{0}, \quad i \in \mathcal{I} & \text{(11c)} \\ & \sum_{i\in\mathcal{I}} \begin{bmatrix} \boldsymbol{f}_i \\ \boldsymbol{r}_i \times \boldsymbol{f}_i \end{bmatrix} = \begin{bmatrix} -m\boldsymbol{R}_e(\boldsymbol{\theta})\boldsymbol{g} \\ \mathbf{0} \end{bmatrix}, & \text{(11d)} \end{aligned}$$

where $\boldsymbol{R}_e(\boldsymbol{\theta})$ is the mapping from $\boldsymbol{\theta}$ to the corresponding rotation matrix and

$$\boldsymbol{F}_i = \begin{bmatrix} 1 & 0 & 0 \\ \mu_i & -1 & -1 \\ \mu_i & -1 & 1 \\ \mu_i & 1 & -1 \\ \mu_i & 1 & 1 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{n}}_i^T \\ \boldsymbol{S}_i^T \end{bmatrix}$$

with $\boldsymbol{S}_i$ an orthonormal basis for the tangential component of $\boldsymbol{f}_i$, such that (11c) is equivalent to the friction cone constraint (3). This is the same friction model that appears in, e.g., [21]. For multiple objects, each would need a set of Newton-Euler equality constraints (11d). We would also need to include the constraint (5) for any contact points between two of the objects.

The problem (6) 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 $\boldsymbol{R}_e(\boldsymbol{\theta})$, but we did not have a problem solving it with the SLSQP solver [27] from `scipy`.

C. Soft Constraints

Here we provide details the soft constraints used in (9). We soften all of the constraints in (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 balancing 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 (1) for each object to

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

Dividing by the object's mass m ensures that balancing 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 (1) to dominate the other objectives and penalties in the problem (9) just because more objects and contact points have been added to the problem. We found this to be particularly useful with the Cups arrangement (where $N = 28$).

D. Model Predictive Controller Details

Here we provide some additional details about the MPC problem (9). The state and input constraints used for the robot in (9) are

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

where $\bar{\boldsymbol{x}} = [\bar{\boldsymbol{q}}^T, \bar{\boldsymbol{v}}^T, \bar{\dot{\boldsymbol{v}}}^T]^T$, $\underline{\boldsymbol{x}} = -\bar{\boldsymbol{x}}$, $\underline{\boldsymbol{u}} = -\bar{\boldsymbol{u}}$, and $\mathbf{1}_n$ denotes an n-dimensional vector of ones.

As stated in the main manuscript, (9) is solved using sequential quadratic programming. The quadratic program (QP) subproblems are solved using the QP solver HPIPM [24]. The automatic differentiation library CppAD [28] is used to obtain the Jacobians required to construct the QP subproblems.

E. Low-level Joint Controller

The MPC problem (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 (9). Suppose we compute a new MPC policy using (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

$$\boldsymbol{u}(\tau) = \boldsymbol{K}(\tau)(\boldsymbol{x}^\star(\tau) - \boldsymbol{x}(\tau)) + \boldsymbol{k}(\tau), \tag{12}$$

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, (9) is discretized and linearized to form a quadratic program (QP), which is solved using an interior point method (IPM) [24]. 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 IPM used to solve the QP (see [24] as well as [29] and [30] for more details). The upshot of (12) 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 (12) 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 (9) is solved in a separate process. We limit policy updates to at most once every $10\,\mathrm{ms}$ and we use (12) to generate commands at the robot's control frequency of $125\,\mathrm{Hz}$.

F. State Estimation

Two Kalman filters are used for state estimation in our hardware experiments. One is used to estimate the state of the robot $\boldsymbol{x}$ and the other is used to estimate the state of the projectile $\boldsymbol{b}$ during the dynamic obstacle avoidance experiments in Sec. VIII-B.2. Both the robot and projectile models are linear, allowing us to use the standard linear Kalman filter (see e.g. [31]). We are provided with position measurements for both systems: for the robot, the pose of the mobile base is provided by a Vicon motion capture system, and the joint angles of the arm are provided by its joint encoders. The position of the projectile is also obtained from the Vicon system. In the following, we describe the discrete-time equations of motion and the covariance matrices required for each Kalman filter.

1) Robot Kalman Filter: Discretizing (10) gives us the discrete-time model

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

where

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

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

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

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 $\boldsymbol{Q} = 10\boldsymbol{I}_9$, $\bar{\boldsymbol{R}} = 0.001\boldsymbol{I}_9$, and $\boldsymbol{P}_0 = 0.1\boldsymbol{I}_{27}$.

2) Projectile Kalman Filter: 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

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

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

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

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. In experiment we use process covariance $\bar{\boldsymbol{Q}}_b = \bar{\boldsymbol{B}}_b \boldsymbol{Q}_b \bar{\boldsymbol{B}}_b^T$ with $\boldsymbol{Q}_b = 1000\boldsymbol{I}_3$, measurement covariance $\bar{\boldsymbol{R}}_b = 0.001\boldsymbol{I}_3$, and initial state covariance $\boldsymbol{P}_{b_0} = \boldsymbol{I}_6$.

References

[1] 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.

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

[3] 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.

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

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

[6] 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.

[7] 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.

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

[9] 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. on Industrial Automation, Information and Communications Technology*, 2014, pp. 15–21.

[10] 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. on Humanoid Robots*, 2018, pp. 1–6.

[11] 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.

[12] 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.

[13] 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. on Intelligent Robots and Systems*, 2022, pp. 223–230.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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. on Intelligent Robots and Systems*, 2020, pp. 6718–6725.

[26] 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. on Instrumentation and Measurement*, vol. 67, no. 4, pp. 811–820, 2018.

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

[28] "CppAD: A package for differentiation of C++ algorithms." [Online]. Available: https://github.com/coin-or/CppAD

[29] 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.

[30] 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.

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