Autonomous Robots manuscript No.
(will be inserted by the editor)

Robust Trajectory Optimization Under Frictional Contact with Iterative Learning

Jingru Luo · Kris Hauser

the date of receipt and acceptance should be inserted later

Abstract Optimization is often difficult to apply to robots due to the presence of errors in model parameters, which can cause constraints to be violated during execution on the robot. This paper presents a method to optimize trajectories with large modeling errors using a combination of robust optimization and parameter learning. In particular it considers the context of contact modeling, which is highly susceptible to errors due to uncertain friction estimates, contact point estimates, and sensitivity to noise in actuator effort. A robust time-scaling method is presented that computes a dynamically-feasible, minimum-cost trajectory along a fixed path under frictional contact. The robust optimization model accepts confidence intervals on uncertain parameters, and uses a convex parameterization that computes dynamically-feasible motions in seconds. Optimization is combined with an iterative learning method that uses feedback from execution to learn confidence bounds on modeling parameters. It is applicable to general problems with multiple uncertain parameters that satisfy a monotonicity condition that requires parameters to have conservative and optimistic settings. The method is applied to manipulator performing a "waiter" task, on which an object is moved on a carried tray as quickly as possible, and to a simulated humanoid locomotion task. Experiments demonstrate this method can compensate for large modeling errors within a handful of iterations.

Jingru Luo
Indiana University at Bloomington
now at Robert Bosch LLC
E-mail: luojing@indiana.edu

Kris Hauser
Duke University
100 Science Dr. Box 90291
Durham, NC 27708 USA E-mail: kris.hauser@duke.edu

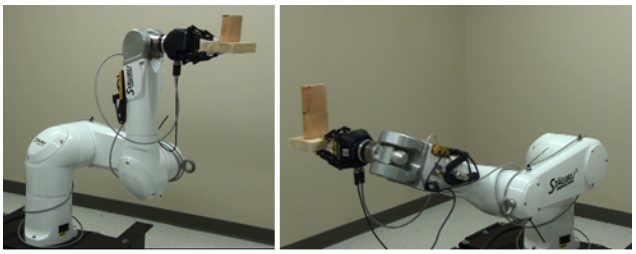

Fig. 1: Left: The waiter task for a single block. Right: the task for a stack of two blocks.

1 Introduction

Optimization is difficult to apply to physical robots because there is a fundamental tradeoff between optimality and robustness. Optimal trajectories pass precisely at the boundary of feasibility, so errors in the system model or disturbances in execution will usually cause feasibility violations. For example, optimal motions in the presence of obstacles will cause the robot to graze an object's surface. It is usually impractical to obtain extremely precise models. This is particularly true regarding dynamic effects, such as inertia matrices and hysteresis, and contact estimation, including points, normals, and friction coefficients. In legged locomotion, such errors may cause a catastrophic fall, and in nonprehensile manipulation, such errors may cause the object to slip, tip, or fall. Moreover, accurate execution of trajectories is becoming more difficult as robotics progressively adopts compliant, human-safe actuators that are less precise than the highly-geared motors of traditional industrial robots. A final source of error is numerical error in optimization algorithms, such as the resolution of constraint checks along a trajectory using point-wise collocation. One way to increase robustness is to add a margin of error to optimization constraints, e.g., by assuming very small frictions, conservative velocity bounds, or collision avoidance margins. However, this approach leads to unnecessarily slow executions and/or difficulties in margin tuning.

This paper presents an iterative learning approach in which a robot 1) learns the errors in its models given execution feedback, 2) incorporates estimated errors into optimization, and 3) repeats the process until it converges to successful and/or near-optimal executions. Specifically, it addresses tasks under frictional contact with significant friction uncertainty and execution errors, such as legged locomotion and object manipulation tasks, and its main contributions lie in a robust trajectory optimization module and an execution

feedback module. The robust optimization accepts confidence intervals on the uncertain parameters and computes a dynamically-feasible time-parameterized trajectory along a fixed geometric path with contact constraints. The iterative learning step uses observations from execution to adjust uncertainty intervals to achieve robust, successful trajectories.

Our contributions help the learning process proceed at interactive rates. Although trajectory optimization is widely used in robotic motion planning and optimal control, optimization under frictional contact remains challenging. For tractability of computation we consider optimizing only velocity along a fixed path given dynamic constraints (the *time-scaling* problem). This paper formulates a new fast time-scaling method for generating an optimal, robust, dynamically feasible time-parameterized trajectory along a fixed geometric path for a robot in frictional contact. We formulate the contact constraints on the system as a linear program that is solved at each constraint evaluation point of the time-scaling optimization. This transforms the optimization problem into a two level hierarchical optimization with the confidence intervals on uncertain parameters working as margins in both levels. The decomposed optimization runs much faster (in seconds) than the monolithic large optimization (in minutes).

We test our methods on a manipulator performing a "waiter" task (Fig. 1). Imagining a robotic waiter serving customers, objects are placed on a tray carried by the robot and the goal is to move the objects as quickly as possible without causing them tipping over. Modeling and execution uncertainties are inevitable. For irregular-shaped objects and objects made from unknown materials, it is difficult to have the precise values on the contact friction and COM of the objects. Moreover, there could be large disturbances acting on the tray in a dynamic environment similar to a busy restaurant. Using the proposed framework, the robot would "practice" a handful of times to learn estimates of parameter and execution uncertainty, and then maintain those estimates to optimize trajectories robustly in the future.

We handle two types of uncertainties: (i) *parameter estimation errors* for parameters of constant value over time, such as coefficient of friction (COF), hardware calibration, contact point position, center of mass (COM) of a manipulated object, and (ii) *execution errors* from control and disturbances. These are handled via two types of execution feedback:

1. Binary execution feedback (success or failure) is used to estimate unobservable parameters of the dynamic model. Our method accepts estimated ranges of the chosen parameter, and then uses a bisection search method to determine whether we have over- or under-estimated its value, converging to a value that leads to a feasible execution within ϵ of optimal. For multiple parameters, we use a branch-and-bound search method to explore the multi-dimensional parameter error space.
2. Joint sensor feedback on the executed trajectory lets us estimate execution errors (e.g., random disturbances). We construct confidence intervals on the deviation between executed and planned trajectory as margins into optimization.

Our method can consider both types of error either individually or simultaneously. In its current implementation our method assumes that the position and velocity error are small, and we focus on second-order uncertainties that are the most relevant to problems with contact.

Experiments evaluate the proposed method on a block manipulation task in simulation and on hardware with a Stäubli TX90L industrial manipulator (Fig.1). Experiments also test the proposed multi-parameter branch-and-bound method with a simulated Hubo+ humanoid robot performing a legged movement. Results demonstrate our robust trajectory optimization can generate dynamically-feasible trajectories within 10 seconds on a standard PC, and the iterative learning approach is able to produce robust task executions within a handful of iterations.

2 Related Work

The problem of generating time-optimal, dynamically-feasible trajectory of a given path without considering contact was initially solved by Bobrow et al [4] and Shin and McKay [31] and later enhanced by [7, 8, 22, 32]. As this earlier work used numerical integration, it tended to suffer from numerical errors [17]. Improvements in robustness and scalability [14, 20] have emerged using a convex optimization formulation of the time scaling problems introduced in [33]. Recent work has extended the time-scaling problem to handle frictional contact [16]. We further extend the time-scaling concept to incorporate robustness margins in friction parameters and execution disturbances.

Although the problem of time-optimal path parameterization is well-studied for robots operating in free-space [3, 34], this problem has not been widely considered for robots that make or break contact. Several authors have considered optimizing both joint trajectories and timing for legged robots [9, 12, 28, 29]. A trajectory optimization method in Cartesian space for solving humanoid motions is presented in [12] and is suit-

able for generating motion primitives. In [9], a contact-before-motion planner, which grows a search tree using a rough trajectory, is proposed. [28, 29] present a direct trajectory optimization method which formulates the reaction forces as optimization parameters. But due to the size of the optimization problem, these techniques are often extremely slow, taking minutes or hours to complete. Some numerical approaches have been presented for dexterous manipulation [21, 25] as well as non-prehensile grasp [19, 24, 27], but extending them to high dimensional problems has proven computationally challenging.

Recent work [16] considers contact forces in optimization and presented a fast convex time-scaling method with polytope projection techniques to precompute dynamic feasible sets. Similar to [16], our work avoids the computation of exact contact forces to speed up the optimization. But we use a different formulation on the contact constraints through a transformation into smooth linear programs at constraint evaluation points, which allows us to consider robustness to execution errors without severely increasing running time. We also illustrate how to incorporate the coupled dynamic constraints between manipulated objects as well as the robot.

Robotic applications cannot escape the uncertainties of imperfect system modeling, robot control, and execution disturbances. Many learning methods have been proposed to handle uncertainties of different types: learning from demonstration [1] for computing a task-specific policy, model learning with machine learning techniques [26] and control learning [5, 30]. In this work, we utilize ideas from previous work and use an iterative learning approach based on a binary search method and a robust trajectory optimization to learn the uncertainties from modeling and execution. The robust optimization models acceleration errors as range estimation (similar to [2]) and optimizes a nonlinear problem to generate a trajectory that is robust up to a given confidence value. Perhaps the most similar approach to ours is the method of Lengagne et al (2011) that replans an optimized path using constraints revised from execution data [18]. Our approach iterates more than once to achieve a desired level of convergence, and learns from multiple sources of error.

Portions of this work were presented in the conference paper [23]. The novel contributions of this paper include the multi-parameter learning method and locomotion experiments presented in Section 7.

3 Problem Formulation

The goal of our method is to generate a robust, dynamically feasible, and near-optimal trajectory for a robot manipulating an object under frictional contact and imperfect knowledge about the system model and random execution errors. The robot is allowed to execute trajectories repeatedly to improve its estimates. However, its feedback will only consist of the executed joint trajectories and success/failure of the manipulation; no external observations like object motion capture or pressure sensors are available.

In the time-scaling problem, we assume that a geometric path (without timing information) is given. This is indeed a limitation of our method. In future extensions, we would like to consider optimizing the shape of the path. Also, we consider three types of errors in our experiments: modeling errors in the coefficient of friction, modeling errors in contact locations, and random disturbances in the motion execution. Our work may be applicable to other sources of error, but this must be examined on a case-by-case basis.

3.1 Summary of method

The method operates as follows:

1. The robot is given a geometric path, robot-object contact points, and initial confidence intervals of the uncertainty in modeling parameters and execution errors. (These estimated ranges may be pessimistic or optimistic; the robot will adapt them later via execution feedback.)
2. Robust time-scaling optimization is applied to generate a trajectory that is feasible under the given confidence intervals.
3. The optimized path is executed. Success / failure and sensing feedback are used to improve the confidence intervals.
4. Steps 2 and 3 are iterated until success and/or a desired level of convergence is achieved.

3.2 Robot Motion Constraints

Given a twice-differentiable geometric path $q(s) : s \in [0,1] \mapsto R^n$ for a robot of n degrees of freedom (DOFs), we wish to find a time parameterization $s(t) : t \in [0, t_f] \mapsto [0,1]$ and a final time t_f such that an objective function f_{obj} is minimized and a set of constraints is satisfied.

The objective function f_{obj} considered here is the minimum execution time, but could also include other terms such as minimum jerk or energy. The constraints

of our problem include velocity, acceleration and torque limits given by

$$\dot{q}_{min} \leq \dot{q} \leq \dot{q}_{max} \tag{1}$$
$$\ddot{q}_{min} \leq \ddot{q} \leq \ddot{q}_{max} \tag{2}$$
$$\tau_{min} \leq \tau \leq \tau_{max}. \tag{3}$$

Also, for safety reasons we usually require the robot starts and ends at a stop, which is enforced using constraints

$$\dot{q}(0) = 0, \quad \dot{q}(1) = 0. \tag{4}$$

We are interested in obtaining dynamically-feasible motion with robot in contact. In addition to the geometric path $q(s)$, we assume the input also contains the following information:

1. If there exist $N_p > 1$ contact phases, the domain of q is divided into sections $sp_0 = 0, sp_1, ..., sp_{N_p} = 1$ in which contacts are constant over the range $\{q(u) \quad | \quad sp_i < u < sp_{i+1}\}$.
2. The set F_i of contact points and normals in each phase $i = 1, \ldots, N_p$.

For simplicity, we describe our formulation and methods in the simplest case, which only has one contact phase. Extensions to handle multiple contact phases are presented in Section 4.5.

Contact enforces two additional constraints: contact forces are limited within friction cones, and maintaining the contact states requires the contact points have zero velocity in world space. Let $f_1, ..., f_m$ be the contact forces at points $p_1, ..., p_m$ respectively. The system dynamics is given by

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau + \sum_i J_i(q)^T f_i \tag{5}$$

where M is the mass matrix, C is the centrifugal and Coriolis forces which are quadratic in the joint velocities, G is the generalized gravity vector, and $J_i(q)$ is the Jacobian of point p_i with respect to robot's configuration q. We assume that the initial path is chosen to respect the condition that fixed contact positions have zero velocity in world space: $J_i(q)\frac{dq}{ds} = 0$. Also, we consider the friction constraints $f_i \in FC_i$ for $i = 1, \ldots, m$, where FC_i is the friction cone at contact position p_i. For the purposes of this paper, we assume all friction cones are convex and we approximate the nonlinear cones by a set of linear constraints expressed as

$$A_i f_i \leq 0 \text{ for } i = 1, ..., m. \tag{6}$$

Since each friction cone is dependent on a coefficient of friction estimate μ_i we may write the dependence explicitly as $FC_i \equiv FC_i(\mu_i)$ and $A_i \equiv A_i(\mu_i)$.

3.3 Extra Constraints for the Object Manipulation Task

For the "waiter" task, to move the objects together with the robot, we must ensure the linear acceleration α_L and angular acceleration α_A of the object can be supported by the contact forces and gravity, therefore the Newton-Euler equations impose the following two dynamic constraints on the system:

$$\begin{aligned} m_{obj}\alpha_L &= \sum_i f_i + m_{obj} g \\ I_{obj}\alpha_A &= \sum_i (p_i - c) \times f_i \end{aligned} \tag{7}$$

where m_{obj} and I_{obj} are the mass and inertia matrix of the block respectively, p_i is the contact point, and c is the center of mass (all quantities given in world coordinates). Given the configuration, velocity and acceleration of the robot, the accelerations of the object can be computed through the Jacobian and Hessian of the object's pose in the robot frame as follows:

$$\begin{pmatrix} \alpha_L \\ \alpha_A \end{pmatrix} = J(q)\ddot{q} + \dot{q}^T H(q)\dot{q}. \tag{8}$$

4 Robust Time-Scaling with Friction

This section presents a robust time-scaling optimization method for the dynamic equations presented above. Computation speed is improved by transforming the constraints on the system dynamics and contact forces into a linear program so that the outer optimization problem size is reduced to be independent of the number of contact points. Robustness against execution errors is achieved by optimizing over a confidence interval that estimates the noise on the execution acceleration.

4.1 Robust Motion Constraints with Confidence Intervals

Errors introduce nontrivial changes in the way dynamic constraints are evaluated; for example in the "waiter" task, the *executed* acceleration of the object, rather than the *planned* acceleration, determines whether the contact forces are sufficient to move the blocks without tipping over. Moreover, the robot's velocity and acceleration determines the object's acceleration in a nontrivial manner.

Errors on the robot's execution velocity and acceleration can be modeled as joint-wise confidence intervals $\dot{q} - \dot{q}_{plan} \in [\dot{q}_{low}, \dot{q}_{upp}]$ and $\ddot{q} - \ddot{q}_{plan} \in [\ddot{q}_{low}, \ddot{q}_{upp}]$ where $\dot{q}_{plan}$ is the planned velocity and $\ddot{q}_{plan}$ is the planned acceleration. For simplicity, our method ignores the errors

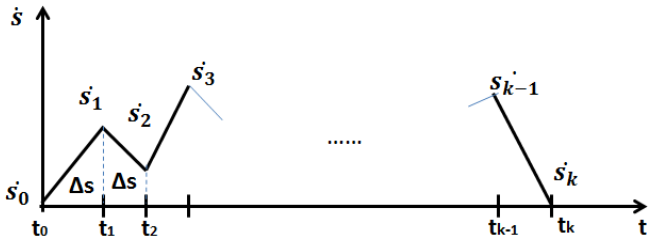

Fig. 2: Velocity $\dot{s}$ of the time scaling.

in $\dot{q}$ because the errors are usually two orders of magnitude smaller than acceleration errors in the industrial and humanoid robots that we evaluate.

As described in later sections, we will estimate joint-wise execution errors as $\ddot{q}_{exc} = \ddot{q}_{plan} + e$ where e is subject to Gaussian distribution $N(U, \Delta^2)$ to account for the uncertainties from execution and unknown disturbances. We convert the distribution to a confidence interval $[\ddot{q}_{low}, \ddot{q}_{upp}]$ as follows:

$$\ddot{q}_{upp} = U + K \cdot \Delta, \quad \ddot{q}_{low} = U - K \cdot \Delta \tag{9}$$

where K controls the confidence ratio on uncertainties.

When we apply the confidence interval to the dynamic constraints, we wish to ensure that all constraints related to $\ddot{q}$ in (2), (5) and (8) are satisfied with respect to the *execution* accelerations. As a result, this imposes additional upper and lower constraints on the *planned* accelerations.

For confidence intervals on the friction coefficient $[\mu_{low}, \mu_{upp}]$, it is clear that we only need to formulate the lower end of the confidence interval μ_{low} because any solution for the lower friction case is also a solution for the higher friction case.

4.2 Time-Scaling Function

The proposed time scaling method is a robust version of the method of [16]. The constraints on the velocity and acceleration require the time-scaling function $s(t)$ to be at least twice differentiable. So we define the velocity of the path $s(t)$ as a piecewise linear interpolation between $\dot{S} = \{\dot{s_0}, ..., \dot{s_N}\}$ (see Fig. 2) and then $s(t)$ becomes a piecewise quadratic curve consisting of N segments $s_i(t), i = 1, ..., N$ with constant acceleration in each segment. We also assume each segment to have a uniform duration Δs to simplify the computation. Therefore, the objective function of minimizing execution time is

$$f_{obj}(\dot{S}) = \sum_{i=0}^{N} \frac{\Delta s}{\dot{s}_i + \dot{s_{i+1}}}. \tag{10}$$

Using interval arithmetic notation, we can express the executed derivatives as follows:

$$\dot{q}(s, \dot{s}) \in \frac{dq}{ds}\dot{s} + [\dot{q}_{low}, \dot{q}_{upp}] \tag{11}$$

$$\ddot{q}(\dot{s}, \ddot{s}) \in \frac{d^2q}{ds^2}\dot{s}^2 + \frac{dq}{ds}\ddot{s} + [\ddot{q}_{low}, \ddot{q}_{upp}]. \tag{12}$$

Since we assume negligible velocity error, the system dynamics (5) can be represented in terms of $\dot{s}$, $\ddot{s}$ as follows:

$$a(s)\ddot{s}+b(s)\dot{s}^2+c(s) = \tau+\sum_i J_i(q)^T f_i - M(q)[\ddot{q}_{low}, \ddot{q}_{upp}] \tag{13}$$

where $a(s) = M(q)\frac{dq}{ds}$, $b(s) = M(q)\frac{d^2q}{ds^2} + C(q, \frac{dq}{ds})$ and $c(s) = G(q)$. The last term on the right hand side involves a product of a matrix and a vector interval, and is computed using interval arithmetic.

The optimization can now be performed over a time parameterization $s(t)$, with constraints (1 - 3) and (7) enforced at discretized values $s_1, ..., s_N$ along the s coordinate. Note that significant speed gains can be achieved by computing the coefficients $\frac{dq}{ds}(s)$, $\frac{d^2q}{ds^2}(s)$, $a(s)$, $b(s)$, $c(s)$ only once for all discretized values of s.

4.3 Hierarchical Optimization

We discretize constraints along the optimized trajectory by enforcing them at N collocation points ([13]). To solve this problem, a naïve method is to parameterize both the robot trajectory expressed in terms of $\dot{s}$, $\ddot{s}$ as in (11 - 12) and the vector of contact forces f along the motion, and then solve the following optimization problem:

$$\begin{aligned}
&\min_{\dot{S}, \mathbf{f}_1, ..., \mathbf{f}_m} f_{obj}(\dot{S}) \\
&\text{s.t. at all } s = s_k, \text{ for k=1}, \ldots, N \\
&\quad \frac{dq}{ds}\dot{s} \in [\dot{q}_{min}, \dot{q}_{max}] \\
&\quad \frac{d^2q}{ds^2}\dot{s}^2 + \frac{dq}{ds}\ddot{s} + [\ddot{q}_{low}, \ddot{q}_{upp}] \in [\ddot{q}_{min}, \ddot{q}_{max}] \\
&\quad a(s)\ddot{s} + b(s)\dot{s}^2 + c(s) + M(q)[\ddot{q}_{low}, \ddot{q}_{upp}] \\
&\qquad - \sum_i^m J_i(q)^T f_{k,i} \in [\tau_{min}, \tau_{max}] \\
&m_{obj}\alpha_L(\dot{s}, \ddot{s}) = \sum_i^m f_{k,i} + m_{obj}g \\
&I_{obj}\alpha_A(\dot{s}, \ddot{s}) = \sum_i^m (p_i - c) \times f_{i,j} \\
&A_i f_{k,i} \leq 0 \text{ for } i = 1, ..., m
\end{aligned} \tag{14}$$

where we express α_L and α_A as functions of $\dot{s}$ and $\ddot{s}$ via replacing (11) and (12) into (8).

In this formulation, each $\mathbf{f}_k$ is the stacked vector of contact forces $f_{k,1}, \ldots, f_{k,m}$ at collocation point k, containing $3m$ entries in total. As a result, this naïve formulation has $(1+3m)N$ parameters and $3nN$ constraints. This could be extremely expensive at high values of m and N.

We present an alternate hierarchical method that decomposes this large problem into an inner problem embedded into an outer problem, both of which are simpler than the formulation above. It uses the fact that for each value of s along the motion, contact forces may be chosen independently of one another. So, at each constraint evaluation point, a solution to (3), (6), (13) and (7) can be solved independently via a relatively small linear program to verify whether the given values of $\dot{s}$ and $\ddot{s}$ are valid:

$$\begin{aligned}
&\text{Given } \dot{s} \text{ and } \ddot{s}, \text{ find } f = (f_1, ..., f_m) \text{ such that} \\
&\sum_i J_i(q)^T f_i \leq (a(s)\ddot{s} + b(s)\dot{s}^2 + c(s)) + e_{low} - \tau_{min} \\
&-\sum_i J_i(q)^T f_i \leq \tau_{max} - (a(s)\ddot{s} + b(s)\dot{s}^2 + c(s)) - e_{upp} \\
&m_{obj}\alpha_L(\dot{s}, \ddot{s}) = \sum_i^m f_i + m_{obj} g \\
&I_{obj}\alpha_A(\dot{s}, \ddot{s}) = \sum_i^m (p_i - c) \times f_i \\
&A_i f_i \leq 0 \text{ for } i = 1, ..., m
\end{aligned} \tag{15}$$

where $[e_{low}, e_{upp}]$ is the interval resulting from the product $M(q)[\ddot{q}_{low}, \ddot{q}_{upp}]$.

The outer optimization problem becomes

$$\begin{aligned}
&\text{Find } \dot{S}^* = \underset{\dot{S}}{\text{argmin}}\ f_{obj}(\dot{S}) \text{ such that} \\
&for\ k = 1, ..., N \\
&\dot{q}_{min} \leq (11) \leq \dot{q}_{max} \\
&\ddot{q}_{min} \leq (12) \leq \ddot{q}_{max} \\
&(15) \text{ is satisfied.}
\end{aligned} \tag{16}$$

This approach reduces the number of outer optimization variables to N and each iteration of the outer optimization requires solving N independent inner LPs (15). Each LP requires solving for relatively few variables ($3m$) and can be solved quickly. Overall, the hierarchical optimization is much faster than the naïve method, even for large number of contact points.

But this approach has a problem that, as formulated, the LP constraint is a black box binary test, which cannot be handled easily in numerical optimization routines that require the constraints to be differentiable with respect to $\dot{S}$. We use a formulation for turning this constraint into a piecewise linear, numerical form.

Let us rewrite (15) in a standard LP form as follows:

$$\begin{aligned}
&\text{Given } \dot{s} \text{ and } \ddot{s}, \text{ find } f = (f_1, ..., f_m) \text{ such that} \\
&Wf \leq z(\dot{s}, \ddot{s})
\end{aligned} \tag{17}$$

where W includes the linearized friction cone constraints A_i and the Jacobian $J_i(q)^T$ which transforms the contact force into joint torques. $z(\dot{s}, \ddot{s})$ corresponds to the RHS of the inequalities in (15). By introducing an auxiliary variable v, we can transform this into a minimization problem whose solution is negative if and only if there is a solution to (15). We define the optimization variable $x = (f_1, ..., f_m, v)^T$ and formulate the objective function as Cx where $C = (0, ..., 0, 1)$. The linear program becomes

$$\begin{aligned}
&\text{Solve } v^*(\dot{s}, \ddot{s}) = \min_x Cx \text{ such that} \\
&Wf - z(\dot{s}, \ddot{s}) \leq v\bar{1}
\end{aligned} \tag{18}$$

where $\bar{1}$ is the vector of all 1s. The optimal value can be determined using a standard LP algorithm. We now replace the constraint at the outer level optimization with the continuous, piecewise-linear constraint $v^*(\dot{s}, \ddot{s}) \leq 0$. We can determine the subderivatives $\frac{\partial v^*}{\partial \dot{s}}$ and $\frac{\partial v^*}{\partial \ddot{s}}$ analytically through Karush Kuhn Tucker (KKT) multipliers of the LP (see Appendix). This calculation assumes a unique active set at the optimum, which holds almost everywhere. Using this formulation we are able to solve the outer optimization using standard nonlinear programming (NLP) techniques like sequential quadratic programming (SQP).

4.4 Initial Guess

Although it has been proven that the minimum-time time-scaling problem as formulated is convex [33, 16], it is useful to obtain a good initial solution to speed up the outer optimization. To do so we maximize the velocities $\dot{S}$ of the time scaling function pointwise, so we compute the maximum possible $\dot{s}^0$ as our initial guess according to the constraint $\dot{q}_{min} \leq \dot{q} = \frac{dq}{ds}\dot{s} \leq \dot{q}_{max}$. Since the bounds $(\frac{dq}{ds})_{min}$ and $(\frac{dq}{ds})_{max}$ can be computed explicitly from the input path, we get $[(\frac{dq}{ds})_{min}, (\frac{dq}{ds})_{max}]\dot{s} \in [\dot{q}_{min}, \dot{q}_{max}]$, then $\dot{s}^0$ can be approximated as

$$\dot{s}^0 = \min(\min_i \left(\left| \frac{q^i_{max}}{(\frac{dq^i}{ds})_{max}} \right| \right), \min_i \left(\left| \frac{q^i_{min}}{(\frac{dq^i}{ds})_{min}} \right| \right)) \tag{19}$$

where $i = 1, ..., n$ iterates through each joint of the robot.

4.5 Multiple Contact Phases

In the presentation above, we assumed only one contact phase (no contacts are made or broken). But our method can handle situations with $N_p > 1$ contact phases with a small modification. Each contact phase is distinguished by the set of contact points F_p and friction coefficients, leading to different constraints of (13) and (6). We model the time-scaling as an optimization over k piecewise quadratic function as in section 4, so the objective function in (16) becomes

$$f_{obj}(\dot{S}) = \sum_{p=1}^{N_p} \sum_{i=0}^{k} \frac{\Delta s_p}{\dot{s}_{p,i} + \dot{s}_{p,i+1}} \tag{20}$$

where Δs_p is the uniform interval in p^{th} contact phase.

Since feasibility at a single collocation point only depends on the two optimization variables $\dot{s}_i$ and $\dot{s}_{i+1}$ of appropriate contact phase, no changes need to be made for the constraints formulated in (16).

5 Iterative Learning Uncertainties

The second stage of our method is an iterative learning technique for improving confidence intervals given execution feedback. A conservative estimate of friction and execution errors could overly constrain the problem and generate a very slow motion, while an optimistic estimate could cause execution failure. To approach the "sweet spot" we present two methods. The first method uses binary success/failure feedback to perform a bisection search on a chosen modeling parameter, (e.g., friction). Bisection continues until a desired convergence threshold is reached. This approach is most suitable for unobservable parameters like friction. The second method incorporates feedback on execution velocity and acceleration to measure confidence intervals more accurately.

5.1 Bisection Search on Friction Coefficient

Ideally, the optimized trajectory should move the blocks as fast as possible without tipping them over, but this is difficult to achieve due to modeling uncertainties. Instead, what we can achieve is to produce an trajectory that is ϵ-near the time-optimal trajectory, where ϵ-*near* implies that the generated trajectory can be executed successfully, while executing the trajectory sped up by a factor $1+\epsilon$ fails. Here ϵ controls how close the generated motion approaches the true (unobservable) feasibility boundary of the fastest possible motion. A smaller ϵ indicates faster executions with lower margin of error, at the expense of more learning iterations.

Beginning from the initial confidence interval on the parameter of interest, a bisection search is conducted to find a lower bound that is nearly optimal according to the speed-up parameter ϵ. We apply this to the coefficient of friction (COF). In manipulation it controls the perpendicular magnitudes of the contact forces and therefore affects the maximum accelerations applied on the object. It is also a difficult parameter to estimate accurately without specialized sensors.

The bisection search method (see Alg. 1) works by setting upper and lower bounds, μ_{upp} and μ_{low}, on the COF value μ and optimizing the problem formulated in (16) with μ. With the assumption that the execution is relatively accurate or the error is negligible, we set the acceleration confidence interval to be empty, namely $\ddot{q}_{low} = \ddot{q}_{upp} = 0$. The optimized motion is executed, and if it fails (blocks tip over or shift largely beyond a predefined threshold), then we update μ_{upp} with the current COF value μ and repeat. Otherwise, we update the lower bound on μ. We then execute a sped-up version of this trajectory. If the sped-up execution fails, then we have converged successfully to a ϵ-near optimal trajectory. We can also terminate if the difference between upper and lower bounds becomes less than a given tolerance δ ($\delta = 0$ is an acceptable value).

Algorithm 1 Bisection Search for COF

```
μ_low ← 0, μ_upp ← μ_init
while μ_upp − μ_low > δ do
    μ ← (μ_low + μ_upp)/2
    traj ← RobustOptimize(μ)
    if Execute(traj)= Success then
        traj′ ←Speedup(traj, 1 + ϵ )
        if Execute(traj′) ≠ Success then return μ_low
        else
            μ_low ← μ
    else
        μ_upp ← μ
return μ_low
```

5.2 Iterative Learning with Robust Optimization and Binary Search

To consider errors on both parameter estimation and execution, we modify the binary search method for the modeling parameter in Sec. 5.1 by adding an inner loop that learns and bounds execution disturbances via trajectory feedback.

To some extent, naïve introduction of measured confidence intervals into optimization can compensate for these disturbances. However, we have observed that many disturbances are trajectory-dependent (e.g., speed-dependent control errors, air resistance, etc.) Again we find that conservative estimates of errors e with a large standard deviation Δ lead to slower optimized trajectories. Hence, our method learns characteristic disturbances during the iterative learning phase.

Our approach optimizes the trajectory with confidence interval $[\ddot{q}_{low}, \ddot{q}_{upp}]$ from the last iteration, and update this confidence interval from the current execution feedback. This process repeats until the confidence interval becomes stable. Here stability is judged if the differences between each endpoint of the confidence interval for the last iteration and the current one become falls below a threshold. To estimate a per-trajectory confidence interval, we assume that for each time step the disturbance arises from a Gaussian noise model $N(U, \Delta^2)$.

Our algorithm Iterative Learning with Robust Optimization and Binary Search (ILROBS) is shown in Alg. 2. The input K (used in Eq. (9)) is a user-specified parameter indicating the desired number of standard deviations of disturbances to which the model should be robust. In other words, the expected likelihood of success $\Phi(K)$ where Φ is the cumulative distribution of the standard normal distribution; and so K should be chosen according to the *68 - 95 - 99.7 Rule* [6]. After each execution the algorithm computes the noise model $N(U, \Delta^2)$ from $\ddot{q}_{exc} - \ddot{q}_{plan}$. These are then used to determine the acceleration confidence interval according to (9) for the given K.

The challenge in this algorithm is to guess whether a given execution failed due to an overestimated friction coefficient or an unlucky disturbance (or contrariwise, succeeded due to random luck). In the former, the COF estimate should be lowered, and in the latter, further execution feedback may be needed to accurately estimate the probability of success. If we have observed that a given trajectory has succeeded all of $N_e(K) = \lceil 1/(1-\Phi(K)) \rceil$ times, then we believe the success rate is at least $\Phi(K)$. So, if any one of them fails, we bisect.

Algorithm 2 ILROBS (K)

$\mu_{low} \leftarrow 0$, $\mu_{upp} \leftarrow \mu_{init}$
$U \leftarrow 0$, $\Delta \leftarrow 0$
while $\mu_{upp} - \mu_{low} > \delta$ **do**
  $\mu \leftarrow (\mu_{low} + \mu_{upp})/2$
  Set $\ddot{q}_{low} \leftarrow U - K \cdot \Delta$ and $\ddot{q}_{upp} \leftarrow U + K \cdot \Delta$.
  traj $\leftarrow$ RobustOptimize(μ)
  for $i = 1, \ldots, N_e(K)$ **do**
    result $\leftarrow$ Execute(traj)
    if result $\neq$ *Success* **then**
      $\mu_{upp} \leftarrow \mu$
      Estimate U and Δ from executions.
      Return to outer loop.
  Estimate U and Δ from executions
  $\mu_{low} \leftarrow \mu$
return (μ_{low}, U, Δ)

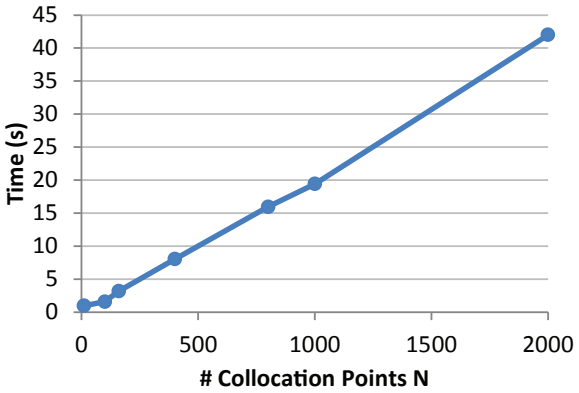

Fig. 3: Computation time versus number of time-domain collocation points for the two-block waiter task.

6 Experiments

We first test how the number of collocation points affects the computation time. And we also conduct two experiments corresponding to the two situations introduced in Sec. 5. SNOPT [10] was used for solving the outer optimization problem and the GNU GLPK library [11] was used for the inner linear program. The computation was carried on a laptop with 2.9GHz processor and the time for trajectory optimization for all the physical robot experiments are within 10 seconds.

6.1 Computation Time vs Number of Collocation Points

The number of constraint checking points decides how many constraints must be evaluated and therefore affects the speed of optimization. We test the change of computation time in terms of the number of collocation points for the example of moving a stack of two blocks (Fig. 1, right). With 8 contacts points for the stack of two blocks, the computation time is shown in Fig. 3 for different number of collocation points. To keep computation times below 10 s per iteration, 200 collocation points are used throughout the following experiments.

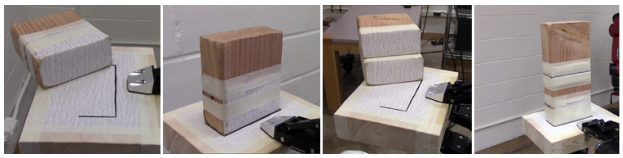

Fig. 4: The contact surfaces are wrapped with sand paper. One block and a stack of two blocks are placed on the plate shown on 2nd and 4th picture respectively.

Iter	μ	Time(s)	Rslt	Spdup	μ_{upp}	μ_{low}
1	1	3.229	F		1	0
2	0.5	3.232	F		0.5	0
3	0.25	3.704	F		0.25	0
4	0.125	5.237	S	S	0.25	0.125
5	0.1875	4.276	S	F	0.25	0.1875

Table 1: Binary search on COF according to Alg. 1 for the example of moving a stack of two blocks. Time column indicates execution time of optimized path. With $\epsilon = 0.05$, the COF converges on an ϵ-near optimal trajectory with $\mu = 0.1875$ as in the highlighted row (i.e., speeding up the motion by 5% tips over the blocks).

6.2 Binary Search COF with Hardware Execution

To study the effect of friction uncertainty, the Stäubli TX90L industrial manipulator is used. A RobotiQ hand is installed on the manipulator to hold a plate with blocks on it. We wrap the contacting surfaces between blocks and plate with sand paper (Fig. 4). The COF is roughly estimated as $\mu = 1$ by tilting the plate with the block on it and checking the inclination at which the block starts to slide.

First, the optimized trajectory with $\mu = 1$ was too fast during execution on the physical robot, causing the objects to slide in the one-block case and to wobble and fall over in the two-block case. Next, we applied the binary search method on COF to compensate for the un-modeled uncertainties which caused execution failure. Table 1 lists the binary search parameters for the two-block example. After five iterations it yields an ϵ-near optimal value for COF and Fig. 5 shows the snapshots of the final motion execution.

We note that the converged parameter value $\mu = 0.1875$ is less than a fifth of the empirically determined value of $\mu = 1$. This is because the COF parameter acts as a proxy for all other un-modeled uncertainties, such as low-level controller errors and estimation errors in center-of-mass, contact points, etc. It is important to note that, the converged execution time of 4.276 is not far from the optimistic time of 3.229. This is because the optimization slows down the trajectory *only in the portions for which friction is the limiting constraint.*

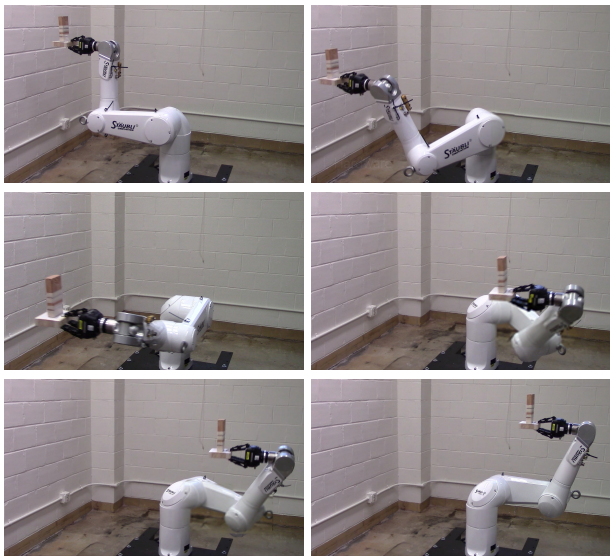

Fig. 5: Snapshots of executing the motion generated from Alg. 1 for moving a two-block stack.

6.3 Iterative Learning with Robust Optimization in Simulation

In the second experiment, we introduce random disturbances during execution, and incorporate trajectory feedback into optimization as described in Sec. 5.2. Because it is difficult to inject random disturbances to a real robot in the lab, we use a simulator for these examples. A rigid body simulator based on Open Dynamics Engine using a PID controller with feedforward gravity compensation torques [15] is used to track the planned trajectory $(t, q, \dot{q})$.

This example introduces both random disturbances and errors on model parameters. The optimization's initial COF estimate is 1, while it is set to 0.5 in simulation. The uncertainties in trajectory execution are simulated by introducing random forces on the robot. A random horizontal force $F_{disturb} = (x, y, 0)$ with x and y subject to a Gaussian distribution $N(0, 2)$ (in Newtons) is added to the tray at each simulation step (see Fig. 6).

The results of each outer loop of Alg. 2 are listed in Table 2. Here we run the algorithm with $K = 1$ (corresponding to a success rate of 68%). Success ratios are estimated via 100 Monte Carlo trials. After 6 iterations, we learned the COF and a confidence interval of acceleration error that achieves a success rate of 74%.

7 Extension to Multi-Parameter Learning

We now present a generalization of the iterative bisection method to handle multiple uncertain parameters.

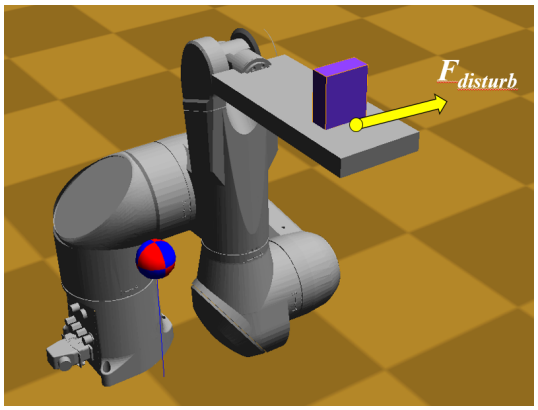

Fig. 6: In simulation tests, random horizontal forces with 2 N std. dev. are introduced on the tray at each simulation step.

Iter	μ	Time(s)	Ratio(%)	μ_{upp}	μ_{low}
1	1	1.88	0	1	0
2	0.5	2.20	0	0.5	0
3	0.25	3.09	100	0.5	0.25
4	0.375	2.52	98	0.5	0.375
5	0.4375	2.30	74	0.5	0.4375
6	0.46875	2.28	16	0.46875	0.4375

Table 2: Outer iterations of Alg. 2 for the one-block waiter task in simulation. Time column indicates execution time of optimized path. With $K = 1$, $\delta = 0.05$, the algorithm converges in 6 iterations.

For example, both the center of mass and the friction of a held object may be unknown, or a walking robot with multiple limbs in contact may have uncertain friction for each limb. We assume each of these parameters is not directly observable; observable uncertainties such as acceleration errors can simply be bounded as demonstrated above. Instead, we only observe a boolean feedback about whether the execution of the robustly optimized trajectory succeeds or not. Like in the case of friction, we make an *ordered uncertainty* assumption in that execution conservativeness is monotonic in each uncertain parameter value.

To estimate a single parameter, boolean feedback is sufficient to narrow down the confidence interval via bisection. But for multiple parameters, it is unclear from the outset whether it is possible to correctly identify the cause of an execution error. For example, one parameter may be set optimistically and another pessimistically, and the algorithm should use the failure/success signal to adjust the confidence intervals of both parameters correctly.

We present three multi-parameter learning algorithms based on a branch-and-bound search, with increasingly sophisticated heuristics to learn with fewer execution examples. To perform each optimization step

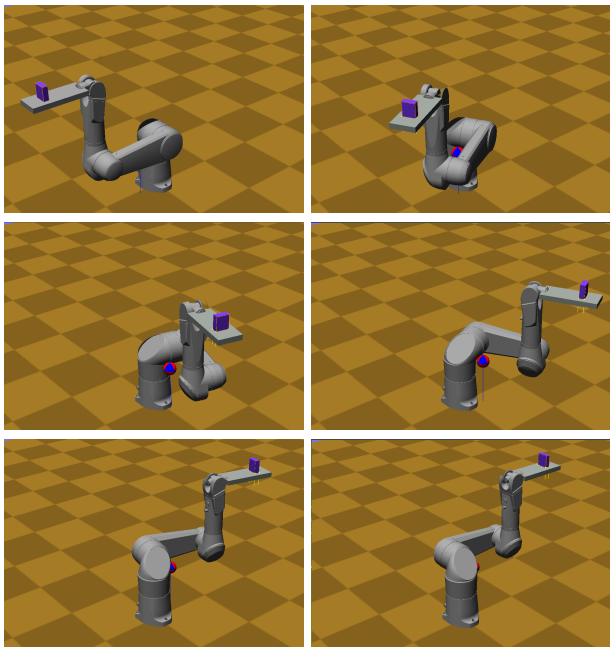

Fig. 7: Snapshots of moving a block under disturbances in simulation.

we use the robust time-scaling method as presented in Section 4.2.

7.1 Problem Formulation

We formalize the multi-parameter, ordered uncertainty learning problem as follows. Let $\theta = (\theta_1, \ldots, \theta_k)$ be an uncertain parameter, and let θ_{act} be the actual setting of these parameters. Let $P(\theta)$ be the optimization problem solved under parameter estimates θ. We assume that we have a initial confidence hyper-rectangle $\mathcal{H} = [a_1, b_1] \times \cdots \times [a_k, b_k]$ containing θ_{act}, and that the problem family satisfies some monotonicity conditions:

1. If $\theta \leq \theta'$ element-wise, then the optimized cost for estimates θ is lower than that of θ'.
2. If $\theta' \geq \theta$ element-wise, then a successful execution of θ implies successful execution of θ'.
3. There exists a feasible solution to the most conservative optimization $P(b_1, ..., b_k)$, and the execution of this solution is successful.

Informally, these say that optimized trajectories for parameter estimates closer to $(a_1, \ldots, a_k)$ are less conservative and those closer to $(b_1, ..., b_k)$ are more conservative.

More precisely, condition 1 can be stated as $f^\star(\theta) \leq f^\star(\theta')$ for all $\theta \leq \theta'$ where $f^\star$ is the optimized cost given some parameter estimate. Condition 2 can be stated that $g(\theta) \implies g(\theta')$ for all $\theta' \geq \theta$ where g is a predicate

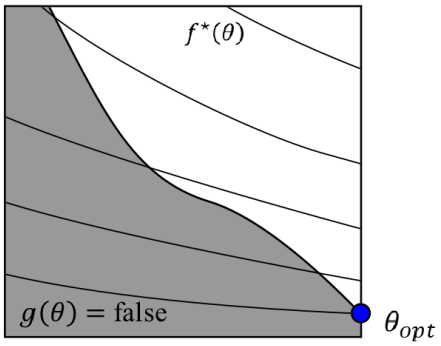

Fig. 8: Illustrating the multi-parameter learning problem. The search space is the multi-parameter hypercube, and the shaded region is the subset for which the optimized solution leads to a failed execution, $g(\theta) =$ false. We wish to minimize the optimal cost function $f^\star(\theta)$ (contour lines shown) such that the optimized solution is executable.

giving the execution feasibility of the path optimized given some parameter estimate. Condition 3 states that $f^\star(b_1, \ldots, b_k) < \infty$ and $g(b_1, \ldots, b_k) =$ true.

Formally, we wish to develop a learning procedure that converges toward the optimal value:

$$\begin{aligned} \theta_{opt} &= \arg\min_{\theta \in \mathcal{H}} f^\star(\theta) \\ &\text{such that} \\ g(\theta) &= \text{true}. \end{aligned} \tag{21}$$

It is also beneficial to minimize the number of evaluations of g because each evaluation requires the robot to execute a trajectory.

Note that if the optimization model is a correct model of the physical world, then $\theta_{opt} = \theta_{act}$. In general, however, the model may differ and hence they may be unequal.

Without loss of generality let us assume the domain is a unit hypercube; arbitrary hyper-rectangles can be converted to the hypercube through scaling and translation. A visualization of a 2-parameter learning problem is given in Fig. 8. The contour plots depict level-sets of optimal cost surface $f^\star(\theta)$, while the shaded area shows the set of infeasible executions. Note that both are monotonic in each parameter according to the above assumptions.

7.2 Basic Branch-and-Bound Method

The branch-and-bound search method proceeds constructs a sequence of increasingly fine boxes $\mathcal{H}_0 = \mathcal{H}$, $\mathcal{H}_1, \mathcal{H}_2, \ldots$ such that the maximal (most conservative)

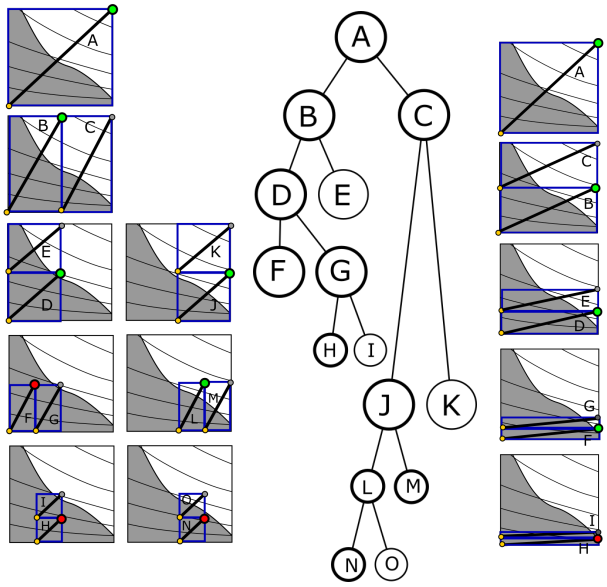

Fig. 9: (a) Search nodes in the proposed (basic) branch-and-bound method are hyper-rectangles. Parameter vectors for which optimization is performed are indicated in yellow, those that are successfully executed are drawn in green, and those for which execution fails are drawn in red. Highlighted circles indicate expanded nodes. The round-robin dimension selection technique is shown here. (b) A search tree for the basic method, with expanded nodes highlighted. (c) The greedy dimension selection method yields fewer search nodes.

point of each box is feasible and converges toward θ_{opt}. The method provided here is different from standard branch-and-bound techniques because it solves a constrained optimization problem where the constraint is expressed in "black box" form, i.e., provides only boolean "in" or "out" feedback. It also uses the monotonicity assumptions of the problem to quickly determine bounds.

The basic method is illustrated in Fig 9. It maintains a search tree of boxes, and at each iteration, the following steps are performed:

1. A box $\mathcal{H}_k$ is chosen from the search front for expansion.
2. The solution $x^\star(\theta_{max})$ at its maximal point θ_{max} is computed, executed and tested for feasibility (in other words, we evaluate $g(\theta_{max})$). If it fails, we continue from Step 1.
3. Subdivide the box along some dimension, and add child nodes to the search tree. A basic method for selecting the dimension uses a round-robin method (Fig. 9(a,b)).

We order boxes by increasing cost of the optimized solution at their minimal points $f^\star(\theta_{min})$. The algorithm

is run until no nodes have minimum cost greater than ϵ plus the cost of best parameter setting found so far.

The most relevant complexity measure of such an algorithm is the number of optimization invocations and the number of execution trials. Optimization is run at most twice per generated node, while execution is run at most once per expanded node.

Suppose the finest resolution of leaf boxes is ϵ_θ. Then, the final depth of the search tree is $k \log_2 \epsilon_\theta$. In typical cases, the search proceeds essentially in greedy fashion toward the optimum. Thus, the algorithm performs $O(k \log_2 \epsilon_\theta)$ executions and optimizations.

An alternative search strategy is a *greedy method* that selects the splitting dimension adaptively by running k different optimizations and picking the split that leads to the greatest decrease in the cost function. This has the effect of reducing the function value faster than round robin. However, for each search node it runs k optimizations. This multiplicative factor of k is usually worthwhile because optimizations are often faster than repeating a physical experiment. The result is illustrated in Fig. 9(c).

However for both the basic and greedy algorithms, the worst case complexity is exponential in atypical cases where the optimal level set aligns with the constraint boundary. In these cases, the algorithm may explore an exponential number of nodes, $O((1/\epsilon_\theta)^{k-1})$, because the constraint boundary has dimensionality $k-1$ and the algorithm will cover the surface with leaf nodes.

7.3 Informed Multi-Parameter Learning

When the number of uncertain parameters k is large, even greedy dimension selection may require an excessive number of optimizations and executions to get close to an optimal solution. This section presents a method that uses the structure of the optimization problem to better inform the learning sequence. In particular, it efficiently identifies the (hopefully small) subset of relevant parameters based on the optimization feasibility constraints, and suggests informative parameter settings based on that subset. This strategy improves convergence rate beyond bisection, and in many cases, the method can achieve convergence within $O(\log_2 \epsilon)$ execution trials.

The basic principle behind the method is as follows: if an uncertain parameter does not affect a constraint that is active at the optimal solution of the underlying optimization problem, then changing that parameter (by a small amount) will not affect the optimal solution, and hence the executed path will be the same as before.

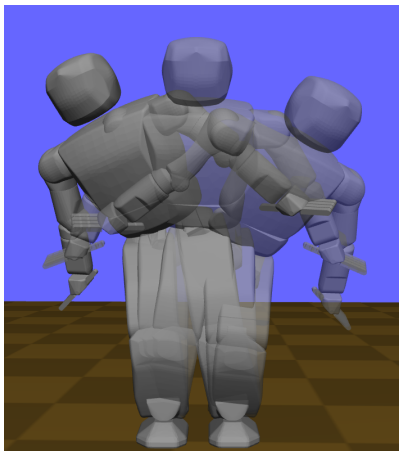

Fig. 10: The humanoid robot path taken in the multi-parameter learning example.

Based on this principle, we reduce the branching factor of the search by splitting on only the dimensions of uncertainty that affect the active set. Non-active dimensions are irrelevant and do not need exploring. To do so, after a trajectory $x = x^\star(\theta_{max})$ corresponding to the maximum point on a given hypercube $\mathcal{H}_i$ is successfully executed, we consider which subset of $\mathcal{H}_i$ would cause x to become infeasible. Since each uncertain parameter affects some set of optimization constraints, the sub-hypercube $\mathcal{H}_{infeas} \subseteq \mathcal{H}_i$ that may create infeasibility can be easily determined by sweeping the each parameter along its range in $\mathcal{H}_i$ until it causes x to become infeasible. For irrelevant parameters, this range will be empty. For constraints on the active set, this range will be the entire interval spanned by $\mathcal{H}_i$. We then shrink the node's bounding box to $\mathcal{H}_{infeas}$ and then choose a dimension to split as chosen by the greedy strategy.

7.4 Simulation Experiments

We tested the above method on a 4-parameter learning problem for humanoid robot moving under the zero moment point (ZMP) constraint. A geometric path, illustrated in Fig. 10, is generated for the Hubo+ robot to transfer weight from one foot to the other. The center of mass starts in the center of the support polygon of the robot's right foot and is transferred 1.5 cm away from the edge of the robot's left foot. Here, a physics simulation stands in from the real world.

The ZMP condition is a widely used balance criterion for dynamic bipedal walking on flat ground. The zero moment point is the point on the plane at which

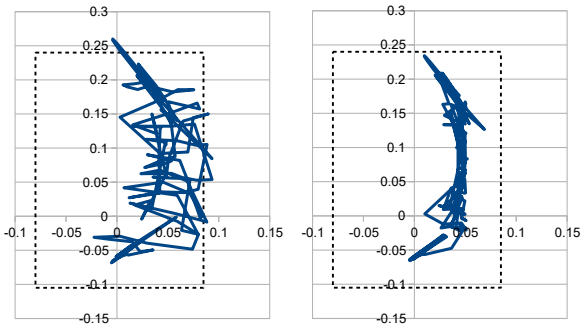

Fig. 11: Left: with the originally estimated support rectangle, the optimized ZMP path leaves the true support polygon during execution. Right: with a shrunken (more conservative) rectangle, the ZMP path leaves the planned rectangle but stays within the true support polygon.

the torque about the contact wrench is zero:

$$\begin{aligned} ZMP_x &= c_x - \ddot{c}_x(c_z - h_g)/g \\ ZMP_y &= c_y - \ddot{c}_y(c_z - h_g)/g \end{aligned} \tag{22}$$

where c is the center of mass position, h_g is the height of the ground plane, and g is the gravitational acceleration. This equation assumes that the center of mass moves horizontally, i.e., the height c_z is held constant. The ZMP criterion states that the ZMP must stay within the support polygon throughout the trajectory.

The learner has several sources of error:

- It is given as input a support rectangle that over-approximates the "true" support polygon. (We put "true" in quotes because the ZMP-in-support polygon criterion is an oversimplification of the simulation engine's contact physics).
- It assumes perfect execution of the path with no actuation errors, while the simulation emulates servomotors with PID gains and joint friction.
- It performs dynamic constraint checking at 100 collocation points, while the simulation time step is 1 kHz.

The four uncertain parameters are the four margins of the support rectangle's edges, and are given error ranges of up to 4 cm (Fig 11). Here the x axis indicates the robot's forward direction. With the most optimistic estimate (no error) the ZMP trajectory leaves the true support polygon, and with the most pessimistic estimate, the ZMP trajectory stays inside, but takes 2.41 s to execute. Using the proposed informed approach, a better solution was learned after 61 optimizations and 10 executions (Fig. 12). The resulting margins, listed in ccw order from left, are 4 cm, 4 cm, 2.125 cm, and 3.75 cm. Note that the method ignores learning the left and bottom edges because they are irrelevant to the feasibility of the executed path. The learned path execution time is 1.86 s.

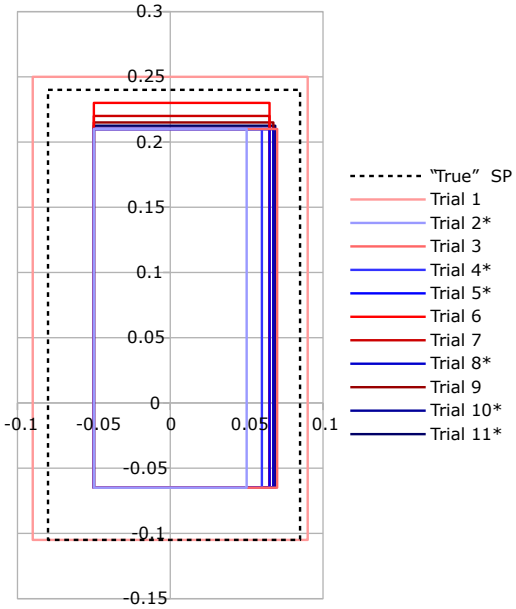

Fig. 12: Learning the support polygon. Note that only the right and top margins are adjusted after the first feasible execution. Iterations marked with * indicate successful executions.

8 Conclusion

This paper presented a fast time-scaling optimization method for generating time-optimal, dynamically feasible trajectory along a given geometric path for a robot in contact. By carefully formulating the problem as a non-linear optimization with the contact constraints as a linear program, we reduced the problem size and are able to solve the problem quickly (in seconds).

We apply this method to manipulation and locomotion problems under contact, and we present algorithms to handle the modeling and execution uncertainties by incorporating execution feedback. Our experiments demonstrate that the combination of optimization and uncertainty learning can generate motions that i) are near-optimal in the presence of modeling errors; and ii) can be executed multiple times with a success rate that depends on a user specified parameter.

There is substantial room for future work. First, we hope to consider other sources of error, such as inertial parameters of objects. Second, we consider all modeling parameters to be decoupled, and future work should

address coupled effects, such as uncertainty in the L2-norm of a parameter vector. Another source of information that could be used is the time of a detected failure, which could help identify which constraint errors are likely to have caused the failure, and hence may allow the method to learn more quickly. We hope to extend our method to optimize trajectory shapes in addition to timing. Finally, a weakness of our method is that it requires that a trajectory be executed several times from the same initial conditions with multiple failures, whereas for many applications failures are costly and restarting is tedious. Producing an on-line learning approach that avoids execution failures with high probability would be an interesting extension.

Acknowledgment

This work is partially supported under NSF grant IIS # 1218534 and CAREER # 3332066.

Appendix

Here we derive the formula for the derivative of the objective function of an inequality-constrained LP with respect to changes in the constraint RHS. Consider the LP

$$\min_x c^T x \text{ such that } Ax - b(t) \leq 0. \tag{23}$$

By the first-order KKT conditions, the optimal solution satisfies

$$\begin{aligned} c + A^T\mu &= 0 \\ Ax^* - b &\leq 0 \\ \mu^T(Ax^* - b) &= 0 \\ \mu &\leq 0 \end{aligned} \tag{24}$$

where μ is the vector of KKT multipliers (unrelated to the friction coefficient) and x^* is the optimal solution. If the LP is bounded, x^* lies at a corner point of the feasible region, and is defined by active constraints with $A_i x^* - b_i = 0$ and $\mu_i < 0$. Denote $\hat{A}$ and $\hat{b}$ respectively as the matrix/vector containing the rows of A and b corresponding to active constraints, and let $\hat{\mu}$ be the set of active multipliers. Then the equations

$$\begin{aligned} c + \hat{A}^T\hat{\mu} &= 0 \\ \hat{A}x^* - \hat{b} &= 0 \end{aligned} \tag{25}$$

must be satisfied, with $\hat{A}$ an invertible matrix. Hence, $x^*(t) = \hat{A}^{-1}\hat{b}(t)$, and $\frac{d}{dt}(c^T x^*) = c^T\hat{A}^{-1}\hat{b}'(t)$. Since $\hat{\mu}^T = -c^T\hat{A}^{-1}$, we obtain

$$\frac{d}{dt}(c^T x^*) = -\hat{\mu}^T\hat{b}'(t) = -\mu^T b'(t) \tag{26}$$

as desired.

References

1. Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
2. Dimitris Bertsimas and Aurélie Thiele. Robust and data-driven optimization: Modern decision-making under uncertainty. *INFORMS Tutorials in Operations Research: Models, Methods, and Applications for Innovative Decision Making*, 2006.
3. John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.
4. J. E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.
5. Douglas A Bristow, Marina Tharayil, and Andrew G Alleyne. A survey of iterative learning control. *Control Systems, IEEE*, 26(3):96–114, 2006.
6. George W Cobb, Jeffrey A Witmer, and Jonathan D Cryer. *An Electronic Companion to Statistics*. Cogito Learning Media New York, 1997.
7. D. Constantinescu and E. A. Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *J. of Robotic Systems*, 17:223–249, 2000.
8. O. Dahl and L. Nielsen. Torque limited path following by on-line trajectory time scaling. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1122 –1128 vol.2, May 1989. doi: 10.1109/ROBOT.1989.100131.
9. Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec, and Sylvain Garsault. Planning support contact-points for acyclic motions and experiments on hrp-2. In *Experimental Robotics*, pages 293–302. Springer, 2009.
10. Philip E. Gill, Walter Murray, Michael, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization, 1997.
11. GNU. Gnu linear programming kit (glpk). URL `http://www.gnu.org/software/glpk/glpk.html`. (accessed April 16, 2015).
12. K. Harada, K. Hauser, T. Bretl, and J.-C. Latombe. Natural motion generation for humanoid robots. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.
13. Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

14. Kris Hauser. Fast interpolation and time-optimization on implicit contact submanifolds. In *Robotics: Science and Systems*, 2013.
15. Kris Hauser. Robust contact generation for robot simulation with unstructured meshes. In *International Symposium on Robotics Research, Singapore*, 2013.
16. Kris Hauser. Fast interpolation and time-optimization with contact. *The International Journal of Robotics Research*, 33(9):1231–1250, 2014.
17. T. Kunz and M Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Robotics: Science and Systems*, July 2012.
18. S. Lengagne, N. Ramdani, and P. Fraisse. Planning and fast replanning safe motions for humanoid robots. *IEEE Transactions on Robotics*, 27(6): 1095–1106, Dec 2011. ISSN 1552-3098. doi: 10.1109/TRO.2011.2162998.
19. Puttichai Lertkultanon and Quang-Cuong Pham. Dynamic non-prehensile object transportation. In *Int. Conf. on Control Automation Robotics Vision (ICARCV)*, pages 1392–1397, Dec 2014.
20. Thomas Lipp and Stephen Boyd. Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6):1297–1311, 2014. doi: 10.1080/00207179.2013.875224. URL `http://dx.doi.org/10.1080/00207179.2013.875224`.
21. C Karen Liu. Dextrous manipulation from a grasping pose. *ACM Transactions on Graphics (TOG)*, 28(3):59, 2009.
22. Jingru Luo and K. Hauser. Interactive generation of dynamically feasible robot trajectories from sketches using temporal mimicking. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3665 –3670, may 2012.
23. Jingru Luo and Kris Hauser. Robust trajectory optimization under frictional contact with iterative learning. In *Robotics: Science and Systems*, Jun 2015.
24. Kevin M Lynch and Matthew T Mason. Dynamic underactuated nonprehensile manipulation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 2, pages 889–896. IEEE, 1996.
25. Igor Mordatch, Zoran Popović, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144. Eurographics Association, 2012.
26. Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12 (4):319–340, 2011.
27. Quang-Cuong Pham, Stéphane Caron, Puttichai Lertkultanon, and Yoshihiko Nakamura. Planning truly dynamic motions: Path-velocity decomposition revisited. *arXiv preprint arXiv:1411.4045*, 2014.
28. Michael Posa and Russ Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Workshop on the Algorithmic Foundations of Robotics*, 2012.
29. Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.
30. Stefan Schaal and Christopher G Atkeson. Learning control in robotics. *Robotics & Automation Magazine, IEEE*, 17(2):20–29, 2010.
31. Kang Shin and N. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Trans. on Automatic Control*, 30: 531–541, 1985. doi: 10.1109/TAC.1985.1104009.
32. J.-J.E. Slotine and H.S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Trans. on Robotics and Automation*, 5(1):118 –124, Feb 1989. ISSN 1042-296X. doi: 10.1109/70.88024.
33. D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Trans. Automatic Control*, 54(10): 2318 –2327, October 2009. ISSN 0018-9286. doi: 10.1109/TAC.2009.2028959.
34. Oskar von Stryk and Roland Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, 37(1):357–373, 1992.