Trajectory Optimization with Optimization-Based Dynamics

Taylor A. Howell[1], Simon Le Cleac'h[1], Sumeet Singh[2], Pete Florence[2], Zachary Manchester[3], Vikas Sindhwani[2]

***Abstract*—We present a framework for bi-level trajectory optimization in which a system's dynamics are encoded as the solution to a constrained optimization problem and smooth gradients of this lower-level problem are passed to an upper-level trajectory optimizer. This optimization-based dynamics representation enables constraint handling, additional variables, and non-smooth behavior to be abstracted away from the upper-level optimizer, and allows classical unconstrained optimizers to synthesize trajectories for more complex systems. We provide a path-following method for efficient evaluation of constrained dynamics and utilize the implicit-function theorem to compute smooth gradients of this representation. We demonstrate the framework by modeling systems from locomotion, aerospace, and manipulation domains including: acrobot with joint limits, cart-pole subject to Coulomb friction, Raibert hopper, rocket landing with thrust limits, and planar-push task with optimization-based dynamics and then optimize trajectories using iterative LQR.**

***Index Terms*—Motion and Path Planning, Optimization and Optimal Control, Dynamics**

I. Introduction

TRAJECTORY optimization is a powerful tool for synthesizing trajectories for nonlinear dynamical systems. Indirect methods, in particular, are able to efficiently find optimal solutions to this class of problem by returning dynamically feasible trajectories via rollouts and utilizing gradients of the system's dynamics.

Classically, indirect methods like iterative LQR (iLQR) [1] utilize *explicit* dynamics representations, which can be directly evaluated and differentiated in order to return gradients. In this work, we present a more general framework for *optimization-based* dynamics representations. This latter class enables partial elimination of trajectory-level constraints by absorbing them into the dynamics representation.

We formulate optimization-based dynamics as a constrained optimization problem and provide a path-following method for efficient evaluation of the dynamics at each time step. The implicit-function theorem is utilized to differentiate through this representation, and we exploit intermediate results from our path-following method to return useful, smooth gradients to an upper-level optimizer.

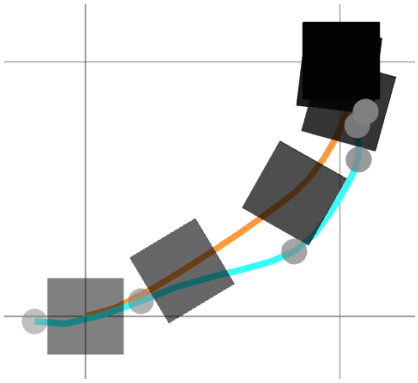

Fig. 1: Optimized trajectories for planar-push task. The pusher (blue) and block (orange) paths are shown for a plan that maneuvers the block from a pose at the origin to a goal pose.

To demonstrate the capabilities of this representation, we utilize optimization-based dynamics and iLQR in a bi-level optimization framework to perform trajectory optimization. We provide a number of optimization-based dynamics models and examples in simulation including: an acrobot with joint limits, a cart-pole experiencing friction, gait generation for a Raibert hopper, a belly-flop soft landing for a rocket subject to thrust limits, and a planar-push manipulation task. We compare our approach to MuJoCo, contact-implicit trajectory optimization, and gradients generated with randomized smoothing.

Specifically, our contributions are:

- A novel framework for optimization-based dynamics that can be used with trajectory-optimization methods that require gradients
- A path-following method that can efficiently evaluate constrained optimization problems and return smooth gradients
- A collection of optimization-based dynamics models and examples from locomotion, aerospace, and manipulation domains that demonstrate the proposed bi-level trajectory-optimization framework

In the remainder of this paper, we first review related work on bi-level approaches to trajectory optimization in Section II. Next, we present background on the iLQR algorithm, the

Manuscript received: September 9, 2021; Revised December 6, 2021; Accepted February 31, 2022.

This paper was recommended for publication by Editor Stephen J. Guy upon evaluation of the Associate Editor and Reviewers' comments.

*This work was supported by Google Research.

[1]Taylor A. Howell and Simon Le Cleac'h are with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305, USA {thowell,simonlc}@stanford.edu

[2] Sumeet Singh, Pete Florence, and Vikas Sindhwani are with Robotics at Google, New York City, NY 10011 and Mountain View, California 94043 USA {ssumeet,peteflorence,sindhwani}@google.com

[3]Zachary Manchester is with the The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA zacm@cmu.edu

Digital Object Identifier (DOI): see top of this page.

implicit-function theorem, and implicit integrators in Section III. Then, we present our optimization-based dynamics representation, as well as a path-following method for solving this problem class in Section IV. We provide a collection of optimization-based dynamics models that are utilized to perform trajectory optimization, and comparisons, in Section V. Finally, we conclude with discussion and directions for future work in Section VI.

II. Related Work

Bi-level optimization [2] is a framework where an upper-level optimizer utilizes the results, i.e., solution and potentially gradients, of a lower-level optimization problem. Approaches typically either implicitly solve the lower-level problem and compute gradients using the solution, or explicitly represent the optimality conditions of the lower-level problem as constraints in the upper-level problem. For example, the MuJoCo simulator [3] has been employed in an implicit bi-level approach for whole-body model-predictive control of a humanoid robot [4]. The lower-level simulator solves a convex optimization problem in order to compute contact forces for rigid-body dynamics, and the results are utilized to perform rollouts and return finite-differenced gradients to the upper-level iLQR optimizer. In contrast, explicit approaches have directly encoded the linear-complementarity-problem contact dynamics as constraints in a direct trajectory-optimization method [5], [6]. A related approach formulates contact dynamics as lower-level holonomic constraints [7]. Implicit integrators, which are a specific instance of optimization-based dynamics and are widely used in collocation methods [8], have been explored with differentiable dynamic programming generally [9] and specifically for models that require cloth [10] or contact [11] simulation. Implicit dynamics have also been trained to represent non-smooth dynamics [12], although this work has not been utilized for trajectory optimization. Direct methods with implicit lower-level problems have been used in locomotion applications for tracking reference trajectories with model-predictive control [13] and a semi-direct method utilizes a lower-level friction problem for planning through contact events [14]. A related, sequential operator splitting framework is proposed in [15] and a derivative-free method that generates gradients via randomized smoothing for iLQR is also proposed [16]. In this work we focus on implicit lower-level problems that can include cone constraints and indirect methods, specifically iLQR, as the upper-level optimizer.

III. Background

In this section we provide background on the iLQR algorithm, implicit-function theorem, and implicit integrators.

A. Iterative LQR

iLQR is an algorithm for trajectory optimization that solves instances of the following problem:

$$\begin{aligned} \underset{u_{1:T-1}}{\text{minimize}} \quad & g_T(x_T) + \sum_{t=1}^{T-1} g_t(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t), \quad t = 1, \dots, T-1, \\ & (x_1 \text{ given}). \end{aligned} \tag{1}$$

For a system with state $x_t \in \mathbf{R}^n$, control inputs $u_t \in \mathbf{R}^m$, time index t, initial state x_1, and discrete-time dynamics $f_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}^n$, the algorithm aims to minimize an objective with stage-cost functions, $g_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$, and terminal-cost function, $g_T : \mathbf{R}^n \to \mathbf{R}$, over a planning horizon T.

The algorithm utilizes gradients of the objective and dynamics, and exploits the problem's temporal structure. The state trajectory is defined implicitly by the initial state x_1 and only the control trajectory $u_{1:T-1}$ is parameterized as decision variables. Closed-loop rollouts enable this indirect method to work well in practice. The overall complexity is linear in the planning horizon and cubic in the control-input dimension [17]

While efficient, the algorithm does not natively handle additional general equality or inequality constraints. This limitation has led to many variations of the classic algorithm in order to accommodate constraints in some capacity at the solver level. Box constraints have been incorporated at each step in the backward pass in order to enforce control limits [18]. An alternative approach embeds controls in barrier functions which smoothly approximate constraints [19]. Augmented Lagrangian methods and constrained backward and forward passes have also been proposed for handling general constraints [20], [21].

B. Implicit-Function Theorem

An implicit function, $r : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}^n$, is defined as

$$r(z^*; \theta) = 0, \tag{2}$$

for solutions $z^* \in \mathbf{R}^n$ and problem data $\theta \in \mathbf{R}^p$.

At an equilibrium point, $z^*(\theta)$, the sensitivities of the solution with respect to the problem data, i.e., $\partial z / \partial \theta$, can be computed under certain conditions [22]. First, we approximate (2) to first order:

$$\frac{\partial r}{\partial z} \delta z + \frac{\partial r}{\partial \theta} \delta \theta = 0, \tag{3}$$

and then the sensitivity of the solution is computed as:

$$\frac{\partial z}{\partial \theta} = -\left(\frac{\partial r}{\partial z}\right)^{-1} \frac{\partial r}{\partial \theta}. \tag{4}$$

In the case that $\partial r / \partial z$ is not full rank, an approximate solution, e.g., least-squares, can be computed.

C. Implicit Integrators

Unlike explicit integrators, i.e., $x_{t+1} = f_t(x_t, u_t)$, implicit integrators are an implicit function of the next state [23], [24],

which cannot be separated from the current state and control input:

$$f_t(x_{t+1}, x_t, u_t) = 0, \tag{5}$$

and are often used for numerical simulation of stiff systems due to improved stability and accuracy compared to explicit integrators, at the expense of increased computation [25].

For direct trajectory-optimization methods that parameterize states and controls and allow for dynamic infeasibility during iterates, such integrators are easily utilized since the state at the next time step is already available as a decision variable to the optimizer [8]. However, for indirect methods, like iLQR, that enforce strict dynamic feasibility at each iteration via rollouts, evaluating (5) requires a numerical solver to find a solution $x_{t+1} = z^*(\theta)$ that satisfies this implicit function for problem data $\theta = (x_t, u_t)$.

In practice, the next state can be found efficiently and reliably using Newton's method. Typically, the current state is used to initialize the solver and less than 10 iterations are required to solve the root-finding problem to machine precision.

Having satisfied (5) to find the next state, we can compute the dynamics Jacobians using the implicit-function theorem. First, the dynamics are approximated to first order:

$$\frac{\partial f_t}{\partial x_{t+1}}\delta x_{t+1} + \frac{\partial f_t}{\partial x_t}\delta x_t + \frac{\partial f_t}{\partial u_t}\delta u_t = 0, \tag{6}$$

and then we solve for δx_{t+1}:

$$\delta x_{t+1} \quad = \quad -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\Big(\frac{\partial f_t}{\partial x_t}\delta x_t \;+\; \frac{\partial f_t}{\partial u_t}\delta u_t\Big). \tag{7}$$

The Jacobians:

$$\frac{\partial x_{t+1}}{\partial x_t} = -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\frac{\partial f_t}{\partial x_t}, \tag{8}$$

$$\frac{\partial x_{t+1}}{\partial u_t} = -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\frac{\partial f_t}{\partial u_t}. \tag{9}$$

are returned at each time step.

IV. Optimization-Based Dynamics

In the previous section, we presented dynamics with implicit integrators that can be evaluated during rollouts and differentiated. In this section, we present a more general representation: optimization-based dynamics, which solve a constrained optimization problem in order to evaluate dynamics and use the implicit-function theorem to compute gradients by differentiating through the problem's optimality conditions.

A. Problem Formulation

For dynamics we require: fast and reliable evaluation, gradients that are useful to an upper-level optimizer, and (ideally) tight constraint satisfaction. We consider problems of the form:

$$x_{t+1} \in z^*(\theta) = \underset{z \in \mathcal{K} \;|\; c(z;\theta)=0}{\arg\min} \; l(z;\theta) \tag{10}$$

Algorithm 1 Differentiable Path-Following Method

procedure PathFollowing(z, θ)
  Parameters: $\beta = 0.5, \gamma = 0.1,$
    $\epsilon_\mu = 10^{-4}, \epsilon_r = 10^{-8}$
  Initialize: $\lambda = 0, \nu \in \mathcal{K}^*, \mu = 1.0, w_\mu = \{\}$
  $\bar{r} = r(w; \theta, \mu)$
  Until $\mu < \epsilon_\mu$ **do**
    $\Delta w = (\Delta z, \Delta\lambda, \Delta\nu) = (\partial r/\partial w)^{-1}\bar{r}$
    $\alpha \leftarrow 1$
    Until $z - \alpha\Delta z \in \mathcal{K}$, $\nu - \alpha\Delta\nu \in \mathcal{K}^*$ **do**
      $\alpha \leftarrow \beta\alpha$
    $\bar{r}_+ = r(w - \alpha\Delta w; \theta, \mu)$
    Until $\|\bar{r}_+\| < \|\bar{r}\|$ **do**
      $\alpha \leftarrow \beta\alpha$
      $\bar{r}_+ = r(w - \alpha\Delta w; \theta, \mu)$
    $w \leftarrow w - \alpha\Delta w$
    $\bar{r} \leftarrow \bar{r}_+$
    If $\|\bar{r}\| < \epsilon_r$ **do**
      $w_\mu \leftarrow w_\mu \cup w$
      $\mu \leftarrow \gamma\mu$
  $\partial w/\partial\theta \leftarrow \mathrm{IFT}(w_\mu, \theta)$ ▷ Eq. 4
  Return $w, \partial w/\partial\theta$
end procedure

with decision variable $z \in \mathbf{R}^k$, problem data $\theta \in \mathbf{R}^p$, objective $\ell : \mathbf{R}^k \times \mathbf{R}^p \to \mathbf{R}$, equality constraints $c : \mathbf{R}^k \times \mathbf{R}^p \to \mathbf{R}^q$, and cone $\mathcal{K}$, which compactly represents combinations of free, positive-orthant, and second-order-cone constraints. Many problems from robotics can be specified in this form. For example, the objective could be the Lagrangian for a system, the cone constraints can represent joint limits, the problem data might comprise the current state and control, $\theta = (x_t, u_t)$, and the next state belongs to the optimal solution set.

B. Path-Following Method

One of the primary challenges in optimizing (10) is selecting a solver that is well-suited to the requirements imposed by dynamics representations. Solvers for this class of problem are generally categorized as first-order projection [26], [27], [28] or second-order path-following methods [29], [30]. The first approach optimizes (10) by splitting the problem and alternating between inexpensive first-order methods, and potentially non-smooth, projections onto the cone. The second approach formulates and optimizes barrier subproblems, using second-order methods, along a central path, eventually converging to the cone's boundary in the limit [31]. Second-order semi-smooth methods also exist [32].

The first-order projection-based methods, while fast, often can only achieve coarse constraint satisfaction and, importantly, the gradients returned are usually subgradients, which are less useful to an optimizer. In contrast, path-following methods exhibit fast convergence, can achieve tight constraint satisfaction [33], and can return smooth gradients using a relaxed central-path parameter.

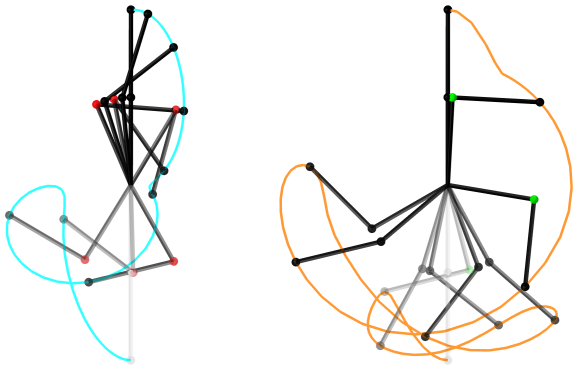

Fig. 2: Optimized trajectories for acrobot systems without (left) and with (right) joint limits performing a swing-up. Red and green elbow colors indicate violation or non-violation of the joint limits, respectively.

Based on these characteristics, we utilize a path-following method [33] (see Section 14.3) to optimize (10). The idea behind this approach is that the cone constraints are handled using a logarithmic barrier and a sequence of relaxed, and easier to solve subproblems, optimized using Newton's method, converges to a solution of the original problem.

The optimality conditions for a barrier subproblem are:

$$\partial \ell(z;\theta)/\partial z + (\partial c(z;\theta)/\partial z)^T \lambda - \nu = 0, \tag{11}$$

$$c(z;\theta) = 0, \tag{12}$$

$$z \circ \nu = \mu \mathbf{e}, \tag{13}$$

$$z \in \mathcal{K},\ \nu \in \mathcal{K}^*, \tag{14}$$

with duals $\lambda \in \mathbf{R}^q$ and $\nu \in \mathbf{R}^k$, cone product denoted with the $\circ$ operator, central-path parameter $\mu \in \mathbf{R}_+$, and dual cone $\mathcal{K}^*$ [29]. We consider free: $\mathbf{R}^d$; nonnegative: $\mathbf{R}^d_+$; second-order:

$$\mathcal{Q}^d = \{(a_1, a_{2:d}) \in \mathbf{R} \times \mathbf{R}^{d-1} \mid \|a_{2:d}\|_2 \leq a_1\}; \tag{15}$$

and the Cartesian product of these cones. For nonnegative cones, the cone product is an element-wise product and $\mathbf{e} = \mathbf{1}$. For second-order cones, the cone product is:

$$a \circ b = (a^T b, a_1 b_{2:d} + b_1 a_{2:d}), \tag{16}$$

and $\mathbf{e} = (1, 0_{d-1})$ [29]. The nonnegative and second-order cones are self dual, i.e., $\mathcal{K} = \mathcal{K}^*$, while the dual of the free cone is the zero cone, i.e., $\{0\}$.

To find a stationary point of (11-14), for a fixed central-path parameter μ, we consider $w = (z, \lambda, \nu) \in \mathbf{R}^{k+q+k}$ and a residual $r : \mathbf{R}^{k+q+k} \times \mathbf{R}^p \times \mathbf{R}_+ \to \mathbf{R}^{k+p+k}$ comprising (11-13). A search direction, $\Delta w \in \mathbf{R}^{k+q+k}$, is computed using the residual and its Jacobian with respect to the decision variables at the current point. A backtracking line search is performed to ensure that a candidate step respects the cone constraints and reduces the norm of the residual. Once the residual norm is below a desired tolerance, we cache the current solution w_μ, the central-path parameter is decreased, and the procedure is repeated until the central-path parameter is below a desired tolerance. We refer to [31], [33] for additional background on these methods.

TABLE I: Comparison of objective, iterations, goal constraint violation, and solve time between MuJoCo and optimization-based dynamics with nominal and joint-limited models for acrobot. When limited, the MuJoCo model violates the joint limits and fails at the swing-up task.

	MuJoCo	MuJoCo (limited)	ours	ours (limited)
obj.	395.6	1.9e6	48.9	84.2
iter.	207	95	256	907
con.	3e-4	0.2	5e-4	1e-3
time (s)	0.9	1.0	0.5	6.5

To differentiate (10), we apply the implicit-function theorem (4) to the residual at an intermediate solution, w_μ. In practice, we find that performing implicit differentiation with a solution point having a relaxed central-path parameter returns smooth gradients that improve the convergence behavior of the upper-level optimizer. The complete algorithm is summarized in Algorithm 1 and we provide an open-source implementation of a path-following solver.

V. Examples

We formulate optimization-based dynamics models and use iLQR to perform bi-level trajectory optimization for a number of examples that highlight how these representations can be constructed and demonstrate that trajectories for non-smooth and constrained dynamics can be optimized. Additionally, we provide a comparison of our approach with MuJoCo using finite-difference gradients, contact-implicit trajectory optimization, and gradients generated via randomized smoothing.

Throughout, we use implicit midpoint integrators, quadratic costs, and for convenience, employ an augmented Lagrangian method to enforce any remaining trajectory-level constraints (e.g., terminal constraints), not handled implicitly by the dynamics representation. Our implementation and all of the experiments are available here: https://github.com/thowell/optimization_dynamics.

A. Acrobot with Joint Limits

We model an acrobot [34] with joint limits on the actuated elbow. These limits are enforced with a signed-distance constraint,

$$\phi(q) = \begin{bmatrix} \pi/2 - q_{\text{e}} \\ q_{\text{e}} + \pi/2 \end{bmatrix} \geq 0, \tag{17}$$

where $q \in \mathbf{R}^2$ is the system's configuration and q_{e} is the elbow angle. Additional constraints,

$$\lambda \circ \phi(q) = 0, \quad \lambda \geq 0, \tag{18}$$

enforce physical behaviors that impact impulses $\lambda_t \in \mathbf{R}^2$ can only be non-negative, i.e., repulsive not attractive, and that they are only applied to the system when the joint reaches a limit. Relaxing the complementarity constraint via a central-path parameter, introducing a slack variable for the signed-distance function ϕ, and combining this reformulation with

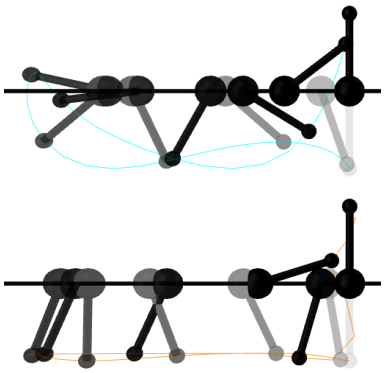

Fig. 3: Optimized trajectories for cart-pole performing a swing-up without (top, friction coefficient $= 0$) and with (bottom, friction coefficient $= 0.35$) joint friction.

the system's dynamics results in a problem formulation (11-14) that can be optimized with Algorithm 1.

Unlike approaches that include joint limits at the solver level, including the impact (18) and limit (17) constraints at the dynamics level enables impact forces encountered at joint stops to be optimized and applied to the system.

The system has $n = 4$ states and $m = 1$ controls. We plan a swing-up trajectory over a horizon $T = 101$ with a time step $h = 0.05$. The optimizer is initialized with random controls.

We compare unconstrained and joint-limited systems, the optimized motions are shown in Fig. 2. The unconstrained system violates joint limits, while the joint-limited system reaches the limits without exceeding them and finds a motion utilizing additional pumps. Additionally, we compare our optimization-based dynamics with the MuJoCo physics simulator and gradients provided via finite-differencing. For the unconstrained system, the optimizer is able to successfully find a swing-up trajectory. For the joint-limited system, the trajectory optimizer fails. We also note that MuJoCo is unable to enforce hard joint limits and that such constraints typically require tuning the solver parameters to produce realistic looking behavior. The results are summarized in Table I.

B. Cart-Pole with Coulomb Friction

A cart-pole [34] is modeled that experiences Coulomb friction [35] on both its slider and pendulum arm. The friction force that maximally dissipates kinetic energy is the solution to the following optimization problem:

$$\begin{aligned} \underset{b}{\text{minimize}} \quad & v^T b \\ \text{subject to} \quad & \|b\|_2 \leq N, \end{aligned} \tag{19}$$

where $v \in \mathbf{R}^{d-1}$ is the joint velocity, $b \in \mathbf{R}^{d-1}$ is the friction force, and $N \in \mathbf{R}_+$ is the friction-cone limit [36]. The optimality conditions for the cone program's barrier subproblem are:

$$\begin{bmatrix} y^T & v^T \end{bmatrix}^T - \eta = 0, \tag{20}$$

$$\beta_1 - N = 0, \tag{21}$$

$$\beta \circ \eta = \mu \mathbf{e}, \tag{22}$$

$$\beta, \eta \in \mathcal{Q}^d, \tag{23}$$

with $\beta \in \mathbf{R}^d$, such that $b = \beta_{2:d}$, and dual variables $y \in \mathbf{R}$ and $\eta \in \mathbf{R}^d$ for the equality and cone constraints, respectively.

The system has $n = 4$ states, $m = 1$ controls, and dimension $d = 2$. We make the modeling assumption that N, which is a function of the friction coefficient, is fixed for both joints. We plan a swing-up trajectory for a horizon $T = 51$ and time step $h = 0.05$. The optimizer is initialized with an impulse at the first time step and zeros for the remaining initial control inputs.

We compare the trajectories optimized for systems with increasing amounts of friction, i.e., increasing N, in Fig. 3. In the presence of friction, the system performs a more aggressive manuever at the end of the trajectory in order to achieve the swing-up. The results are summarized in Table II.

TABLE II: Comparison of objective, iterations, goal constraint violation, and solve time for cart-pole model with different friction coefficients.

friction	0.0	0.01	0.1	0.25	0.35
obj.	5.0	5.5	11.6	76.3	163.1
iter.	456	485	406	472	943
con.	1e-4	5e-4	6e-4	1e-3	5e-3
time (s)	0.36	2.8	3.1	4.6	9.8

C. Hopper Gait

We generate a gait for a Raibert hopper [37]. The system's dynamics are modeled as a nonlinear complementarity problem [13]. The 2D system has four generalized coordinates: lateral and vertical body positions, body orientation, and leg length. There are $m = 2$ control inputs: a body moment and leg force. The state is $n = 8$, comprising the system's current and previous configurations. The horizon is $T = 21$ with time step $h = 0.05$. The initial state is optimized and a trajectory-level periodicity constraint is employed to ensure that the first and final states, with the exception of the lateral positions, are equivalent. Finally, we initialize the solver with controls that maintain the system in an upright configuration.

We compare our implicit bi-level approach to a direct method for contact-implicit trajectory optimization [38] that transcribes the problem and solves it with Ipopt [39]. We use the same models, cost functions, and comparable optimizer parameters and tolerances.

The complexity of the classic iLQR algorithm is: $O(Tm^3)$ [17], while a direct method for trajectory optimization that exploits the problem's temporal structure is: $O\Big(T(n^3{+}n^2m)\Big)$ [40]. The path-following method generally has complexity: $O(a^3)$, although specialized solvers can reduce this cost, where a is the number of primal and dual variables [33].

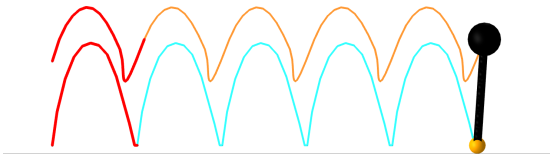

Fig. 4: Hopper gait. Optimized template (red) for the system's body (orange) and foot (blue) trajectories is repeated to form a gait.

TABLE III: Comparison between contact-implicit trajectory optimization (direct) and optimization-based dynamics + iLQR (ours) for hopper-gait examples. The objective, iterations, gait constraint violation, and solve times are compared.

	direct 1	direct 2	direct 3	ours 1	ours 2	ours 3
obj.	3.5	20.4	2.9	3.5	19.8	2.4
iter.	122	114	107	38	47	25
con.	1e-9	1e-9	1e-9	3e-4	3e-4	9e-4
time (s)	2.0	2.0	1.8	0.3	0.4	0.3

The complexity of optimization-based dynamics and iLQR is typically dominated by the cost of the path-following method, which is incurred at each time step in the planning horizon, so the overall complexity is $O(Ta^3)$. The hopper problem has $a = 20$ and $n = 8$ and $m = 2$ with our approach. For the contact-implicit approach, the controls are augmented with contact variables, increasing the dimension to $m = 17$ at each time step. As a result, contact-implicit trajectory optimization has lower complexity in the case where a structure exploiting direct method is employed, although this is not necessarily the case when a general-purpose solver like Ipopt is called.

An optimized gait is shown in Fig. 4. We find that our implicit approach requires fewer, but more expensive iterations, while contact-implicit trajectory optimization requires more, but less expensive iterations. Additionally, our approach more consistently returns good trajectories. Numerical comparison results are provided in Table III.

D. Rocket Landing

We plan a soft-landing for a rocket with 6 degrees of freedom that must transition from a horizontal to vertical orientation prior to landing while respecting thrust constraints. Unlike prior work that enforces such constraints at the optimizer level [41], we instead enforce these constraints at the dynamics level. This enables the optimizer to provide smooth controls to the dynamics, which then internally constrain the input to satisfy the system's limits at every iteration.

The cone projection problem, which finds the thrust vector that is closest to the optimizer's input thrust while satisfying constraints, is formulated as:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \tfrac{1}{2}\|u-\bar{u}\|_2^2 \\ \text{subject to} \quad & \|u_{2:3}\|_2 \leq s u_1, \\ & u_{\min} \leq u_1 \leq u_{\max} \end{aligned} \tag{24}$$

where $u, \bar{u} \in \mathbf{R}^3$ are the optimized and provided thrust vectors, respectively, u_1 is the component of thrust along the longitudinal axis of the rocket, $u_{\min}$ and $u_{\max}$ are limits on this value, and $s \in \mathbf{R}_+$ is a scaling parameter.

The system has $n = 12$ states and $m = 3$ controls that are first projected using (24) before being applied to dynamics. The planning horizon is $T = 61$ and time step is $h = 0.05$. The controls are initialized with small random values, the initial pose is offset from the target, and the rocket has initial downward velocity. A constraint enforces the rocket's final pose, a zero velocity, vertical-orientation configuration in a landing zone. The scaling parameter s is set to one.

The optimizer requires 547 iterations and takes 8.1 seconds to find a plan that successfully reorients the rocket prior to landing while respecting the thrust constraints. Without the cone constraint, the optimizer requires 521 iterations and 1.9 seconds to find a solution. However, the thrust cone constraint is violated at the first time steps. The position trajectory is shown in Fig. 5.

E. Planar Push

For the canonical manipulation problem of planar pushing [42], we optimize the positions and forces of a robotic manipulator's circular end-effector in order to slide a planar bock into an oriented goal pose (Fig. 1).

The system's end-effector is modeled as a fully actuated particle in 2D that can move the block (with 2D translation and orientation) via impact and friction while the two systems are in contact. The block is modeled with point friction at each of its four corners and a signed-distance function, modeled as a p-norm with $p = 10$, is employed that enables the end-effector to push on any of the block's sides.

The system has $n = 10$ states, $m = 2$ controls, the planning horizon is $T = 26$, and the timestep is $h = 0.1$. We initialize the end-effector with a control input that overcomes stiction to move the block. The block is initialized at the origin with the pusher in contact on its left side, and the block is constrained at the trajectory-level to reach a goal pose.

We compare the smooth gradients returned by differentiating through our path-following solver with randomized smoothing to generate a zero-order gradient bundle [16]. We use $N = 100$ samples and noise sampled from a zero-mean unit Gaussian with scaling $\sigma = 1.0e\text{-}4$. Ultimately, we find that both approaches result in similar convergence behavior of the upper-level optimizer. However, the gradient-bundle approach necessitates parallel computation of dynamics evaluations (which we do not explore in our comparison) in order to be a tractable approach, whereas the implicit-function theorem is much more efficient in a serial-computational setting. Results are provided in Table IV.

VI. Discussion

In this work we have presented a bi-level optimization framework that utilizes lower-level optimization-based dynamics for trajectory optimization. This representation enables expressive dynamics by including constraint handling, internal states, and additional physical behavior modeling at the dynamics level, abstracted away from the upper-level optimizer,

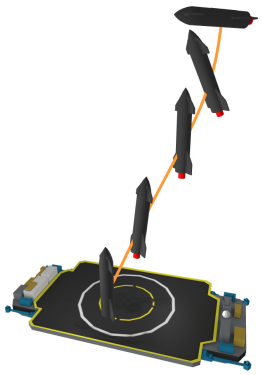

Fig. 5: Position trajectory (orange) for rocket performing soft landing. The system first reorients from a horizontal to vertical pose before landing with zero velocity in a target zone while respecting thrust constraints.

enabling classically unconstrained solvers to optimize motions for more complex systems.

One of the primary drawbacks to this approach is the lack of convergence guarantees for finding a solution that satisfies the dynamics representation. In practice, we find that the solver converges reliably. However, there are cases where the solver fails to meet specified tolerances. In this event we have the optimizer return the current solution and its sensitivities. We find that occasional failures like this often do not impair the overall trajectory-optimization solve and that the optimizer can eventually find a dynamically feasible trajectory. Just as robust, large-scale solvers such as Ipopt [39] fallback to their restoration mode when numerical difficulties occur, our basic path-following method is likely to be improved by including such a fallback routine, perhaps specific to a particular system. Additionally, we find that standard techniques such as: problem scaling, appropriate hyperparameter selection, and warm-starting greatly improve the reliability of this approach.

Other potential weaknesses are the increased serial nature of the approach and cost of evaluating the dynamics and their gradients. First, iLQR is a serial algorithm, dominated by forward rollouts and backward Riccati recursions that cannot be parallelized. Similarly, the path-following solver is a serial method dominated by a matrix factorization and back substitution that is generally not amenable to parallelization either. Second, because an iterative solver is utilized to evaluate the dynamics, calls to the dynamics are inherently more expensive compared to an explicit dynamics representation. However, such overhead can potentially be mitigated with a problem-specific solver that can exploit specialized constraints or sparsity, unlike an off-the-shelf solver's generic routines.

There are numerous avenues for future work exploring optimization-based dynamics for bi-level trajectory optimization. First, in this work we explore constrained optimization problems solved with a second-order method. Another interesting problem class to consider are energy-based models [43], potentially optimized with first-order Langevin dynamics [44]. Second, we find that returning gradients optimized with a relaxed central-path parameter greatly improves the convergence behavior of the upper-level optimizer. An analysis of, or method for, returning gradients from the lower-level problem that best aid an upper-level optimizer would be useful. Finally, this bi-level framework could be extended to a tri-level setting where the highest-level optimizer autotunes an objective to generate model-predictive-control policies in an imitation-learning setting.

TABLE IV: Comparison between gradients generated as zero-order gradient bundles (GB) and via the implicit-function theorem (I) for planar-push task. *Gradient bundles are evaluated serially.

	translate (GB)	rotate (GB)	translate (I)	rotate (I)
iter.	18	32	17	33
time (s)	+100.0*	+100.0*	8.0	18.3

REFERENCES

[1] D. H. Jacobson and D. Q. Mayne, *Differential dynamic programming*. Elsevier Publishing Company, 1970, no. 24.

[2] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.

[3] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[4] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the HRP-2 humanoid," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3346–3351.

[5] K. Yunt and C. Glocker, "Trajectory optimization of mechanical hybrid systems using sumt," in *9th IEEE International Workshop on Advanced Motion Control, 2006.* IEEE, 2006, pp. 665–671.

[6] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[7] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2536–2542.

[8] O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal Control*. Springer, 1993, pp. 129–143.

[9] W. Jallet, N. Mansard, and J. Carpentier, "Implicit differential dynamic programming," 2021.

[10] S. Zimmermann, R. Poranne, and S. Coros, "Dynamic manipulation of deformable objects with implicit integration," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4209–4216, 2021.

[11] I. Chatzinikolaidis and Z. Li, "Trajectory optimization of contact-rich motions using implicit differential dynamic programming," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2626–2633, 2021.

[12] S. Pfrommer, M. Halm, and M. Posa, "Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations," *arXiv preprint arXiv:2009.11193*, 2020.

[13] S. Le Cleac'h, T. A. Howell, M. Schwager, and Z. Manchester, "Fast contact-implicit model-predictive control," *arXiv preprint arXiv:2107.05616*, 2021.

[14] B. Landry, J. Lorenzetti, Z. Manchester, and M. Pavone, "Bilevel optimization for planning through contact: A semidirect method," *arXiv preprint arXiv:1906.04292*, 2019.

[15] V. Sindhwani, R. Roelofs, and M. Kalakrishnan, "Sequential operator splitting for constrained nonlinear optimal control," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 4864–4871.

[16] H. J. Terry Suh, T. Pang, and R. Tedrake, "Bundled gradients through contact via randomized smoothing," *arXiv preprint arXiv:2109.05143*, 2021.

[17] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming." in *NIPS*. Citeseer, 2007, pp. 1465–1472.

[18] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[19] J. Marti-Saumell, J. Solà, C. Mastalli, and A. Santamaria-Navarro, "Squash-box feasibility driven differential dynamic programming," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 7637–7644.

[20] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7674–7679.

[21] G. Lantoine and R. Russell, "A hybrid differential dynamic programming algorithm for robust low-thrust optimization," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2008, p. 6615.

[22] U. Dini, *Lezioni di analisi infinitesimale*. Fratelli Nistri, 1907, vol. 1.

[23] J. Brüdigam and Z. Manchester, "Linear-time variational integrators in maximal coordinates," *arXiv preprint arXiv:2002.11245*, 2020.

[24] Z. R. Manchester and M. A. Peck, "Quaternion variational integrators for spacecraft dynamics," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 1, pp. 69–76, 2016.

[25] G. Wanner and E. Hairer, *Solving Ordinary Differential Equations II*. Springer Berlin Heidelberg, 1996, vol. 375.

[26] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[27] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, 2016.

[28] M. Garstka, M. Cannon, and P. Goulart, "Cosmo: A conic operator splitting method for large convex problems," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 1951–1956.

[29] A. Domahidi, E. Chu, and S. Boyd, "Ecos: An socp solver for embedded systems," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 3071–3076.

[30] L. Vandenberghe, "The cvxopt linear and quadratic cone program solvers," *Online: http://cvxopt.org/documentation/coneprog.pdf*, 2010.

[31] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[32] A. Ali, E. Wong, and J. Z. Kolter, "A semismooth newton method for fast, generic convex programming," in *International Conference on Machine Learning*. PMLR, 2017, pp. 70–79.

[33] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[34] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832)," *Downloaded in Fall*, 2021.

[35] J. J. Moreau, "On unilateral constraints, friction and plasticity," in *New Variational Techniques in Mathematical Physics*. Springer, 2011, pp. 171–322.

[36] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear Algebra and its Applications*, vol. 284, no. 1-3, pp. 193–228, 1998.

[37] M. H. Raibert, H. B. Brown Jr., M. Chepponis, J. Koechling, J. K. Hodgins, D. Dustman, W. K. Brennan, D. S. Barrett, C. M. Thompson, J. D. Hebert, W. Lee, and B. Lance, "Dynamically stable legged locomotion," Massachusetts Institute of Technology Cambridge Artificial Intelligence Lab, Tech. Rep., 1989.

[38] Z. Manchester and S. Kuindersma, "Variational contact-implicit trajectory optimization," in *Robotics Research*. Springer, 2020, pp. 985–1000.

[39] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[40] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2009.

[41] L. Blackmore, B. Açikmeşe, and D. P. Scharf, "Minimum-landing-error powered-descent guidance for mars landing using convex optimization," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 4, pp. 1161–1171, 2010.

[42] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," *arXiv preprint arXiv:1611.08268*, 2016.

[43] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Jie Huang, "A tutorial on energy-based learning," *Predicting structured data*, vol. 1, no. 0, 2006.

[44] T. Schlick, *Molecular modeling and simulation: an interdisciplinary guide*. Springer, 2010, vol. 2.