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 forces 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.**

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 and return dynamically feasible trajectories by performing rollouts and utilizing gradients of the system's dynamics.

Classically, indirect methods like iterative LQR (iLQR) [10] 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 the properties of our path-following method to return useful, smooth gradients to an upper-level optimizer.

*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, 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

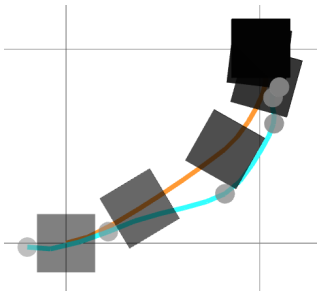

Fig. 1: Optimized trajectories for planar-push task. The pusher (blue) and block (orange) paths are shown for a plan that maneuvers the block from a pose at the origin: $(0, 0, 0)$ to a goal pose: $(0.5,0.5,\pi/4)$.

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.

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 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 [29] 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 [34] has been employed in an implicit bi-level approach for whole-body model-predictive control of a humanoid robot [11]. 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 used the optimality conditions of linear-complementarity-problem contact dynamics as constraints in a direct trajectory-optimization method [25]. A related approach formulates contact dynamics as lower-level holonomic constraints [20]. Implicit integrators, which are a specific instance of optimization-based dynamics and are widely used in collocation methods [36], have been explored with differentiable dynamic programming for models that require cloth simulation [41]. Implicit dynamics have also been trained to represent non-smooth dynamics [24], 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 [14] and a semi-direct method utilizes a lower-level friction problem for planning through contact events [12]. A related, sequential operator splitting framework is proposed in [28]. In this work we focus on implicit lower-level problems 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:

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

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 [31]

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 [32]. 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 [9, 40, 13].

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., $\frac{\partial z}{\partial \theta}$, can be computed [5]. 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 $\frac{\partial r}{\partial z}$ is not full rank, an approximate solution to (4), 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 [4, 18], 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 [39].

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 [36]. 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. We describe dynamics with implicit integrators as:

$$\begin{aligned} x_{t+1}(x_t,u_t) &= f_t(x_t,u_t) \\ &= \underset{x_{t+1}}{\text{find}} f_t(x_{t+1};x_t,u_t) = 0 \end{aligned} \tag{6}$$

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:

$$\frac{\partial f_t}{\partial x_{t+1}}\delta x_{t+1} + \frac{\partial f_t}{\partial x_t}\delta x_t + \frac{\partial f_t}{\partial u_t}\delta u_t = 0, \tag{7}$$

and then we solve for δx_{t+1}:

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

The Jacobians:

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

$$\frac{\partial x_{t+1}}{\partial u_t} = -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\frac{\partial f_t}{\partial u_t}. \tag{10}$$

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:

$$x_{t+1} \in z^*(\theta) = \underset{z \in \mathcal{K} \mid c(z;\theta)=0}{\arg\min} \; l(z;\theta) \tag{11}$$

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 $\mathcal{K}$. Typically, the problem data include 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 (11) 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 [30, 23, 7] or second-order path-following methods [6, 35]. The first approach optimizes (11) 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 [3]. Second-order semi-smooth methods also exist [1].

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 [22], and can return smooth gradients using a relaxed central-path parameter.

Based on these characteristics, we utilize a path-following method [22] to optimize (11). The optimality conditions for a barrier subproblem are:

$$\nabla_z \ell(z;\theta) + \nabla_z c(z;\theta)^T \lambda - \nu = 0, \tag{12}$$

$$c(z;\theta) = 0, \tag{13}$$

$$z \circ \nu = \mu \mathbf{e}, \tag{14}$$

$$z \in \mathcal{K}, \nu \in \mathcal{K}^*, \tag{15}$$

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 $\mathcal{K}^*$ [6]. We consider free: $\mathbf{R}^d$; nonnegative: $\mathbf{R}^d_+$; second-order:

$$\mathcal{Q}^d = \{(a_1, a_{2:d}) \in \mathbf{R} \times \mathbf{R}^{d-1} \mid \|a_{2:d}\|_2 \le a_1\}; \tag{16}$$

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:

$$a \circ b = (a^T b, a_1 b_{2:d} + b_1 a_{2:d}), \tag{17}$$

and $\mathbf{e} = (1, 0_{d-1})$ [6]. 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 (12-15), 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 (12-14). 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

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**
    $\Delta w=(\Delta z,\Delta\lambda,\Delta\nu)=(\frac{\partial r}{\partial w})^{-1}\bar{r}$
    $\alpha\leftarrow 1$
    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**
      $w_\mu\leftarrow w_\mu\cup w$
      $\mu\leftarrow\gamma\mu$
  $\frac{\partial w}{\partial\theta}\leftarrow\text{IFT}(w_\mu,\theta)$ ▷ Eq. 4
  Return $w,\frac{\partial w}{\partial\theta}$
end procedure

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.

To differentiate (11), 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.

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, demonstrate that non-smooth and constrained dynamics can be optimized, provide a comparison of smooth gradients, and compare implicit and explicit bi-level approaches.

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/motion_planning/examples/implicit_dynamics`.

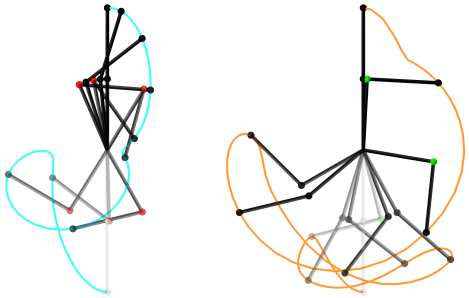

Fig. 2: Optimized trajectories for acrobot systems without (left) and with (right) joint limits performing a swing-up. The systems are visualized at 0.5 second increments. Red and green elbow colors indicate violation or non-violation of the joint limits, respectively.

A. Acrobot with Joint Limits

We model an acrobot [33] with joint limits on the actuated elbow. These limits are enforced with a signed-distance constraint,

$$\phi(q)=\begin{bmatrix}\pi/2-q_e\\ q_e+\pi/2\end{bmatrix}\geq 0, \tag{18}$$

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

$$\lambda\circ\phi(q)=0,\quad \lambda\geq 0, \tag{19}$$

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 the system's dynamics results in a problem formulation (12-15) that can be optimized with Algorithm 1.

Unlike approaches that include joint limits at the solver level, including the impact (19) and limit (18) 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.

B. Cart-Pole with Coulomb Friction

A cart-pole [33] is modeled that experiences Coulomb friction [21] on both its slider and pendulum arm. The friction

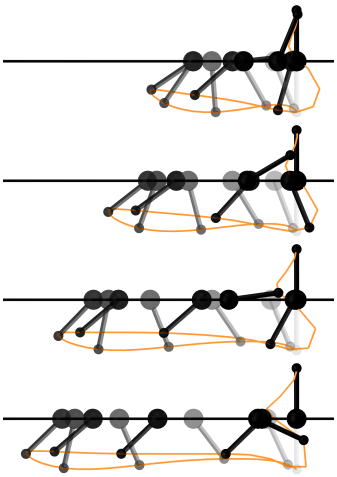

Fig. 3: Optimized trajectories for cart-pole performing a swing-up. The trajectories becoming increasingly aggressive in order to achieve a swing-up as friction is increased (top to bottom).

force that maximally dissipates kinetic energy is the solution to the following optimization problem:

$$\begin{aligned} \underset{b}{\text{minimize}} \quad & v^T b \\ \text{subject to} \quad & \|b\|_2 \leq N, \end{aligned} \tag{20}$$

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 [16]. The optimality conditions for the cone program's barrier subproblem are:

$$\begin{bmatrix} y^T & v^T \end{bmatrix}^T - \eta = 0, \tag{21}$$

$$\beta_1 - N = 0, \tag{22}$$

$$\beta \circ \eta = \mu \mathbf{e}, \tag{23}$$

$$\beta, \eta \in \mathcal{Q}^d, \tag{24}$$

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. We observe that the system must perform more aggressive maneuvers in order to succeed at the swing-up task in the presence of increasing friction.

C. Hopper Gait

We generate gaits for a Raibert hopper [26]. The system's dynamics are modeled as a nonlinear complementarity problem [14]. 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.

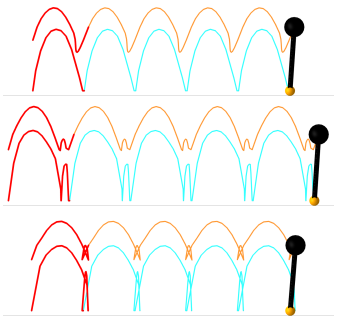

Fig. 4: Hopper gaits. Optimized templates (red) for the system's body (orange) and foot (blue) trajectories. The top trajectory finds a single-hop gait, the middle finds a two-hop gait, and the bottom finds a two-hop gait that bounces back before hopping forward.

By tuning the cost functions we generate qualitatively different gaits. The optimizer finds single-hop and multi-hop trajectories that satisfy the periodicity constraint. We repeat the optimized template to form a gait, three are shown in Fig. 4.

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 [2], 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:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \tfrac{1}{2}\|u - \bar{u}\|_2^2 \\ \text{subject to} \quad & \|u_{2:3}\|_2 \leq s u_1, \\ & u_{\min} \leq u_1 \leq u_{\max} \end{aligned} \tag{25}$$

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.

TABLE I: Comparison of smooth gradients, parameterized by central-path parameter μ and tested over five random seeds for random initial control inputs, on the number of iLQR iterations (mean $\pm$ standard deviation) required for acrobot swing-up problem.

1e-0	1e-1	1e-2	1e-3	1e-4
failure	128 ± 9	206 ± 10	582 ± 133	630 ± 68

The system has $n = 12$ states and $m = 3$ controls that are first projected using (25) before being applied to dynamics. The planning horizon is $T = 101$ 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 finds a plan that successfully reorients the rocket prior to landing while respecting the thrust constraints, which would otherwise be violated at the first time steps without the thrust projection. The normalized optimized controls and position trajectory are provided in Fig. 5.

E. Planar Push

For the canonical manipulation problem of planar pushing [8], 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 impulses and friction forces while the two systems are in contact. The block is modeled with point friction at each of its four corners and a single signed-distance function is employed that enables the end-effector to impact and push on any of the block's sides.

The system has $n = 10$ states, $m = 2$ controls, and $j = 10$ forces are optimized. 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.

F. Comparison of Smooth Gradients

We compare the effect of smooth gradients on the upper-level optimizer by varying the value of the central-path parameter used by the implicit-function theorem to compute the gradients of dynamics. The results for the acrobot swingup problem, provided in Table I, are computed over five trials of random initial control inputs for each central-path value. We find that iLQR requires more iterations when the central-path value is small and less iterations using smooth gradients—until the gradients are too smooth and the optimizer fails to find a swing-up trajectory.

TABLE II: Comparison between implicit and explicit bi-level approaches for hopper-gait and planar-push examples. The number of iterations, final cost, and maximum constraint violation are compared between: (ours) *implicit*, optimization-based dynamics with iLQR, and *explicit*, a direct method using Ipopt.

implicit	hopper 1	hopper 2	hopper 3	push 1	push 2
iter.	28	42	30	18	19
cost	-29.5	-192.1	-34.4	-32.6	-104.2
con.	1.5e-4	1.5e-4	2.3e-4	9.2e-4	1.4e-4
explicit	hopper 1	hopper 2	hopper 3	push 1	push 2
iter.	81	69	394	57	133
cost	-15.98	226.9	-22.5	-33.0	-104.1
con.	5.5e-9	5.5e-9	2.3e-6	3.0e-10	1.8e-8

G. Comparison with Explicit Bi-Level Optimization

We compare our implicit bi-level approach to an explicit one that is optimized using a direct method with Ipopt [37]. For the hopper-gait and planar-push examples, we explicitly represent the contact-dynamics optimality conditions as constraints and directly optimize trajectories [17]. We use the same models, cost functions, and comparable optimizer parameters and tolerances. The results are provided in Table II. We find that our implicit approach requires fewer, but more expensive iterations, while the explicit approach requires more, but less expensive iterations. Additionally, the implicit approach typically finds a lower final cost while the explicit approach is able to better satisfy the goal and periodicity constraints.

The complexity of the classic iLQR algorithm is: $O(Tm^3)$ [31], while a direct method for trajectory optimization that exploits the problem's temporal structure is: $O\Big(T(n^3 + n^2 m)\Big)$ [38]. 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 [22]. The complexity of optimization-based dynamics and iLQR is $O(Ta^3)$ and is typically dominated by the cost of the path-following method, which is incurred at each time step in the planning horizon. The hopper problems have $a = 12$ and the planar push problems have $a = 35$. For the explicit bi-level approach, we augment the control with the additional decision variables, increasing the control dimension m at each time step. Then, the hopper problems have $n = 8$ and $m = 9$. The planar push problems have $n = 10$ and $m = 49$.

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 more expressive dynamics by including constraint handling, internal states, and additional physical behavior modeling at the dynamics level, abstracted away from the

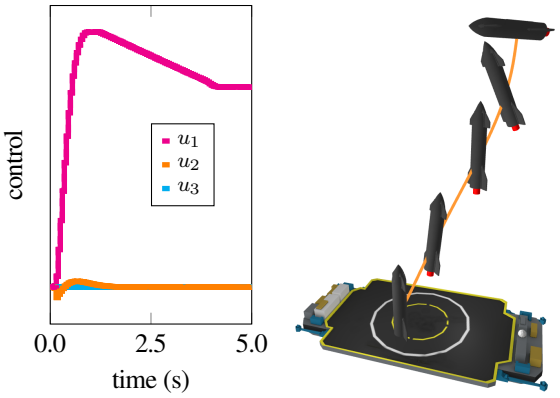

Fig. 5: Normalized optimized controls (left) and position trajectory (right) for rocket performing soft landing. The system first reorients from a horizontal to vertical pose before landing with zero velocity in a target zone while respecting thrust constraints.

upper-level optimizer, 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 of finding a solution for the dynamics representation. In practice, we find that occasionally the dynamics solver will fail to converge. In this event the optimizer simply returns 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 [37] 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.

Another potential weakness of this method is overall complexity or wall-clock time for a generic implementation. 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 an LU decomposition and solve 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 [15], potentially optimized with first-order Langevin dynamics [27]. 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.

References

[1] Alnur Ali, Eric Wong, and J. Zico Kolter. "A semismooth Newton method for fast, generic convex programming". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 70–79.

[2] Lars Blackmore, Behçet Açikmeşe, and Daniel P. Scharf. "Minimum-landing-error powered-descent guidance for Mars landing using convex optimization". In: *Journal of Guidance, Control, and Dynamics* 33.4 (2010), pp. 1161–1171.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[4] Jan Brüdigam and Zachary Manchester. "Linear-time variational integrators in maximal coordinates". In: *arXiv preprint arXiv:2002.11245* (2020).

[5] Ulisse Dini. *Lezioni di analisi infinitesimale*. Vol. 1. Fratelli Nistri, 1907.

[6] Alexander Domahidi, Eric Chu, and Stephen Boyd. "ECOS: An SOCP solver for embedded systems". In: *2013 European Control Conference (ECC)*. IEEE. 2013, pp. 3071–3076.

[7] Michael Garstka, Mark Cannon, and Paul Goulart. "COSMO: A conic operator splitting method for large convex problems". In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 1951–1956.

[8] François Robert Hogan and Alberto Rodriguez. "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics". In: *arXiv preprint arXiv:1611.08268* (2016).

[9] Taylor A. Howell, Brian E. Jackson, and Zachary 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.

[10] David H. Jacobson and David Q. Mayne. *Differential dynamic programming*. 24. Elsevier Publishing Company, 1970.

[11] Jonas Koenemann et al. "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.

[12] Benoit Landry et al. "Bilevel Optimization for Planning through Contact: A Semidirect Method". In: *arXiv preprint arXiv:1906.04292* (2019).

[13] Gregory Lantoine and Ryan Russell. "A hybrid differential dynamic programming algorithm for robust low-thrust optimization". In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2008, p. 6615.

[14] Simon Le Cleac'h et al. "Linear Contact-Implicit Model-Predictive Control". In: *arXiv preprint arXiv:2107.05616* (2021).

[15] Yann LeCun et al. "A tutorial on energy-based learning". In: *Predicting structured data* 1.0 (2006).

[16] Miguel Sousa Lobo et al. "Applications of second-order cone programming". In: *Linear Algebra and its Applications* 284.1-3 (1998), pp. 193–228.

[17] Zachary Manchester and Scott Kuindersma. "Variational contact-implicit trajectory optimization". In: *Robotics Research*. Springer, 2020, pp. 985–1000.

[18] Zachary R. Manchester and Mason A. Peck. "Quaternion variational integrators for spacecraft dynamics". In: *Journal of Guidance, Control, and Dynamics* 39.1 (2016), pp. 69–76.

[19] Josep Marti-Saumell et al. "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] Carlos Mastalli et al. "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.

[21] Jean Jacques Moreau. "On unilateral constraints, friction and plasticity". In: *New Variational Techniques in Mathematical Physics*. Springer, 2011, pp. 171–322.

[22] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Second. Springer, 2006.

[23] Brendan O'Donoghue et al. "Conic optimization via operator splitting and homogeneous self-dual embedding". In: *Journal of Optimization Theory and Applications* 169.3 (2016), pp. 1042–1068.

[24] Samuel Pfrommer, Mathew Halm, and Michael Posa. "ContactNets: Learning Discontinuous Contact Dynamics with Smooth, Implicit Representations". In: *arXiv preprint arXiv:2009.11193* (2020).

[25] Michael Posa, Cecilia Cantu, and Russ Tedrake. "A direct method for trajectory optimization of rigid bodies through contact". In: *The International Journal of Robotics Research* 33.1 (2014), pp. 69–81.

[26] Marc H. Raibert et al. *Dynamically Stable Legged Locomotion*. Tech. rep. Massachusetts Institute of Technology Cambridge Artificial Intelligence Lab, 1989.

[27] Tamar Schlick. *Molecular modeling and simulation: an interdisciplinary guide*. Vol. 2. Springer, 2010.

[28] Vikas Sindhwani, Rebecca Roelofs, and Mrinal Kalakrishnan. "Sequential operator splitting for constrained nonlinear optimal control". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 4864–4871.

[29] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. "A review on bilevel optimization: From classical to evolutionary approaches and applications". In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 276–295.

[30] Bartolomeo Stellato et al. "OSQP: An operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672.

[31] Yuval Tassa, Tom Erez, and William D. Smart. "Receding Horizon Differential Dynamic Programming." In: *NIPS*. Citeseer. 2007, pp. 1465–1472.

[32] Yuval Tassa, Nicolas Mansard, and Emo Todorov. "Control-limited differential dynamic programming". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1168–1175.

[33] Russ Tedrake. "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for MIT 6.832)". In: *Downloaded in Fall* (2021).

[34] Emanuel Todorov, Tom Erez, and Yuval 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.

[35] Lieven Vandenberghe. "The CVXOPT linear and quadratic cone program solvers". In: *Online: http://cvxopt.org/documentation/coneprog.pdf* (2010).

[36] Oskar Von Stryk. "Numerical solution of optimal control problems by direct collocation". In: *Optimal Control*. Springer, 1993, pp. 129–143.

[37] Andreas Wächter and Lorenz T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical Programming* 106.1 (2006), pp. 25–57.

[38] Yang Wang and Stephen Boyd. "Fast model predictive control using online optimization". In: *IEEE Transactions on Control Systems Technology* 18.2 (2009), pp. 267–278.

[39] Gerhard Wanner and Ernst Hairer. *Solving Ordinary Differential Equations II*. Vol. 375. Springer Berlin Heidelberg, 1996.

[40] Zhaoming Xie, C. Karen Liu, and Kris Hauser. "Differential dynamic programming with nonlinear constraints". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 695–702.

[41] Simon Zimmermann, Roi Poranne, and Stelian Coros. "Dynamic manipulation of deformable objects with implicit integration". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 4209–4216.