NUMERICAL OPTIMIZATION FOR THINGS THAT MOVE: SIMULATION, PLANNING, AND CONTROL

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Taylor Athaniel Howell
January 2023

Contents

List of Tables

List of Figures

Acknowledgments

I am forever grateful to my research advisor Zachary Manchester for the years of mentorship and collaboration. His limitless energy and patience encouraged me at every stage to improve as a researcher and fostered my love for optimization.

My Stanford advisor, Allison Okamura, went beyond the call of duty to make my Ph.D. possible. Her graciousness throughout my time at Stanford provided me a feeling of complete support and enabled me to pursue my dream.

As an undergraduate at the University of Utah, Jake Abbott was an invaluable role model, providing early mentorship and instilling within me a drive for excellence in research.

To my labmates and collaborators, Simon Le Cleac'h, Brian Jackson, and Kevin Tracy, it was an incredible privilege to learn and work alongside you. I cherish the time we spent discussing ideas, making progress on challenging problems, and the shared excitement toward advancing robotics research.

The opportunities to perform research at Google Brain and DeepMind under the wonderful mentorship of Vikas Sindhwani and Yuval Tassa opened my mind to new and exciting research ideas and instilled a prioritization for excellent software engineering in my work.

To my friends from Stanford, you made these years in California beautiful. And to my oldest friend Brian, thank you for all of our long conversations and the inspiration to take a path that has brought so much fulfillment and excitement into my life.

Finally, I am deeply grateful to my parents for their endless love and support; encouraging me to pursue my interests and fostering a creative environment where it felt possible to achieve anything I set out to accomplish.

Chapter 1

Introduction

This dissertation takes an optimization-first approach to the development of tools for simulation, planning, and control for robotic systems. The first chapter contains technical background on numerical optimization that will be extensively utilized in this work. Each of the following chapters is based on a research paper and includes discussion of limitations and future work.

Chapter 2 provides background on numerical optimization [1] topics explored in this work, including trajectory optimization [2], simulation of rigid-body dynamics with contact [3], predictive control [4], and implicit differentiation [5]. First, planning is formalized as a trajectory optimization problem and two important algorithms: the linear quadratic regular [6] and a popular variant for non-convex problems, iterative linear quadratic regulator [7], are highlighted. Next, simulation for non-smooth mechanical systems experiencing impact and friction is formulated as a complementarity problem [8]. This is followed by a summary of predictive control. Then, two important algorithms for solving constrained optimization problems that are utilized extensively in this work, augmented Lagrangian [9] and interior-point [10] methods, are outlined. Finally, implicit differentiation is presented as an approach for efficiently differentiating through a numerical solver [11, 12].

Chapter 3 is based on the paper [13]. This is joint work with Simon Le Cleac'h, Zico Kolter, Mac Schwager, and Zachary Manchester. In this chapter we develop and implement a differentiable physics engine for rigid-body dynamics with contact: Dojo. This engine builds upon prior work for simulating smooth dynamical systems in maximal coordinates using variational integrators [14]. In this work, we develop a new friction model that utilizes techniques from cone programming [15] to support nonlinear friction cones in order to improve simulated stick-slip behavior. Next, we formulate a novel complementarity problem [16] that incorporates this friction model, maximal-coordinates representations, and a classic impact model that is amenable to optimization with interior-point methods [10]. To efficiently and reliably solve this problem to high accuracy we develop a custom primal-dual interior-point solver. Finally, we employ implicit differentiation [5] to

compute smooth analytical gradients that provide useful information through contact events by exploiting intermediate results from the solver. Experimental results including: simulation, planning, policy optimization, and system identification demonstrate the capabilities of the engine and the advantages of implicit gradients. An open-source implementation of the engine: `Dojo.jl`, written in the Julia programming language [17], is provided.

Chapter 4 is based on the paper [18]. This is joint work with Brian Jackson and Zachary Manchester. In this chapter we develop and implement an optimizer that is specialized for trajectory optimization problems with equality and inequality constraints: ALTRO. This work builds on previous work that adds support for constraints to the iterative linear quadratic regulator (iLQR) algorithm via the augmented Lagrangian method [19, 20]. First, we devise a novel square-root backward pass, inspired by the square-root Kalman filter [21], that has improved numerical conditioning properties. Next, we present problem reformulations for free-time and infeasible-initialization problems that endow our iLQR-based algorithm with capabilities previous limited to direct trajectory optimization solvers. Finally, we devise a solution-polishing phase for the algorithm that takes coarse solutions from the primary iLQR phase and utilizes an active-set method [1] to rapidly refine the solution to high precision. An open-source implementation of the optimizer, written in Julia, `TrajectoryOptimization.jl` is provided.

Chapter 5 is based on the paper [22]. This work was developed during an internship at Google Brain and is in collaboration with Simon Le Cleac'h, Sumeet Singh, Pete Florence, Zachary Manchester, and Vikas Sindhwani. In this chapter we develop a bi-level approach for planning with a model represented as a constrained optimization problem. An upper-level problem, optimized with iLQR [23], plans trajectories and a lower-level interior-point solver [24] optimizes a dynamics model at each time step that minimizes an objective subject to equality and cone constraints. Implicit differentiation [5] is utilized to compute derivatives through the lower-level solver which are subsequently utilized in the upper-level planning problem. Manipulation and locomotion examples are provided to demonstrate the approach.

Chapter 6 is based on the paper [25]. This work is a collaboration with Kevin Tracy, Simon Le Cleac'h, and Zachary Manchester. We develop and implement a general-purpose solver for non-convex optimization problems: CALIPSO. Current state-of-the-art solvers, specifically: Ipopt [26] and SNOPT [27], have poor support for complementarity and second-order-cone constraints, which are necessary for many robotics applications. In this work we combined primal-dual augmented Lagrangian [28] and cone programming [29] ideas, as well as heuristics from Ipopt, to support these constraints in the non-convex problem setting. We analyze the algorithmic properties of the solver on a simple contact-implicit trajectory optimization problem [30] by considering the linear independence constraint qualification [31]. Then, we empirically validate the solver's performance for state-triggered constraints [32], a collection of contact-implicit trajectory optimization problems, and a policy optimization scenario. An open-source implementation of the solver, `CALIPSO.jl`,

written in Julia, is provided.

Chapter 7 is based on the paper [33], in collaboration with Chunjiang Fu and Zachary Manchester. In this work we develop an algorithm to optimize robust feedback policies by formulating a single optimization problem that jointly optimizes a reference trajectory, a feedback tracking policy, and a set of deterministically chosen sample trajectories. Our work builds on the prior work of DIRTREL [34], which propagates uncertainty through linear dynamics and explicitly differentiates through the linear quadratic regulator problem in order to optimize a tracking policy. In contrast, our method deterministically propagates uncertainty through nonlinear dynamics using the unscented transform [35] and directly optimizes the parameters of a policy. We demonstrate empirically that this generalized method exactly recovers linear quadratic regulator [6] policies in the case of linear dynamics, a quadratic objective, and Gaussian disturbances. Additionally, we demonstrate the capabilities of our method to optimize robust tracking policies with a collection of motion planning problems for underactuated, nonlinear dynamical systems, including synthesizing an output feedback policy.

Chapter 8 is based on the paper [36], and is joint work with Simon Le Cleac'h, Mac Schwager, and Zachary Manchester. Hardware experiments are a collaboration with Shuo Yang, Chi-Yen Lee, John Zhang, and Arun Bishop. We present a predictive control algorithm for controlling systems that make and break contact with their environments. Online, the policy tracks a reference trajectory that is generated offline using contact-implicit trajectory optimization [37]. To make the policy amenable to fast online optimization, the contact dynamics model, represented as a complementarity problem, is strategically approximated along the reference trajectory using a Taylor expansion. The resulting planning model comprises a sequence of time-varying linear complementarity problems. To further reduce the online computational cost of the planning model, we devise a custom linear-system solver that leverages offline partial factorization. We implement a custom primal-dual interior-point solver to optimize the resulting lower-level complementarity problems at each step in the planning horizon and a custom Gauss-Newton method for the upper-level planning problem. Real-time rates for the policy are achieved in simulation for a collection of locomotion examples. Hardware experiments performed on a Unitree Go1 quadruped [38] demonstrate the method's real-time performance and robustness to large external disturbances.

Chapter 9 is based on the paper [39] and is co-authored with Yuval Tassa, in collaboration with Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, and Tom Erez, during an internship at DeepMind. In this chapter we present an open-source tool, MuJoCo MPC, for real-time behavior synthesis using predictive control algorithms, built on top of the MuJoCo physics engine [40]. The aim of this work is to democratize predictive control algorithms by providing fast planners written in C/C++ that extensively utilize multi-threading; asynchronous simulation and planning that enables the tool to run locally on limited-compute hardware; and interactive simulation in order to accelerate behavior design for robotics applications, which can often take minutes, hours, or days

using available model-based or learning methods. Additionally, we present a simple derivative-free optimizer, Predictive Sampling, that is easy to understand and simple to implement. An open-source implementation, `MuJoCo MPC`, is provided.

Ultimately, this work aims to leverage powerful tools from numerical optimization to advance the capabilities of robotic systems. My hope is that this work, and further research that may build upon it, will enable robots to be more dynamic, perform useful everyday tasks, and eventually, help us explore our universe.

Chapter 2

Background

This chapter provides foundational background for numerical optimization, trajectory optimization, rigid-body dynamics with contact, predictive control, and implicit differentiation.

2.1 Numerical Optimization

Problems from every field of research can be formulated as numerical optimization problems. A myriad of algorithms have been designed to solve these problems ranging from grid search to derivative-free black-box methods to large-scale first-order gradient methods to efficient second-order approaches based on Newton's method. This thesis primarily focuses on constrained optimization with second-order methods for modeling dynamical systems, planning trajectories to generate useful behaviors, and control for robotic systems.

Problem. A canonical representation:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c(x) \\ \text{subject to} \quad & g(x) = 0, \\ & h(x) \in \mathcal{K}, \end{aligned} \tag{2.1}$$

has decision variables $x \in \mathbf{R}^n$, objective $c : \mathbf{R}^n \to \mathbf{R}$, equality constraints $g : \mathbf{R}^n \to \mathbf{R}^m$, and cone constraints $h : \mathbf{R}^n \to \mathbf{R}^p$ in the Cartesian product of cones $\mathcal{K}$.

The designer formulates their problem (2.1) and a numerical optimizer searches for a solution x^* that minimizes the objective while satisfying each of the constraints.

In the non-convex setting, there are limited guarantees that the optimizer will find the globally optimal solution, and for challenging problems, even converge to a locally optimal solution. For the robotics applications explored in this work, modeling non-smooth rigid-body dynamics with contact

and fast optimization for control, these difficulties are of paramount importance.

2.2 Trajectory Optimization

Many optimization problems are of a temporal nature, for example, planning state and action sequences for robot motion planning. It is possible to exploit this structure with specialized numerical solvers and significantly reduce the computation complexity of optimizing such problems.

Problem. The trajectory optimization problem is formulated as:

$$\begin{array}{lll} \underset{x_{1:T}, u_{1:T-1}}{\text{minimize}} & c_T(x_T) + \sum_{t=1}^{T-1} c_t(x_t, u_t) & \\ \text{subject to} & f_t(x_t, u_t) = x_{t+1}, & t = 1, \ldots, T-1, \\ & g_t(x_t, u_t) = 0, & t = 1, \ldots, T, \\ & h_t(x_t, u_t) \in \mathcal{K}_t, & t = 1, \ldots, T, \end{array} \tag{2.2}$$

for a dynamical system with state $x_t \in \mathbf{R}^{n_t}$, control inputs $u_t \in \mathbf{R}^{m_t}$, time index t, discrete-time dynamics $f_t : \mathbf{R}^{n_t} \times \mathbf{R}^{m_t} \to \mathbf{R}^{n_{t+1}}$, and stage-wise objective $c_t : \mathbf{R}^{n_t} \times \mathbf{R}^{m_t} \to \mathbf{R}$, equality constraints $g_t : \mathbf{R}^{n_t} \times \mathbf{R}^{m_t} \to \mathbf{R}^{p_t}$, and cone constraints $h_t : \mathbf{R}^{n_t} \times \mathbf{R}^{m_t} \to \mathbf{R}^{q_t}$, with the Cartesian product of convex cones $\mathcal{K}_t$, over a planning horizon T.

For robot dynamics and planning, problems with sizes on the order of hundreds to tens of thousands of decision variables can be efficiently optimized with second-order optimization algorithms, based on Newton's method. However, naïve algorithms will have cubic complexity in the planning horizon, state, action, and constraint dimensions.

Fortunately, certain problem formulations and specialized numerical solvers can greatly reduce this complexity, enabling online optimization in many cases.

2.3 Finite-Horizon Linear Quadratic Regulator

The canonical linear quadratic regulator (LQR) problem is a trajectory optimization problem (2.2) with affine dynamics and a quadratic objective. This convex quadratic program [15] can be solved efficiently using an algorithm derived from dynamic programming [41].

Problem. The LQR problem:

$$\begin{array}{ll}\underset{x_{1:T},\,u_{1:T-1}}{\text{minimize}} & \frac{1}{2}x_T^T W_T x_T + w_T^T x_T + \sum_{t=1}^{T-1} \frac{1}{2}x_t^T W_t x_t + w_t^T x_t + \frac{1}{2}u_t^T R_t u_t + r_t^T u_t + x_t^T H_t u_t \\ \text{subject to} & x_{t+1} = A_t x_t + B_t u_t + C_t, \quad t = 1, \ldots, T-1, \\ & (x_1 \text{ given}),\end{array} \tag{2.3}$$

was originally formulated by Kalman [6] and has problem data: $W_t \in \mathbf{S}_+^{n_t}, w_t \in \mathbf{R}^{n_t}, R_t \in \mathbf{S}_{++}^{m_t}, r_t \in \mathbf{R}^{m_t}, H_t \in \mathbf{R}^{n_t \times m_t}, A_t \in \mathbf{R}^{n_{t+1} \times n_t}, B_t \in \mathbf{R}^{n_{t+1} \times m_t}, C_t \in \mathbf{R}^{n_{t+1}}$, and $x_1 \in \mathbf{R}^{n_1}$ that are provided by the designer.

The globally optimal solution is represented as the following feedback policy:

$$u = \pi_t(x) = K_t x + k_t, \tag{2.4}$$

where:

$$K_t = -(R_t + B_t^T P_{t+1} B_t)^{-1}(H_t^T + B_t^T P_{t+1} A_t), \tag{2.5}$$

$$k_t = -(R_t + B_t^T P_{t+1} B_t)^{-1}(r_t + B_t^T P_{t+1} C_t + B_t^T p_{t+1}). \tag{2.6}$$

These terms (2.5, 2.6) are computed recursively using dynamic programming.

Dynamic programming solution. First, we assume that the value function has a quadratic form:

$$V_t(x) = \frac{1}{2}x^T P_t x + p_t^T x, \tag{2.7}$$

with $P_t \in \mathbf{S}_{++}^{n_t}$ and $p_t \in \mathbf{R}^{n_t}$. At the final time step T the value function is determined:

$$P_T = W_T, \tag{2.8}$$

$$p_t = w_T. \tag{2.9}$$

Now, in a backward fashion, the value function is computed at preceding time steps:

$$\begin{aligned} P_t = W_t + A_t^T P_{t+1} A_t + K_t^T (R_t + B_t^T P_{t+1} B_t) K_t \\ + K_t^T (H_t^T + B_t^T P_{t+1} A_t) + (H_t + A_t^T P_{t+1} B_t) K_t, \end{aligned} \tag{2.10}$$

$$\begin{aligned} p_t = w_t + A_t^T P_{t+1} C_t + A_t^T p_{t+1} + K_t^T (R_t + B_t^T P_{t+1} B_t) k_t \\ + K_t^T (r_t + B_t^T P_{t+1} C_t + B_t^T p_{t+1}) + (H_t + A_t^T P_{t+1} B_t) k_t. \end{aligned} \tag{2.11}$$

The feedback policy is then computed using the optimal value function.

Optimal trajectories. The optimal trajectory, $\tau = (x_1, u_1, \ldots, x_T)$, and associated duals, $\Lambda = (\lambda_1, \ldots, \lambda_{T-1})$, are computed via a closed-loop forward rollout of the dynamics using the optimal policy from the initial state x_1:

$$u_t = K_t x_t + k_t, \tag{2.12}$$

$$x_{t+1} = A_t x_t + B_t u_t + C_t, \tag{2.13}$$

$$\lambda_t = P_t x_t + p_t. \tag{2.14}$$

2.4 Iterative Linear Quadratic Regulator

For non-convex trajectory optimization problems, Iterative LQR (iLQR) is an efficient algorithm that leverages solutions to LQR problems (2.3).

Problem. The following instance:

$$\begin{array}{ll} \underset{u_{1:T-1}}{\text{minimize}} & c_T(x_T) + \sum_{t=1}^{T-1} c_t(x_t, u_t) \\ \text{subject to} & x_{t+1} = f_t(x_t, u_t), \quad t = 1, \ldots, T-1, \\ & (x_1 \text{ given}), \end{array} \tag{2.15}$$

has (potentially) non-convex discrete-time dynamics and costs. The algorithm requires twice differentiable objective and dynamics, and utilizes derivatives of these functions. Additionally, only the controls are optimized and the state trajectory is recovered via forward rollouts.

Backward pass. A Taylor series approximates the objective to second order and the dynamics to first order about a nominal trajectory, $\bar{\tau} = (\bar{x}_1, \bar{u}_1, \ldots, \bar{x}_T)$, denoted with an overbar ($\bar{}$). This

expansion is utilized as problem data:

$$W_t = [c_t]_{xx}(\bar{x}_t, \bar{u}_t), \tag{2.16}$$
$$R_t = [c_t]_{uu}(\bar{x}_t, \bar{u}_t), \tag{2.17}$$
$$H_t = [c_t]_{xu}(\bar{x}_t, \bar{u}_t), \tag{2.18}$$
$$w_t = [c_t]_{x}(\bar{x}_t, \bar{u}_t), \tag{2.19}$$
$$r_t = [c_t]_{u}(\bar{x}_t, \bar{u}_t), \tag{2.20}$$
$$A_t = [f_t]_{x}(\bar{x}_t, \bar{u}_t), \tag{2.21}$$
$$B_t = [f_t]_{u}(\bar{x}_t, \bar{u}_t), \tag{2.22}$$
$$C_t = 0, \tag{2.23}$$

to solve an LQR problem (2.3).

Forward pass. The resulting LQR feedback policy (2.4) is then utilized to rollout the dynamics from an initial state x_1:

$$u_t = \bar{u}_t + K_t(x_t - \bar{x}_t) + \alpha k_t, \tag{2.24}$$
$$x_{t+1} = f_t(x_t, u_t). \tag{2.25}$$

Importantly, because the expansion used to solve the LQR problem is only valid in a (potentially small) region around the nominal trajectory, which is not known *a priori*, a line search over the parameter $\alpha \in [0, 1]$ is performed.

The trajectory with the lowest cost defined by the objective is selected as the new nominal trajectory, i.e., $\bar{\tau} \leftarrow \tau$. The procedure of alternating backward and forward passes is typically performed repeatedly until the change in objective value between iterations becomes small or a desired objective value is achieved.

2.5 Rigid-Body Dynamics with Contact

Non-smooth rigid-body dynamics with contact can be posed as a complementarity problem [8]. In this formulation, both impacts and friction are modeled as constraints on the system's configuration and contact impulses.

Impact. For a system with a single contact point, we define a configuration-dependent signed-distance function, $\phi : \mathbf{Q} \to \mathbf{R}$:

$$\phi(q) \geq 0. \tag{2.26}$$

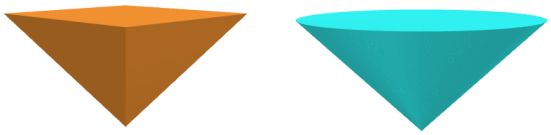


Figure 2.1: Friction-cone comparison. Linearized double-parameterized (left) and nonlinear second-order (right) cones.

An impact impulse with magnitude $\gamma \in \mathbf{R}$ is applied at the contact points in the direction of the surface normal in order to enforce (2.26) and prevent interpenetration. A non-negative constraint:

$$\gamma \geq 0, \tag{2.27}$$

enforces physical behavior that an impulse is repulsive (e.g., the floor does not attract bodies), and the complementarity condition:

$$\gamma \cdot \phi(q) = 0, \tag{2.28}$$

encodes switching behavior, e.g., enforcing zero force if the body is not in contact and allows non-zero force during contact.

Friction. Coulomb friction [42] models stick-slip behavior with friction that instantaneously maximizes the dissipation of kinetic energy between two objects in contact while remaining within a friction-cone. For a single contact point, this physical phenomenon can be modeled by the following optimization problem:

$$\begin{array}{ll} \underset{b}{\text{minimize}} & \nu^T b \\ \text{subject to} & \|b\|_2 \leq \mu\gamma, \end{array} \tag{2.29}$$

where $\nu \in \mathbf{R}^2$ is the tangential velocity at the contact point, $b \in \mathbf{R}^2$ is the friction force, and $\mu \in \mathbf{R}_+$ is the coefficient of friction between two objects [42].

This problem is naturally a convex second-order cone program, and can be efficiently and reliably solved [43]. However, classically, an approximate version of (2.29):

$$\begin{array}{ll} \underset{\beta}{\text{minimize}} & \nu^T D^T \beta \\ \text{subject to} & \beta^T \mathbf{1} \leq \mu\gamma, \\ & \beta \geq 0, \end{array} \tag{2.30}$$

formulated as a linear program, is instead solved. Here, the friction cone is linearized, (Fig. 2.1) and the friction vector, $\beta \in \mathbf{R}^{2d}$, is over parameterized and subject to additional non-negative constraints [8], and $D \in \mathbf{R}^{2d \times 2}$ is the parameterization mapping. Finer discretization (i.e., larger d)

of the friction cone can be utilized to better approximate the original problem, at the expense of a larger and computationally more expensive optimization problem.

The optimality conditions for (2.30) are:

$$\psi \cdot [\mu\gamma - \mathbf{1}^T\beta] = 0, \tag{2.31}$$

$$\beta \circ [D\nu - \psi\mathbf{1}] = 0, \tag{2.32}$$

$$\psi, \beta, [\mu\gamma - \mathbf{1}^T\beta], [D\nu - \psi\mathbf{1}] \geq 0, \tag{2.33}$$

where $\psi \in \mathbf{R}$ is a dual variable that can be interpreted as the magnitude of the contact-point tangential velocity, $\mathbf{1}$ is a vector of ones, and $\circ$ is an element-wise product.

The primary drawback to this formulation is that the optimized friction force will naturally align with the vertices of the cone approximation, which may not align with the velocity vector of the contact point. This can lead to simulation artifacts such as creep. The reason this set of constraints is often preferred in practice is because they satisfy the requirements for a linear complementarity problem.

Linear complementarity problem. Contact dynamics are classically simulated with a velocity-based time-stepping scheme formulated as a linear complementarity problem (LCP). This formulation combines the system's smooth dynamics with impact (2.26-2.28) and friction constraints (2.31-2.33) in order to optimize the next velocity, $v \in \mathbf{R}^{n_v}$ of the system by finding the physically correct contact impulses, $\lambda = (\beta, \gamma)$. The next configuration $q \in \mathbf{R}^{n_q}$ is recovered from the solution: $q = q_- + hv$. Values at the previous time step are indicated with negative subscripts ($_-$). For a single contact point, dynamics are formulated as follows:

$$\begin{aligned} \text{find} \quad & v, \lambda, \psi \\ \text{subject to} \quad & M(q_-)(v - v_-)/h + C(q_-, v_-) = J(q_-)^T\lambda, \\ & \gamma \circ [\phi(q_-) + \phi_q(q_-) \cdot hv] = 0, \\ & \psi \cdot \left[\mu\gamma - \mathbf{1}^T\beta\right] = 0, \\ & \beta \circ [DJ(q_-)v + \psi\mathbf{1}] = 0, \\ & \phi, \gamma, [\mu\gamma - \mathbf{1}^T\beta], [DJ(q_-)v + \psi\mathbf{1}], \beta, \psi \geq 0, \end{aligned} \tag{2.34}$$

where $M \in \mathbf{S}^{n_v}_{++}$ is the mass matrix, $C \in \mathbf{R}^{n_v}$ is the dynamics bias, $J \in \mathbf{R}^{(2d+1)\times n_v}$ is the contact Jacobian that maps impulses at the contact point into the velocity space, and $h \in \mathbf{R}_{++}$ is the time step. The terms M, C, and J are evaluated at the previous configuration, and a first-order approximation of the signed distance function at the previous configuration is utilized. This formulation generalizes to multiple contacts.

Specialized solvers exist for LCPs. Typically, these optimizers rely on active-set methods that

utilize pivoting algorithms to search the space of valid complementarity conditions [44].

2.6 Contact-Implicit Trajectory Optimization

Direct trajectory optimization can utilize the LCP contact dynamics formulation to plan trajectories for systems that make and break contact with their environment [30] without requiring hybrid dynamics [45] or explicitly enumerating all of the possible sequences of contact configurations. This enables the optimizer to potentially generate motion plans without pre-specified contact plans using task-level specifications via an objective.

Problem. The contact-implicit trajectory optimization problem:

$$\begin{aligned}
&\underset{\substack{x_{1:T},u_{1:T-1},\\ \lambda_{1:T-1},\psi_{1:T-1}}}{\text{minimize}} && c_T(x_T) + \sum_{t=1}^{T-1} c_t(x_t, u_t) && && (2.35)\\
&\text{subject to} && M(q_t)(v_{t+1} - v_t)/h + C(q_t, v_t) - B(q_t)u - J(q_t)^T\lambda_t = 0, && t = 1, \ldots, T-1,\\
& && q_t + hv_{t+1} - q_{t+1} = 0, && t = 1, \ldots, T-1,\\
& && \gamma_t \circ \phi(q_{t+1}) = 0, && t = 1, \ldots, T-1,\\
& && \psi_t \cdot \left[\mu\gamma_t - \mathbf{1}^T\beta_t\right] = 0, && t = 1, \ldots, T-1,\\
& && \beta_t \circ [DJ(q_t)v_{t+1} + \psi_t\mathbf{1}] = 0, && t = 1, \ldots, T-1,\\
& && \phi(q_{t+1}), \gamma_t, [\mu\gamma_t - \mathbf{1}^T\beta], [DJ(q_t)v_{t+1} + \psi_t\mathbf{1}], \beta_t, \psi_t \geq 0, && t = 1, \ldots, T-1,\\
& && (x_1 \text{ given}), &&
\end{aligned}$$

with states, $x = (q, v)$, directly encodes the LCP constraints and aims to minimize an objective. This formulation generalizes to multiple contacts.

Complementarity reformulation. To work well in practice with general-purpose off-the-shelf solvers for non-convex problems, the complementarity constraints are reformulated using an exact ℓ_1-norm penalty [37]:

$$\begin{array}{lll} \text{find} & a, b & \\ \text{subject to} & a \circ b = 0, & \\ & a, b \geq 0 & \end{array} \quad \rightarrow \quad \begin{array}{lll} \underset{a,b,s}{\text{minimize}} & \rho s & \\ \text{subject to} & s\mathbf{1} - a \circ b \geq 0, & \\ & a, b, s \geq 0 & \end{array} \tag{2.36}$$

This formulation relaxes the complementarity constraints and empirically results in superior convergence properties. As $\rho \to \infty$, we have $s \to 0$, and the original formulation is recovered. Despite the reformulation's practical performance, it requires additional decision variables and careful selection of the initial penalty parameter, $\rho \in \mathbf{R}_+$.

2.7 Predictive Control

A powerful tool for controlling complex systems is predictive control (PC) [46]. These algorithms leverage fast dynamics and planning algorithms to perform online optimization. This framework has been successfully deployed in numerous real-world settings including: control for chemical and nuclear processes [47, 48], navigation for autonomous vehicles [49], and whole-body control of humanoid robots [50].

In this framework, a model and the current state are utilized to predict the future evolution of the system under candidate action sequences. Numerical optimization is performed online to search for improved sequences. The best plan is utilized to control the system until a new (hopefully) improved plan has been computed.

Problem. The predictive control policy:

$$u \leftarrow \pi(x) \begin{cases} \underset{x_{1:T}, u_{1:T}}{\text{minimize}} & \sum_{t=1}^{T} c_t(x_t, u_t) \\ \text{subject to} & x_{t+1} = f_t(x_t, u_t), \quad t = 1, \dots, T, \\ & x_1 = x \end{cases} \tag{2.37}$$

is formulated as an optimization problem where T is the planning horizon, typically much shorter than the horizon we care to control the system. The optimized action sequence is utilized to return an action from the policy. If replanning is performed at a high rate, the resulting sequence of open-loop plans will effectively perform feedback.

Practical enhancements. There are three key enhancements for PC algorithms that lead to its practical performance. First, actions returned by the policy from approximate solutions to the internal optimization problem are often highly effective. As a result, since it is not necessary to solve the problem to convergence the computational burden is substantially reduced, enabling faster replanning, which typically leads to superior feedback.

Second, myopic planning horizons are typically sufficient to achieve desired long-term behavior (i.e., minimizing total returns over the full planning horizon). This enables simpler models that may be easier to construct and less expensive to evaluate to be used during forward prediction since simulating accurate long-term behavior is not necessary.

Third, the temporal nature of the control problem results in sequential policy evaluations that will solve similar optimization problems. As a result, warm starting, where the solution to a previous problem is used as a starting point for the current problem, can be effectively utilized. Employing this technique can greatly reduce the computational cost of policy evaluations.

2.8 Augmented Lagrangian Method

For equality-constrained problems, the augmented Lagrangian method [9] is a simple and effective algorithm, particularly when the problem is non-convex. This method transforms the constrained problem into an unconstrained form, then alternates between minimizing an augmented objective and performing updates to the dual variables and a penalty parameter.

Problem. An equality-constrained problem:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c(x) \\ \text{subject to} & g(x) = 0, \end{array} \tag{2.38}$$

is transformed into an unconstrained problem by introducing dual variables, $\lambda \in \mathbf{R}^m$, and a quadratic penalty parameterized by $\rho \in \mathbf{R}_+$:

$$L_{\mathcal{A}}(x; \lambda, \rho) = c(x) + \lambda^T g(x) + \frac{\rho}{2} g(x)^T g(x), \tag{2.39}$$

where we refer to $L_{\mathcal{A}}$ as the *augmented Lagrangian* for the problem (2.38).

Primal method. The classic method alternates between minimizing the augmented Lagrangian (2.39) and performing outer updates on the dual variables and penalty:

$$\lambda \leftarrow \lambda + \rho g(x), \quad \rho \leftarrow \phi(\rho), \tag{2.40}$$

until a solution to the original problem (2.38) is found [9]. This typically requires ten or fewer outer updates and a simple update, $\phi : \mathbf{R}_+ \to \mathbf{R}_+$, that scales the penalty by a constant value, works well in practice. Throughout, subscripts are used to denote derivatives and we drop the variable dependence of the functions for clarity. The KKT system for this method is:

$$\Big[c_{xx} + \rho g_x^T g_x + \sum_{i=1}^{m} (\lambda^{(i)} + \rho g^{(i)}) g_{xx}^{(i)}\Big] \Delta x = -\Big[c_x + g_x^T (\lambda + \rho g)\Big]. \tag{2.41}$$

Search directions $\Delta x \in \mathbf{R}^n$ are computed by solving the linear system (2.41) for fixed values of the dual variables and penalty. Newton's method with a line search is utilized to compute iterates that satisfy the KKT conditions, or residual, to a desired tolerance.

While this simple algorithm is effective at finding low to medium quality solutions, the KKT matrix becomes increasingly ill-conditioned as the penalty is increased in order to achieve better satisfaction of the equality constraints, degrading the quality of the Newton step. As a result, this method is best suited for applications where finding approximate solutions quickly is desirable, but highly accurate solutions are unnecessary.

Primal-dual method. To address the deficiencies of the primal method, a primal-dual method introduces additional dual variables, $y \in \mathbf{R}^m$, and constraints:

$$y = \lambda + \rho g(x), \tag{2.42}$$

in order to utilize an alternative KKT system with better numerical properties.

Combining these constraints (2.42) with the primal KKT system (2.41) and performing a simple manipulation of the new equations yields the *primal-dual* augmented-Lagrangian KKT system:

$$\begin{bmatrix} c_{xx} + \sum_{i=1}^{m} y^{(i)} g_{xx}^{(i)} & g_x^T \\ g_x & -\frac{1}{\rho} I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} c_x + g_x^T y \\ g + \frac{1}{\rho}(\lambda - y) \end{bmatrix}. \tag{2.43}$$

In contrast to the primal system (2.41), this system (2.43) does not become ill-conditioned as the penalty is increased because this term does not appear in the KKT matrix—only its inverse appears (i.e., $-\frac{1}{\rho} I$), which actually enhances the conditioning of the system by performing dual regularization [51, 28, 52]. Further, this method implicitly satisfies the linear independence constraint qualification (LICQ) [1], necessary for most second-order methods for constrained optimization, because the KKT matrix remains full rank even in cases where g_x is rank deficient as a result of the dual regularization [31].

Additionally, the KKT conditions now contain relaxed constraints (i.e., $g(x) + \frac{1}{\rho}(\lambda - y)$) that are particularly helpful for complementarity constraints since these elements of the residual are only satisfied in the limit as outer updates are performed.

Similar to the primal method, search directions $(\Delta x, \Delta y)$ are computed by solving the linear system (2.43) for fixed values of the dual variables estimates and penalty. Newton's method with a line search is utilized to compute iterates that satisfy the KKT conditions, or residual, to a desired tolerance. Outer updates to the dual estimates and penalty parameter are performed as before.

2.9 Interior-Point Method

For problems with hundreds to tens of thousands of decision variables and cone constraints, interior-point methods are efficient and return highly accurate solutions.

Problem. A cone program:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c(x) \\ \text{subject to} & x \in \mathcal{K}, \end{array} \tag{2.44}$$

minimizes an objective while satisfying a cone constraint. For robotics applications, common cones include the positive orthant (i.e., inequality constraints):

$$x^{(i)} \geq 0, \quad i = 1, \ldots, n \quad \rightarrow \quad x \in \mathbf{R}^n_+, \tag{2.45}$$

and second-order cones:

$$\|x^{(2:n)}\|_2 \leq x^{(1)} \quad \rightarrow \quad x \in \mathcal{Q}^n. \tag{2.46}$$

Primal method. An interior-point method transforms the original constrained problem (2.44) into an unconstrained form using a logarithmic barrier:

$$L_{\mathcal{B}}(x; \kappa) = c(x) - \kappa\phi(x), \tag{2.47}$$

where we refer to $L_{\mathcal{B}}$ as the *barrier Lagrangian*. The barrier functions are:

$$\phi(x) = \sum_{i=1}^{n} \log(x^{(i)}), \tag{2.48}$$

and

$$\phi(x) = \frac{1}{2}\log\Big((x^{(1)})^2 - (x^{(2:n)})^T x^{(2:n)}\Big), \tag{2.49}$$

for the positive orthant and second-order cone, respectively.

The classic method alternates between minimizing the barrier Lagrangian (2.47) while taking steps that ensure the cone constraint remains satisfied, and outer updates to the central-path parameter, $\kappa \in \mathbf{R}_+$, until a solution to the original problem (2.44) is found [15]. An effective strategy for the update is to decrease the parameter by a constant factor, $\kappa \to 0$.

The KKT system for this method is:

$$\Big[c_{xx} - \kappa\phi_{xx}\Big]\Delta x = -\Big[c_x - \kappa\phi_x\Big]. \tag{2.50}$$

As the central-path parameter is decreased, the logarithmic barrier becomes a closer approximation to the indicator function, which has an infinite cost if a constraint is violated and is otherwise zero [15]. While simple, this approach suffers from numerical ill-conditioning, similar to the primal augmented Lagrangian method, as the central-path parameter approaches zero, degrading a solver's ability to find accurate solutions to the original problem (2.44).

Primal-dual method. To address the conditioning issues of the primal method, additional dual variables, $z \in \mathbf{R}^n$, and complementarity constraints are introduced to form a new KKT system:

$$\begin{bmatrix} c_{xx} & -I \\ \mathbf{diag}(z) & \mathbf{diag}(x) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta z \end{bmatrix} = - \begin{bmatrix} c_x - z \\ x \circ z - \kappa \mathbf{e} \end{bmatrix}, \tag{2.51}$$

$$x \in \mathcal{K}, z \in \mathcal{K}^*. \tag{2.52}$$

Similar to the primal-dual augmented Lagrangian method, this KKT matrix does not depend on the central-path parameter, resulting in significantly better numerical conditioning than primal methods. The complementarity constraints in the KKT conditions are relaxed (i.e., $x \circ z - \kappa \mathbf{e}$), only being satisfied in the limit as the central-path parameter is decreased to zero. For the positive orthant, the target $\mathbf{e} = \mathbf{1}$ and the cone product $\circ$ denotes an element-wise product. For the second-order cone, the target is: $\mathbf{e} = (1, 0^{(n-1)})$, with cone product: $a \circ b = (a^T b,\, a^{(1)} b^{(2:n)} + b^{(1)} a^{(2:n)})$ [53, 29]. Both cones are self dual [15], and $\mathcal{K} = \mathcal{K}^*$.

With the new system, search directions $(\Delta x, \Delta z)$ are computed by solving the linear system (2.51) for fixed values of the central path parameter. Newton's method with a line search is utilized to compute iterates that satisfy the KKT conditions, or residual, to a desired tolerance, while respecting the cone constraints on the primal and dual variables. Updates to the central-path parameter are performed as before.

2.10 Implicit Differentiation

Many optimization problems can be formulated as finding a fixed point to an implicit function. An implicit function, $r : \mathbf{R}^a \times \mathbf{R}^b \to \mathbf{R}^a$, is defined as:

$$r(w^*; \theta) = 0, \tag{2.53}$$

for a fixed-point $w^* \in \mathbf{R}^a$ and problem data $\theta \in \mathbf{R}^b$. Further, it is often useful to compute the sensitivity of this solution with respect to the problem data, i.e., differentiate through the implicit function.

Computing this sensitivity is possible by applying the implicit-function theorem [5]. First, the function is approximated to first order at the solution:

$$\cancelto{0}{r} + \frac{\partial r}{\partial w} \delta w + \frac{\partial r}{\partial \theta} \delta \theta = 0. \tag{2.54}$$

Then, it is possible to solve for the relationship:

$$\frac{\partial w^*}{\partial \theta} = -\Big(\frac{\partial r}{\partial w}\Big)^{-1} \frac{\partial r}{\partial \theta}, \tag{2.55}$$

and recover the sensitivity.

Newton's method is typically employed to find solutions. When the method succeeds, the sensitivity (2.55) can be computed and the factorization of $\partial r/\partial w$ used to find the solution is reused to efficiently compute sensitivities at very low computational cost, using only back-substitution. In the case that $\partial r/\partial w$ is not full rank, an approximate solution, e.g., least-squares [15], can be computed, or regularization can be employed. Additionally, each element of the sensitivity can be computed in parallel.

Differentiating through an optimization problem. Implicit differentiation can also be used to compute the sensitivity of solutions, found by numerical optimizers, with respect to the optimization problem data. Numerical solvers find fixed points, comprising primal and dual variables, to the gradient of the problem's Lagrangian. This approach has been successfully applied to quadratic programs [11], and more generally, convex cone programs [12].

While it is possible to compute derivatives via finite-difference schemes, this approach requires at least as many calls to the solver as there are problem data elements. An alternative approach to computing gradients through a solver is to unroll the algorithm and utilizing the chain rule to differentiate sequentially through each iteration [54]. However, in practice, this approach requires truncating the number of iterates, which can lead to low-accuracy solutions, and this approach can be plagued by numerical issues that lead to exploding or vanishing gradients. Implicit differentiation is generally more efficient and able to return highly accurate sensitivities.

Chapter 3

Differentiable Contact Dynamics

We present Dojo, a differentiable physics engine for robotics that prioritizes stable simulation, accurate contact physics, and differentiability with respect to states, actions, and system parameters. Dojo achieves stable simulation at low sample rates and conserves energy and momentum by employing a variational integrator. A nonlinear complementarity problem, with second-order cones for friction, models hard contact and is reliably solved using a custom primal-dual interior-point method. Special properties of the interior-point method are exploited using implicit differentiation to efficiently compute smooth gradients that provide useful information through contact events. We demonstrate Dojo's unique ability to simulate hard contact while providing smooth, analytic gradients with a number of examples, including trajectory optimization, policy optimization, and system identification.

Dojo: A Differentiable Physics Engine for Robotics. Taylor A. Howell*, Simon Le Cleac'h*, Zico Kolter, Mac Schwager, and Zachary Manchester. arXiv 2203.00806. 2022.

3.1 Introduction

The last decade has seen immense resources devoted to network architectures, optimization algorithms, and the construction of large datasets by the learning community. These efforts have been leveraged to make impressive advances in robotics including: dexterous manipulation [55, 56], quadrupedal locomotion [57, 58], and pixels-to-torques control [59]. In contrast, there has been comparatively little work on the lowest level of the robotics stack: the *physics engine*. We argue that deficiencies in current widely used physics engines form a key bottleneck that must be overcome to enable future advancements in robotics.

Physics engines that simulate rigid-body dynamics with contact are utilized for trajectory optimization, reinforcement learning, system identification, and dataset generation for domains ranging from locomotion to manipulation. To be of practical value in real-world applications and overcome the sim-to-real gap an engine should provide stable simulation, accurately emulate a robot's dynamics, and ideally, be differentiable to enable use of efficient gradient-based optimization methods.

In recent years, a number of physics engines [60, 61, 62, 63, 64, 65] have been developed and utilized for robotics. In this work, we address key deficiencies of these tools including: high sample rates required for stable simulation that exacerbate the vanishing/exploding gradient problem and substantially increase sample complexities for rollout-based optimization methods, interpenetration of rigid bodies (e.g., a robot foot sinking through the floor) or creep (e.g., objects that should be at rest incorrectly sliding), and lack of informative gradients through contact events (e.g., subgradients resulting from naïve differentiation of non-smooth dynamics) or expensive gradients that require a large number of calls to the engine (e.g., finite-difference or stochastic-sampling schemes).

The Dojo physics engine is designed from the ground up to enable better and easier optimization for motion planning, control, reinforcement learning, system identification, and high-quality dataset generation. By taking a physics- and optimization-first approach to physics-engine design, we significantly advance the state of the art in *stable simulation*, *accurate contact physics*, and *differentiability* for robot simulation. Key attributes of Dojo include:

- Variational integration for stable simulation that is not sensitive to time step size
- A nonlinear complementarity problem (NCP) model for accurate contact dynamics
- A custom primal-dual interior-point method for reliably solving the NCP
- Smooth, analytic gradients that provide useful information through contact events

In the remainder of this chapter, we first provide an overview of related state-of-the-art physics engines in Section 3.2. Next, we present Dojo, and its key features, in Section 3.3. Simulation, planning, policy optimization, and system identification examples are presented in Section 3.4. Finally, we discuss Dojo's limitations in Section 3.5 and provide closing remarks in Section 3.6.

3.2 Related Work

This section provides an overview of physics engines that are commonly used in robotics. Table 3.1 summarizes the features of these tools and compares them to Dojo.

Many physics engines being developed and used in practice today were not designed for real-world robotics applications, and it is common for these tools to prioritize the *appearance* of realism over actual physical accuracy. Additionally, many of these physics engines were designed primarily for graphics and animation applications where fast simulation rates are prioritized and general-purpose numerical-optimization routines that do not natively support key elements from robotics domains, like cone constraints or quaternions, are commonplace. In this section, we provide background on a collection of popular engines, including: discussion of physical fidelity, underlying optimization routines, and ability to return gradients.

In the learning community, *MuJoCo* [40] has become a standard for benchmarking reinforcement learning algorithms using the OpenAI Gym environments [66]. MuJoCo utilizes minimal-coordinates representations and employs both semi-implicit Euler and explicit fourth-order Runge-Kutta integrators to simulate multi-body systems. These integrators often require small time steps, particularly for systems experiencing contact, and typically sample rates of hundreds to thousands of Hertz are required for stable simulation, which a mature and efficient implementation is able to achieve much faster than real-time rates. However, the high simulation rates can prove a challenge for control tasks like reinforcement learning settings where vanishing or exploding gradients are exacerbated over long horizons with many time steps.

Impact and friction are modeled using a smooth, convex contact model [67]. While this approach reliably computes contact forces, it introduces artificial damping, allowing the system to experience unphysical interpenetration and forces at a distance (i.e., while not in contact) and the default friction model introduces additional artifacts like velocity drift during sliding. Additionally, achieving good simulation behavior often requires system-specific tuning of multiple solver parameters. Further, the "soft" contact model is computed using a *primal* optimization method, meaning that as parameters are set to produce "hard," or more realistic contact, the underlying optimization problem becomes increasingly ill-conditioned and difficult to solve. As a result, it is often not possible to eliminate unphysical artifacts from the simulation and produce realistic results. Finally, analytical gradients are not provided by the engine, requiring finite-difference schemes.

Drake [60] was designed for robotics applications and its contact dynamics primarily rely on the classic time-stepping contact model that solves a linear complementarity problem (LCP) at each time step [8]. To satisfy the LCP problem formulation, a number of approximations are made to the dynamics and contact model, including the use of an approximate friction cone. To ensure stability of the simulation, small time steps are used where linearizations of the dynamics are valid, but importantly, the engine can achieve accurate hard contact. General-purpose LCP solvers that are typically used rely on a pivoting method like Lemke's algorithm [16]. Randomized smoothing has

been proposed as a method for returning gradients through contact [68] with this model. Higher-fidelity models are available for patch contacts [69], but they are computationally more expensive, requiring sophisticated higher-order implicit integrators.

The popular robotics simulator *Gazebo* [70] can utilize several different physics engines to simulate multi-body contact dynamics; Bullet [65] and DART [71] are common choices. Similar to Drake, these engines model hard contact dynamics with a LCP formulation. Automatic differentiation tools have also been utilized to compute gradients [65]. However, because of the discontinuous nature of contact dynamics, this approach will sometimes return sub-gradients, which do not provide useful information through contact events. Heuristics have been proposed to enumerate contact modes in order to select informative sub-gradients [62]. However, this approach scales poorly with the number of mode switches.

Engines designed for hardware accelerators (e.g., GPUs), including *Brax* [61] and *PhysX* [72], utilize simplified contact models that are amenable to low-precision data types (e.g., Float32). The benefit is massive parallel computation. However, these engines often require system-specific tuning to achieve stable simulation and the results are typically lower-fidelity.

Building on previous work [73], Dojo utilizes the open-source maximal-coordinates dynamics library `ConstrainedDynamics.jl` and efficient graph-based linear-system solver `GraphBasedSystems.jl` to compute smooth dynamics. However, unlike this prior work, Dojo has an improved contact model, specifically with regard to friction and its representation in the graph structure. Additionally, Dojo implements an efficient and reliable differentiable interior-point solver for complementarity problems.

The properties and characteristics of these existing engines are summarized in Table 3.1. We find that none of the existing engines prioritize two of the most important attributes for robotics: physical accuracy and useful differentiability. This motivates our development of a new physics engine for robotics applications.

Table 3.1: Comparison of physics engines used for robotics.

Engine	**Integrator**	**State**	**Contact**	**Solver**	**Gradients**
MuJoCo [40]	RK4	minimal	soft	Newton	finite-difference
Drake [60]	implicit Euler	minimal	soft/hard	LCP	randomized-smoothing
Bullet [74]	implicit Euler	minimal	soft/hard	LCP	sub-gradient
DART [71]	implicit Euler	minimal	hard	LCP	sub-gradient
PhysX [72]	explicit	minimal	soft	iterative	finite-difference
Brax [61]	explicit	maximal	soft	iterative	sub-gradient
Dojo	variational	maximal	hard	NCP	implicit gradient

3.3 Dojo

We now introduce Dojo's contact dynamics model and custom interior-point solver. This section presents the key algorithms and subroutines, including: variational integrators, the contact model, a differentiable primal-dual interior-point solver, and the computation of smooth gradients. An open-source implementation of the engine is provided.

3.3.1 Maximal-coordinates representation

Most multi-body physics engines utilize minimal- or joint-coordinates representations for dynamics because of the small number of states and convenience of implementation. This results in small, but dense systems of linear equations. In contrast, maximal-coordinates explicitly represent the position, orientation, and velocities of each body in a multi-body system. This produces large, sparse systems of linear equations that can be efficiently solved, including in the contact setting. We provide an overview, largely based on prior work [14], of this representation.

A single rigid body is defined by its mass and inertia, and has a configuration, $x = (p, q) \in \mathbf{X} = \mathbf{R}^3 \times \mathbf{H}$, comprising a position p and unit quaternion q, where $\mathbf{H}$ is the space of four-dimensional unit quaternions. We define the implicit discrete-time dynamics $F : \mathbf{X} \times \mathbf{X} \times \mathbf{X} \to \mathbf{R}^6$ as:

$$F(x_-, x, x_+) = 0, \tag{3.1}$$

where we indicate the previous and next time steps with minus $(-)$ and plus $(+)$ subscripts, respectively, and the current time step without decoration. A variational integrator is employed to simulate the next state of the system and has desirable energy and momentum conservation properties [3]. Linear and angular velocities are handled implicitly via finite-difference approximations.

For a multi-body system with bodies a and b connected via a joint—common types include: revolute, prismatic, and spherical—we introduce a constraint, $k : \mathbf{X} \times \mathbf{X} \to \mathbf{R}^l$, that couples the two bodies:

$$k^{ab}(x_+^a, x_+^b) = 0. \tag{3.2}$$

An impulse, $j \in \mathbf{R}^l$, where l is equal to the six degrees-of-freedom of an unconstrained body minus the joint's number of degrees-of-freedom, acts on both bodies to satisfy the constraint. The implicit integrator for the multi-body system has the form,

$$\begin{bmatrix} F^a(x_-^a, x^a, x_+^a) + K^a(x^a, x^b)^T j^{ab} \\ F^b(x_-^b, x^b, x_+^b) + K^b(x^a, x^b)^T j^{ab} \\ k^{ab}(x_+^a, x_+^b) \end{bmatrix} = 0, \tag{3.3}$$

where $K : \mathbf{X} \times \mathbf{X} \to \mathbf{R}^{l \times 6}$ is a mapping from the joint to the maximal-coordinates space and is related to the Jacobian of the joint constraint.

We can generalize (3.3) to include additional bodies and joints. For a system with N bodies and M joints we define a maximal-coordinates configuration $z = (x^{(1)}, \ldots, x^{(N)}) \in \mathbf{Z}$ and joint impulse $j = (j^{(1)}, \ldots, j^{(M)}) \in \mathbf{J}$. We define the implicit discrete-time dynamics of the maximal-coordinates system as:

$$F(z_-, z, z_+, j) = 0, \tag{3.4}$$

where $F : \mathbf{Z} \times \mathbf{Z} \times \mathbf{Z} \times \mathbf{J} \to \mathbf{R}^{6N}$. In order to simulate the system we find z_+ and j that satisfy (3.4) for a provided z_- and z using Newton's method.

By exploiting the mechanism's structure, we can efficiently perform root finding on (3.4) (see [14] for additional details). This structure is manifest as a graph of the mechanism, where each body and joint is considered a node, and joints have edges connecting bodies (Fig. 3.1). Because the mechanism structure is known *a priori*, a permutation matrix can be precomputed and used to perform efficient sparse linear algebra during simulation. For instance, in the case where the joint constraints form a system without loops, the resulting sparse system can be solved in linear time with respect to the number of links.

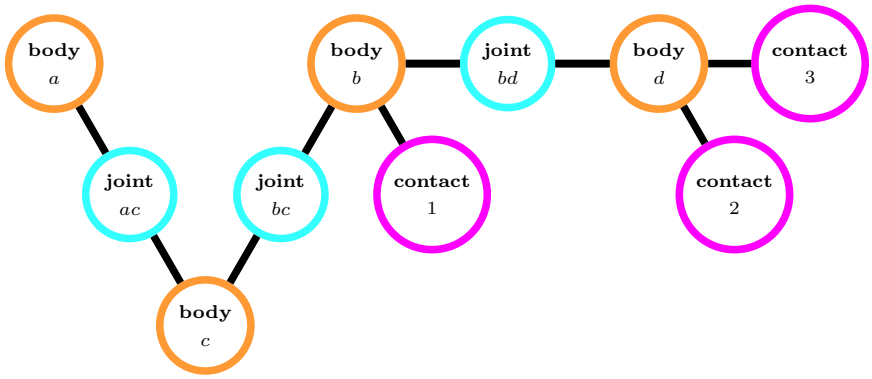


Figure 3.1: Graph structure for maximal-coordinates system with 4 bodies, 3 joints, and 3 points of contact.

3.3.2 Variational integrator

We use a specialized implicit integrator that preserves energy and momentum, natively handles quaternions, and alleviates spurious artifacts that commonly arise from contact interactions.

Dojo utilizes a maximal-coordinates state representation [14]. Each body has linear:

$$m(p_+ - 2p + p_-)/h - hmg - A(p)^T j - hf = 0, \tag{3.5}$$

and rotational:

$$\sqrt{1 - \psi_+^T \psi_+} J\psi_+ + \psi_+ \times J\psi_+ - \sqrt{1 - \psi^T \psi} J\psi + \psi \times J\psi - B(q)^T j - h^2 \tau / 2 = 0, \tag{3.6}$$

dynamics specified by mass $m \in \mathbf{R}_{++}$, inertia $J \in \mathbf{S}^3_{++}$, gravity $g \in \mathbf{R}^3$, and time step $h \in \mathbf{R}_{++}$. Equations (3.5, 3.6) are essentially second-order centered-finite-difference approximations of Newton's second law and Euler's equation for the rotational dynamics where $q_+ = q \cdot (\sqrt{1 - \psi_+^T \psi_+}, \psi_+)$ is recovered from a three-parameter representation $\psi \in \mathbf{R}^3$ [75], respectively. Joint impulses $j \in \mathbf{J}$ have linear $A : \mathbf{R}^3 \to \mathbf{R}^{\dim(\mathbf{J}) \times 3}$ and rotational $B : \mathbf{H} \to \mathbf{R}^{\dim(\mathbf{J}) \times 3}$ mappings into the dynamics. The configuration of a body $z^{(i)} = (p^{(i)}, q^{(i)}) \in \mathbf{R}^3 \times \mathbf{H}$ comprises a position and orientation represented as a quaternion. Forces and torques $f, \tau \in \mathbf{R}^3$ can be applied to the bodies.

Both (3.5) and (3.6) are derived by approximating Hamilton's Principle of Least-Action using a simple midpoint scheme [3, 37]. This approach produces *variational* integrators that automatically conserve momentum and energy [3].

3.3.3 Rigid-body dynamics with contact

Impact and friction behaviors are modeled, along with the system's dynamics, as a complementarity problem. This model simulates hard contact without requiring system-specific solver tuning. Additionally, contacts between a system and the environment are treated as a single graph node connected to a rigid body (Fig 3.1). As a result, the engine retains efficient linear-time complexity for open-chain mechanical systems.

Dojo uses the classic impact model (2.26-2.27) and in the following section we present a Coulomb friction model that utilizes an exact nonlinear friction cone.

Second-order friction cone. In contrast to the LCP formulation, we utilize the optimality conditions of (2.29) in a form amenable to a primal-dual interior-point solver. The associated cone program is,

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & v^T \beta_{(2:3)} \\ \text{subject to} \quad & \beta_{(1)} = \mu\gamma, \\ & \|\beta_{(2:3)}\|_2 \leq \beta_{(1)}, \end{aligned} \tag{3.7}$$

where subscripts indicate vector indices. The relaxed optimality conditions for (3.7) in interior-point form are:

$$v - \eta_{(2:3)} = 0, \tag{3.8}$$

$$\beta_{(1)} - \mu\gamma = 0, \tag{3.9}$$

$$\beta \circ \eta = \kappa \mathbf{e}, \tag{3.10}$$

$$\|\beta_{(2:3)}\|_2 \leq \beta_{(1)}, \; \|\eta_{(2:3)}\|_2 \leq \eta_{(1)}, \tag{3.11}$$

with dual variable $\eta \in \mathbf{R}^3$ associated with the second-order-cone constraints, and central-path parameter, $\kappa \in \mathbf{R}_+$. The benefits of this model are increased physical fidelity and fewer optimization

variables, without substantial increase in computational cost.

Complementarity problem. Rigid-body dynamics with a single contact (this formulation extends to multiple contacts) is simulated using a time-stepping scheme that solves the feasibility problem,

$$\begin{aligned} \text{find} \quad & z_+, j, \gamma, \beta, \eta \\ \text{subject to} \quad & F(z_-, z, z_+, j, \lambda, u) = 0, \\ & v_c(z, z_+) - \eta_{(2:3)} = 0, \\ & \beta_{(1)} - \mu\gamma = 0, \\ & \gamma \cdot \phi(z_+) = \kappa, \\ & \beta \circ \eta = \kappa \mathbf{e}, \\ & \gamma, \phi(z_+) \geq 0, \; \|\beta_{(2:3)}\|_2 \leq \beta_{(1)}, \; \|\eta_{(2:3)}\|_2 \leq \eta_{(1)}, \end{aligned} \tag{3.12}$$

with actions $u = (f^{(1)}, \tau^{(1)}, \ldots, f^{(N)}, \tau^{(N)}) \in \mathbf{U}$, and contact impulses $\lambda = (\beta_{(2:3)}, \gamma) \in \mathbf{\Lambda}$. The system's smooth dynamics $F : \mathbf{Z} \times \mathbf{Z} \times \mathbf{Z} \times \mathbf{J} \times \mathbf{\Lambda} \times \mathbf{U} \to \mathbf{R}^{6N}$ comprise linear and rotational dynamics (3.5-3.6) for each body [14]. The central-path parameter $\kappa \in \mathbf{R}_+$ and target $\mathbf{e}$ [29] are utilized by the interior-point solver in the following section.

Solving the feasibility problem finds a maximal-coordinates state representation. In many applications it is desirable to utilize a minimal-coordinates representation (e.g., direct trajectory optimization where algorithm complexity scales with the state dimension). Dojo includes functionality to analytically convert between representations, as well as formulate and apply the appropriate chain rule in order to differentiate through a representation transformation.

To simulate a system forward in time one step, given a control input and state comprising the previous and current configurations, solutions to a sequence of barriers problems (3.12) are found with $\kappa \to 0$. The central-path parameter has a physical interpretation as being the softness of the contact model. A value $\kappa = 0$ corresponds to exact "hard" or inelastic contact, whereas a relaxed value produces soft contact where contact forces can occur at a distance. The primal-dual interior-point solver described in the next section adaptively decreases this parameter in order to efficiently and reliably converge to hard contact solutions. In practice, the engine is set to converge to small values for simulation in order to simulate accurate physics. Intermediate solutions (i.e., $\kappa \neq 0$) are cached and later utilized to compute smooth gradients in order to provide useful information through contact events.

3.3.4 Solver

To efficiently and reliably satisfy (3.12), we developed a custom primal-dual interior-point solver for NCPs with support for cone constraints and quaternions. The algorithm is largely based upon Mehrotra's predictor-corrector algorithm [24, 1], while implementing non-Euclidean optimization techniques to handle quaternions [76] and borrowing features from CVXOPT [29] to handle cones.

The primary advantages of this algorithm are the correction to the classic Newton step, which can greatly reduce the iterations required by the solver (often halving the total number of iterations), and feedback on the problem's central-path parameter that helps avoid premature ill-conditioning and adaptively drives the complementarity violation to zero in order to reliably simulate hard contact.

Problem. The solver aims to satisfy instantiations of the following problem:

$$\begin{aligned} \text{find} \quad & a, b, c \\ \text{subject to} \quad & E(a, b, c; \theta) = 0, \\ & b \circ c = \kappa \mathbf{e}, \\ & b, c \in \mathcal{K}, \end{aligned} \tag{3.13}$$

with decision variables $a \in \mathbf{R}^{n_a}$ and $b, c \in \mathbf{R}^{n_b}$, equality-constraint set $E : \mathbf{R}^{n_a} \times \mathbf{R}^{n_b} \times \mathbf{R}^{n_b} \times \mathbf{R}^{n_\theta} \to \mathbf{R}^{n_a + n_b}$, problem data $\theta \in \mathbf{R}^{n_\theta}$; and where $\mathcal{K}$ is the Cartesian product of positive-orthant and second-order cones [15].

Interior-point methods aim to satisfy a sequence of relaxed problems with $\kappa > 0$ and $\kappa \to 0$ in order to reliably converge to a solution of the original problem (i.e., $\kappa = 0$). This continuation approach helps avoid premature ill-conditioning and is the basis for numerous convex and non-convex interior-point solvers [1].

The LCP formulation is a special-case instantiation (3.13) where the constraint set is affine in the decision variables, the cone is the positive orthant. Most general-purpose solvers for LCPs rely on active-set methods that strictly enforce $\kappa = 0$ at each iteration.

Residual and Jacobians. The interior-point solver aims to find a fixed point for the residual:

$$r(w; \theta, \kappa) = \begin{bmatrix} E(w; \theta) \\ b^{(1)} \circ c^{(1)} - \kappa \mathbf{1} \\ \vdots \\ b^{(n_\mathcal{K})} \circ c^{(n_\mathcal{K})} - \kappa \mathbf{e} \end{bmatrix}, \tag{3.14}$$

while respecting the cone constraints. The Jacobian of this residual with respect to the decision variables,

$$R(w; \theta) = \partial r(w; \theta, \cdot) / \partial w, \tag{3.15}$$

is used to compute a search direction. For convenience, we denote $w = (a, b, c)$. After a solution $w^*(\theta, \kappa)$ is found, the Jacobian of the residual with respect to the problem data,

$$D(w; \theta) = \partial r(w; \theta, \cdot)/\partial\theta, \tag{3.16}$$

is used to compute the sensitivity of the solution. These Jacobians are not explicitly dependent on the central-path parameter.

The non-Euclidean properties of quaternion variables are handled with modifications to these Jacobians (3.15) and (3.16) by right multiplying each with a matrix H containing attitude Jacobians [76] corresponding to the quaternions in x and θ, respectively:

$$\bar{R}(w; \theta) = R(w; \theta)H_R(w), \tag{3.17}$$

$$\bar{D}(w; \theta) = D(w; \theta)H_D(\theta). \tag{3.18}$$

Euclidean variables have corresponding identity blocks. This modification accounts for the implicit unit-norm constraint on each quaternion variable and improves the convergence behavior of the solver.

Analytical line search for cones. To ensure the cone variables strictly satisfy their constraints, a cone line search is performed for a candidate search direction. For the update:

$$y \leftarrow y + \alpha\Delta, \tag{3.19}$$

with step size α and search direction Δ, the solver finds the largest $\alpha \in [0, 1]$ such that $y + \alpha\Delta \in \mathcal{K}$. The step-size is computed analytically for the positive orthant:

$$\alpha = \min\left(1, \max_{k|\Delta_{(k)}<0}\left\{-\frac{y_{(k)}}{\Delta_{(k)}}\right\}\right), \tag{3.20}$$

and second-order cone:

$$\nu = y_{(1)}^2 - y_{(2:k)}^T y_{(2:k)}, \tag{3.21}$$

$$\zeta = y_{(1)}\Delta_{(1)} - y_{(2:k)}^T \Delta_{(2:k)}, \tag{3.22}$$

$$\rho_{(1)} = \frac{\zeta}{\nu}, \tag{3.23}$$

$$\rho_{(2:k)} = \frac{\Delta_{(2:k)}}{\sqrt{\nu}} - \frac{\zeta/\sqrt{\nu} + \Delta_{(1)}}{y_{(1)}/\sqrt{\nu} + 1}\frac{y_{(2:k)}}{\nu}, \tag{3.24}$$

$$\alpha = \begin{cases} \min\left(1, \frac{1}{\|\rho_{(2:k)}\|_2 - \rho_{(1)}}\right), & \|\rho_{(2:k)}\|_2 > \rho_{(1)}, \\ 1, & \text{otherwise.} \end{cases} \tag{3.25}$$

The line search over all individual cones is summarized in Algorithm 1.

Algorithm 1 Analytical Line Search For Cones

procedure ConeLineSearch($w, \Delta, \tau^{\text{ort}}, \tau^{\text{soc}}$)
  $\alpha_y^{\text{ort}} \leftarrow \alpha(y^{(1)}, \tau^{\text{ort}} \Delta^{y^{(1)}})$ ▷ Eq. 3.20
  $\alpha_z^{\text{ort}} \leftarrow \alpha(z^{(1)}, \tau^{\text{ort}} \Delta^{z^{(1)}})$ ▷ Eq. 3.20
  $\alpha_y^{\text{soc}} \leftarrow \min_{i \in \{2,\ldots,n\}} \alpha(y^{(i)}, \tau^{\text{soc}} \Delta^{y^{(i)}})$ ▷ Eq. 3.25
  $\alpha_z^{\text{soc}} \leftarrow \min_{i \in \{2,\ldots,n\}} \alpha(z^{(i)}, \tau^{\text{soc}} \Delta^{z^{(i)}})$ ▷ Eq. 3.25
  Return $\min(\alpha_y^{\text{ort}}, \alpha_z^{\text{ort}}, \alpha_y^{\text{soc}}, \alpha_z^{\text{soc}})$

Candidate update. The variables are partitioned: $a = (a^{(1)}, \ldots, a^{(p)})$, where $i = 1$ are Euclidean variables and $i = 2, \ldots, p$ are each quaternion variables; and $b = (b^{(1)}, \ldots, b^{(n)})$, $c = (c^{(1)}, \ldots, c^{(n)})$, where $j = 1$ is the positive-orthant and the remaining $j = 2, \ldots, n$ are second-order cones. For a given search direction, updates for Euclidean and quaternion variables are performed. The Euclidean variables in a use a standard update:

$$a^{(1)} \leftarrow a^{(1)} + \alpha \Delta^{(1)}, \tag{3.26}$$

For each quaternion variable, the search direction exists in the space tangent to the unit-quaternion hypersphere and is 3-dimensional. The corresponding update for $i = 2, \ldots, p$ is:

$$a^{(i)} \leftarrow L(a^{(i)}) \varphi(\alpha \Delta^{(i)}), \tag{3.27}$$

where $L : \mathbf{H} \rightarrow \mathbf{R}^{4 \times 4}$ is a matrix representing a left-quaternion matrix multiplication, and $\varphi : \mathbf{R}^3 \rightarrow \mathbf{H}$ is a mapping to a unit quaternion. The standard update is used for the remaining decision variables b and c.

Violation metrics. Two metrics are used to measure progress: The constraint violation,

$$r_{\text{vio}} = \| r(w; \theta) \|_\infty, \tag{3.28}$$

and complementarity violation,

$$\kappa_{\text{vio}} = \max_i \{ \| b^{(i)} \circ c^{(i)} \|_\infty \}. \tag{3.29}$$

The problem (3.13) is considered solved when $r_{\text{vio}} < r_{\text{tol}}$ and $\kappa_{\text{vio}} < \kappa_{\text{tol}}$.

Centering. The solver adaptively relaxes (3.13) by computing the centering parameters μ and σ. These values provide an estimate of the cone-constraint violation and determine the value of the central-path parameter that a correction step will aim to satisfy. These values rely on the degree of the cone [29]:

$$\mathbf{deg}(\mathcal{K}) = \sum_{i=1}^{n_\mathcal{K}} \mathbf{deg}(\mathcal{K}^{(i)}) = \mathbf{dim}(\mathcal{K}^{(1)}) + n_\mathcal{K} - 1, \tag{3.30}$$

the complementarity violations:

$$\mu = \frac{1}{\mathbf{deg}(\mathcal{K})} \sum_{i=1}^{n_\mathcal{K}} (b^{(i)})^T c^{(i)}, \tag{3.31}$$

and affine complementarity violations:

$$\mu^{\text{aff}} = \frac{1}{\mathbf{deg}(\mathcal{K})} \sum_{i=1}^{n_\mathcal{K}} (b^{(i)} + \alpha \Delta^{b^{(i)}})^T (c^{(i)} + \alpha \Delta^{c^{(i)}}), \tag{3.32}$$

as well as their ratio:

$$\sigma = \min\left(1, \max\left(0, \mu^{\text{aff}}/\mu\right)\right)^3, \tag{3.33}$$

As the algorithm makes progress, it aims to reduce these violations.

Algorithm. The interior-point algorithm used to solve (3.13) is summarized in Algorithm 2. Additional tolerances $\tau \in [0.9, 1]$ are used to improve numerical reliability of the solver. The algorithm parameters include $\tau_{\text{max}}^{\text{soc}}$ to prevent the iterates from reaching the boundaries of the cones too rapidly during the solve, τ_{min} to ensure we are aiming at sufficiently large steps, and β is the decay rate of the step size α during the line search. In practice, r_{tol} and κ_{tol} are the only parameters the user might want to tune.

Finally, the algorithm outputs a solution $w^*(\theta, \kappa)$ that satisfies the solver tolerance levels and, optionally, the implicit gradients of the solution with respect to the problem parameters θ.

For an instance of problem (3.13), the algorithm is provided problem data and an initial point, which is projected to ensure that the cone variables are initially feasible with some margin. Next, an affine search direction (i.e., predictor) is computed that aims for zero complementarity violation. Using this direction, a cone line search is performed followed by a centering step that computes a target relaxation for the computation of the corrector search direction. A second cone line search is then performed for this new search direction. A subsequent line search is performed until either the constraint or complementarity violation is reduced. The current point is then updated, a new affine search direction is computed, and the procedure repeats until the violations satisfy the solver tolerances.

Algorithm 2 Primal-Dual Interior-Point Solver

procedure OPTIMIZE($a_0, b_0, c_0, \theta, \mathcal{K}$)
  Parameters: $\tau^{\text{soc}}_{\text{max}} = 0.99, \tau_{\text{min}} = 0.95$
  $r_{\text{tol}} = 10^{-5}, \kappa_{\text{tol}} = 10^{-5}, \beta = 0.5$
  Initialize: $a = a_0, b = b_0 \in \mathcal{K}, c = c_0 \in \mathcal{K}$
  $r_{\text{vio}}, \kappa_{\text{vio}} \leftarrow$ VIOLATION(w) ▷ (3.28, 3.29)
  Until $r_{\text{vio}} < r_{\text{tol}}$ **and** $\kappa_{\text{vio}} < \kappa_{\text{tol}}$ **do**
    $\Delta^{\text{aff}} \leftarrow -\bar{R}^{-1}(w;\theta) r(w;\theta,0)$
    $\alpha^{\text{aff}} \leftarrow$ CONESEARCH($w, \Delta^{\text{aff}}, 1, 1$)
    $\mu, \sigma \leftarrow$ CENTER($b, c, \alpha^{\text{aff}}, \Delta^{\text{aff}}$) ▷ (3.30-3.33)
    $\kappa \leftarrow \max(\sigma\mu, \kappa_{\text{tol}}/5)$
    $\Delta \leftarrow -\bar{R}^{-1}(w;\theta) r(w;\theta,\kappa)$
    $\tau^{\text{ort}} \leftarrow \max(\tau_{\text{min}}, 1 - \max(r_{\text{vio}}, \kappa_{\text{vio}})^2)$
    $\tau^{\text{soc}} \leftarrow \min(\tau^{\text{soc}}_{\text{max}}, \tau^{\text{ort}})$
    $\alpha \leftarrow$ CONESEARCH($w, \Delta, \tau^{\text{ort}}, \tau^{\text{soc}}$)
    $c^*_{\text{vio}}, \kappa^*_{\text{vio}} \leftarrow r_{\text{vio}}, \kappa_{\text{vio}}$
      $\hat{w} \leftarrow$ UPDATE(w, Δ, α) ▷ (3.26, 3.27)
      $r_{\text{vio}}, \kappa_{\text{vio}} \leftarrow$ VIOLATION($\hat{w}$) ▷ (3.28, 3.29)
    Until $r_{\text{vio}} \leq c^*_{\text{vio}}$ **or** $\kappa_{\text{vio}} \leq \kappa^*_{\text{vio}}$ **do**
      $\alpha \leftarrow \beta\alpha$
      $\hat{w} \leftarrow$ UPDATE(w, Δ, α) ▷ (3.26, 3.27)
      $r_{\text{vio}}, \kappa_{\text{vio}} \leftarrow$ VIOLATION($\hat{w}$) ▷ (3.28, 3.29)
    end
    $w \leftarrow \hat{w}$
  end
  $\partial w^*/\partial\theta \leftarrow -\bar{R}^{-1}(w^*;\theta)\bar{D}(w^*;\theta)$ ▷ (2.55)
  Return $w, \partial w^*/\partial\theta$

3.3.5 Gradients

Interior-point methods solve non-smooth problems by optimizing a smooth barrier sub-problems, where the degree of smoothing is parameterized by the central-path parameter κ. In the contact setting, we employ this same technique to return gradients that are informative through contact events. Intermediate solutions, $w^*(\theta, \kappa > 0)$, are differentiated using the implicit-function theorem (2.55) to compute smooth *implicit gradients*. In practice, we find that these gradients greatly improve the performance of gradient-based optimization methods, consistent with the long history of interior-point methods. Dojo's gradients are compared with sub-gradients and randomized smoothing in Fig. 3.2.

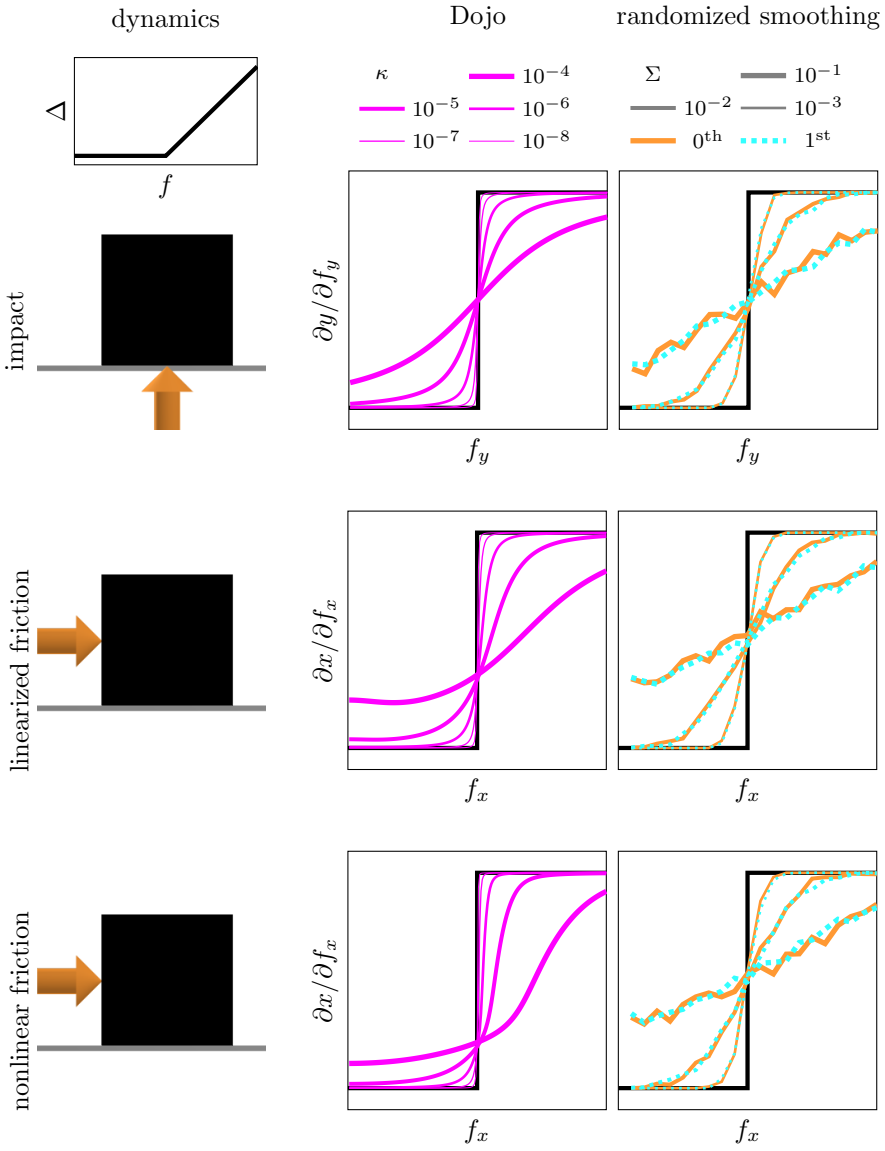


Figure 3.2: Gradient comparison between subgradients (black), randomized-smoothing gradients [68] (orange, blue) and Dojo's analytic gradients (magenta). The dynamics for a box in the XY plane that is resting on a flat surface and displaced an amount Δ by a force f (top left). Randomized smoothing gradients (right column) are computed using 500 samples with varying covariances Σ. Dojo's gradients (middle column) are computed for different values of central-path parameter κ. Compared to Dojo, the randomized smoothing method produces noisy derivatives that are many times more expensive to compute.

A wall-clock-time comparison of these gradients and sampled gradients is provided in Table 3.2, demonstrating that implicit gradients are more than an order of magnitude faster to compute.

Table 3.2: Compute-time ratio between Dojo's gradient and simulation-step evaluations for maximal-coordinates representation. We compute the engine's implicit gradient with respect to the initial configuration, velocity and control input. For a large system like Atlas, using a finite-difference (FD) scheme to evaluate the dynamics Jacobian in maximal coordinates would require at least 400 simulation-step evaluations. Alternatively, Dojo computes this Jacobian at the cost of approximately 4 simulation-step evaluations: a potential 100 times speedup on a single thread.

	Atlas	**Humanoid**	**Quadruped**	**Ant**	**Half-Cheetah**
Dojo	3.7	4.9	2.5	2.3	1.2
FD	472.6	194.7	170.3	197.0	94.8

The problem data for each simulation step include: the previous and current configurations, control input, and additional terms like the time step, friction coefficients, and parameters of each body.

3.3.6 Implementation

An open-source implementation, `Dojo.jl`, written in Julia, is available and a Python interface, `dojopy`, is also included. These tools, and the experiment, are available at:

`https://github.com/dojo-sim`

3.4 Results

Dojo's capabilities are highlighted through a collection of examples, including: simulating physical phenomena, gradient-based planning with trajectory optimization, policy-optimization, and real-to-sim system identification. The current implementation supports point, sphere, and capsule collisions with flat surfaces. All of the experiments were performed with an Intel Core i9-10885H CPU and 32GB of memory.

3.4.1 Simulation

The simulation accuracy of Dojo and MuJoCo is compared in a number of illustrative scenarios.

Impact comparison. The Atlas humanoid is simulated dropping on to a flat surface (Fig. 3.3). The system comprises 31 bodies, resulting in 403 maximal-coordinates states, and has 36 actuated degrees-of-freedom. Each foot has four contact points. The current implementation of Dojo simulates this system in real time at 65 Hz.

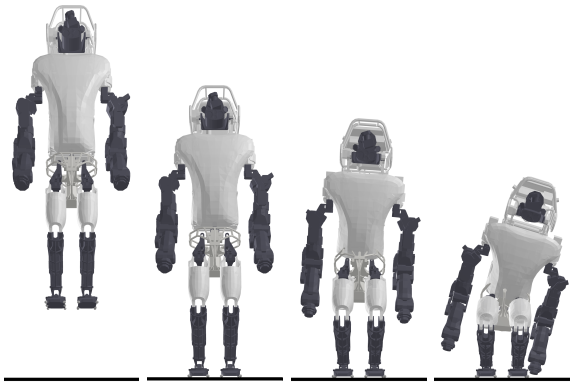


Figure 3.3: Atlas drop simulation. Dojo simulates this system with 403 maximal-coordinates states, 30 joint constraints, 36 inputs, and 8 contact points in real-time at 65 Hz. Dojo respects floor-feet penetration constraints to machine precision, while MuJoCo suffers from centimeters of interpenetration and is unstable at low simulation rates.

A comparison with MuJoCo is performed measuring penetration violations with the floor for different simulation rates (Table 3.3).

Table 3.3: Contact violation for Atlas drop. Comparison between Dojo and MuJoCo for foot contact penetration (millimeters) with the floor for different time steps (seconds). Dojo strictly enforces no penetration. When Atlas lands, its feet remains above the ground by an infinitesimal amount. In contrast, MuJoCo exhibits significant penetration through the floor (i.e., negative values).

Time Step	**0.1**	**0.01**	**0.001**
MuJoCo	failure	−28	−46
Dojo	**+1e-12**	**+1e-7**	**+8e-6**

Friction-cone comparison. The effect of friction-cone approximation is demonstrated by simulating a box that is initialized with lateral velocity before impacting and sliding along a flat surface. For a pyramidal approximation, in the probable scenario where its vertices are not aligned with the direction of motion, velocity drift occurs for a linearized cone implemented in Dojo and MuJoCo (Fig. 3.4).

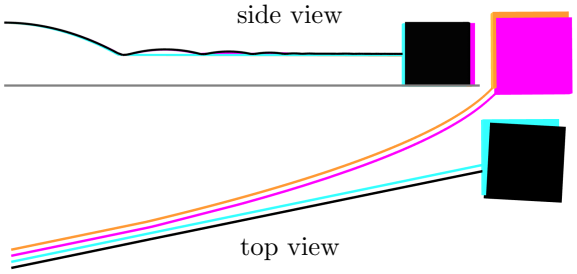


Figure 3.4: Velocity drift resulting from friction-cone approximation. Comparison between a box sliding with approximate cones having four vertices implemented in MuJoCo (magenta) and Dojo (orange) versus MuJoCo's (black) and Dojo's (blue) nonlinear friction cones. Dojo's nonlinear friction cone gives the physically correct straight line motion, while linear friction-cone approximations lead to lateral drift. MuJoCo's nonlinear friction cone exhibits a minor rotational drift.

The complementarity problem with P contact points requires $2P(1 + 2d)$ decision variables for contact and a corresponding number of constraints, where d is the degree of parameterization (e.g., double parameterization: $d = 2$). While it is possible to reduce such artifacts by increasing the number of vertices in the approximation of the second-order cone, this increases the computational complexity. Such approximation is unnecessary in Dojo as we handle the exact nonlinear cone constraint efficiently and reliably with optimization tools from cone programming; the result is accurate sliding.

Energy and momentum conservation. An accurate physics engine conserves important physical quantities like energy and momentum. Following the methodology from [77], we simulate "astronaut," a free-floating humanoid, and measure the drift of these quantities (Fig. 3.6).

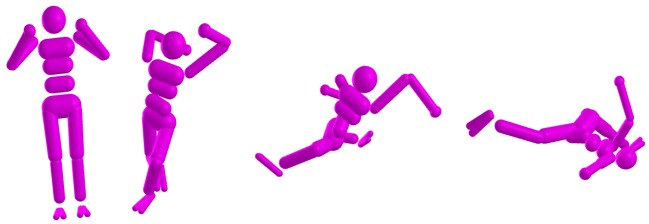


Figure 3.5: Astronaut simulation for energy and momentum conservation test. Joints are initialized with zero velocities and randomly actuated for 1 second. The simulation is visualized, from left to right.

There is no internal damping or springs, joint limits, or contact, and gravity is turned off. The astronaut is initialized with no linear or angular velocity and momentum drift is computed after one

second of uniformly sampled actuation: $u \sim \mathcal{U}(0, 0.05)$. Energy drift is computed over a 100 second period after 1 second of random actuation. MuJoCo exhibits drift in all scenarios. Characteristic of its variational integrator, Dojo conserves both linear and angular momentum to machine precision. Energy does not drift for Dojo but exhibits small bounded oscillations that decrease in amplitude as the time step decreases (Fig. 3.6). Conservation of energy to machine precision with variational integrators is possible and is a topic of current research [78].

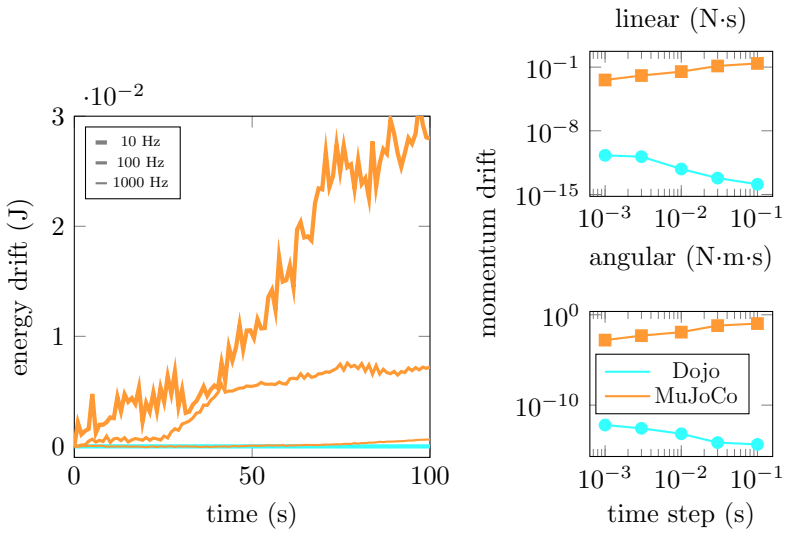


Figure 3.6: Energy and momentum conservation comparison between MuJoCo and Dojo for the astronaut simulation (Fig 3.5) using time steps ranging from 0.001 to 0.1 second. Momentum drift is measured after actuating the astronaut for 1 second with random controls. Energy drift is measured over a 100 second simulation after 1 second of random actuation.

3.4.2 Planning

Iterative LQR [22] utilizes implicit gradients from Dojo to perform trajectory optimization on three systems: planar box, hopper, and quadruped. A comparison is performed with MuJoCo and finite-difference gradients. The results for the quadruped are visualized in Fig. 3.7 and all tested systems are summarized in Table 3.4.

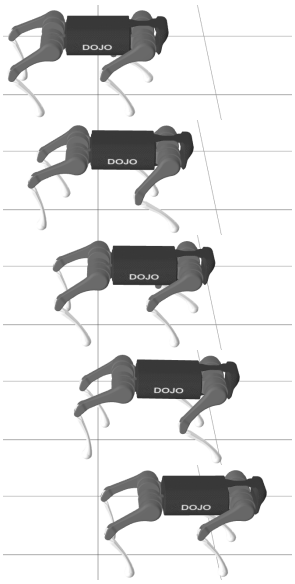


Figure 3.7: Locomotion plan for quadruped generated using trajectory optimization. Time progresses top to bottom.

Box. Inputs are optimized to move a stationary rigid body that is resting on a flat surface (Fig. 3.2) to a goal location that is either to the right or up in the air 1 meter. The planning horizon is 1 second and the controls are initialized with zeros. Dojo uses a time step $h = 0.1$, whereas MuJoCo uses $h = 0.01$ to prevent significant contact violations with the floor. MuJoCo fails in the scenario with the goal in the air, while Dojo succeeds at both tasks.

Hopper. The hopping robot [79] with $m = 3$ controls and $n = 14$ degrees-of-freedom is tasked with moving to a target pose over 1 second. Similar, although not identical, models and costs are used. Dojo uses a time step $h = 0.05$ whereas MuJoCo uses $h = 0.01$. The hopper is initialized with controls that maintain its standing configuration. Quadratic costs are used to penalize control effort and cost shaping is utilized to set target positions in the air and the goal pose. The optimizer typically finds a single-hop motion.

Quadruped. The torque-controlled Unitree A1 quadruped [80] with $m = 12$ controls and $n = 36$ degrees-of-freedom is tasked with moving to a goal location over a planning horizon $T = 41$ with time step $h = 0.05$. Controls are initialized to compensate for gravity and there are costs on tracking a target kinematic gait and control inputs. The optimizer finds a dynamically feasible motion that closely tracks the kinematic plan.

Table 3.4: Planning results. Comparison of final cost value, goal constraint violation, and total number of iterations for a collection of systems with maximal (max) and minimal (min) representations, optimized with iterative LQR [81] using Dojo with implicit gradients or MuJoCo (M) with finite-difference gradients.

System	Cost	Violation	Iterations
box right (max)	14.5	3e-3	**30**
box right (M)	**13.5**	**3e-3**	95
box up (max)	**14.5**	**3e-3**	**106**
box up (M)	failure	1.0	-
hopper (max)	10.2	4e-3	**57**
hopper (min)	**8.9**	**1e-3**	96
hopper (M)	26.7	2e-3	66
quadruped (min)	2e-2	3e-4	20

In the examples, gradients are computed with $\kappa = 3e{-}4$. Overall, we find that final results from both engines are similar. However, importantly, MuJoCo is enforcing soft contact whereas Dojo simulates hard contact. Further, MuJoCo requires a time step $h = 0.01$ for stable simulation, whereas Dojo works well with $h = 0.05$.

3.4.3 Policy optimization

Gym-like environments [66, 82]: ant and half-cheetah are implemented in Dojo and we train static linear policies for locomotion. As a baseline, we employ Augmented Random Search (ARS) [83], a gradient-free approach coupling random search with a number of simple heuristics. For comparison, we train the same policies using augmented gradient search (AGS) which replaces the stochastic-gradient estimation of ARS with the Dojo's implicit gradients. Policy rollouts are visualized in Fig. 3.8 and results are summarized in Table 3.5.

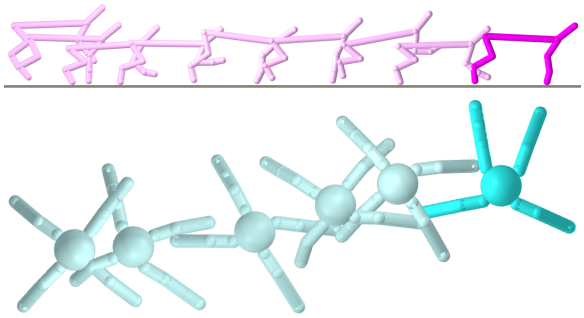


Figure 3.8: Learned policy rollouts for half-cheetah (top) and ant (bottom) generated using Augmented Random Search. Time progresses left to right.

Half-cheetah. This planar system [66] with $m = 6$ controls and $n = 18$ degrees-of-freedom is rewarded for forward velocity and penalized for control effort over a horizon $T = 80$ with time step $h = 0.05$.

Ant. The system [66] has $m = 8$ controls and $n = 28$ degrees-of-freedom and is rewarded for forward motion and staying alive and is penalized for control effort and contact over a horizon $T = 150$ with time step $h = 0.05$.

Table 3.5: Policy optimization results. Comparison of total reward, number of simulation-step and gradient evaluations for policies trained with Augmented Random Search (ARS) [83] and Augmented Gradient Search (AGS). The results are averaged over the best 3 out of 5 runs with different random seeds.

System	Reward	Simulation	Gradient
half-cheetah (ARS)	46 ± 24	3e+4	0
ant (ARS)	64 ± 15	2e+5	0
half-cheetah (AGS)	44 ± 24	**5e+3**	5e+3
ant (AGS)	54 ± 28	**2e+4**	2e+4

First, we are able to successfully train policies using this simple learning algorithm in Dojo's hard contact environments. Second, MuJoCo requires smaller $h = 0.01$ time steps for stable simulation, whereas Dojo is stable with $h = 0.05$. Third, our initial experiments indicate that it is possible to utilize Dojo's implicit gradients for more efficient training compared to the derivative-free approach.

3.4.4 System identification

System identification is performed on an existing real-world dataset of trajectories collected by throwing a box on a table with different initial conditions [84]. We learn a set of parameters

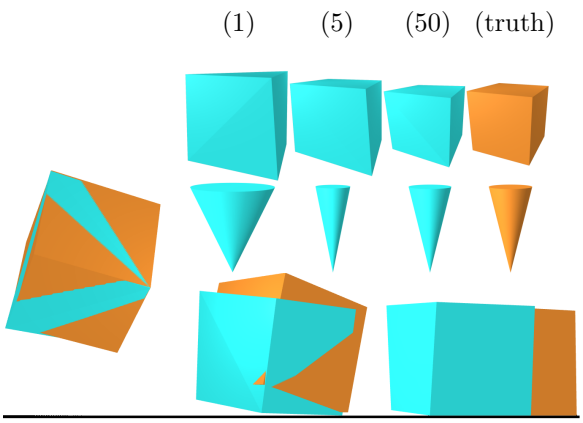


Figure 3.9: System identification. Top right: Learning box geometry and friction cone to less than 5% error. Bottom: Simulated trajectory of the box using the learned properties (blue) compared to ground truth (orange).

$\theta = (\mu, p^{(1)}, \ldots, p^{(8)})$ that include the friction coefficient μ, and 3-dimensional vectors $p^{(i)}$ that represent the position of vertex i of the box with respect to its center of mass.

Each trajectory is decomposed into $T-2$ triplets of consecutive configurations: $Z = (z_-, z, z_+)$, where T is the number of time steps in the trajectory. Using the initial conditions z_-, z from a tuple, and an estimate of the system's parameters θ, Dojo performs one-step simulation to predict the next state, $\hat{z}_+$. Implicit gradients are utilized by a Gauss-Newton method to learn the system parameters.

The parameters are learned by minimizing the following loss:

$$\mathcal{L}(\mathcal{D}, \theta) = \sum_{Z \in \mathcal{D}} L(Z, \theta) = \sum_{Z \in \mathcal{D}} \frac{1}{2} ||\mathbf{Dojo}(z_-, z; \theta) - z_+||_W^2, \tag{3.34}$$

where $||\cdot||_W$ is a weighted norm, which aims to minimize the difference between the ground-truth trajectories and physics-engine predictions. We use gradients:

$$\frac{\partial L}{\partial \theta} = \frac{\partial \mathbf{Dojo}}{\partial \theta}^T W \left(\mathbf{Dojo}(z_-, z; \theta) - z_+\right), \tag{3.35}$$

and approximate Hessians:

$$\frac{\partial^2 L}{\partial \theta^2} \approx \frac{\partial \mathbf{Dojo}}{\partial \theta}^T W \frac{\partial \mathbf{Dojo}}{\partial \theta}. \tag{3.36}$$

Gradients are computed with $\kappa = 3e{-}4$.

After training, the learn parameters are within 5% of the geometry and best-fit friction coefficient for the box from the dataset. We complete the transfer from real-to-sim and simulate the learned

system parameters in Dojo, comparing it to the ground-truth dataset trajectories. Results are visualized in Fig. 3.9.

3.5 Limitations

The computation cost of Dojo per simulation step is greater than many existing engines. The greater computational complexity enables superior simulation accuracy at the cost of increased wall-clock time and is a trade-off that must be considered by the user for a particular application. Additionally, because Dojo solves a nonlinear complementarity problem (i.e, non-convex problem) at each time step the engine provides no guarantees of converging to a solution. In practice, we do no find this to be a problem, but for time- or safety-critical applications this should be a consideration. Further, it remains to be seen how well sim-to-real transfer works, in comparison to existing engines, particularly for manipulation tasks with a large number of contact interactions.

3.6 Future work

Dojo is designed from physics- and optimization-first principles to enable better gradient-based optimization for motion planning, control, reinforcement learning, and system identification. The engine makes several advancements over previous state-of-the-art engines for robotics: First, the variational integrator enables stable simulation at low sample rates. Second, the contact model includes an improved friction model that eliminates artifacts like creep, particularly for sliding, and hard contact for impact is achieved to machine precision. This should enable superior sim-to-real transfer for both locomotion and manipulation applications. The underlying interior-point solver, developed specifically for solving NCPs, is numerically robust and requires practically no hyperparameter tuning for good performance across numerous systems, and handles cone and quaternion variables. Third, the engine efficiently returns implicit gradients that are smooth and analytical, providing useful information through contact events. Fourth, in addition to building and providing an open-source tool, the physics and optimization algorithms presented can improve many existing engines.

In terms of features, reliability, and wall-clock time, MuJoCo–the product of a decade of excellent software engineering–is impressive. As development of Dojo continues, we expect to make significant progress in all of these areas. However, fundamentally, Dojo's approach of solving an NCP with a primal-dual interior-point method will likely be computationally more expensive compared to MuJoCo's convex soft-contact model, which can never entirely recover accurate solutions even at higher sampling frequencies. This is the fundamental trade-off Dojo makes for robotics applications: greater computational cost for accurate physics and smooth gradients.

A number of future improvements to Dojo are planned. First, Dojo currently implements simple

collision detection (e.g., sphere-halfspace, sphere-sphere). Natural extensions include support for convex primitives and curved surfaces. Another improvement is adaptive time stepping. Similar to advanced numerical integrators for stiff systems, Dojo should take large time steps when possible and adaptively modify the time step in cases of numerical difficulties or physical inaccuracies. Finally, hardware-accelerator support for Dojo would potentially enable faster simulation and optimization.

Perhaps the most important remaining question is whether the physics and optimization improvements from this work translate into better transfer of simulation results to successes on real-world robotic hardware. In this thrust, future work will explore the transfer of control policies trained in Dojo to hardware and deployment of the engine in model predictive control frameworks.

In conclusion, we have presented a new physics engine, Dojo, specifically designed for robotics. This tool is the culmination of a number of improvements to the contact dynamics model and underlying optimization routines, aiming to advance state-of-the-art physics engines for robotics by improving physical accuracy and differentiability.

Acknowledgments

The authors would like to thank Jan Brüdigam for `ConstrainedDynamics.jl`, an open-source library which served as a foundation for Dojo's maximal-coordinates state representation, as well as early technical discussions and support; and Suvansh Sanjeev for assistance with the Python interface. Toyota Research Institute provided funds to support this work.

3.7 Appendix: Quaternion Algebra

In this section we introduce a set of conventions for notating standard quaternion operations, adopted from [14, 76], and employed in the rotational part of our variational integrator (3.6).

Quaternions are written as four-dimensional vectors:

$$q = (s, v) = (s, v_1, v_2, v_3) \in \mathbf{H}, \tag{3.37}$$

where s and v are scalar and vector components, respectively. Dojo employs unit quaternions (i.e., $q^T q = 1$) to represent orientation, providing a mapping from the local body frame to a global inertial frame.

Quaternion multiplication is represented using linear algebra (i.e., matrix-vector and matrix-matrix products). Left and right quaternion multiplication,

$$q^a \otimes q^b = \begin{bmatrix} s^a s^b - (v^a)^T v^b \\ s^a v^b + s^b v^a + v^a \times v^b \end{bmatrix} = L(q^a) q^b = R(q^b) q^a, \tag{3.38}$$

where $\times$ is the standard vector cross product, is represented using the matrices:

$$L(q) = \begin{bmatrix} s & -v^T \\ v & sI_3 + \mathbf{skew}(v) \end{bmatrix} \in \mathbf{R}^{4\times4}, \tag{3.39}$$

$$R(q) = \begin{bmatrix} s & -v^T \\ v & sI_3 - \mathbf{skew}(v) \end{bmatrix} \in \mathbf{R}^{4\times4}, \tag{3.40}$$

where,

$$\mathbf{skew}(x) = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}, \tag{3.41}$$

is defined such that,

$$\mathbf{skew}(x)y = x \times y, \tag{3.42}$$

and I_3 is a 3-dimensional identity matrix. The vector component of a quaternion,

$$v = Vq, \tag{3.43}$$

is extracted using the matrix:

$$V = \begin{bmatrix} \mathbf{0} & I_3 \end{bmatrix} \in \mathbf{R}^{3\times4}, \tag{3.44}$$

and quaternion conjugate:

$$q^\dagger = \begin{bmatrix} s \\ -v \end{bmatrix} = Tq, \tag{3.45}$$

is computed using:

$$T = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & -I_3 \end{bmatrix} \in \mathbf{R}^{4\times4}. \tag{3.46}$$

Chapter 4

Fast Constrained Trajectory Optimization

Trajectory optimization is a widely used tool with many important applications in robotic motion planning and control. However, most existing algorithm implementations fall into one of two categories: either they rely on general-purpose off-the-shelf solvers for non-convex problems that are numerically robust and capable of handling constraints but tend to be slow, or they use custom numerical methods that are fast but lack robustness and have limited or no ability to deal with constraints. This chapter presents ALTRO (Augmented Lagrangian TRajectory Optimizer), a novel algorithm for solving constrained trajectory optimization problems that bridges this gap by offering fast convergence, numerical robustness, and the ability to handle general state and action constraints.

ALTRO: A Fast Solver for Constrained Trajectory Optimization. Taylor A. Howell*, Brian Jackson*, and Zachary Manchester. International Conference on Intelligent Robots and Systems. 2019.

4.1 Introduction

Trajectory optimization is a powerful tool for planning, enabling the synthesis of dynamic motions for complex underactuated robotic systems. This general framework can be applied to robots with nonlinear dynamics and constraints where other motion planning paradigms—such as sample-based planning, inverse dynamics, or differential flatness—are impractical or ineffective.

Direct methods transcribe states and actions as decision variables and solve (2.2) using general-purpose solvers such as SNOPT [27] or Ipopt [26], and tend to be versatile and robust. It is straight forward to provide an initial state trajectory to the solver in such methods, even if it is dynamically infeasible. Direct transcription [85] and direct collocation [86] are common direct algorithms.

Alternatively, indirect methods leverage the structure of (2.2) to solve a sequence of smaller sub-problems using dynamic programming. These are anytime algorithms, meaning they are always dynamically feasible, allowing state and action trajectories at any iteration to be used in a tracking controller. However, it is often difficult to find a suitable initial guess for the action trajectory. Historically, indirect methods have been considered less robust and less suitable for reasoning about general state and action constraints, but they tend to be fast and amenable to implementation in embedded systems. Methods include Differential Dynamic Programming (DDP) [7] and Iterative LQR (iLQR)[81], as well as various shooting methods [87].

In this chapter we present ALTRO (Augmented Lagrangian TRajectory Optimizer), a trajectory optimization algorithm that combines the best characteristics of both direct and indirect methods, namely: speed, small problem size, numerical robustness, handling of state and action constraints, anytime dynamic feasibility, and infeasible state initialization. Our contributions are a number of enhancements to an algorithm that utilizes iLQR within an augmented Lagrangian framework to handle constraints, including:

- a numerically robust square-root backward pass
- reformulations for infeasible initialization and free-time problems
- a solution polishing phase that returns highly accurate solutions

In the remainder of the chapter, Section 4.2 provides an overview of related work. Then, Section 4.3 provides background on inequality-constrained augmented Lagrangian methods and matrix square roots. Next, we present ALTRO in Section 4.4. In Section 4.5 we highlight these solver enhancements with examples including a cart-pole and car. Finally, we discuss the algorithm's limitations in Section 4.6 and directions for future work in Section 4.7.

4.2 Related Work

Several methods have been proposed to incorporate constraints into indirect methods: box constraints on actions [88] and stage-wise inequality constraints on the states [89, 90] have been handled by solving a constrained quadratic program at each step of the backward pass; a projection method was devised that satisfies linearized terminal state and stage state-action constraints [91]; augmented Lagrangian methods have been proposed [20], including hybrid approaches that also solve constrained quadratic programs for stage state-action constraints [19, 90]; and mixed state-action constraints have also been handled using a penalty method [92].

4.3 Background

In this section we provide background on optimizing inequality-constrained problems using augmented Lagrangian methods and matrix square roots.

Inequality-constrained augmented Lagrangian method. Classically, augmented Lagrangian methods are utilized to handle equality-constrained optimization problems (2.38). However, with small modifications to the algorithm it is possible to also handle inequality constraints [93].

For mixed problems:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c(x) \\ \text{subject to} \quad & g(x) \{\leq, =\} 0, \end{aligned} \tag{4.1}$$

we formulate an augmented Lagrangian with indicator penalties:

$$L_{\mathcal{A}}(x; \lambda, \rho) = c(x) + \lambda^T g(x) + \frac{1}{2} g(x)^T I_\rho g(x), \tag{4.2}$$

where I_ρ is a diagonal matrix defined as:

$$I_\rho^{(i)} = \begin{cases} 0 & \text{if } g^{(i)}(x) < 0 \wedge \lambda^{(i)} = 0 \wedge i \in \textbf{inequality}, \\ \rho & \text{otherwise.} \end{cases} \tag{4.3}$$

The dual update becomes:

$$\lambda_+^{(i)} = \begin{cases} \lambda^{(i)} + \rho g^{(i)}(x) & i \in \textbf{equality}, \\ \max\left(0, \lambda^{(i)} + \rho g^{(i)}(x)\right) & i \in \textbf{inequality}, \end{cases} \tag{4.4}$$

and the penalty update:

$$\rho_+ = \phi(\rho), \tag{4.5}$$

remains a monotone update. The augmented Lagrangian procedure remains the same: minimize

(4.2) for fixed duals and penalty values, dual update (4.4), penalty update (4.5), and repeat until convergence to a desired constraint tolerance is achieved.

Matrix square roots. For matrices $A, B \in \mathbf{S}^n_{++}$, we can utilize the individual matrix square-roots $\sqrt{A}$ and $\sqrt{B}$ to compute $\sqrt{A+B}$ as follows:

$$A + B = [\sqrt{A}^T \ \sqrt{B}^T] \begin{bmatrix} \sqrt{A} \\ \sqrt{B} \end{bmatrix}, \tag{4.6}$$

$$= F^T F, \tag{4.7}$$

$$= R^T Q^T Q R, \tag{4.8}$$

$$= R^T R, \tag{4.9}$$

where $F \in \mathbf{R}^{2n \times n}$ is the matrix factorized by a QR decomposition, the product of two orthogonal matrices is the identity (i.e., $Q^T Q = I$), and the upper triangular matrix returned by the procedure:

$$\sqrt{A+B} = R, \tag{4.10}$$

is the solution. We define:

$$\sqrt{A+B} = \mathbf{QR}\Big(\begin{bmatrix} \sqrt{A} \\ \sqrt{B} \end{bmatrix} \Big). \tag{4.11}$$

The complexity of this operation is $\mathbf{O}(n^3)$ using the Householder algorithm.

We can also compute $\sqrt{A-B}$ using the individual matrix square-roots. We perform rank-one downdates on $\sqrt{A}$ using the rows of $\sqrt{B}$:

$$\sqrt{A-B} = \mathbf{DD}\Big(\sqrt{A}, \sqrt{B}\Big). \tag{4.12}$$

The complexity of this operation is also $\mathbf{O}(n^3)$.

4.4 ALTRO

ALTRO is a two-stage algorithm. The first stage rapidly solves a constrained trajectory optimization problem to a coarse tolerance using iLQR within an augmented Lagrangian framework, similar to [20]. The optional secondary stage performs solution polishing, utilizing this coarse solution to warm start an active-set method that achieves high-precision constraint satisfaction. Problem reformulations for infeasible-initialization and free-time problems work both stages.

Problem. The ALTRO algorithm solves problems of the form:

$$\begin{aligned} \underset{u_{1:T-1}}{\text{minimize}} \quad & c_T(x_T) + \sum_{t=1}^{T-1} c_t(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t), && t = 1, \dots, T-1, \\ & g_t(x_t, u_t) \{\leq, =\} 0, && t = 1, \dots, T-1, \\ & g_T(x_T) \quad \{\leq, =\} 0, \\ & (x_1 \text{ given}). \end{aligned} \tag{4.13}$$

4.4.1 Constrained Iterative Linear Quadratic Regulator

The problem (4.13) is reformulated for the augmented Lagrangian method:

$$\begin{aligned} \underset{u_{1:T-1}}{\text{minimize}} \quad & \ell_T(x_T; \lambda_T, \rho) + \sum_{t=1}^{T-1} \ell_t(x_t, u_t; \lambda_t, \rho) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t), && t = 1, \dots, T-1, \\ & (x_1 \text{ given}), \end{aligned} \tag{4.14}$$

where:

$$\ell_t(x_t, u_t; \lambda_t, \rho) = c_t(x_t, u_t) + \lambda_t^T g_t(x_t, u_t) + \frac{1}{2} g_t(x_t, u_t)^T I_\rho g_t(x_t, u_t), \tag{4.15}$$

$$\ell_T(x_T; \lambda_T, \rho) = c_T(x_T) + \lambda_T^T g_T(x_T) + \frac{1}{2} g_T(x_T)^T I_\rho g_T(x_T). \tag{4.16}$$

The augmented Lagrangian objective (4.15, 4.16) and the dynamics satisfy the iterative linear quadratic regulator form. As a result, for fixed values of the dual variables and penalty parameter we can directly utilize the standard algorithm. Data for the LQR sub-problems are now:

$$W_t = [c_t]_{xx}(\bar{x}_t, \bar{u}_t) + [g_t]_x(\bar{x}_t, \bar{u}_t)^T I_\rho [g_t]_x(\bar{x}_t, \bar{u}_t), \tag{4.17}$$

$$R_t = [c_t]_{uu}(\bar{x}_t, \bar{u}_t) + [g_t]_u(\bar{x}_t, \bar{u}_t)^T I_\rho [g_t]_u(\bar{x}_t, \bar{u}_t), \tag{4.18}$$

$$H_t = [c_t]_{xu}(\bar{x}_t, \bar{u}_t) + [g_t]_x(\bar{x}_t, \bar{u}_t)^T I_\rho [g_t]_u(\bar{x}_t, \bar{u}_t), \tag{4.19}$$

$$w_t = [c_t]_x(\bar{x}_t, \bar{u}_t) + [g_t]_x(\bar{x}_t, \bar{u}_t)^T \Big(\lambda_t + I_\rho g_t(\bar{x}_t, \bar{u}_t) \Big), \tag{4.20}$$

$$r_t = [c_t]_u(\bar{x}_t, \bar{u}_t) + [g_t]_u(\bar{x}_t, \bar{u}_t)^T \Big(\lambda_t + I_\rho g_t(\bar{x}_t, \bar{u}_t) \Big), \tag{4.21}$$

$$A_t = [f_t]_x(\bar{x}_t, \bar{u}_t), \tag{4.22}$$

$$B_t = [f_t]_u(\bar{x}_t, \bar{u}_t), \tag{4.23}$$

$$C_t = 0, \tag{4.24}$$

where a Gauss-Newton approximation of the constraints is utilized.

4.4.2 Square-root backward pass

For augmented Lagrangian methods to achieve fast convergence the penalty term must be increased to large values, which can result in severe numerical ill-conditioning. The condition number of a matrix is the ratio of its maximum and minimum eigenvalues. In the case of iLQR, the value-function matrix can become ill-conditioned due to the recursive nature of the backward pass.

To help mitigate this issue, we introduce a numerically robust backward pass inspired by the square-root Kalman filter [21] and derive a recursive expression for this matrix square root:

$$S_t = \sqrt{P_t}. \tag{4.25}$$

At the final time step:

$$S_T = \mathbf{cholesky}(W_T), \tag{4.26}$$

is computed using a Cholesky factorization. At subsequent steps in the recursion, we need to modify (2.10):

$$P_t = \begin{bmatrix} I \\ K_t \end{bmatrix}^T \begin{bmatrix} X_t & Y_t^T \\ Y_t & Z_t \end{bmatrix} \begin{bmatrix} I \\ K_t \end{bmatrix}, \tag{4.27}$$

represented here in matrix form, where:

$$X_t = W_t + A_t^T P_{t+1} A_t, \tag{4.28}$$
$$Y_t = H_t + A_t^T P_{t+1} B_t, \tag{4.29}$$
$$Z_t = R_t + B_t^T P_{t+1} B_t, \tag{4.30}$$

to depend on the square-root matrix from the the following (i.e., $t+1$) time step. The Cholesky factorization is:

$$P_t = \begin{bmatrix} I \\ K_t \end{bmatrix}^T \begin{bmatrix} \Psi_t^T & O \\ \Lambda_t^T & \Gamma_t^T \end{bmatrix} \begin{bmatrix} \Psi_t & \Lambda_t \\ O & \Gamma_t \end{bmatrix} \begin{bmatrix} I \\ K_t \end{bmatrix} = \begin{bmatrix} \Psi_t + \Lambda_k K_k \\ \Gamma_t K_t \end{bmatrix}^T \begin{bmatrix} \Psi_t + \Lambda_k K_k \\ \Gamma_t K_t \end{bmatrix}, \tag{4.31}$$

where:

$$\Psi_t = \sqrt{X_t} = \mathbf{cholesky}(W_t) + S_{t+1} A_t, \tag{4.32}$$
$$\Lambda_t = \Psi_t^{-T} Y_t, \tag{4.33}$$
$$\Gamma_t = \sqrt{Z_t - Y_t^T X_t^{-1} Y_t} = \mathbf{DD}\Big(\mathbf{cholesky}(R_t) + S_{t+1} B_t, \Psi_t^{-T} Y_t\Big). \tag{4.34}$$

The square-root update is then:

$$S_t = \mathbf{QR}\Big(\begin{bmatrix} \Psi_t + \Lambda_k K_k \\ \Gamma_t K_t \end{bmatrix} \Big), \tag{4.35}$$

and replaces (2.10).

4.4.3 Infeasible initialization

Desired state trajectories can often be identified (e.g., from sampling-based planners or expert knowledge), whereas determining a sequence of actions that will produce a desired state trajectory is often challenging. Infeasible initialization is enabled by modifying the nominal discrete-time dynamics $d_t : \mathbf{R}^n \times \mathbf{R}^{m_t} \to \mathbf{R}^{n_{t+1}}$:

$$x_{t+1} = f_t(x_t, u_t) = d_t(x_t, v_t) + s_t, \tag{4.36}$$

with slacks $s_t \in \mathbf{R}^{n_t}$ to make the system artificially fully actuated, by augmenting the actions at each time step such that $u_t = (v_t, s_t) \in \mathbf{R}^{m_t + n_{t+1}}$. The dynamics Jacobians are then:

$$[f_t]_x = [d_t]_x, \tag{4.37}$$

$$[f_t]_u = \begin{bmatrix} [d_t]_v & I \end{bmatrix}. \tag{4.38}$$

Given a trajectory, $\hat{z} = (\hat{x}_1, \hat{v}_1, \ldots, \hat{x}_T)$, the slacks at each time step are initialized as:

$$s_t = \hat{x}_{t+1} - d_t(\hat{x}_t, \hat{v}_t), \tag{4.39}$$

the difference between the nominal dynamics and the provided trajectory. Additionally, constraints:

$$g_t(x_t, u_t) = s_t = 0, \quad t = 1, \ldots, T-1, \tag{4.40}$$

are added to ensure that the algorithm converges to solutions for the nominal dynamics. Importantly, due to the properties of the augmented Lagrangian method, these constraints will be satisfy in the limit, allowing the optimizer to iteratively reduce the infeasibility and converge to solutions for the nominal dynamics.

4.4.4 Free-time problem

For many planning problems, it is desirable to allow the optimizer to determine the total duration of the trajectory. This is accomplished by encoding the time step, $h \in \mathbf{R}_{++}$, as a decision variable.

The dynamics are formulated as:

$$x_{t+1} = f_t(x_t, u_t) = \begin{bmatrix} d_t(y_t, u_t) \\ s_t \end{bmatrix}, \tag{4.41}$$

where the actions, $u_t = (v_t, s_t) \in \mathbf{R}^{m_t+1}$, are augmented with an additional variable, $s_t = \sqrt{h_t} \in \mathbf{R}$, representing the square-root of the time step, ensuring positive values; and an augmented state, $x_t = (y_t, s_{t-1}) \quad t = 2, \dots T$, that propagates slack variables along the trajectory. The dynamics Jacobians are then:

$$[f_t]_x = \begin{bmatrix} [d_t]_y \\ 0 \end{bmatrix}, \tag{4.42}$$

$$[f_t]_u = \begin{bmatrix} [d_t]_v & [d_t]_s \\ 0 & 1 \end{bmatrix}. \tag{4.43}$$

Additional constraints are added to enforce equality between time steps:

$$g_t(x_t, u_t) = x_t^{(s)} - u_t^{(s)} = 0, \quad t = 2, \dots, T, \tag{4.44}$$

to prevent the optimizer from exploiting time-discretization errors in the system dynamics.

4.4.5 Solution polishing

The primary phase of ALTRO solves problems to low or medium precision, with respect to constraint satisfaction, and returns a solution $\bar{z} = (\bar{x}_1, \bar{u}_1, \dots, \bar{x}_T)$. A solution-polishing phase then utilizes this solution to warm start an active-set method [1] that can often achieve machine-precision constraint satisfaction in a handful of iterations.

Problem. Phase two optimizes the following minimum-norm [1] problem:

$$\begin{array}{ll} \underset{z}{\text{minimize}} & \|z - \bar{z}\|_J \\ \text{subject to} & \bar{E}(z) = 0, \end{array} \tag{4.45}$$

over the objective space:

$$J = \begin{bmatrix} [c_1]_{xx} & [c_1]_{xu} & & \\ [c_1]_{ux} & [c_1]_{uu} & & \\ & & \ddots & \\ & & & [c_T]_{xx} \end{bmatrix}, \tag{4.46}$$

subject to the set of active constraints, include the dynamics, equality, and active inequality constraints:

$$E(z) = \begin{bmatrix} f_1(x_1, u_1) - x_{t+1} \\ \vdots \\ f_{T-1}(x_{T-1}, u_{T-1}) - x_T \\ \bar{g}_1(x_1, u_1) \\ \vdots \\ \bar{g}_T(x_T) \end{bmatrix}, \tag{4.47}$$

denoted with an overbar ($^-$).

Algorithm. We iteratively perform the following update:

$$\Delta z = J^{-1}\bar{E}_z^T(\bar{E}_z J^{-1}\bar{E}_z^T)^{-1}\bar{E}_z, \tag{4.48}$$

$$z \leftarrow z + \alpha\Delta z, \tag{4.49}$$

using a line search procedure on the parameter $\alpha \in [0, 1]$ to ensure that:

$$\|\bar{E}(z + \alpha\Delta z)\|_\infty < \|\bar{E}(z)\|_\infty, \tag{4.50}$$

at each iteration. For additional efficiency, the factorization $(\bar{E}_z J^{-1}\bar{E}_z^T)^{-1}$ is reused, and is only updated when convergence slows or the active set changes.

Implementation. Our implementation of ALTRO is available at:

`https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl`.

4.5 Results

In this section we highlight ALTRO's capabilities by planning trajectories for a cart-pole and a simple car.

4.5.1 Cart-pole

A swing-up is planned for a cart-pole system [94]. We utilize third-order Runge Kutta integration, a fixed time step $h = 0.05$, and a planning horizon $T = 101$. Additionally, we encode action limits, $-3 \leq u \leq 3$, and a goal state constraint, $x_T = (0, \pi, 0, 0)$.

Constraints. In Fig. 4.1, we compare the trajectories optimized with (orange) and without (blue) action limits. ALTRO is able to enforce action limits while successfully planning a swing-up.

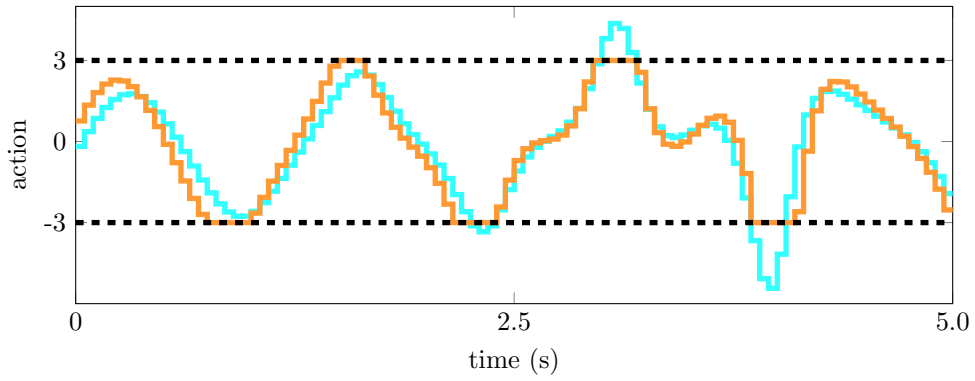


Figure 4.1: Cart-pole action trajectory comparison with (orange) and without (blue) control limits.

Backward pass condition number. In Fig. 4.2 we compare the standard LQR backward pass with ALTRO's square-root version. The square-root version consistently reduces the condition number by a few orders of magnitude over multiple augmented Lagrangian dual updates.

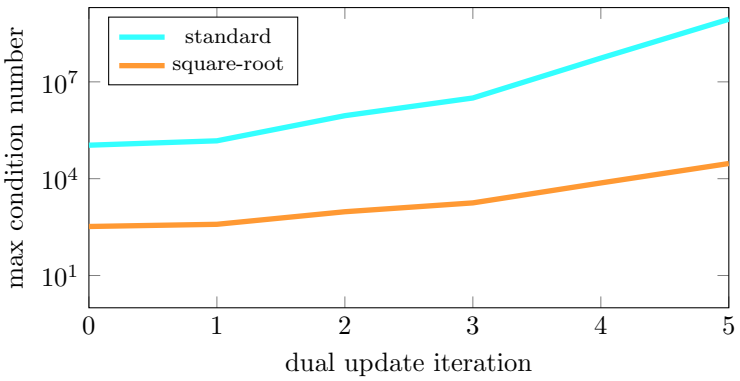


Figure 4.2: Condition number comparison between standard (blue) and square-root (orange) backward pass.

Solution polishing. In this final cart-pole example, in Fig. 4.3, we compare constrained iLQR with ALTRO's solution-polishing phase. After the maximum constraint violation is less 1e-3, the algorithm switches from phase one to phase two. By performing polishing, ALTRO is able to rapidly achieve machine-precision constraint satisfaction (1e-8) compared to the slow tail convergence characteristic of constrained iLQR.

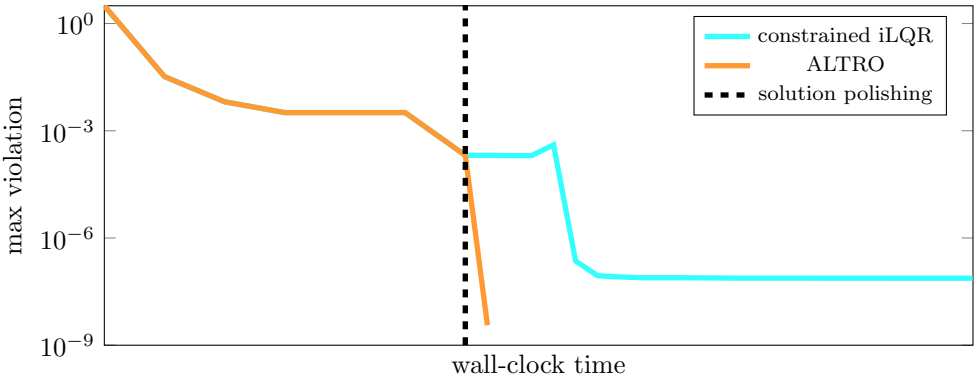


Figure 4.3: Constraint convergence comparison between constrained iLQR (blue) and ALTRO with solution polishing (orange) in terms of wall-clock time.

4.5.2 Car

A simple car, modeled as a simple wheeled system [95], is tasked with moving from a start to goal pose. We utilize midpoint integration, a nominal time step $h = 0.05$, and a planning horizon $T = 101$. Additionally, we encode action limits, $-1 \leq u \leq 1$, and a goal state constraint, $x_T = (1, 1, 0)$.

Infeasible initialization. We provide the optimizer with a linear interpolation of states between the start and goal poses. The resulting position trajectory is shown in Fig. 4.4. With infeasible initialization 132 iterations are required versus 146 iterations for standard initialization.

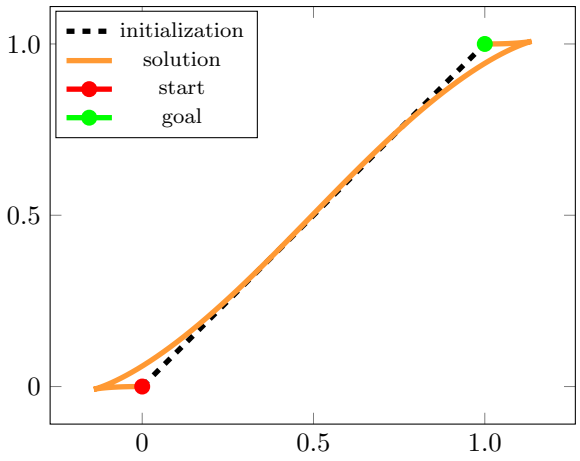


Figure 4.4: Plan (orange) from start (red) to goal (green) positions for car with infeasible-state initialization (black).

Free-time problem. We now solve the same problem with our free-time formulation. A large cost on the (square-root) time step variables encourages the plan to reach the goal quickly. In Fig. 4.5 we compare the action trajectories for the free-time (orange) and nominal problems (blue).

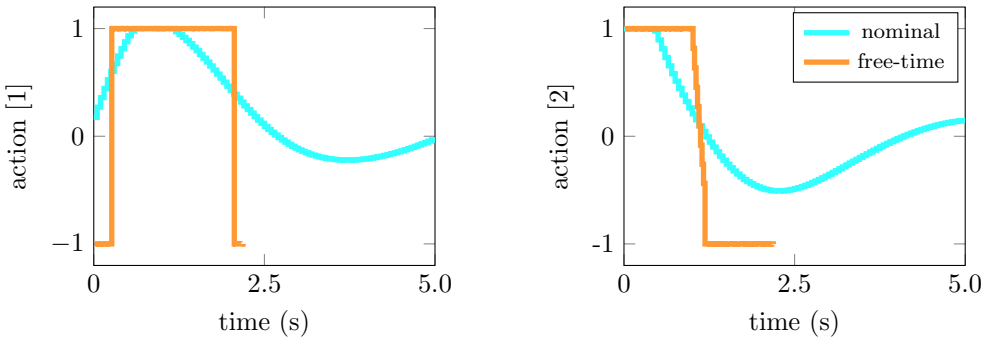


Figure 4.5: Comparison of nominal (blue) and free-time (orange) solutions for car planning problem.

The nominal problem takes 5 seconds, whereas the free-time problem finds a 2.2 second plan with bang-bang controls.

4.6 Limitations

In our experiments, we find that the free-time formulation is more sensitive to initializations compared with direct methods and that careful problem design is required to achieve goods results. We hypothesize that making the time step a decision variable adversely effects the forward rollouts.

Similarly, the algorithm often fails to utilize the infeasible initialization if the augmented Lagrangian penalty is not properly initialized. As a result, the designer should carefully select the initial penalty value to balance rapidly setting the slack values to zero and retaining the initialization.

Finally, predictive control applications typically perform a single update before returning a new solution. In the case of ALTRO, this would require good warm starting of both the nominal trajectory and the augmented Lagrangian dual and penalty variables. Heuristics for warm starting the latter values in order to enable rapid convergence of the constraints remain for future work.

4.7 Future work

We have presented a versatile high-performance trajectory optimization algorithm, ALTRO, that combines the advantages of both direct and indirect methods: fast convergence, infeasible initialization, anytime dynamic feasibility, and high-precision constraint satisfaction.

Future work will implement the algorithm in C/C++ for deployment on hardware in a predictive control framework. Additional improvements include parallel line searches for increased efficiency and support for additional constraints in the augmented Lagrangian framework in order to handle second-order and positive semidefinite cones [15]. A final avenue to pursue is making the solver differentiable with respect to problem data (e.g., objective terms or model parameters) via implicit differentiation [96] for auto-tuning [97] or learning applications.

Chapter 5

Planning with Optimization-Based Dynamics

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 an interior-point method for efficient evaluation of constrained dynamics and utilize implicit differentiation 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.

Trajectory Optimization with Optimization-Based Dynamics. Taylor A. Howell, Simon Le Cleac'h, Sumeet Singh, Pete Florence, Zachary Manchester, and Vikas Sindhwani. Robotics and Automation Letters. 2022.

5.1 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) [23] 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 an interior-point method for efficient evaluation of the dynamics at each time step. Implicit differentiation is utilized to compute derivatives for this representation, and we exploit intermediate results from our interior-point method to return useful, smooth gradients to an upper-level optimizer.

To demonstrate the capabilities of this representation, we utilize optimization-based dynamics and iLQR in a bi-level optimization framework for planning. 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
- An interior-point 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 chapter, we first review related work on bi-level approaches to trajectory optimization in Section 5.2. Next, we provide background on implicit integrators in Section 5.3. Then, we present our optimization-based dynamics representation, including an interior-point method for solving this problem class in Section 5.4. We provide a collection of optimization-based dynamics models that are utilized to perform trajectory optimization, and comparisons, in Section 5.5. Finally, we conclude with a discussion of limitations to this approach in Section 5.6 and directions for future work in Section 5.7.

5.2 Related Work

Bi-level optimization [98] 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 [40] has been employed in an implicit bi-level approach for whole-body model predictive control of a humanoid robot [99]. 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-difference 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 [100, 30]. A related approach formulates contact dynamics as lower-level holonomic constraints [101]. Implicit integrators, which are a special-case of optimization-based dynamics and are widely used in collocation methods [102], have been explored with differentiable dynamic programming generally [103] and specifically for models that require cloth [104] or contact [105] simulation. Implicit dynamics have also been trained to represent non-smooth dynamics [84], 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 [36] and a semi-direct method utilizes a lower-level friction problem for planning through contact events [106]. A related, sequential operator splitting framework is proposed in [107] and a derivative-free method that generates gradients via randomized smoothing for iLQR is also proposed [68]. 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.

5.3 Background

In this section we provide background on implicit integrators, which we will generalize to optimization-based dynamics in the following section.

5.3.1 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 [14, 75], which cannot be separated from the current state and control input:

$$f_t(x_{t+1}, x_t, u_t) = 0, \tag{5.1}$$

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 [108].

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 [102]. However, for indirect methods, like iLQR, that enforce strict dynamic feasibility at each iteration via rollouts, evaluating (5.1) requires a numerical solver to find a solution that satisfies this implicit function for given problem data.

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

$$\frac{\partial f_t}{\partial x_{t+1}}\delta x_{t+1} + \frac{\partial f_t}{\partial x_t}\delta x_t + \frac{\partial f_t}{\partial u_t}\delta u_t = 0, \tag{5.2}$$

and then we solve for δx_{t+1}:

$$\delta x_{t+1} = -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\Big(\frac{\partial f_t}{\partial x_t}\delta x_t + \frac{\partial f_t}{\partial u_t}\delta u_t\Big). \tag{5.3}$$

The Jacobians:

$$\frac{\partial x_{t+1}}{\partial x_t} = -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\frac{\partial f_t}{\partial x_t}, \tag{5.4}$$

$$\frac{\partial x_{t+1}}{\partial u_t} = -\Big(\frac{\partial f_t}{\partial x_{t+1}}\Big)^{-1}\frac{\partial f_t}{\partial u_t}. \tag{5.5}$$

are returned at each time step.

5.4 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.

5.4.1 Problem

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:

$$x_{t+1} \in z^*(\theta) = \underset{z \in \mathcal{K} \mid c(z;\theta)=0}{\arg\min} \ \ell(z;\theta), \tag{5.6}$$

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.

5.4.2 Interior-point method

One of the primary challenges in optimizing (5.6) 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 [109, 110, 111] or second-order interior-point methods [53, 29]. The first approach optimizes (5.6) 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 [15]. Second-order semi-smooth methods also exist [112].

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, interior-point methods exhibit fast convergence, can achieve tight constraint satisfaction [1], and can return smooth gradients using a relaxed central-path parameter.

Based on these characteristics, we utilize an interior-point method [1] to optimize (5.6). 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:

$$\partial\ell(z;\theta)/\partial z + (\partial c(z;\theta)/\partial z)^T \lambda - \nu = 0, \tag{5.7}$$

$$c(z;\theta) = 0, \tag{5.8}$$

$$z \circ \nu = \mu \mathbf{e}, \tag{5.9}$$

$$z \in \mathcal{K},\ \nu \in \mathcal{K}^*, \tag{5.10}$$

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 $\mathcal{K}^*$ [53]. We consider free, nonnegative, and second-order cones.

To find a stationary point of (5.7-5.10), 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 (5.7-5.9). 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 [15, 1] for additional background on these methods.

5.4.3 Implicit differentiation

To differentiate (5.6), we apply the implicit-function theorem (2.55) 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.

5.4.4 Algorithm

The complete algorithm is summarized in Algorithm 3 and we provide an open-source implementation of an interior-point solver.

Algorithm 3 Differentiable Interior-Point Method

procedure OPTIMIZE(z, θ)
  Parameters: $\beta = 0.5, \gamma = 0.1,$
    $\epsilon_\mu = 10^{-4}, \epsilon_r = 10^{-8}$
  Initialize: $z \in \mathcal{K}$, $\lambda = 0, \nu \in \mathcal{K}^*, \mu = 1.0, w_\mu = \{\}$
  $\bar{r} = r(w; \theta, \mu)$
  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**
      $w_\mu \leftarrow w_\mu \cup w$
      $\mu \leftarrow \gamma\mu$
  $\partial w / \partial\theta \leftarrow$ **Differentiate**(w_μ, θ) ▷ Eq. 2.55
  Return $w, \partial w / \partial\theta$

5.5 Results

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, models, and all of the experiments are available here:

`https://github.com/thowell/optimization_dynamics`.

5.5.1 Acrobot with joint limits

We model an acrobot [94] with joint limits on the actuated elbow. These limits are enforced with a signed-distance constraint:

$$\phi(q) = \begin{bmatrix} \pi/2 - q_{\text{e}} \\ q_{\text{e}} + \pi/2 \end{bmatrix} \geq 0, \tag{5.11}$$

where $q \in \mathbf{R}^2$ is the system's configuration and q_{e} is the elbow angle. Additional constraints (2.26-2.28) encode impacts when joint limits are reached. 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 (5.7-5.10) that can be optimized with Algorithm 3.

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. 5.1.

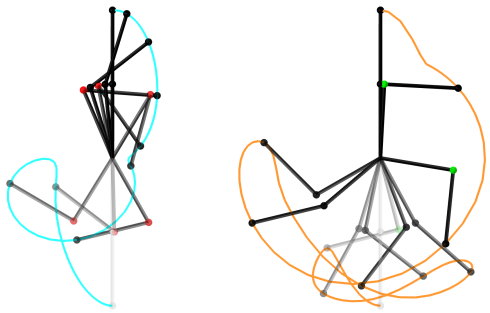


Figure 5.1: Optimized trajectories for acrobot systems without (left) and with (right) joint limits performing a swing-up. Red and green elbow colors indicate violation or non-violation of the joint limits, respectively.

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 5.1.

Table 5.1: 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)**
objective	395.6	1.9e6	48.9	84.2
iterations	207	95	256	907
violation	3e-4	0.2	5e-4	1e-3
time (s)	0.9	1.0	0.5	6.5

5.5.2 Cart-pole with Coulomb friction

A cart-pole [94] is modeled that experiences Coulomb friction [42] on both its slider and pendulum arm. We use the constraints (3.11) to model stick-slip behavior.

The system has $n = 4$ states and $m = 1$ controls. 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, see Fig. 5.2.

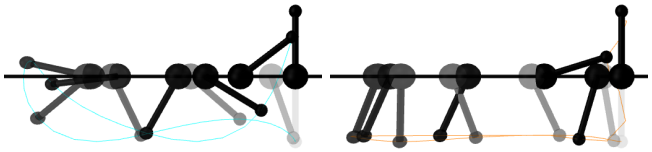


Figure 5.2: Optimized trajectories for cart-pole performing a swing-up without (left, friction coefficient = 0) and with (right, friction coefficient = 0.35) joint friction.

In the presence of friction, the system performs a more aggressive maneuver at the end of the trajectory in order to achieve the swing-up. The results are summarized in Table 5.2.

Table 5.2: 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**
objective	5.0	5.5	11.6	76.3	163.1
iterations	456	485	406	472	943
violation	1e-4	5e-4	6e-4	1e-3	5e-3
time (s)	0.36	2.8	3.1	4.6	9.8

5.5.3 Hopper gait

We generate a gait for a Raibert hopper [79]. The system's dynamics are modeled as a nonlinear complementarity problem [36]. 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 [37] that transcribes the problem and solves it with Ipopt [26]. We use the same models, cost functions, and comparable optimizer parameters and tolerances.

The complexity of the classic iLQR algorithm is: $O(Tm^3)$ [113], while a direct method for trajectory optimization that exploits the problem's temporal structure is: $O\Big(T(n^3 + n^2m)\Big)$ [114]. The interior-point method for the dynamics generally has complexity: $O(a^3)$, although specialized solvers can reduce this cost, where a is the number of primal and dual variables [1]. The complexity of optimization-based dynamics and iLQR is typically dominated by the cost of the interior-point 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. 5.3.

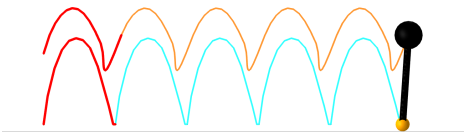


Figure 5.3: Hopper gait. Optimized template (red) for the system's body (orange) and foot (blue) trajectories is repeated to form a gait.

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 5.3.

Table 5.3: 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)**
objective	3.5	20.4	2.9	3.5	**19.8**	**2.4**
iterations	122	114	107	**38**	**47**	**25**
violation	**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**

5.5.4 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 [115], 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.

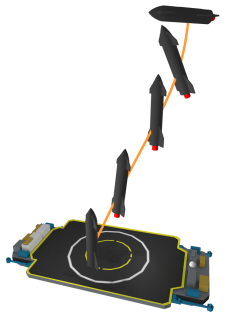


Figure 5.4: Position trajectory (orange) for rocket performing soft landing. The system first reorients from a horizontal to vertical pose before landing with zero velocity in a target zone while respecting thrust constraints.

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

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \tfrac{1}{2}\|u-\bar{u}\|_2^2 \\ \text{subject to} \quad & \|u_{2:3}\|_2 \le u_1, \\ & u_{\min} \le u_1 \le u_{\max}, \end{aligned} \tag{5.12}$$

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.

The system has $n = 12$ states and $m = 3$ controls that are first projected using (5.12) 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 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 optimized position trajectory is shown in Fig. 5.4.

5.5.5 Planar push

For the canonical manipulation problem of planar pushing [116], we optimize the positions and forces of a robotic manipulator's circular end-effector in order to slide a planar block into an reoriented goal pose (Fig. 5.5).

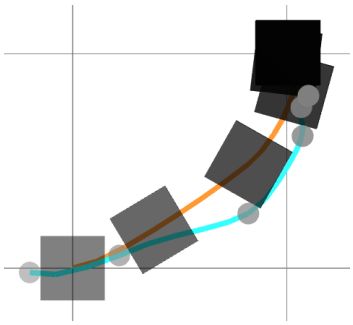


Figure 5.5: Optimized trajectories for planar-push task. The pusher (blue) and block (orange) paths are shown for a plan that maneuvers the block from a pose at the origin to a goal pose.

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 time step 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 interior-point solver with randomized smoothing to generate a zero-order gradient bundle [68]. 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 5.4.

Table 5.4: Comparison between gradients generated as zero-order gradient bundles and via the implicit-function theorem for planar-push translation (T) and rotation (R) tasks. *Gradient bundles are evaluated serially.

	Bundle (T)	**Bundle (R)**	**Implicit (T)**	**Implicit (R)**
iterations	18	**32**	**17**	33
time (s)	+100.0*	+100.0*	**8.0**	**18.3**

5.6 Limitations

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 overall planning and that the optimizer can eventually find a dynamically feasible trajectory. Just as robust, large-scale solvers such as Ipopt [26] fallback to their restoration mode when numerical difficulties occur, our basic interior-point 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 interior-point 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.

5.7 Future Work

In this work we have presented a bi-level optimization framework that utilizes lower-level optimization-based dynamics for planning. 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, enabling classically unconstrained solvers to optimize motions for more complex systems.

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 [117], potentially optimized with first-order Langevin dynamics [118]. 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.

Chapter 6

Optimization with Conic and Complementarity Constraints

We present a new solver for non-convex trajectory optimization problems that is specialized for robotics applications. CALIPSO, or the Conic Augmented Lagrangian Interior-Point SOlver, combines several strategies for constrained numerical optimization to natively handle second-order cones and complementarity constraints. It reliably solves challenging motion-planning problems that include contact-implicit formulations of impacts and Coulomb friction and state-triggered constraints where general-purpose non-convex solvers like SNOPT and Ipopt fail to converge. Additionally, CALIPSO supports efficient differentiation of solutions with respect to problem data, enabling bi-level optimization applications like auto-tuning of feedback policies. Reliable convergence of the solver is demonstrated on a range of problems from manipulation, locomotion, and aerospace domains. An open-source implementation of this solver is available.

CALIPSO: A Differentiable Solver for Trajectory Optimization with Conic and Complementarity Constraints. Taylor A. Howell, Kevin Tracy, Simon Le Cleac'h, and Zachary Manchester. International Symposium of Robotic Research. 2022.

6.1 Introduction

Trajectory optimization is a powerful tool for offline generation of complex behaviors for dynamic systems, as well as online as a planner or feedback controller within model predictive control frameworks. The use of constraints greatly enhances the ability of a designer to generate desirable solutions, enforce safe behaviors, and model physical phenomena. Unfortunately, many constraint types that have important and direct applications to robotics are poorly handled by existing general-purpose non-convex solvers [1] or differential dynamic programming (DDP) algorithms [23].

Second-order cones [15], which commonly appear as friction-cone [42] or thrust-limit [115] constraints, present difficulties for these solvers due to their nondifferentiability at commonly occurring states, like when the friction or thrust forces are zero. Common reformulations of these constraints for solvers like SNOPT [27] and Ipopt [26] are typically non-convex and fail to work well in practice [119].

Contact dynamics, including impact and friction, are naturally modeled with complementarity constraints [120]. This formulation constrains contact forces to only take on non-zero values when the distance between objects is zero. State-triggered constraints [121], in which constraints switch on or off in different regions of the state space, can similarly be modeled with complementarity constraints. However, these constraints violate the linear independence constraint qualification (LICQ), a fundamental assumption in the convergence theory of standard second-order solvers [1, 122].

In this work, we present a new solver for trajectory optimization, CALIPSO: Conic Augmented Lagrangian Interior-Point SOlver. The development of this solver is motivated by challenging non-convex motion-planning problems that require second-order cone and complementarity constraints; in particular, contact-implicit trajectory optimization [30, 37] for locomotion and manipulation. CALIPSO combines a number of ideas and algorithms from constrained numerical optimization to solve these difficult problems reliably.

Second-order cones are handled using an interior-point method [29] that exploits the convexity of these constraints for strict enforcement without the need for linear approximations. All equality constraints are handled using an augmented Lagrangian method [9], which does not require the constraints to satisfy LICQ and is robust to the degeneracies that can arise from complementarity constraints [31]. Additionally, both the interior-point and augmented-Lagrangian methods are formulated in a primal-dual fashion to enhance the numerical robustness and performance of the solver. The computation of Newton steps on the combined primal-dual augmented-Lagrangian and interior-point KKT conditions are reformulated as a symmetric linear system to enable the use of fast, direct linear algebra methods [123]. Finally, the implicit-function theorem [5, 11] is utilized to efficiently differentiate through solutions, enabling bi-level optimization applications such as auto-tuning of model predictive control policies.

Our specific contributions are:

- A differentiable trajectory optimization solver with native support for second-order cones and reliable handling of complementarity constraints
- A novel, combined, interior-point and augmented-Lagrangian algorithm for non-convex optimization
- A symmetric reformulation of the combined method's KKT system for fast symmetric linear-system solvers
- A collection of benchmark robot motion-planning problems that contain second-order cone and complementarity constraints
- An open-source implementation of the solver written in Julia

In the remainder of this chapter, we first provide an overview of related work on trajectory optimization methods in Section 6.2. Then, we provide the necessary background on complementarity constraints and LICQ in Section 6.3. Next, we present CALIPSO and its key algorithms and routines in Section 6.4. We then demonstrate CALIPSO on a collection of robot motion-planning examples in Section 6.5. Finally, we conclude with a discussion of limitations in Section 6.6 and future work in Section 6.7.

6.2 Related work

Trajectory optimization problems are solved with methods that are classically categorized as either indirect or direct [2]. Indirect methods include shooting methods, DDP [23], and iterative LQR (iLQR) [81]. These methods exploit the temporal structure of the problem and utilize a Riccati backward pass to compute updates for control variables followed by forward simulation rollouts to update the states. Classically, apart from the dynamics, these methods do not include support for constraints. In recent years, various approaches have extended these methods to handle constraints [88, 18, 22, 101, 124, 125, 126]. However, reliable constraint handling and solution accuracy for these methods is still challenging in many scenarios.

Direct methods, in contrast, directly transcribe the trajectory optimization problem as a constrained non-convex problems, with both states and controls as decision variables [102]. The transcription is provided to a general-purpose solver such as SNOPT or Ipopt. This approach is generally reliable and enables robust constraint handling. However, these solvers do not exploit the temporal structure of the trajectory optimization problem like indirect methods, and historically have been thought to converge to solutions more slowly as a result. To address this limitation, direct solvers tailored for the underlying trajectory optimization problem structure have been proposed [114] and are available as commercial tools [127].

While providing reliable constraint handling for equality and inequality constraints in many scenarios, direct methods lack support for second-order cones [15] and exhibit difficulties handling complementarity constraints [120]. The exact handling of second-order cones as inequality constraints with these solvers results in poor practical performance because the nondifferentiable point of the cone is frequently visited in robotics applications (e.g., an object resting on a surface with zero friction force). Classic reformulations are non-convex and similarly exhibit poor and unreliable convergence [119]. Recently, sequential convexification approaches have been developed that are able to handle second-order cones by iteratively solving convex approximations of the original problem using general-purpose cone solvers [32, 128].

The complementarity constraints that can arise in robotics problems are generally difficult for solvers to handle because they are non-convex and violate LICQ. To overcome this, a number of problem reformulations and constraint relaxation techniques have been proposed and explored for interior-point [122, 129] and augmented Lagrangian [31] methods, but these approaches largely remain *ad hoc* and are unavailable in state-of-the-art solvers for trajectory optimization, which we compare in Table 6.1.

Table 6.1: Comparison of general-purpose and trajectory optimization solvers.

Solver	Method	Accuracy	Second-Order	Complementarity	Differentiable
Ipopt [26]	direct	high	×	×	×
SNOPT [27]	direct	high	×	×	×
CVX [130]	direct	high	✓	×	✓
ALTRO [18]	indirect	medium	✓	×	×
Trajax [131]	indirect	medium	×	×	✓
GuSTO [128]	direct	high	✓	×	×
FORCES [127]	direct	high	×	×	×
Drake [60]	direct	high	×	×	×
CALIPSO	direct	high	✓	✓	✓

6.3 Background

In this section, we provide background on complementarity constraints and LICQ.

6.3.1 Complementarity constraints

Contact-implicit trajectory optimization [30] optimizes trajectories for systems that make and break contact with their environments and represents dynamics using complementarity constraints. For example, optimizing motion over a single time step for an actuated particle in a single dimension,

resting on a surface, modeled with impact such that it cannot pass through the floor:

$$\begin{aligned} \underset{z,u,\gamma}{\text{minimize}} \quad & \tfrac{1}{2}(z - z_{\text{g}})^2 + \tfrac{1}{2}u^2 \\ \text{subject to} \quad & m\big(z/h + gh\big) = \gamma + u, \\ & z \cdot \gamma = 0, \\ & z, \gamma \geq 0, \end{aligned} \tag{6.1}$$

with position $z \in \mathbf{R}$, control input $u \in \mathbf{R}$, contact impulse $\gamma \in \mathbf{R}$, mass $m \in \mathbf{R}_{++}$, gravity $g \in \mathbf{R}_{+}$, time step $h \in \mathbf{R}_{+}$, and goal $z_{\text{g}} \in \mathbf{R}$. These constraints are derived from a constrained discrete Lagrangian [37]. The set of constraints on z and γ are collectively referred to as a complimentary constraint, and are sometimes abbreviated $z \perp \gamma$. This formulation does not require pre-specified contact-mode sequences or hybrid dynamics since the solver is able to optimize physically correct contact dynamics at each time step.

6.3.2 Linear independence constraint qualification

General-purpose second-order solvers that rely on Newton's method to compute search directions (e.g., SNOPT and Ipopt) assume that the constraints provided by the user satisfy the LICQ in the neighborhood of solutions. Certain classes of constraints, including complementarity conditions that naturally arise in contact dynamics, often do not satisfy this assumption. We demonstrate how this assumption is violated with a simple contact-implicit trajectory optimization problem (6.1). The Lagrangian for the problem is:

$$L(z, u, \gamma, a, b, c, d) = \frac{1}{2}(z - z_{\text{g}})^2 + \frac{1}{2}u^2 + a\big(m(z/h + gh) - \gamma - u\big) + bz\gamma - cz - d\gamma, \tag{6.2}$$

where $a, b, c, d \in \mathbf{R}$ are the Lagrange multipliers, or "dual variables," associated with constraints. The KKT system is:

$$\begin{bmatrix} 1 & b & 0 & m/h & \gamma & -1 & 0 \\ b & 0 & 0 & 1 & z & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ m/h & -1 & -1 & 0 & 0 & 0 & 0 \\ \gamma & z & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & z & 0 \\ 0 & d & 0 & 0 & 0 & 0 & \gamma \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\gamma \\ \Delta u \\ \Delta a \\ \Delta b \\ \Delta c \\ \Delta d \end{bmatrix} = - \begin{bmatrix} (z - z_{\text{g}}) + am/h + b\gamma - c \\ a + bz - d \\ u - a \\ m\big(z/h + gh\big) - \gamma - u \\ z \cdot \gamma \\ c \cdot z \\ d \cdot \gamma \end{bmatrix}, \tag{6.3}$$

$$z, \gamma, c, d, \geq 0, \tag{6.4}$$

with first-order necessary (KKT) conditions (right-hand side) and KKT matrix (left-hand side). The Newton step that a standard second-order solver would take to drive these KKT conditions to zero is computed by solving this system (6.3).

In the scenario where the particle is above the surface (i.e., $z > 0$) the contact impulse must be zero (i.e., $\gamma = 0$). As a result, the fifth and seventh rows of the KKT matrix will be linearly dependent, resulting in non-unique optimal dual variables and violation of LICQ. A similar result occurs when $\gamma > 0$ and $z = 0$. Consequently, the Newton step is not well defined in these cases, causing difficulties for the solver. While a myriad of *ad hoc* heuristics exist to alleviate this problem, we pursue a more rigorous approach in the following sections based on ideas from constrained numerical optimization.

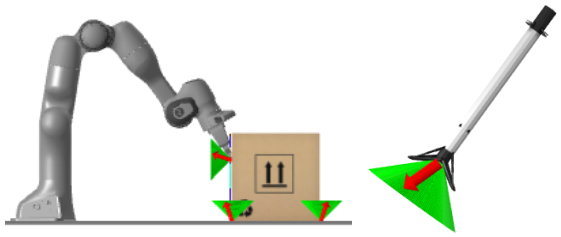


Figure 6.1: Robotics applications with second-order cone constraints include Coulomb friction at contact points in manipulation tasks (left) and thrust limits on rockets (right). Cone constraints are shown in green, while force vectors are shown in red.

6.4 CALIPSO

CALIPSO is a differentiable primal-dual augmented Lagrangian interior-point solver for non-convex optimization problems with second-order cone and complementarity constraints. Its standard problem formulation is:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c(x;\theta) \\ \text{subject to} \quad & g(x;\theta) = 0, \\ & h(x;\theta) \in \mathcal{K}, \end{aligned} \tag{6.5}$$

with decision variables $x \in \mathbf{R}^n$, problem data $\theta \in \mathbf{R}^d$, equality constraints $g : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}^m$, and constraints $h : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}^p$ in cone $\mathcal{K} = \mathbf{R}^q_{++} \times Q^{(1)}_{l_1} \times \cdots \times Q^{(j)}_{l_j}$ comprising a q-dimensional inequality and j second-order cones, each of dimension l_i. Internally, problem (6.5) is reformulated and additional slack variables $r \in \mathbf{R}^m$ and $s \in \mathbf{R}^p$, associated with the equality and cone constraints,

respectively, are introduced, and the following modified problem is formed:

$$\begin{array}{ll} \underset{x,r,s}{\text{minimize}} & c(x;\theta) + \lambda^T r + \frac{\rho}{2} r^T r - \kappa \sum_{i=1}^{p} \log(s^{(i)}) \\ \text{subject to} & g(x;\theta) - r = 0, \\ & h(x;\theta) - s = 0, \\ & s \in \mathcal{K}, \end{array} \tag{6.6}$$

The modified problem's Lagrangian is:

$$\begin{aligned} L(w;\theta,\lambda,\rho,\kappa) = c(x;\theta) + y^T(g(x;\theta) - r) + z^T(h(x;\theta) - s) \\ + \lambda^T r + \frac{\rho}{2} r^T r - \kappa \sum_{i=1}^{p} \log(s^{(i)}). \end{aligned} \tag{6.7}$$

For convenience we denote the concatenation of all of the solver's variables as $w = (x, r, s, y, z, t)$. The KKT system is:

$$\begin{bmatrix} L_{xx} + \epsilon_p I & 0 & 0 & g_x^T & h_x^T & 0 \\ 0 & (\rho + \epsilon_p) I & 0 & -I & 0 & 0 \\ 0 & 0 & \epsilon_p I & 0 & -I & -I \\ g_x & -I & 0 & -\epsilon_d I & 0 & 0 \\ h_x & 0 & -I & 0 & -\epsilon_d I & 0 \\ 0 & 0 & P_s & 0 & 0 & P_t - \epsilon_d I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta r \\ \Delta s \\ \Delta y \\ \Delta z \\ \Delta t \end{bmatrix} = - \begin{bmatrix} c_x + g_x^T y + h_x^T z \\ \lambda + \rho r - y \\ -z - t \\ g - r \\ h - s \\ s \circ t - \kappa \mathbf{e} \end{bmatrix},$$

$$J \Delta w = -R, \tag{6.8}$$

where and $P_s, P_t \in \mathbf{S}^p$ are the cone-product Jacobians, and $\epsilon_p, \epsilon_d \in \mathbf{R}_+$ are additional primal and dual regularization terms, respectively.

6.4.1 Search direction

The nominal KKT system (6.8) without regularization (i.e., $\epsilon_p, \epsilon_d = 0$) is non-symmetric, potentially with undesirable eigenvalues that will not return a descent direction. The solver modifies this system for faster computation and a more reliable search direction.

Symmetric KKT system. To be amenable to fast solvers for symmetric linear systems, the nominal KKT system is reformulated:

$$\begin{bmatrix} L_{xx} + \epsilon_p I & g_x^T & h_x^T \\ g_x & -\left(\frac{1}{\rho+\epsilon_p} I\right) & 0 \\ h_x & 0 & -\left(\epsilon_d I + ((P_s + \epsilon_p \bar{P}_t)^{-1} \bar{P}_t)\right) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = - \begin{bmatrix} L_x \\ \bar{L}_y \\ \bar{L}_z \end{bmatrix}, \tag{6.9}$$

using Schur complements [15], where $\bar{L}_y = L_y + \frac{1}{\rho+\epsilon_p} L_r$, $\bar{L}_z = L_z + (P_s + \epsilon_p \bar{P}_t)^{-1}(\bar{P}_t L_s + L_t)$, and $\bar{P}_t = P_t - \epsilon_d I$. The remaining search directions:

$$\Delta r = (\Delta y + L_r)/(\rho + \epsilon_p), \tag{6.10}$$

$$\Delta s = (P_s + \epsilon_p \bar{P}_t)^{-1}(\bar{P}_t \Delta z + \bar{P}_t L_s + L_r), \tag{6.11}$$

$$\Delta t = -\Delta z + \epsilon_p \Delta s - L_s, \tag{6.12}$$

are recovered from the solution (6.9). Iterative refinement [1] is performed to improve the quality of the search directions computed using the symmetric system (6.9 - 6.12).

Inertia correction. To ensure a unique descent direction, the left-hand side of the symmetric KKT system (6.9) is corrected to have an inertia of n positive, $m + p$, negative, and no zero-valued eigenvalues. This is accomplished via adaptive regularization that increases the regularization terms, and subsequently reduces the values when possible to limit unnecessary corrections, using a heuristic developed for Ipopt [26].

6.4.2 Line search

Cone variables are initialized being strictly feasible at the start of each solve. A filter line search that is a slight modification of the one used by Ipopt [26] is performed to ensure an improvement to the solution is achieved at each step of the algorithm and step sizes are initially chosen using a fraction-to-the-boundary rule to ensure that the cone constraints remain strictly satisfied. Additionally, a separate step size is computed for the candidate cone dual variables t to avoid unnecessarily restricting progress of the remaining variables.

After satisfying the cone constraints, a filter line search is performed with the merit function:

$$\varphi(x, r, s; \theta, \lambda, \rho, \kappa) = c(x; \theta) + \lambda^T r + \frac{\rho}{2} r^T r - \kappa \sum_{i=1}^{p} \log(s^{(i)}), \tag{6.13}$$

and violation metric:

$$\eta = \|(g(x) - r, h(x) - s)\|_1/(m + p). \tag{6.14}$$

The step size is further decremented until either the merit or violation metric is decreased [132]. A

filter:

$$\mathcal{F} = \{(\varphi^{(1)}, \eta^{(1)}), \ldots, (\varphi^{(p)}, \eta^{(p)})\}, \tag{6.15}$$

comprises a set of p previously accepted points. A candidate point must satisfy:

$$\hat{\varphi} < \varphi^{(i)} \ \vee \ \hat{\eta} < \eta^{(i)}, \quad i = 1, \ldots, v, \tag{6.16}$$

for each previous point in the filter. Then, the Armijo condition [1]:

$$\hat{\varphi} < \varphi + \epsilon_a \alpha (\varphi_x^T \Delta x + \varphi_r^T \Delta r + \varphi_s^T \Delta s), \tag{6.17}$$

with tolerance $\epsilon_a \in \mathbf{R}_+$, must be satisfied in order to accept a candidate point. Finally, the filter is augmented with the candidate point:

$$\mathcal{F} \leftarrow \mathcal{F} \cup (\hat{\varphi}, \hat{\eta}). \tag{6.18}$$

6.4.3 Cone-product Jacobians

The cone-product Jacobian's P_s and P_t have known structure and decompose by cone. This enables fast matrix products and inverses. For inequalities:

$$P_a(a, b) = \mathbf{diag}(b), \quad P_a(a, b)^{-1} = \mathbf{diag}(1/b_1, \ldots, 1/b_q), \tag{6.19}$$

are diagonal matrices. For second-order cones:

$$P_a(a, b) = \begin{bmatrix} b^{(1)} & (b^{(2:l)})^T \\ b^{(2:l)} & b^{(1)} I \end{bmatrix}, \tag{6.20}$$

is an arrowhead matrix and its inverse has complexity linear in the dimension of the matrix [133].

6.4.4 Fraction-to-the-boundary

Step sizes α for cone-variable updates are selected to ensure that the fraction-to-the-boundary rule [26]:

$$s + \alpha \Delta s - (1 - \tau) s \in \mathcal{K}, \tag{6.21}$$

is satisfied for a cone. The parameter $\tau \in [0, 1]$ helps prevent cone variables from reaching their respective boundaries too quickly and is increased during outer updates. The fraction-to-the-boundary value is increased during each outer update.

6.4.5 Iterative refinement

A drawback to computing search directions using the symmetric system is the potential worsening of the numerical conditioning of the system. As a result, the error:

$$e = R + J\Delta w, \tag{6.22}$$

will be nonzero. To account for this, we correct the search direction by performing iterative refinement [1]. A linear system:

$$J\Delta e = -e, \tag{6.23}$$

is solved using the error as the residual in order to compute a correction Δe. This correction is then utilized to update the search direction:

$$\Delta w \leftarrow \Delta w + \Delta e. \tag{6.24}$$

This procedure (6.22 - 6.24) is repeated until the norm of the error is below a desired tolerance.

In the case where iterative refinement fails (e.g., exceeding the solver's maximum number of refinement iterations), new candidate points are evaluated using an alternative search direction computed using the regularized non-symmetric system and an LU factorization.

6.4.6 Outer updates

Convergence of a subproblem (6.6) occurs for fixed values of λ, ρ, and κ, when the criteria: $\|R\|_\infty \leq \gamma_\kappa \kappa$, is met for $\gamma_\kappa \in \mathbf{R}_+$. This criteria does not require strict satisfaction of subproblems and decreases the total number of iterations required by the solver. Outer updates on the central-path parameter and the penalty value are subsequently performed:

$$\kappa \leftarrow \max\Big(\kappa_{\min}, \min(\psi_\kappa \cdot \kappa, \kappa^{\zeta_\kappa})\Big), \quad \rho \leftarrow \min\Big(\rho_{\max}, \max(\phi_\rho \cdot \rho, 1/\kappa)\Big). \tag{6.25}$$

The updates are clipped to prevent unnecessarily small/large values.

6.4.7 Initialization

Given an initial guess for the primal variables x_{init}, the solver's variables are initialized with:

$$r = g(x_{\text{init}}),\ s_{\text{ineq}} = \mathbf{1}_q,\ s_{\text{so}}^{(i)} = (1, 1/10 \cdot \mathbf{1}_{l-1}),\ y = 0_m,\ z = 0_p,\ t = s, \tag{6.26}$$

where s_{ineq} and s_{so} are inequality and second-order variables, respectively, and are initialized strictly feasible. While more complex schemes may be effective, particularly for specific problems, this simple initialization works well in practice as a default setting.

Algorithm 4 CALIPSO

procedure OPTIMIZE($x, \theta, c, g, h, \mathcal{K}, \gamma$)
  Parameters: $\kappa = 1, \rho = 1, \lambda = 0$
  Initialize: r, s, y, z, t
  Until $\|R(w; \theta, y, \infty, 0)\|_\infty < \gamma_R$
    Until $\|R(w; \theta, \lambda, \rho, \kappa)\|_\infty < \gamma_\kappa \kappa$ **do**
      inertia correction: ϵ_p, ϵ_d ▷ Eq. (6.8)
      search direction: $\Delta w = (\Delta x, \Delta r, \Delta s, \Delta y, \Delta z, \Delta t)$ ▷ Eqs. (6.9 - 6.12)
      cone line search: α, α_t
      candidate: $\hat{x} = x + \alpha \Delta x$, $\hat{r} = r + \alpha \Delta r$, $\hat{s} = s + \alpha \Delta s$
      filter: $\hat{\varphi}$, $\hat{\eta}$
      update: $w \leftarrow (\hat{x}, \hat{r}, \hat{s}, y + \alpha \Delta y, z + \alpha \Delta z, t + \alpha_t t)$
    outer update: λ, ρ, κ ▷ Eq. (6.25, 2.41)
  Differentiate: $\partial w / \partial \theta$ ▷ Eq. 2.55
  Return $w, \partial w / \partial \theta$

6.4.8 Solution derivatives

The solution $w^*(\theta)$ returned by CALIPSO is differentiable with respect to its problem data θ. At a solution point, the residual is approximately zero (2.53), and sensitivities (2.55) are computed using $\partial R / \partial w = J$, which has already been computed and factorized, and:

$$\partial R / \partial \theta = (L_{x\theta}, 0_{m \times d}, 0_{p \times d}, g_\theta, h_\theta, 0_{p \times d}). \tag{6.27}$$

Additionally, the sensitivity of the solution with respect to each element of the problem data can be computed in parallel.

6.4.9 Implementation

The CALIPSO solver is summarized in Algorithm 4. An open-source implementation of the solver, `CALIPSO.jl`, written in Julia, is provided. The solver, and the following examples, are available at:

`https://github.com/thowell/CALIPSO.jl`.

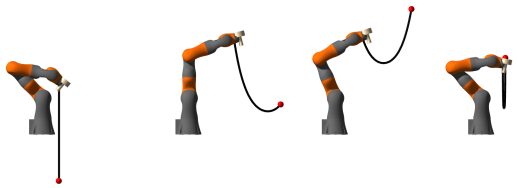
Transcribing problems for CALIPSO requires specifying the objective and constraint functions, and the number of primal variables. Trajectory-optimization problems are automatically formulated into the standard form (6.5). Gradients and sparse Jacobians are generated symbolically using the Julia package `Symbolics.jl`.

6.5 Results

We highlight the capabilities of CALIPSO by optimizing a collection of motion-planning problems from manipulation, locomotion, and aerospace domains that require second-order cone and complementarity constraints while transcribing constraints without approximation. Next, we demonstrate the ability to differentiate through the solver and auto-tune policies for non-convex underactuated robotic systems. Additional details about the experimental setups are available in the open-source implementation. A collection of non-convex problems are provided in the Appendix.

6.5.1 Contact-implicit trajectory optimization

CALIPSO is utilized to optimize contact-implicit trajectory optimization problems (2.35). The contact dynamics [30] are directly transcribed without modification. Comparisons are performed with Ipopt using the default MUMPS linear-system solver, and results are summarized in Table 6.2.



(a) ball-in-cup

(b) drifting

(c) quadruped gait

(d) bunny-hop

Figure 6.2: Contact-implicit trajectory optimization examples optimized with CALIPSO. (a) Ball attached to a string is swung into a cup by optimizing end-effector positions and forces. (b) Autonomous car plans a drifting maneuver in order to parallel park. (c) Gait for a quadruped is optimized via a single step and loop constraint. (d) Bicycle robot performs a bunny hop over an obstacle to reach a goal state.

Ball-in-cup. The position and applied forces of a robotic manipulator's end-effector are optimized to swing a ball into a cup (Fig. 6.2a). A string between the end-effector and ball is modeled with inequality and complementarity constraints. CALIPSO finds a physically realistic motion plan that is verified with inverse kinematics in simulation, while the Ipopt solution is of poor quality and violates physics by applying nonnegligible force to the ball while the string is slack.

Drifting. A parallel-park maneuver is planned that requires an autonomous vehicle to drift (i.e., plan a trajectory with both sticking and sliding contact) into a goal configuration between two

parked vehicles (Fig. 6.2b). The system is modeled as a Dubins car [95] with Coulomb friction applied to the wheels. Ipopt exhibits extremely poor converge and violates the friction cones to solve this problem, whereas CALIPSO finds a high-quality solution that leverages the nonlinear friction cone to slide into the narrow parking spot.

Quadruped gait. A gait is planned for a planar quadruped by optimizing a single step with a loop constraint (Fig. 6.2c). Ipopt struggles to converge, returning a solution with large complementarity violations and a reference that is unusable for online tracking. In contrast, CALIPSO finds a reference trajectory that satisfies the contact dynamics.

Bunny-hop. A bicycle robot performs a bunny-hop over an obstacle (Fig. 6.2d). The rider is modeled as a mass with actuated prismatic joints attached to the bike at each wheel. CALIPSO is able to converge to a trajectory where the bicycle hops over the obstacle by manipulating the rider mass, while Ipopt takes an order of magnitude more iterations to converge.

In summary, Ipopt struggles to return solutions that are useful for robotics applications, whereas CALIPSO reliably returns high-quality solutions while using exact constraint specifications.

Table 6.2: Comparison between CALIPSO and Ipopt for final objective value, constraint violation, and total iterations on contact-implicit trajectory optimization problems. Cases that failed to converge (i.e., Ipopt falling back to restoration mode) are highlighted in red. Without user-tuned smoothing and problem reformulations, Ipopt performs poorly on these examples and returns solutions that are unusable for robotics applications. In contrast, CALIPSO returns high-quality solutions and does not require approximating constraints.

Solver	Objective	Violation	Iterations
Ipopt	68.18	2.60e-2	205
CALIPSO	**11.96**	**4.86e-5**	**131**

(a) ball-in-cup

Solver	Objective	Violation	Iterations
Ipopt	8.66	1.00e-1	194
CALIPSO	**0.24**	**1.38e-5**	**189**

(b) drifting

Solver	Objective	Violation	Iterations
Ipopt	1855.18	1.13e-1	2000
CALIPSO	**574.84**	**5.32e-4**	**178**

(c) quadruped gait

Solver	Objective	Violation	Iterations
Ipopt	1503.42	**8.96e-8**	1409
CALIPSO	**462.61**	1.76e-6	**101**

(d) bunny-hop

6.5.2 State-triggered constraints

A trigger condition $\Gamma : \mathbf{R}^n \to \mathbf{R}$ encodes the logic: $\Gamma(x) > 0 \implies h(x) \geq 0$, that a constraint is enforced only when the trigger is satisfied. Such state-triggered constraints are utilized within various aerospace applications [121, 32] and commonly utilize a non-smooth formulation (left):

$$\min(0, -\Gamma(x)) \cdot h(x) \leq 0 \quad \rightarrow \quad \begin{aligned} \Gamma_+ - \Gamma_- &= \Gamma(x), \\ h_+ - h_- &= h(x), \\ \Gamma_+ \cdot h_- &= 0, \\ \Gamma_+, \Gamma_-, h_+, h_- &\geq 0, \end{aligned} \tag{6.28}$$

that linearizes poorly and can violate LICQ. In this work, we employ an equivalent complementarity formulation (right) to land an entry-vehicle in an environment with keep-out zones adjacent to the landing site. The resulting solution (Fig. 6.3) demonstrates CALIPSO's ability to find solutions that satisfy complementarity constraints. Ipopt returns a solution that violates the keep-out zone for the majority of the trajectory.

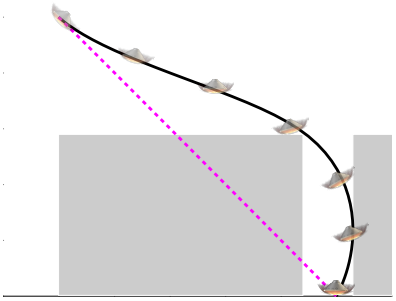


Figure 6.3: Entry-vehicle soft-landing plan that must avoid elevated regions to the left and right of the landing zone (gray), represented as state-triggered constraints. The unconstrained and constrained solutions are shown in magenta and black respectively.

6.5.3 Predictive control auto-tuning

CALIPSO is utilized offline to plan reference trajectories and online as the tracking controller in a model predictive control (MPC) policy. The policy aims to track the reference, given an updated state estimate from a simulation, by re-planning over a reduced horizon to compute a new (feedback) control that is then applied to the system.

The policy's cost weights are treated as problem data to be optimized and the solution computed by the policy (i.e., the re-optimized control inputs) are differentiated with respect to these parameters (2.55) in order to compute gradients that are utilized to automatically tune the policy. The metric for tuning the policy consists of quadratic costs on tracking the reference. Gradient descent with a line search is used to update the cost weights by differentiating through rollouts of the policy.

The policy weights are initialized with all ones and we compare the performance of the auto-tuned policy after 10 gradient steps with open-loop and untuned policies on swing-up tasks for cart-pole and acrobot systems. For both systems, the auto-tuned policy outperforms the baselines; results are summarized in Table 6.3.

Table 6.3: Comparison of tracking error between open-loop and both untuned and auto-tuned model predictive control policies in simulation. By differentiating through CALIPSO, gradient-based optimization is able to rapidly improve controller performance without requiring designer input.

	Open-Loop	**MPC (untuned)**	**MPC (tuned)**
cart-pole	5.11e4	15.06	**0.79**
acrobot	1.38e4	439.26	**0.04**

6.6 Limitations

In the non-convex setting it is generally not possible to guarantee that the optimizer will find a globally optimal solution. Further, while prior work analyzes the convergence properties for line-search filter methods [132], we leave this analysis for CALIPSO to future work.

Despite the numerical improvements of the solver, in many cases we still find that contact-implicit trajectory optimization problems are difficult to optimize. In practice we find that good initialization is crucial, but results that generate qualitatively different contact sequences compared to the initialization are rare.

6.7 Future Work

In this work, we have presented a new solver for trajectory optimization problems with second-order cone and complementarity constraints: CALIPSO. The solver prioritizes reliability and numerical robustness, and offers specialized constraint support that enables planning for challenging tasks arising in manipulation, locomotion, and aerospace applications while enabling the designer to exactly transcribe constraints without requiring problem reformulations. Additionally, the solver is differentiable with respect to its problem data, allowing it to be called by efficient, gradient-based, upper-level optimization routines for applications like policy auto-tuning. To the best of our knowledge, no existing solver offers this collection of unique features.

Future work will explore support for additional cones, which naturally fit within CALIPSO's interior-point framework, including semidefinite cones that are of interest in many control applications [134]—particularly settings with nonlinear dynamics. Additionally, extending the Julia implementation to C/C++ will potentially enable real-time performance of MPC policies onboard robots with limited computing hardware. The solver has potential to support state-triggered constraints online in safety-critical applications or be utilized in feedback loops for contact-implicit model predictive control [36] with systems that make and break contact with their environments.

Acknowledgments

The authors would like to thank Shane Barratt for helpful discussions related to cone programming. This work was supported by Frontier Robotics, Innovative Research Excellence, Honda R&D Co., Ltd. and an Early Career Faculty Award from NASA's Space Technology Research Grants Program (Grant Number 80NSSC21K1329).

6.8 Appendix: Non-Convex Optimization Problems

Three small, non-convex problems are optimized with CALIPSO.

Wächter problem. Motivating the development of a number of Ipopt's key algorithms, the following problem:

$$\begin{array}{ll} \underset{x_1,x_2,x_3}{\text{minimize}} & x_1 \\ \text{subject to} & x_1^2 - x_2 - 1 = 0, \\ & x_1 - x_3 - \frac{1}{2} = 0, \\ & x_2, x_3 \geq 0, \end{array} \tag{6.29}$$

when initialized with a point, $x_1 < 0$, $x_2 > 0$, $x_3 > 0$, causes many infeasible-start interior-point methods to fail [135]. Given the point $x_{\text{init}} = (-2, 3, 1)$, CALIPSO finds the optimal solution $x^* = (1, 0, \frac{1}{2})$ in 17 iterations.

Maratos problem. The following problem:

$$\begin{array}{ll} \underset{x_1,x_2}{\text{minimize}} & 2(x_1^2 + x_2^2 - 1) - x_1, \\ \text{subject to} & x_1^2 + x_2^2 - 1 = 0, \end{array} \tag{6.30}$$

highlights the Maratos effect [1], which often requires a solver to perform second-order corrections. CALIPSO's use of an augmented Lagrangian for the equality constraints allows for the omission of second-order corrections, allowing convergence on this problem from a starting point $x_{\text{init}} = (2, 1)$ to the optimal solution $x^* = (1, 0)$ in 6 iterations.

Complementarity problem. Challenging complementarity-constrained problems such as:

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & (x_1 - 5)^2 + (2x_2 + 1)^2 \\
\text{subject to} \quad & 2(x_2 - 1) - \tfrac{3}{2}x_2 + x_3 - \tfrac{1}{2}x_4 + x_5 = 0, \\
& 3x_1 - x_2 - x_6 - 3 = 0, \\
& -x_1 + \tfrac{1}{2}x_2 - x_7 + 4 = 0, \\
& -x_1 - x_2 - x_8 + 7 = 0, \\
& x_3 \cdot x_6 = 0, \\
& x_4 \cdot x_7 = 0, \\
& x_5 \cdot x_8 = 0, \\
& x \geq 0,
\end{aligned}
\tag{6.31}
$$

often require manual reformulations in order to be solved by general-purpose optimizers. With CALIPSO, we directly specify the problem, initialize the solver with $x_{\text{init}} = 0$, and find the optimal solution $x^* = (1, 0, 2, 0, 0, 0, 3, 6)$ in 12 iterations.

Chapter 7

Direct Policy Optimization

We present an approach for approximately solving discrete-time stochastic optimal-control problems by combining direct trajectory optimization, deterministic sampling, and policy optimization. Our feedback motion-planning algorithm uses a quasi-Newton method to simultaneously optimize a reference trajectory, a set of deterministically chosen sample trajectories, and a parameterized policy. We demonstrate that this approach exactly recovers LQR policies in the case of linear dynamics, quadratic objective, and Gaussian disturbances. We also demonstrate the algorithm on several nonlinear, underactuated robotic systems to highlight its performance and ability to handle control limits, safely avoid obstacles, and generate robust plans in the presence of unmodeled dynamics.

Direct Policy Optimization using Deterministic Sampling and Collocation. Taylor A. Howell, Chunjiang Fu, and Zachary Manchester. Robotics and Automation Letters. 2021.

7.1 Introduction

Trajectory optimization (TO) is a powerful tool for solving deterministic optimal-control problems in which accurate models of the system and its environment are available. However, when disturbances or unmodeled dynamics are significant, a stochastic optimal-control approach, in which a feedback policy is optimized directly, can often produce more robust performance [136].

Unfortunately, general solution methods for solving stochastic optimal-control problems suffer from the curse of dimensionality and are only applicable to low-dimensional systems in practice. To scale to many interesting robotic systems, approximations must be made. Typically, these include simplifying or linearizing dynamics, approximating value functions or policies with polynomials or neural networks, or approximating distributions with Gaussians or Monte Carlo sampling.

We present Direct Policy Optimization (DPO), a computationally tractable algorithm for finding approximate solutions to stochastic optimal-control problems that jointly optimizes a small number of trajectories and a policy. This algorithm combines several key ideas:

- Joint optimization of parameterized policies along with trajectories.
- Deterministic sampling of trajectories and approximation of expectations using the unscented transform.
- Direct collocation for enforcing dynamics along trajectories without performing rollouts.
- Use of large-scale constrained non-convex solvers based on quasi-Newton methods for fast and robust convergence.

In contrast to many other approaches, DPO is able to easily enforce constraints like torque limits and obstacle avoidance, makes extensive use of analytical models and their derivatives, and is extremely sample efficient.

We first provide background for the discrete-time stochastic optimal-control problem, present related work on feedback motion planning in Section 7.2, and give an overview of the unscented transform in Section 7.3. In Section 7.4, we present the DPO algorithm. We then demonstrate that DPO exactly recovers LQR policies when the dynamics are linear, the objective is quadratic, and disturbance inputs are Gaussian, and provide examples using DPO for several non-convex, underactuated-robot control problems in Section 7.5. Section 7.6 offers discussion of the experimental results and limitations. Finally, we summarize our work and propose directions for future research in Section 7.7.

7.2 Related Work

Model-based approaches often tackle problem (7.1) in a decoupled fashion: First, generate a reference trajectory assuming no disturbances; then, design a tracking feedback policy to reject disturbances.

Using collocation methods [102] or differential dynamic programming (DDP) [23] to optimize a trajectory, then tracking it with a time-varying LQR controller has achieved impressive results for complex, real-world systems [137, 138].

There are two primary drawbacks to these classic synthesis techniques: First, there is no explicit consideration of uncertainty or disturbances. Robustness is often achieved heuristically via *post hoc* Monte Carlo testing and tuning. Second, by decoupling the synthesis of the reference trajectory and policy, performance is often sacrificed.

DIRTREL [34] optimizes a reference trajectory with an additional cost term that penalizes a linearized approximation of the tracking error from an LQR policy. Chance constraints are approximated by enforcing constraints at a finite number of samples. There is also a variation of DDP that can account for multiplicative noise applied to the controls [139]. Unlike these methods, DPO directly optimizes policies and can propagate uncertainty through nonlinear dynamics.

Several learning-based approaches exist that directly optimize feedback policies. Policy gradient methods [140, 141] use first-order or derivative-free methods to optimize parameterized policies, typically without direct access to the underlying dynamics model. Domain randomization [142] can be employed to vary model and environment parameters to encourage policy robustness. Random search [83], stochastic gradient descent [143], and Newton methods [144] have also been used to optimize linear feedback and MPC policies [97, 96]. A major benefit of stochastic methods is their inherent exploration of the policy space.

Guided Policy Search (GPS) [145] is a hybrid approach that alternates between optimizing sample trajectories and fitting a policy. DDP is used to generate high-reward trajectories around the current policy that are subsequently employed to improve the policy. This procedure is then iterated with a regularizer to keep new trajectories close to those generated by the previous policy.

While GPS does not make strong assumptions about state and disturbance distributions, it relies heavily on Monte Carlo techniques that require a large number of samples. As a result, training can require significant time and computational resources. In contrast, DPO leverages analytical models and uses only a small number of deterministically chosen samples, making it much more efficient.

Finally, we note that many of the approaches outlined in this section—including DDP and many of the policy gradient methods—rely on explicit forward simulation or "rollouts" along with some form of backpropagation. These techniques are vulnerable to numerical ill-conditioning issues colloquially known as the "tail-wagging-the-dog problem" in control and the "vanishing" or "exploding" gradient problem in reinforcement learning. In contrast, DPO employs collocation methods that simultaneously optimize state and control trajectories with dynamics enforced as constraints. Such "direct" methods enjoy far better numerical conditioning and robustness, especially with long-horizon plans.

7.3 Background

This section provides brief reviews of the discrete-time stochastic optimal-control problem and the unscented transform, as well as a survey of related solution approaches.

7.3.1 Discrete-time stochastic optimal control

We formulate the discrete-time stochastic optimal-control problem as:

$$\begin{aligned} \underset{\Theta}{\text{minimize}} \quad & \mathbf{E}[J(\tau)] & & \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t, w_t), & & t = 1, \ldots, T-1, \\ & u_t = \pi_t(x_t, \theta_t), & & t = 1, \ldots, T-1, \\ & \mathbf{Prob}(c_j(\tau) > 0) \leq \epsilon_j, & & j = 1, \ldots, k. \end{aligned} \tag{7.1}$$

The system's state, $x_t \in \mathbf{R}^n$, and control inputs, $u_t \in \mathbf{R}^m$, define a trajectory:

$$\tau = (x_1, \ldots, x_T, u_1, \ldots, u_{T-1}) \in \mathbf{R}^z, \tag{7.2}$$

with a subscript denoting the time index t, over a planning horizon T. The initial state, x_1, is a random variable. The discrete-time stochastic dynamics, $f_t : \mathbf{R}^n \times \mathbf{R}^m \times \mathcal{W} \to \mathbf{R}^n$, are subject to random disturbance inputs, $w_t \in \mathcal{W}$. We seek a policy, $\pi_t : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}^m$, parameterized by $\Theta = (\theta_1, \ldots, \theta_{T-1}) \in \mathbf{R}^{p(T-1)}$, to minimize the objective, $J : \mathbf{R}^z \to \mathbf{R}$, with the expectation taken over the initial state and disturbance inputs. Chance constraints, with $c_j : \mathbf{R}^z \to \mathbf{R}$ and probability of violation less than tolerance $\epsilon_j \in \mathbf{R}_+$, can further constrain trajectories.

7.3.2 Unscented transform

The unscented transform is a procedure for propagating a unimodal probability distribution through a nonlinear function using deterministic samples, often referred to as sigma points. This tool is commonly used for state estimation, where it is generally considered to be superior to the extended Kalman filter's linear propagation of covariance matrices [146]. There are many variations of the unscented transform [147]; the version we utilize is visualized in Fig. 7.1 and outlined below.

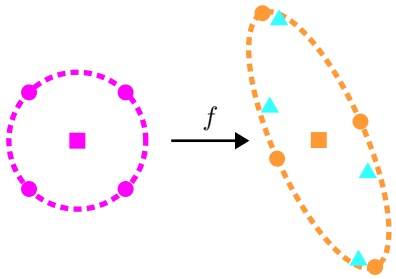


Figure 7.1: Unscented transform visualized in 2D for initial (left) and transformed (right) distributions. Sigma points (circles) with sample mean (square) and sample covariance (dashed) are propagated through a nonlinear function (triangles) and then resampled to compute new sigma points.

Assuming a unimodal state distribution with mean $\mu_t \in \mathbf{R}^n$, covariance $P_t \in \mathbf{S}^n_{++}$, and an uncorrelated zero-mean disturbance distribution with covariance $D_t \in \mathbf{S}^d_{++}$, we generate $N = 2(n + d)$ sigma points:

$$\begin{bmatrix} x_t^{(i)} \\ w_t^{(i)} \end{bmatrix} \leftarrow \begin{bmatrix} \mu_t \\ 0 \end{bmatrix} \pm \beta_t \, \mathbf{col}\left(\sqrt{\begin{bmatrix} P_t & 0 \\ 0 & D_t \end{bmatrix}}\right). \tag{7.3}$$

These samples, denoted with superscripts, are constructed using the column vectors of a square root of the joint covariance matrix. In this work we use the principle (symmetric) matrix square root, although other decompositions, such as the Cholesky factorization, could be employed. The parameter $\beta_t \in \mathbf{R}_+$ controls the spread of the samples around the mean. For a Gaussian distribution propagated through linear dynamics, the unscented transform will exactly recover the updated distribution. For nonlinear systems, the selection of β can affect performance, and has been explored extensively in the literature on unscented Kalman filters [147].

Sample points are first propagated through the policy:

$$u_t^{(i)} = \pi_t(x_t^{(i)}), \tag{7.4}$$

and then sample dynamics:

$$x_{t+1}^{(i)} = f_t^{(i)}(x_t^{(i)}, u_t^{(i)}, w_t^{(i)}), \tag{7.5}$$

in order to compute a sample mean:

$$\mu_{t+1} = \frac{1}{N} \sum_{i=1}^{N} x_{t+1}^{(i)}, \tag{7.6}$$

and sample covariance:

$$P_{t+1} = \frac{1}{2\beta_t^2} \sum_{i=1}^{N} (x_{t+1}^{(i)} - \mu_{t+1})(x_{t+1}^{(i)} - \mu_{t+1})^T, \tag{7.7}$$

for the state distribution at the next time step. Similarly, we compute a sample mean:

$$\nu_t = \frac{1}{N} \sum_{i=1}^{N} u_t^{(i)}, \tag{7.8}$$

and sample covariance:

$$L_t = \frac{1}{2\beta_t^2} \sum_{i=1}^{N} (u_t^{(i)} - \nu_t)(u_t^{(i)} - \nu_t)^T, \tag{7.9}$$

for the control inputs at the current time step.

7.4 Direct Policy Optimization

We now present the Direct Policy Optimization algorithm. DPO makes several strategic approximations to the discrete-time stochastic optimal-control problem: First, the expectation of the objective in (7.1) is approximated using the unscented transform. Second, DPO explicitly optimizes a reference trajectory, $\bar{\tau}$, and N sample trajectories, $\tau^{(i)}$, in order to approximate the stochastic dynamics. We use an overbar to denote reference or nominal quantities and superscripts to denote sample indices throughout the chapter. Next, we seek a local feedback policy that is valid in the neighborhood of the reference trajectory. Finally, chance constraints are approximately enforced by applying inequality constraints to a set of sample points chosen from level sets of the state and input distributions. Using these approximations, we formulate an optimization problem that is amenable to large-scale quasi-Newton solvers.

7.4.1 Objective

DPO minimizes the following objective:

$$J(\bar{\tau}) + \mathbf{E}[S(\bar{\tau}, \tau)], \tag{7.10}$$

with cost function J applied to the reference trajectory and a quadratic tracking-cost function:

$$\begin{aligned} S(\bar{\tau}, \tau) = &(x_T - \bar{x}_T)^T Q_T (x_T - \bar{x}_T) \\ &+ \sum_{t=1}^{T-1} \{(x_t - \bar{x}_t)^T Q_t (x_t - \bar{x}_t) \\ &+ (u_t - \bar{u}_t)^T R_t (u_t - \bar{u}_t)\}, \end{aligned} \tag{7.11}$$

$S : \mathbf{R}^z \times \mathbf{R}^z \to \mathbf{R}$, with $Q_t \in \mathbf{S}_+^n$ and $R_t \in \mathbf{S}_+^m$, that penalizes deviations from the reference trajectory under disturbances. After simple manipulations, we can write (7.10) in terms of the quantities introduced in Sec. 7.3.2:

$$\mathbf{E}[(x_t - \bar{x}_t)^T Q_t (x_t - \bar{x}_t)] = \mathbf{Tr}(P_t Q_t) + (\mu_t - \bar{x}_t)^T Q_t (\mu_t - \bar{x}_t), \tag{7.12}$$

$$\mathbf{E}[(u_t - \bar{u}_t)^T R_t (u_t - \bar{u}_t)] = \mathbf{Tr}(L_t R_t) + (\nu_t - \bar{u}_t)^T R_t (\nu_t - \bar{u}_t). \tag{7.13}$$

Since the expectation in (7.10) depends only on the first and second moments of the state and control distributions, we can efficiently compute it using the unscented transform. As an aside, the second terms in (7.12) and (7.13) are zero in the linear-quadratic Gaussian (LQG) case, and their presence reflects the invalidity of the separation principle in more general settings.

7.4.2 Unscented dynamics

DPO utilizes direct collocation and enforces dynamics for each trajectory via equality constraints at each time step. A unimodal distribution over the state and policy is maintained along the planning horizon by resampling the sample trajectories at each time step using the unscented transform (7.3-7.9); this procedure is summarized in Algorithm 5. For motion-planning applications, maintaining a unimodal distribution over a planning horizon is a reasonable modeling choice because the policy explicitly works to keep sample trajectories near the reference. In addition to computing terms required by the expectation over the objective, resampling at each time step prevents samples from collapsing to the reference at later time steps, encouraging the policy to be robust throughout the entire trajectory.

Initial sample states and disturbance inputs are deterministically drawn from normal distributions, $x_1^{(i)} \sim \mathcal{N}(\mu_1, P_1)$ and $w_t \sim \mathcal{N}(0, D_t)$, using (7.3). The initial reference state is $\bar{x}_1 = \mu_1$. By utilizing the discrete-time dynamics, it is possible to generate non-Gaussian noise for the system and capture model uncertainty.

Algorithm 5 Unscented dynamics

function $g_t(\tau_t^{(1:N)}, D_t)$
  for $i = 1 : N$ **do**
    $x_t^{(i)},\ w_t^{(i)} \leftarrow$ compute sigma points (7.3)
    $u_t^{(i)},\ x_{t+1}^{(i)} \leftarrow$ propagate sigma points (7.4, 7.5)
  $\mu_{t+1},\ \nu_t \leftarrow$ compute sample means (7.6, 7.8)
  $P_{t+1},\ L_t \leftarrow$ compute sample covariances (7.7, 7.9)
  return $x_{t+1}^{(1:N)}$

7.4.3 Feedback policy

Each sample trajectory is subject to a policy constraint:

$$u_t^{(i)} = \pi_t(x_t^{(i)}, \bar{x}_t, \bar{u}_t, \theta_t), \tag{7.14}$$

at each time step. Instead of optimizing global policies, we search for local feedback policies that can depend on the reference trajectory. While these policies can be from any differentiable function class, we focus on linear policies for simplicity and leave extensions to more complex functions to future work.

7.4.4 Chance constraints

Chance constraints are approximated by constraining the reference and sample trajectories. The probability of violation, ϵ, corresponds to particular level sets of the state and control distributions. The sampling parameter β can be selected in order to sample sigma points that approximate these level sets. Constraints are only enforced at these points. While more accurate methods for evaluating chance constraints exist, they are much more computationally demanding. Instead, DPO trades computational tractability for exact guarantees.

7.4.5 Formulation

DPO can be formulated as the following non-convex optimization problem:

$$\begin{aligned}
\underset{\Theta,\bar{\tau},\tau^{(1:N)}}{\text{minimize}} \quad & J(\bar{\tau}) + \sum_{i=1}^{N} S(\bar{\tau}, \tau^{(i)}) && && (7.15)\\
\text{subject to} \quad & \bar{x}_{t+1} = \bar{f}_t(\bar{x}_t, \bar{u}_t, 0), && t = 1, \ldots, T-1,\\
& x_{t+1}^{(1:N)} = g_t(\tau_t^{(1:N)}, D_t), && t = 1, \ldots, T-1,\\
& u_t^{(1:N)} = \pi_t(x_t^{(1:N)}, \bar{x}_t, \bar{u}_t, \theta_t), && t = 1, \ldots, T-1,\\
& c_j(\bar{\tau}) \leq 0, && j = 1, \ldots, k,\\
& c_j(\tau^{(1:N)}) \leq 0, && j = 1, \ldots, k.
\end{aligned}$$

Additional constraints, for example, conditions on the policy parameters or trajectories, can be directly added to the formulation. Off-the-shelf large-scale non-convex solvers like Ipopt [26] and SNOPT [27] can be employed to efficiently optimize (7.15) by taking advantage of sparsity in its associated Jacobian and Hessian matrices.

7.5 Results

The following examples were implemented in Julia, used SNOPT to optimize (7.15), and were performed on a laptop computer with an Intel Core i7-7600U 2.80 GHz CPU and 16 GB of memory.

To verify the performance of DPO, optimized policies are simulated at ten times the sample rate used during optimization, systems are simulated with explicit third-order Runge-Kutta integration, zero-order-hold control interpolation, cubic spline state interpolation, and are subject to additive zero-mean Gaussian noise. TO and DPO utilize the same objective for the reference trajectory. Throughout, the tracking cost (7.11) and LQR policies have the same weights and, unless specified, $\beta = 1$. The reported tracking error is computed using the tracking cost (7.11) for a single trajectory. During optimization we employ implicit midpoint integration and use discrete dynamics with additive noise:

$$x_{t+1}^{(i)} = f_t^{(i)}(x_t^{(i)}, u_t^{(i)}) + w_t^{(i)}. \tag{7.16}$$

We find that additive noise with state augmentation and different sample models is quite general and capable of capturing interesting dynamical effects. Policies use linear feedback to track the reference trajectory:

$$u_t^{(i)} = \bar{u}_t - \theta_t(x_t^{(i)} - \bar{x}_t). \tag{7.17}$$

Our implementation of DPO and additional details for each example are available at:

`http://roboticexplorationlab.org/projects/dpo.html`.

7.5.1 Double integrator

We first demonstrate that DPO recovers the LQR solution for double integrator dynamics:

$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t + w_t, \tag{7.18}$$

with $n = 2$ states, $m = 1$ controls, and $d = 2$ disturbances, regulated to the origin over a horizon $T = 51$. The policy has $p = 2$ parameters at each time step. Initial states are sampled from a distribution with $\mu_1 = 0$ and $P_1 = I$, disturbances have $D_{1:T-1} = I$, the weights are $Q_{1:T} = I$ and $R_{1:T-1} = I$.

The decision variables are initialized by uniformly sampling between -1 and 1 for 1000 trials. The maximum, mean, and standard deviations of the normalized error between the DPO policies and the exact LQR solution computed with the Riccati equation are near the tolerance of the solver: 2.4e-5, 4.0e-7, 8.5e-7, respectively—reinforcing that the approximations made in the derivation of DPO are exact in the LQG case. Additionally, the second term in (7.12) is zero, demonstrating that the separation principle holds for this problem.

7.5.2 Car

We plan a collision-free path through four obstacles for an autonomous car, modeled as a unicycle [95]:

$$\begin{bmatrix} \dot{y} \\ \dot{z} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\phi) \\ v \cdot \sin(\phi) \\ \omega \end{bmatrix}. \tag{7.19}$$

The $n = 3$ states are positions y and z, and orientation ϕ. The $m = 2$ controls are the forward and angular velocities, v and ω. There are $d = 3$ disturbance inputs and the policy has $p = 6$ parameters at each time step. Initial states are sampled from a distribution with $\mu_1 = 0$, $P_1 = \mathbf{diag}(1, 1, 0.1)$, weights are $Q_{1:T-1} = \mathbf{diag}(10, 10, 1)$, $Q_T = 100I$, and $R_{1:T-1} = 0.1I$, and disturbances have $D_{1:T-1} = \mathbf{diag}(\delta, \delta, 0.1\delta)$. The planning horizon is $T = 51$ with fixed time step $h = 0.02$. The choice of $\beta = 1$ over the planning horizon results in constraints being enforced on the 1-sigma level set of the state distribution, corresponding to $\epsilon = 0.32$.

For this scenario, TO finds a path that is in close proximity to the obstacles. In contrast, by considering noise, DPO with $\delta = 0.001$ finds a path that remains a safe distance from the obstacles. Decreasing the noise results in convergence to the TO solution. However, larger disturbance inputs (e.g., $\delta = 0.01$), expose a limitation of DPO; samples can diverge, resulting in a multimodal state distribution with trajectories taking different paths around the obstacles (Fig. 7.2). Ultimately, unlike *ad hoc* techniques like obstacle inflation, DPO considers the coupling of dynamics, constraints, and disturbances to generate safer paths for the system.

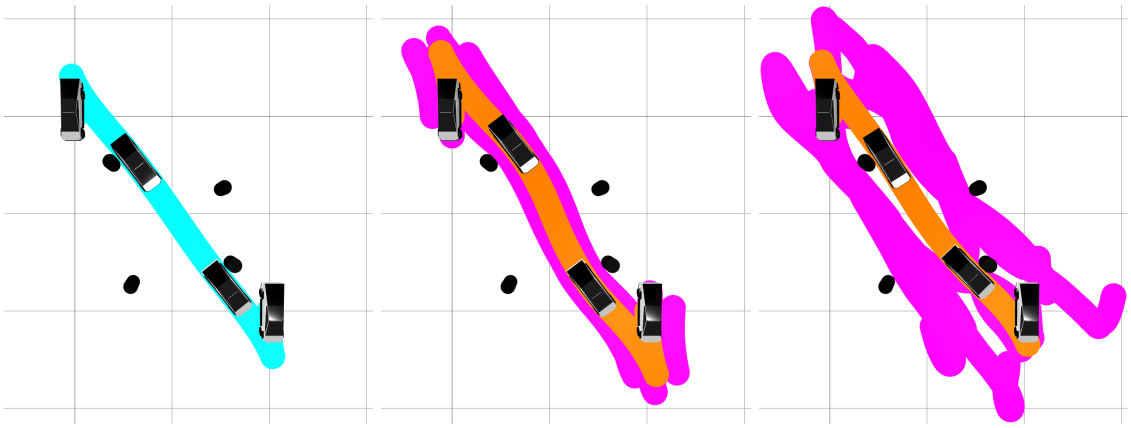


Figure 7.2: Position trajectories for autonomous car avoiding obstacles. TO (left) finds a path that is near the obstacles, whereas DPO with $\delta = 0.001$ (center) finds a path that avoids obstacles with a margin for safety. With larger disturbances, $\delta = 0.01$, samples (magenta) diverge and the DPO policy fails to safely avoid obstacles (right).

7.5.3 Cart-pole

A swing-up trajectory for a cart-pole [94] with a slider that experiences Coulomb friction is synthesized for horizon $T = 51$ and fixed time step $h = 0.1$. The state, $x = (y, \phi, \dot{y}, \dot{\phi})$, has cart position y, pendulum orientation ϕ, and their respective time derivatives. The optimality conditions for the maximum-dissipation principle [42] are used as constraints to explicitly model friction [37] with coefficient $\mu_{\mathrm{f}} = 0.1$ for each trajectory. The cart position is controlled and $m = 1$, there are $d = 4$ disturbance inputs, and the policy has $p = 4$ parameters at each time step. Initial states are sampled from $\mu_1 = 0$ and $P_1 = I$, the weights are $Q_{1:T-1} = \mathbf{diag}(10, 10, 1, 1)$, $Q_T = 100I$, and $R_{1:T-1} = I$, and disturbances have $D_{1:T-1} = 0.001I$.

The performance of LQR tracking reference trajectories optimized subject to friction, as well as the DPO policy, are verified in simulation. Results are provided in Table 7.1 and tracking for the position and orientation is shown in Fig. 7.3.

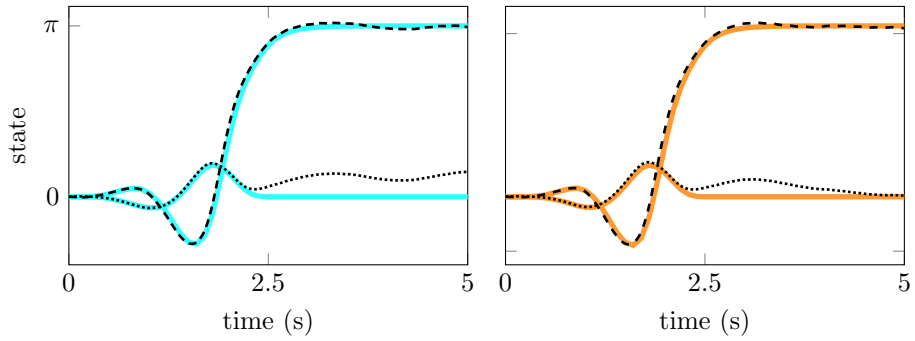


Figure 7.3: Simulated tracking (black) for position (dotted) and orientation (dashed) of cart-pole experiencing Coulomb friction. Comparison between LQR (blue) and DPO (orange) policies.

By first designing a trajectory that explicitly models friction, it is possible to subsequently synthesize an LQR policy for tracking. In contrast, DPO, which can handle nonsmooth dynamics, produces superior tracking by simultaneously optimizing the reference trajectory and policy.

Table 7.1: Tracking error, computed with (7.11), for cart-pole experiencing Coulomb friction. Comparison between LQR and DPO policies.

Error	**LQR**	**DPO**
state	3.18	**2.26**
control	0.84	**0.39**
total	4.02	**2.64**

7.5.4 Rocket landing

We plan a soft landing for a rocket [148]. The nominal system with planar dynamics has state $x = (y, z, \phi, \dot{y}, \dot{z}, \dot{\phi})$, with lateral and vertical positions y and z, orientation ϕ, and their respective time derivatives. The rocket is controlled with gimballed thrust, $u = (F_y, F_z)$.

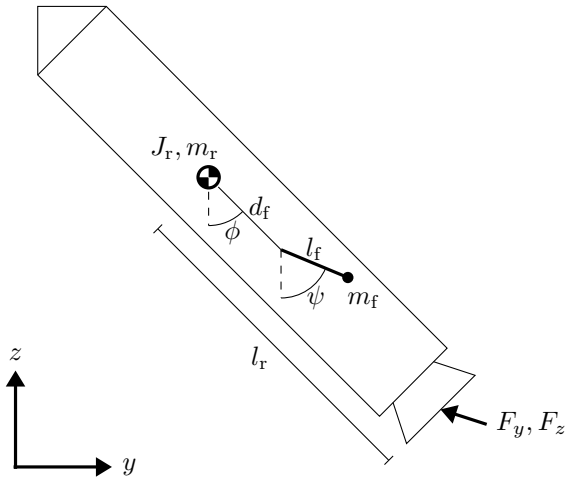


Figure 7.4: Rocket with planar dynamics. The model has lateral and vertical positions y and z, orientation ϕ, and thrust inputs F_y and F_z. Parameters are inertia J_r, mass m_r, and length l_r from center of mass to thruster. A pendulum positioned d_f from rocket center of mass, with orientation ψ, length l_f, and mass m_f, models fuel slosh.

It is initialized with non-zero displacements and velocities relative to its target state: a zero-velocity, vertical-orientation touchdown inside a landing zone.

The rocket experiences fuel slosh during landing that is not accounted for in the nominal model. This is a critical dynamical effect, but it is difficult to model and observe. In practice, these unmodeled effects are typically handled in *ad hoc* ways, often with extensive Monte Carlo simulation and controller tuning. To approximately model fuel slosh, we augment the nominal model with two additional states associated with an unobservable and unactuated pendulum (Fig. 7.4). A common frequency-domain control approach is to include a notch filter at the pendulum's natural frequency in order to prevent excitation of these fuel-slosh dynamics. We instead use DPO to synthesize a policy that is robust to fuel slosh.

The reference trajectory is optimized with the nominal model, while the sample trajectories are subject to the augmented model in order to capture fuel-slosh dynamics, and a linear output feedback policy is optimized that does not have access to the fuel-slosh states. There are $d = 8$ disturbance inputs to the augmented model and the policy has $p = 12$ parameters at each time step. Initial states for the augmented model are sampled with uncertainty $P_1 = I$, weights are $Q_{1:T-1} = 100I$, $Q_T = 1000I$, and $R_{1:T-1} = \mathbf{diag}(1, 1, 100)$, and disturbances have $D_{1:T-1} = 0.001I$. Over the planning horizon $T = 41$ with free final time, thrust limits are enforced.

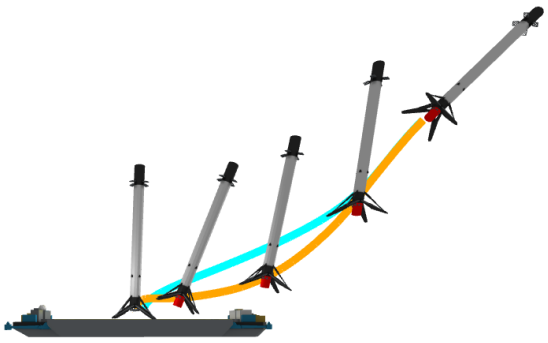


Figure 7.5: Rocket soft-landing position trajectories generated by TO (blue) and DPO (orange). Unlike the TO solution tracked with LQR, the DPO policy successfully lands the rocket despite fuel slosh.

TO finds a 2.72 second solution, whereas DPO finds a longer 2.91 second path to the landing zone. The position trajectories are shown in Fig. 7.5, simulation results with fuel slosh are provided in Table 7.2, and tracking results for the rocket's orientation are shown in Fig. 7.6. Due to fuel slosh, LQR fails to track the path generated by TO and the rocket tips over, whereas the DPO policy successfully lands the rocket.

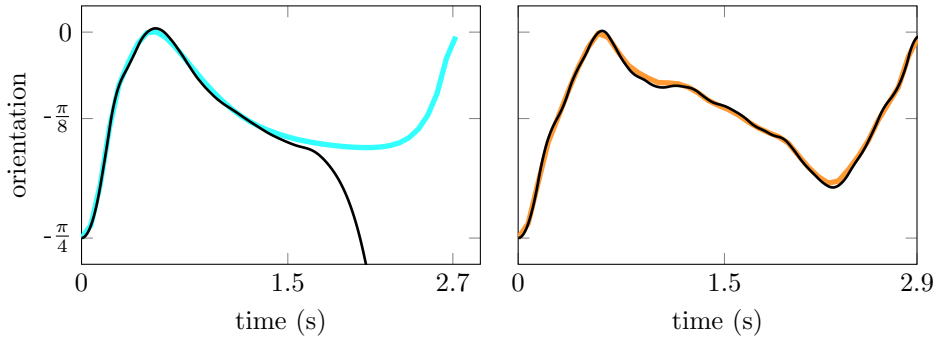


Figure 7.6: Simulated tracking (black) for the orientation of a rocket experiencing fuel slosh during landing. Comparison between LQR (blue) and DPO (orange) polices.

DPO is able to optimize a policy for a system with unobservable dynamical effects by sampling augmented models.

Table 7.2: Tracking error, computed with (7.11), for rocket landing with fuel slosh. Comparison between LQR and DPO policies.

Error	LQR	DPO
state	4725.39	**5.02**
control	75.21	**0.39**
total	4800.60	**5.41**

7.5.5 Quadrotor

A point-to-point maneuver is planned for a quadrotor [149] that experiences a random blade breaking off from a propeller during flight.

A broken propeller is modeled by constraining the corresponding control input to be half its nominal maximum value. We use DPO to optimize a policy that is robust to this event occurring for any of the propellers. The reference model has no broken propellers, but the sample trajectories are optimized with different models from four groups, each experiencing a different broken propeller. There are $n = 12$ states, $m = 4$ controls, $d = 12$ input disturbances, and the policy has $p = 48$ parameters at each time step. Initial states are sampled around the initial nominal state with uncertainty $P_1 = I$, weights are $Q_{1:T-1} = 10I$, $Q_T = 100I$, and $R_{1:T-1} = I$, and disturbances have $D_{1:T-1} = 0.001I$. Thrust limits are enforced over the planning horizon $T = 31$ with a free final time.

TO finds a 2.71 second trajectory, while DPO finds a longer 3.38 second trajectory with lower maximum controls (Fig. 7.7).

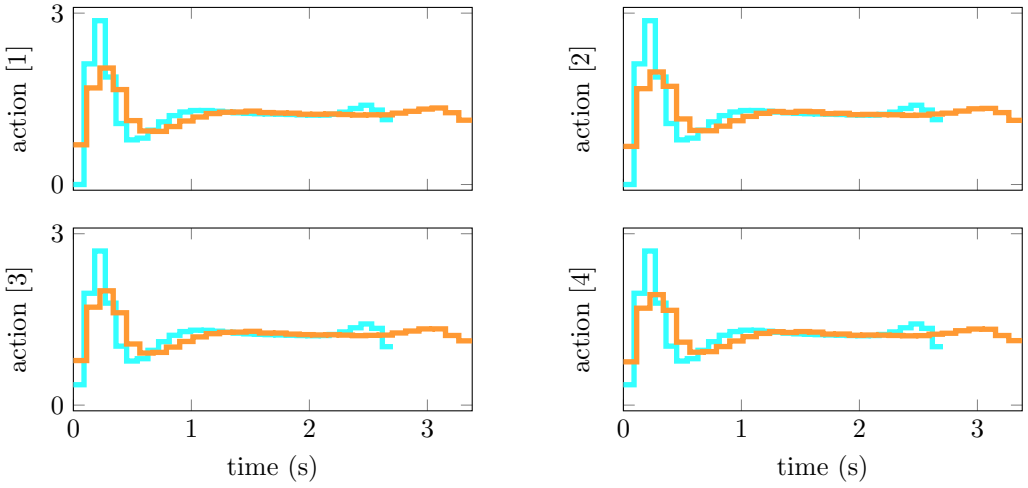


Figure 7.7: Nominal controls for quadrotor performing point-to-point maneuver generated with TO (blue) and DPO (orange). The controls found with DPO are applied over a longer horizon and have lower maximum values compared to TO.

The policies are compared in simulation over 100 initial conditions with noise sampled from $\mathcal{N}(0, 0.001I)$. One set of initial conditions is visualized in Fig. 7.8.

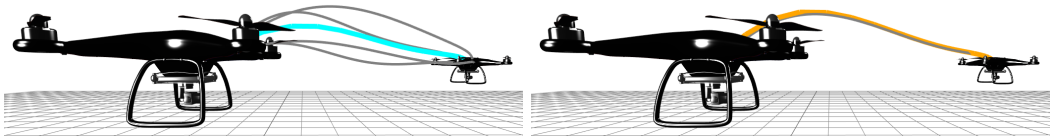


Figure 7.8: Simulated position trajectories (gray) for quadrotor controlled with LQR (top) and DPO (bottom) policies while experiencing different propellers breaking. LQR (blue) has degrading tracking while DPO (orange) has superior, and nearly identical tracking regardless of which propeller breaks.

The average tracking performance over each propeller breaking is provided in Table 7.3. DPO finds a policy with consistent and superior tracking compared with LQR for all cases by optimizing over different models.

Table 7.3: Tracking errors with mean (μ) and standard deviation (σ), computed with (7.11), comparing LQR and DPO policies for quadrotor over 100 initial conditions for each propeller being broken.

Error (μ, σ)	**LQR**	**DPO**
state	(13.43, 12.64)	**(4.74, 5.87)**
control	(0.074, 0.067)	**(0.020, 0.022)**
total	(13.50, 12.71)	**(4.76, 5.89)**

(a) propeller 1 broken

Error (μ, σ)	**LQR**	**DPO**
state	(13.13, 11.98)	**(4.74, 5.87)**
control	(0.073, 0.065)	**(0.020, 0.023)**
total	(13.21, 12.05)	**(4.76, 5.89)**

(b) propeller 2 broken

Error (μ, σ)	**LQR**	**DPO**
state	(3431.91, 12895.93)	**(4.74, 5.88)**
control	(2.21, 7.86)	**(0.020, 0.023)**
total	(3434.12, 12903.75)	**(4.76, 5.90)**

(c) propeller 3 broken

Error (μ, σ)	**LQR**	**DPO**
state	(4843.89, 14892.19)	**(4.74, 5.87)**
control	(3.21, 9.41)	**(0.020, 0.023)**
total	(4847.10, 14901.59)	**(4.76, 5.89)**

(d) propeller 4 broken

7.6 Limitations

We empirically demonstrate through Monte Carlo simulations that DPO recovers LQR policies for a double integrator system. It seems possible that for LQG problems, despite the non-convexity of the policy constraints, there exists a unique global solution for DPO. However, we leave a formal convergence analysis to future work.

The motion-planning examples highlight a number of DPO's capabilities compared to DIRTREL

and GPS. In the cart-pole example, DPO is able to subject each trajectory to discontinuous Coulomb friction calculated with the maximum-dissipation principle, whereas a similar DIRTREL example only applied white-noise disturbances to a single model [34]. This example also highlights how jointly optimizing trajectories and a policy leads to superior performance, a distinction compared to the decoupled approach employed by GPS. Next, in the rocket landing example, DPO optimizes an output feedback policy, whereas DIRTREL is limited to state feedback. Additionally, directly modeling fuel-slosh dynamics with a pendulum is likely simpler than designing input disturbances for DIRTREL to capture the same effects. Finally, with DPO, additional constraints can be applied to any of the decision variables, whereas the ability of GPS to handle constraints using DDP and stochastic gradient descent is limited.

Because we use local optimization methods to solve DPO, providing good initial guesses to the solver is crucial for performance of the algorithm. In practice, we use a standard TO solution to warm start the reference and sample trajectories for DPO. For linear state-feedback policies, we use the corresponding LQR solution as an initial guess. While not necessary in any of our examples, initial guesses for more complex policies could be found by running DPO without the policy constraint, then performing an offline regression to fit approximate policy parameters for warm starting.

For a TO problem solved with a sparsity-exploiting second-order method, the computational complexity is approximately $O\Big(T(n+m)^3\Big)$ [114]. For DPO, we can consider an augmented state and control with dimensions $n(2n+2)$ and $m(2n+2)$, respectively. The complexity of DPO is, therefore, $O\Big(T(n^6+m^3n^3)\Big)$. In the examples, which do not employ a specialized solver, solution times range from seconds to 1.5 hours for a state with $n = 12$ on a laptop computer. It is likely possible to exploit the DPO problem's structure to improve the complexity by using custom linear solvers. First-order methods, which can scale much better, become attractive for DPO as problems become large.

Lastly, LQR is a powerful tool and can likely be tuned to qualitatively match the performance of the linear policies found with DPO in many cases. However, the strength of DPO lies in its ability to explicitly reason about robustness and the complex coupling between dynamics, constraints, and disturbances during synthesis, instead of relying on hand tuning and heuristics.

7.7 Future work

We have presented a new algorithm, Direct Policy Optimization, for approximately solving stochastic optimal-control problems. We demonstrate that the algorithm is exact in the LQG case and is capable of optimizing policies for nonlinear systems that respect control limits and obstacles, and are robust to unmodeled dynamics.

Many interesting avenues for future work remain: Extensions to nonlinear policies could be made by introducing high-dimensional features and regularization or constraints on policy parameters, or

neural networks could be used to parameterize policies since the policy parameters scale much better compared to state and control dimensions. Another potential direction is to optimize a local value function approximation in place of an explicit policy. A much richer class of disturbances and model parameter uncertainty could also be modeled by augmenting the state vector and dynamics. Lastly, more complex systems with larger state and input dimensions may be more amenable to optimization with first-order or matrix-free methods.

Chapter 8

Contact-Implicit Predictive Control

We present a general approach for controlling robotic systems that make and break contact with their environments. Contact-implicit model predictive control (CI-MPC) generalizes linear MPC to contact-rich settings by utilizing a bi-level planning formulation with lower-level contact dynamics formulated as time-varying linear complementarity problems (LCPs) computed using strategic Taylor approximations about a reference trajectory. These dynamics enable the upper-level planning problem to reason about contact timing and forces, and generate entirely new contact-mode sequences online. To achieve reliable and fast numerical convergence, we devise a structure-exploiting interior-point solver for these LCP contact dynamics and a custom trajectory optimizer for the tracking problem. We demonstrate real-time solution rates for CI-MPC and the ability to generate and track non-periodic behaviours in hardware experiments on a quadrupedal robot. We also show that the controller is robust to model mismatch and can respond to disturbances by discovering and exploiting new contact modes across a variety of robotic systems in simulation, including a pushbot, planar hopper, planar quadruped, and planar biped.

Fast Contact-Implicit Model Predictive Control. Simon Le Cleac'h*, Taylor A. Howell*, Shuo Yang, Chi-Yen Lee, John Zhang, Arun Bishop, Mac Schwager, and Zachary Manchester. arXiv 2107.05616. 2022.

8.1 Introduction

Controlling systems that make and break contact with their environments is one of the grand challenges in robotics. Numerous approaches have been employed for controlling such systems, ranging from hybrid-zero dynamics [45, 150, 151], to complementarity controllers [152], to neural-network policies [153, 65], and model predictive control (MPC) [154, 155]. There have also been numerous successes deploying such approaches on complex systems in recent years: direct trajectory optimization and LQR on Atlas [137], smooth-contact models and differential dynamic programming on HRP-2 [40, 156, 99], zero-moment point and feedback linearization on ASIMO [157], and MPC with simplified dynamics models on Cheetah [158] and ANYmal [57]. However, reliable *general-purpose* control techniques that can reason about contact events and can be applied across a wide range of robotic systems without requiring application-specific model simplifications, gait-generation heuristics, or extensive parameter tuning remain elusive.

Our approach combines fast, differentiable rigid-body dynamics with contact, strategic approximations about a reference trajectory, and specialized numerical optimization techniques for the application of local tracking control for systems that experience contact interactions with their environments. The result is a bi-level model predictive control algorithm that can effectively reason about contact changes in the presence of large disturbances while remaining fast enough for real-time execution.

We formulate contact dynamics as a complementarity problem and devise a fast interior-point solver to reliably optimize this feasibility problem. Smooth gradients are efficiently computed through the non-smooth dynamics by exploiting intermediate solutions from within this solver using implicit differentiation. To enable real-time performance for control, we pre-compute linearizations of the system's dynamics, signed-distance functions, and friction cones about a reference trajectory, while explicitly retaining complementarity constraints that encode contact switching behavior, resulting in a sequence of lower-level time-varying linear-complementarity problems (LCP) which represent the model's contact dynamics. An upper-level trajectory optimization problem is then optimized using fast linear algebra. We refer to this algorithm as *Contact-Implicit Model Predictive Control* (CI-MPC).

Finally, we demonstrate that CI-MPC can generate new contact sequences online and reliably track reference trajectories despite significant model mismatch and while large external disturbances are applied for a number of qualitatively different robotic systems, including: a pushbot, and planar hopper, quadruped, and biped systems in simulation; and on Unitree Go1 quadruped hardware [38].

Our contributions are:

- Fast approximate contact dynamics that can be reliably evaluated and efficiently differentiated with a custom interior-point solver
- Structure-exploiting solvers for the contact-dynamics and trajectory optimization problems

- A model predictive control framework for robotic systems with contact dynamics
- A collection of simulation and hardware experiments demonstrating the performance of CI-MPC on a variety of robotic systems across a range of highly dynamic tasks

In the remainder of this paper, we first review related work on control through contact with MPC, as well as complementarity-based contact dynamics in Section 8.2. Next, we present a brief overview of MPC, outline the classic complementarity formulation for contact dynamics, and provide background on interior-point methods and implicit differentiation in Section 8.3. Then, we present CI-MPC in Section 8.4. Results are presented in Section 8.5 including both simulation and hardware experiments. Finally, we discuss our results, limitations of this approach in Section 8.6 and potential directions for future work in Section 8.7.

8.2 Related Work

In this section, we review related work on MPC for the control of dynamical systems that make and break contact with their environments and provide an overview of complementarity-based contact dynamics.

8.2.1 Model Predictive Control

Today, most successful approaches for controlling legged robots utilize MPC in combination with simplified models and heuristics originally pioneered by Raibert for hopping robots [79]. The key insight of this work is that the control problem can be decoupled into a high-level controller that plans body motions while ignoring the details of the leg dynamics, and a low-level controller that determines the necessary leg motions and joint torques to generate the forces and torques on the body determined by the high-level controller.

Arguably the most impressive control work on humanoids has utilized centroidal dynamics with full kinematics to enable Atlas to navigate various scenarios with obstacles [159] and perform parkour [50]. Integrating hardware design and controller synthesis has also recently enabled small humanoids to perform agile acrobatic maneuvers in simulation [160].

There have also been impressive advances for quadrupeds, achieved by designing hardware that aims to closely match the modeling approximations made in the controller, e.g., building very light legs [158]. Whole-body control, which has the benefit of simpler overall control structures and the ability to leverage a system's dynamics, has been achieved at real-time rates on hardware [161]. Approaches that utilize both force-based MPC and whole-body control have also demonstrated agile locomotion [162].

A major limitation of these prior works is that the control policies are highly specialized to a specific robotic system. In this work, we compare CI-MPC to a number of system-specific control

methods that perform quite well for their given system, but do not generalize to other systems, whereas our policy generalizes to many different systems that experience contact interactions while achieving comparable or better performance.

8.2.2 Complementarity-Based Contact Dynamics

The classic approach for simulating rigid-body dynamics with contact interactions is a velocity-based time-stepping scheme formulated as a linear complementarity problem (LCP). The LCP searches for the next state of the system while enforcing impact and friction constraints. Solvers for this class of problems utilize pivoting methods [163], such as Lemke's algorithm [16], or interior-point methods [164]. Implementations of pivoting methods can be found in general-purpose LCP solvers such as PATH [44], or physics engines including: Bullet [74] and DART [71].

Derivatives of LCP-based contact dynamics can be efficiently computing using implicit differentiation [5]. However, the quality of these results is dependent on the method employed for optimization. Pivoting approaches enforce strict complementarity at each iteration, returning solutions at non-differentiable points. As a consequence, this differentiation will return subgradients that make, typically efficient, second-order optimization slower and less reliable. In contrast, interior-point methods relax the complementarity constraints at each iteration, only converging in the limit. These intermediate results can be implicitly differentiated to return smooth gradients [13]. Alternative approaches for computing gradients for contact dynamics include utilizing auto-differentiation tools [65] and analyzing the LCP solution to select subgradients [62].

In addition to simulation, contact dynamics represented as LCPs have been utilized for planning. Collocation approaches [102] directly encode the LCP problem as as constraints in order to enforce contact dynamics, along with an objective specifying desired behavior, in a large non-convex problem [30]. This approach enables the optimizer to plan without pre-specified mode sequences for locomotion and simple manipulation tasks. Subsequent work improved this approach by introducing higher-order integrators for the dynamics and a numerically robust, exact ℓ_1-penalty for handling the complementarity constraints [37]. Alternative rollout-based approaches utilize LCPs for forward simulation and subsequently differentiate through the solution of one-step dynamics in order to compute derivatives for gradient-based optimization [165, 62].

Another popular contact-dynamics formulation is MuJoCo's soft-contact model [40], which solves a convex optimization problem and trades physical realism for fast and reliable performance. An alternative model solved a strictly convex quadratic program [166]. Gradients are computed use a finite-difference scheme. However, this approach is computationally less efficient. Additionally, the LCP complementarity constraints can be relaxed, resulting in a soft-contact model that exhibits improved numerical properties in some scenarios [63].

8.3 Background

In this section, we provide technical background on MPC, complementarity-based contact dynamics, interior-point methods, and implicit differentiation.

8.3.1 Model Predictive Control

Predictive control policies [46] optimize a planning problem:

$$\begin{aligned} \underset{x_{1:T}, u_{1:T-1}}{\text{minimize}} \quad & g_T(x_T) + \sum_{t=1}^{T-1} g_t(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t), \quad t = 1, \ldots, T-1, \\ & (x_1 \text{ given}), \end{aligned} \tag{8.1}$$

for a given initial state in order to compute controls for a dynamical system we aim to control. If planning is performed at a sufficiently high rate, the sequence of open-loop plans provide *feedback*. For a system with state $x \in \mathbf{R}^n$, control $u \in \mathbf{R}^m$, time index t, initial state x_1, and discrete-time dynamics $f : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}^n$, the optimizer aims to minimize an objective with costs, $g : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$, over a planning horizon T.

Solving a (potentially) non-convex problem (8.1) online can be unreliable or computation too expensive. Instead, a proxy problem is solved online that makes strategic approximations. A common simplification is to track a reference trajectory, $\bar{\tau} = (\bar{x}_1, \bar{u}_1, \ldots, \bar{x}_T)$, denoted with an overbar ($\bar{}$), that is precomputed offline. In this setting, the computational complexity for planning is reduced by utilizing dynamics:

$$\delta x_{t+1} = A_t \delta x_t + B_t \delta u_t, \tag{8.2}$$

linearized about the reference, where $A = \partial f(\bar{x}, \bar{u})/\partial x$, $B = \partial f(\bar{x}, \bar{u})/\partial x$, and the decision variables are relative to the reference trajectory, i.e., $\delta a = a - \bar{a}$; and a quadratic objective:

$$\frac{1}{2} \delta x_t^T Q_t \delta x_t + q_t^T \delta x_t + \frac{1}{2} \delta u_t^T R_t \delta u_t + r_t^T \delta u_t, \tag{8.3}$$

where $Q = \partial^2 g(\bar{x}, \bar{u})/\partial x^2, q = \partial g(\bar{x}, \bar{u})/\partial x, R = \partial^2 g(\bar{x}, \bar{u})/\partial u^2, r = \partial g(\bar{x}, \bar{u})/\partial u$, similarly comprise a second-order expansion about the reference trajectory.

This formulation, potentially with additional affine state and control constraints, is commonly referred to as *linear* MPC. Without such constraints, the problem (8.1) with linear dynamics (8.2) and quadratic objective (8.3) is the LQR problem [6] and is efficiently solved with a backward Riccati recursion.

Predictive control iteratively optimizes (8.1), or an approximate version of it (8.2 - 8.3), for a given state, and the optimized controls are utilized to compute an input for the system. After the system evolves, the problem is re-optimized for a new state in order to compute a new control

for the system. By repeating this procedure at a high rate, feedback control is achieved [114]. In practice, applying the controls optimized with time-varying linearized dynamics and quadratic costs, to the actual nonlinear system is extremely effective, especially for applications that track a reference trajectory.

8.3.2 Complementarity-Based Contact Dynamics

Contact dynamics can be simulated with a velocity time-stepping scheme [8], formulated as complementarity problem [16]:

$$\begin{aligned}
\text{find} \quad & q, \lambda, \eta, \beta, \psi && (8.4)\\
\text{subject to} \quad & \left[M_+(q - q_-) - M_-(q_- - q_{--})\right]/h + hC = J^T\lambda + Bu, && (8.5)\\
& \gamma \circ \phi = 0, && (8.6)\\
& \beta \circ [P(q - q_-)/h + \psi\mathbf{1}] = 0, && (8.7)\\
& \psi \circ \left[\mu\gamma - \mathbf{1}^T\beta\right] = 0, && (8.8)\\
& \phi, \gamma \geq 0, && (8.9)\\
& \beta, \psi, [P(q - q_-)/h + \psi\mathbf{1}], [\mu\gamma - \mathbf{1}^T\beta] \geq 0, && (8.10)
\end{aligned}$$

that finds the next configuration of the system $q \in \mathbf{R}^{n_q}$ using implicitly defined velocities, $v = (q - q_-)/h \in \mathbf{R}^{n_v}$. Subscripts indicate a previous time step. This formulation considers a single contact point, but generalizes to systems with multiple contacts. The problem utilizes: the mass matrix $M : \mathbf{R}^{n_q} \to \mathbf{S}^{n_v}_{++}$; dynamics bias $C : \mathbf{R}^{n_q} \times \mathbf{R}^{n_v} \to \mathbf{R}^{n_v}$ that includes Coriolis and gravitational terms; contact Jacobian $J : \mathbf{R}^{n_q} \to \mathbf{R}^{d \times n_v}$ that maps contact forces in the contact frame into the generalized coordinates; input Jacobian $B : \mathbf{R}^{n_q} \to \mathbf{R}^{n_v \times m}$ that maps control inputs, typically joint torques, into the generalized coordinates; $P : \mathbf{R}^{n_q} \to \mathbf{R}^{p(d-1) \times n_v}$ is a mapping from the generalized velocity space to an overparameterized contact tangent space; time step $h \in \mathbf{R}_+$; contact forces $\lambda = (\gamma, \beta) \in \mathbf{R}^d$ defined in the contact frame, with normal force $\gamma \in \mathbf{R}$ and overparameterized friction forces $\beta \in \mathbf{R}^{p(d-1)}$ that are constrained by a linearized friction cone; $\psi \in \mathbf{R}$ is a dual variable associated with friction, representing the magnitude of the contact point velocity; signed-distance function, $\phi : \mathbf{R}^{n_q} \to \mathbf{R}$, that returns distance between a specified contact point on the robot (e.g., feet) and the closest surface in the environment (e.g., the floor); and where $\circ$ is an element-wise (Hadamard) vector product. We use p to denote the overparameterization dimension (often $p = 2$) and d to denote the environment dimension $d = 2$ for planar systems and $d = 3$ otherwise.

The smooth dynamics (8.5) are discretized with a semi-implicit Euler scheme [3]; complementarity constraints (8.6- 8.8) encode contact switching behavior; impact is encoded in (8.6) and (8.9); and friction terms (8.7-8.8) and (8.10), are derived from the maximum dissipation principle [42].

Problem data include previous configurations q_-, q_{--}, time step h and control inputs u. A

nonlinear formulation uses the following mappings:

$$\begin{aligned} &M_+ \leftarrow M(q), && M_- \leftarrow M(q_-), \\ &C \leftarrow C(q, (q - q_-)/h), && J \leftarrow J(q), \\ &B \leftarrow B(q), && P \leftarrow P(q), \\ &\phi \leftarrow \phi(q), \end{aligned} \tag{8.11}$$

evaluating these terms at the next configuration. In practice, a partial linearization of the dynamics is performed to satisfy a linear complementarity problem (LCP) [8], resulting in the following mappings:

$$\begin{aligned} &M_+ \leftarrow M(q_-), && M_- \leftarrow M(q_{--}), \\ &C \leftarrow C(q_-, (q_- - q_{--})/h), && J \leftarrow J(q_-), \\ &B \leftarrow B(q_-), && P \leftarrow P(q_-), \\ &\phi \leftarrow \phi(q_-) + N(q-)(q - q_-), \end{aligned} \tag{8.12}$$

where $N = d\phi/dq$.

8.3.3 Interior-Point Method

Classically, LCPs are solved using active-set methods [44], which strictly enforce complementarity at each iteration. An alternative approach is interior-point methods [164, 1], which relax these conditions during intermediate iterations, only satisfying these constraints in the limit.

LCPs can be generally formulated as:

$$\begin{aligned} \text{find} \quad & x, y, z \\ \text{subject to} \quad & Ex + Fy + f = 0, \\ & Gx + Hy + z + h = 0, \\ & y \circ z = 0, \\ & y, z \geq 0, \end{aligned} \tag{8.13}$$

with decision variables $x \in \mathbf{R}^n$, $y, z \in \mathbf{R}^m$ and problem data $\theta = (E, F, G, H, f, h) \in \mathbf{R}^{n \times n} \times \mathbf{R}^{n \times m} \times \mathbf{R}^{m \times n} \times \mathbf{R}^{m \times m} \times \mathbf{R}^n \times \mathbf{R}^m = \mathbf{R}^p$. Interior-point methods parameterize (8.13) by a central-path parameter $\kappa \in \mathbf{R}_+$ that relaxes the following bilinear constraint:

$$y \circ z = \kappa \mathbf{1}, \tag{8.14}$$

where $\mathbf{1}$ a vector of ones.

The equality and relaxed bilinear constraints form a residual vector or solution map, $r : \mathbf{R}^{n+2m} \times$

$\mathbf{R}^p \times \mathbf{R}_+ \to \mathbf{R}^{n+2m}$, that takes $w = (x, y, z) \in \mathbf{R}^{n+2m}$, the problem data, and central-path parameter as inputs. The problem data and central-path parameter are fixed during optimization. In the context of contact dynamics, these data encode the mechanical properties of the robots, its current configuration and velocity, and properties of the environment like friction coefficients. Newton or quasi-Newton methods are used to find search directions that reduce the norm of the residual and a backtracking line search is employed to ensure that the inequality constraints are strictly satisfied for candidate points at each iteration. Once the residual is optimized to a desired tolerance, the central-path parameter is decreased and the new subproblem is warm-started with the current solution and then optimized. This procedure is repeated in order to find solutions to (8.13) with $\kappa \to 0$ until the central-path parameter, also referred to as complementary slackness, is below a desired tolerance.

Importantly, our interior-point method utilizes a predictor-corrector algorithm [24] that leads to significantly improved convergence. First, the corrector step modifies the pure Newton search direction and typically reduces the number of iterations required for convergence by half (compared to the pure search direction). Second, the central-path parameter is adapted at each iteration to prevent premature numerical ill-conditioning. In practice, we find that this approach is significantly more reliable and has improved convergence behavior compared to prior work that employed relaxed complementarity conditions [37].

In addition to simulating contact by solving a feasibility problem, we would like to compute gradients of these dynamics, requiring us to differentiate through an optimization problem. This is accomplished with implicit differentiation [5].

For the interior-point method (8.13), the residual is:

$$r(w; \theta) = \begin{bmatrix} Ex + Fy + f \\ Gx + Hy + z + h \\ y \circ z - \kappa \mathbf{1} \end{bmatrix}. \tag{8.15}$$

A differentiable interior-point method is summarized in Algorithm 6. Importantly, we can compute gradients for intermediate results, corresponding to non-zero values for the central-path parameter, i.e., $\kappa_{\text{grad}} \neq 0$. For additional details about differentiating through intermediate results of contact dynamics that are solved with interior-point methods, see [13].

Algorithm 6 Differentiable Interior-Point Method

procedure OPTIMIZE(x, θ)
  Settings: $\beta = 0.5, \gamma = 0.1, \epsilon_\kappa = 10^{-6}, \epsilon_r = 10^{-8}$
  Initialize: $y, z = \mathbf{1}, \kappa = 0.1, \kappa_{\text{grad}} = 10^{-4}$
  Until $\kappa < \epsilon_\kappa$ **do**
    $\Delta w = (\frac{\partial r}{\partial w})^{-1} r(w; \theta, \kappa)$
    $\alpha \leftarrow 1$
    Until $(y, z) - \alpha(\Delta y, \Delta z) > 0$ **do** $\alpha \leftarrow \beta\alpha$
    Until $\|r(w - \alpha\Delta w; \theta, \kappa)\| < \|r(w; \theta, \kappa)\|$ **do**
      $\alpha \leftarrow \beta\alpha$
    $w \leftarrow w - \alpha\Delta w$
    If $\|r(w; \theta, \kappa)\| < \epsilon_r$ **do** $\kappa \leftarrow \gamma\kappa$
  $\frac{\partial w}{\partial \theta} \leftarrow$ **Differentiate**$(w, \theta, \kappa_{\text{grad}})$ ▷ Eq. 2.55
  Return w, $\frac{\partial w}{\partial \theta}$

8.4 Contact-Implicit Model Predictive Control

In this section we present Contact-Implicit Model Predictive Control, a tracking policy for systems that make and break contact with their environments. First, we formulate time-varying LCP contact dynamics that are selectively approximated about a reference trajectory. Then, we devise a fast solver for the resulting LCP. Next, we discuss how to compute smooth gradients through the dynamics. A bi-level planning formulation, which utilizes these dynamics for direct trajectory optimization [167], follows. To enable the policy to work well in environments with uncertain terrain we propose a contact-height heuristic. Finally, we summarize the approach and provide an algorithm for CI-MPC.

8.4.1 Time-Varying Contact Dynamics

We formulate alternative LCP dynamics that utilize a reference trajectory, resulting in the following mappings:

$$\begin{aligned} M_+ &\leftarrow M(\bar{q}), & M_- &\leftarrow M(\bar{q}_-), \\ C &\leftarrow C(\bar{q}, (\bar{q} - \bar{q}_-)/h), & J &\leftarrow J(\bar{q}), \\ B &\leftarrow B(\bar{q}), & P &\leftarrow P(\bar{q}), \\ \phi &\leftarrow \phi(\bar{q}) + N(\bar{q})(q - \bar{q}). \end{aligned} \tag{8.16}$$

The LCP problem, formulated for an interior-point method, has the form:

$$\begin{aligned} \text{find} \quad & w \\ \text{subject to} \quad & C(w - \bar{w}) + D(\theta - \bar{\theta}) = 0 \\ & \gamma \circ s_\phi = \kappa \mathbf{1}, \\ & \psi \circ s_\psi = \kappa \mathbf{1}, \\ & \beta \circ \eta = \kappa \mathbf{1}, \\ & \gamma, \psi, \beta, \eta, s_\phi, s_\psi \geq 0, \end{aligned} \tag{8.17}$$

with decision variables $w = (q, \gamma, \beta, \psi, \eta, \beta, s_\phi, s_\psi)$, where slack variables, $s_\phi, s_\psi \in \mathbf{R}$, are introduced for convenience.

Importantly, C and D are matrices that define a linear system of equations resulting from approximations about the reference trajectory and they are pre-computed offline. These contact dynamics:

$$q_{t+1} = \mathbf{LCP}_t(q_{t-1}, q_t, u_t), \tag{8.18}$$

$\mathbf{LCP}_t : \mathbf{R}^{n_q} \times \mathbf{R}^{n_q} \times \mathbf{R}^m \to \mathbf{R}^{n_q}$, solve (8.17) and return the configuration at the next time step. The contact forces at the current time step can also be returned.

8.4.2 Fast Contact Dynamics

The most expensive procedure in evaluating the LCP and computing gradients of a solution is solving the linear system of equations:

$$R_w \Delta w = r, \tag{8.19}$$

required by the interior-point method, where $R_w = \partial r / \partial w$, and Δw is the new search direction.

To reduce the computational cost of this routine, we exploit both the sparsity pattern and the property that most of R_w remains constant across iterations and, therefore, can be pre-factorized offline [168].

We partition the LCP variables (8.13) as follows: $x = q$, $y = (\gamma, \psi, \beta)$, and $z = (\eta, s_\phi, s_\psi)$, and similarly split the residual: $r = (r_x, r_y, r_z)$. The Jacobian's sparsity pattern is:

$$R_w = \begin{bmatrix} E & F & 0 \\ G & H & I \\ 0 & \mathbf{diag}(z) & \mathbf{diag}(y) \end{bmatrix}, \tag{8.20}$$

where I denotes the identity matrix. By exploiting sparsity in the third row of (8.20), we can form

the following condensed system:

$$\begin{bmatrix} E & F \\ G & \tilde{H} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_x \\ \tilde{r}_y \end{bmatrix} \Leftrightarrow \tilde{R}_w \, \Delta\tilde{w} = \tilde{r}, \tag{8.21}$$

where:

$$\tilde{H} = H - \mathbf{diag}(y^{-1} \circ z), \tag{8.22}$$

$$\tilde{r}_y = r_y - y^{-1} \circ r_z, \tag{8.23}$$

$$\Delta z = y^{-1} \circ (r_z - z \circ \Delta y), \tag{8.24}$$

and y^{-1} denotes the element-wise reciprocal of vector y. This term is always well-defined because a line search enforces $y > 0$ at each iteration.

To solve for $\Delta\tilde{w}$, we leverage the fact that, apart from the bottom-right block, $\tilde{R}_w$ can be computed offline. We perform a QR decomposition on the Schur complement of (8.21):

$$Q, R \leftarrow \mathbf{QR}(\tilde{H} - GE^{-1}F), \tag{8.25}$$

and then solve for the search directions:

$$\Delta y = -R^{-1}Q^T(GE^{-1}r_x - \tilde{r}_y), \tag{8.26}$$

$$\Delta x = E^{-1}(r_x - F\Delta y). \tag{8.27}$$

Additionally, E^{-1}, GE^{-1}, and $GE^{-1}F$ are precomputed offline. Finally, after solving for $\Delta\tilde{w}$, we obtain Δz with cheap vector-vector operations (8.24).

For a system with configuration dimension n_q and c contact points, the computational complexity of solving (8.19) with a naive approach is $O\Big((n_q + 2cd)^3\Big)$. Our structure-exploiting approach is $O(8c^3d^3)$ during the online phase. In practice, this provides a factor of 15 speed-up, compared to LAPACK LU, for evaluating the LCP dynamics across all the robotic systems presented in this paper and, in turn, results in a factor of 2.5 speed-up for CI-MPC.

8.4.3 Gradients

Contact dynamics gradients are computed by differentiating through the LCP problem with respect to data: $\theta = (q_{--}, q_-, u)$, which could also include the time step, friction coefficients, and other system values like masses or inertia terms.

At fixed points, w^*_κ, parameterized by a (potentially non-zero) central-path value, gradients are computed using implicit-differentiation. Importantly, at solution where $\kappa \approx 0$, this approach will return subgradients, which often fail to provide useful information through contact events.

However, we exploit intermediate results from the interior-point solver, where $\kappa \neq 0$, in order to compute smooth gradients. Large values of κ will produce smoother gradients than those computed with small values of κ, which more closely approximates a true subgradient at nondifferentiable points. Practically, we find that smooth gradients, computed using intermediate results, provides information through contact events.

8.4.4 Planning

The CI-MPC policy is formulated as:

$$u = \pi(x) \begin{cases} \underset{x_{1:H}, u_{1:H}}{\text{minimize}} & \sum_{t=1}^{H} \frac{1}{2}(x_t - \bar{x}_t)^T Q_t (x_t - \bar{x}_t) + \frac{1}{2}(u_t - \bar{u}_t)^T R_t (u_t - \bar{u}_t) \\ \text{subject to} & x_{t+1} = \mathbf{LCP}_t(x_t, u_t), \quad t = 1, \ldots, H-1, \\ & x_1 = x, \end{cases} \tag{8.28}$$

comprising an upper-level planning problem (8.1) that optimizes a trajectory: $\tau = (x_1, u_1, \ldots, x_H, u_H) \in \mathbf{R}^{(2n_q+m)H}$ of configurations and controls over a horizon H using a state representation: $x_t = (q_{t-1}^{(t)}, q_t^{(t)})$, with two configurations. This problem is solved using a direct trajectory optimization approach. Lower-level LCP problems (8.4) enforce the dynamics with the first state x_1 fixed. For convenience, we overload notation for the LCP dynamics:

$$\mathbf{LCP}_t(x_t, u_t) = \begin{bmatrix} q_t^{(t)} \\ \mathbf{LCP}_t(q_{t-1}^{(t)}, q_t^{(t)}, u_t) \end{bmatrix}, \tag{8.29}$$

for state-based LCP dynamics, and define constraints, $k_t(x_t, u_t, x_{t+1}) = x_{t+1} - \mathbf{LCP}_t(x_t, u_t)$, that couple states across adjacent time steps. The constraint Jacobian:

$$\nabla k = \begin{bmatrix} -B_1 & I & 0 & 0 & 0 & 0 & \\ 0 & -A_2 & -B_2 & I & 0 & 0 & \\ 0 & 0 & 0 & -A_3 & -B_3 & I & \\ & & & & & & \ddots \end{bmatrix}, \tag{8.30}$$

where $k = (k_1, \cdots, k_{H-1}) \in \mathbf{R}^{2n_q(H-1)}$, is comprised of one-step dynamics Jacobians:

$$A_t = \begin{bmatrix} 0 & I \\ \frac{\partial \mathbf{LCP}_t}{\partial q_{t-1}} & \frac{\partial \mathbf{LCP}_t}{\partial q_t} \end{bmatrix}, \; B_t = \begin{bmatrix} 0 \\ \frac{\partial \mathbf{LCP}_t}{\partial u_t} \end{bmatrix}. \tag{8.31}$$

The planning objective is a convex quadratic function (8.3) and velocities are penalized using finite-difference approximations. Because the problem is lifted by using states comprising two configurations, these costs do not introduce coupling across more than one time step. The resulting

Hessian of the objective:

$$\nabla^2 J = \begin{bmatrix} R_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & R_2 & 0 \\ 0 & 0 & 0 & \ddots \end{bmatrix}, \tag{8.32}$$

and its inverse:

$$(\nabla^2 J)^{-1} = \begin{bmatrix} R_1^{-1} & 0 & 0 & 0 \\ 0 & Q_2^{-1} & 0 & 0 \\ 0 & 0 & R_2^{-1} & 0 \\ 0 & 0 & 0 & \ddots \end{bmatrix}, \tag{8.33}$$

are block diagonal and are pre-computed offline.

The resulting KKT system:

$$\begin{bmatrix} \nabla^2 J & \nabla k^T \\ \nabla k & 0 \end{bmatrix} \begin{bmatrix} \Delta\tau \\ \Delta\nu \end{bmatrix} = \begin{bmatrix} \nabla J + \nabla k^T \nu \\ k \end{bmatrix}, \tag{8.34}$$

with dual variables $\nu \in \mathbf{R}^{2n_q(H-1)}$ associated with the constraints, uses a Gauss-Newton approximation of the constraints when computing the Hessian of the Lagrangian and is solved using a sparse LDL^T solver. In the following experiments we utilize QDLDL, a general-purpose sparse solver, for its efficient implementation [123].

8.4.5 Contact-Height Heuristic

To enable the policy to robustly adapt to unknown variations in terrain height, we employ a simple heuristic that we find to be effective in practice. The policy maintains a height estimate, $a \in \mathbf{R}^c$, for each contact and utilizes a modified signed-distance function:

$$\phi_{MPC}(q) = \phi(q) + a, \tag{8.35}$$

that is updated using the current contact height. When contact is detected, the height estimate is updated. In simulation, a threshold on the impact-force magnitude is set; and in practice, force sensors can reliably detect such an event.

This simple heuristic does not affect the structure of (8.20) and only requires c more addition operations to compute r when evaluating the fast contact dynamics (8.19). In our experiments, we find the heuristic to be effective and reliable across unknown terrain for the systems tested.

8.4.6 Algorithm

CI-MPC comprises offline and online stages. The offline stage generates a reference trajectory along with a set of time-varying LCP problems. In this work we employ contact-implicit trajectory optimization [37] to design these references. Additionally, a planning horizon, typically less than the total behavior duration, is specified. During the online stage, planning is performed (8.1) for the current state over the specified horizon, and the optimized control trajectory is used to compute an control that is applied to the system. The CI-MPC policy is summarized in Algorithm 7.

Algorithm 7 Contact-Implicit Model Predictive Control

procedure POLICY
  Offline
    $\bar{\tau} \leftarrow$ generate reference trajectory
    $\mathbf{LCP}_t \leftarrow$ generate fast contact dynamics
    $H \leftarrow$ set planning horizon,
  Online
    For $i = 1, \ldots, \infty$
      $u \leftarrow \pi(x)$ ▷ Eq. (8.28)
      $x \leftarrow \mathbf{dynamics}(x, u)$
    End
End

8.4.7 Heuristics

To enable real-time performance for the policy (8.28), a number of heuristics are employed. First, the planning problem is only solved approximately. Instead of optimizing until convergence, a fix number of iterations are performed and then the current best solution is returned. This enables the current plan to be improved but significantly reduces the total computation required. Generally, we find that performance is greatly improved by returning approximate solutions quickly, enabling replanning with newer state information, compared to returning higher quality solutions to planning problems that are utilizing older state information. Second, the policy extensively utilizes warm starting. Providing the optimizer with a good initial guess for the solution greatly reduces the number of iterations required to converge. Initially, the policy utilizes the reference trajectory. At subsequent evaluations, the previous best solution is used. Third, the LCP problems are solved for a single value of the central path parameter instead of a sequence that converges to zero. In practice, we find that $\kappa \approx$ 1e-4 is a good balance between computation time, physical accuracy, and gradient smoothness. Note, when verifying the performance of the policy in simulation, we solve the nonlinear contact dynamics complementarity problem (8.4) to convergence, i.e., $\kappa =$ 1e-6.

8.5 Results

We demonstrate the CI-MPC algorithm in simulation and on hardware by controlling a variety of robotic systems that make and break contact with their environments. In the examples we show that the policy can generate new contact sequences online; is robust to disturbances, model mismatch, and unknown terrain; and is faster than real-time, see Table 8.1.

Table 8.1: The CI-MPC policy runs at real-time rates meaning that the time required to compute an updated control is always smaller than the reference time step (i.e., budget) and the policy is able to successfully track the specified trajectory. Experiments are run on a computer equipped with an Intel Core i9-9900 CPU and 32GB of memory.

System	**Planning Horizon**	**Time Step**	**Real-Time**
pushbot	1.60 s	0.04 s	✓
hopper	0.10 s	0.01 s	✓
quadruped	0.16 s	0.016 s	✓
biped	0.23 s	0.016 s	✓

The code, including a Julia implementation of the policy and all of the experiments, is available at:

`https://github.com/dojo-sim/ContactImplicitMPC.jl`.

8.5.1 Simulation

We verify the policy performance in simulation where we solve the nonlinear contact dynamics complementarity problem (8.4) to convergence. Additionally, all examples are simulated using a different sample rate, typically 5-10× faster than the reference trajectory, in order to ensure that the policy is robust to sampling rates. For example, if the reference is optimized with a time step of 0.1 seconds, then we simulate the nonlinear dynamics with a smaller time step, 0.01 or 0.02 seconds.

Pushbot. In this example, we demonstrate that our policy can generate qualitatively new, unspecified contact sequences online in order to respond to unplanned disturbances. The system, PushBot, is modeled as an inverted pendulum with a prismatic joint located at the end of the pendulum (Fig. 8.1). There are two control inputs: a torque at the revolute joint and a force at the prismatic joint. The system is located between two walls and has two contact points, one between the prismatic-joint end effector and each wall.

PushBot is tasked with remaining vertical and the policy utilizes a reference trajectory that does not include any contacts. When we apply a large impulse to the system, the policy generates a behavior that commands the prismatic joint to push against the wall in order to stabilize. By

tuning the policy's cost function we can generate different behaviors, including maintaining contact to stabilize and pushing against the wall in order to return to the nominal position. The latter behavior is shown in Fig. 8.1.

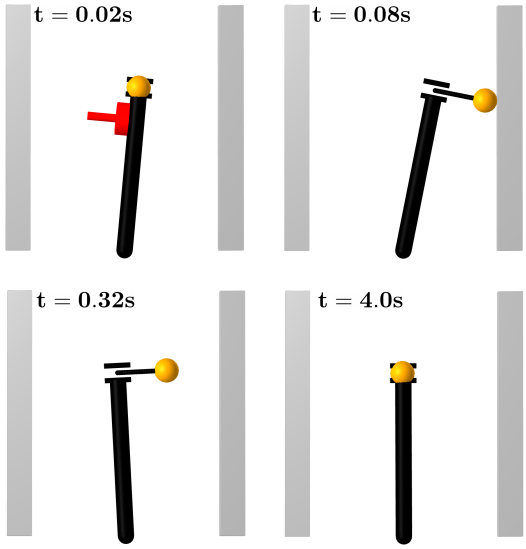


Figure 8.1: Pushbot performing push recovery. A disturbance (red) creates an impulse on the system and the policy generates a new contact sequence that extends the prismatic joint toward the right wall in order to make contact. After stabilizing, pushbot pushes against the wall, eventually breaking contact, in order to return to the nominal upright configuration.

We compare CI-MPC to a method that relies on a mixed-integer quadratic progam (MIQP) formulation [169] applied to a simplified version of the PushBot (an inverted pendulum between two stiff walls). The MIQP minimizes a quadratic objective function subject to piecewise-linearized dynamics. Each linear dynamics domain corresponds to a single contact mode, and discrete decision variables are introduced to encode contact mode switches. Our CI-MPC approach is fast enough to be run online, however, this is not the case for the MIQP policy, as shown in Table 8.2. Moreover, the complexity of the MIQP increases exponentially with the number of contact modes, making it an intractable approach for more complex systems. Warm-starting the MIQP [170] may make this approach more amenable to online optimization.

Table 8.2: Comparison between CI-MPC and MIQP policies for pushbot example. For a fixed replanning rate of 25 Hz, we report the mean and standard deviations for the optimization times and compare this to the associated time budget (0.04 s). Both policies successfully regulate the system around the equilibrium point. However, the MIQP policy is slower than real-time, whereas the CI-MPC policy always remains within time budget, ensuring real-time performance.

Policy	**Planning Time**	**Real-Time**
CI-MPC	$\mathbf{0.014 \pm 0.027}$**s**	✓
MIQP	0.18 ± 0.09 s	×

Hopper. Inspired by the Raibert Hopper [79], we model a 2D hopping robot with $n_q = 4$ generalized coordinates: lateral and vertical positions, body orientation, and leg length, respectively; $m = 2$ controls: body moment, e.g., controlled with an internal reaction wheel, and leg force; and a single contact at the foot.

The centroidal-dynamics modeling assumption we make—consistent with Raibert's work—is to locate the leg and foot mass at the body's center of mass. This results in a configuration-independent mass matrix and no bias term in the dynamics.

The hopper is tasked with locomoting over unknown terrain. The CI-MPC policy uses a reference trajectory that is optimized with a flat surface and no incline. We compare our policy to the Raibert heuristic, which we similarly tune for flat ground and no incline. We observe that our policy is able to adapt to the varying surface heights that range from 0-24 cm and that the robot can slip multiple times and is able to recover while traversing steep inclines. We find that, when tuned well, the Raibert heuristic also works very well on terrains

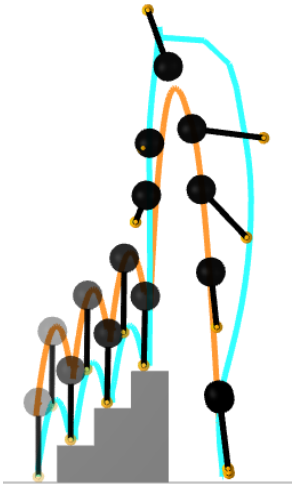


Figure 8.2: Hopper performing parkour. The system tracks the body (orange) and foot (blue) reference trajectories while ascending three stairs before performing a front flip.

Additionally, we task the hopper with climbing a staircase and executing a front flip (Fig. 8.2). This complex trajectory cannot be directly executed using the Raibert heuristic as it is not a periodic hopping gait. Our policy, however, successfully tracks this complex trajectory, illustrating the more general capabilities of CI-MPC. Results are summarized in Table 8.3.

Table 8.3: Comparison between CI-MPC and the Raibert heuristic for a hopper system on 4 scenarios: flat, sinusoidal, and piecewise linear terrains; and a parkour stunt (Fig. 8.2). For each terrain profile, we report the number of hops achieved by the policy. For the parkour scenario, we report if the stunt is successfully completed.

Policy	**Flat**	**Sinusoidal**	**Piecewise**	**Parkour**
CI-MPC	+100	+100	+100	✓
Raibert	+100	+100	+100	×

Planar quadruped. We model a planar quadruped with $n_q = 11$ configuration variables and $m = 8$ control inputs. The system has four contacts, one at each point foot.

The quadruped is tasked with moving to the right over three different terrains: flat, sinusoidal, and piecewise-linear surfaces (Fig. 8.3).

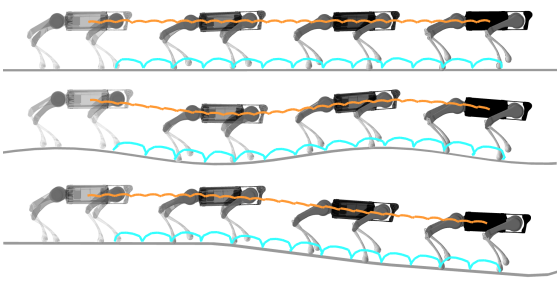


Figure 8.3: Planar quadruped walking over uneven terrain. The reference gait is optimized for flat ground. Our CI-MPC policy, with orange center-of-mass and blue foot position trajectories, is able to adapt online to the unmodeled variation in terrain and track the reference trajectory.

Additionally, we test the robustness of the CI-MPC policy by introducing model mismatch. We provide the policy with the nominal model of the quadruped while the simulator uses a quadruped with a 3-kg payload, representing 25% of its nominal mass.

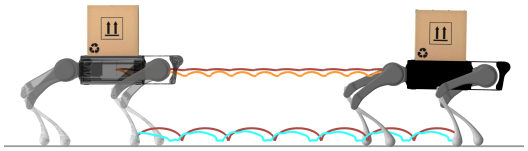


Figure 8.4: Quadruped tracking a reference trajectory (red) while carrying an unmodeled 3-kg payload. We depict the torso (orange) and front-left foot (blue) trajectories.

Despite the unmodeled load, the policy successfully tracks the nominal gait with good performance.

We note that the same CI-MPC policy was used across all quadruped experiments and no retuning was required to transfer from the nominal case (flat terrain, no payload) to more complex scenarios. Further, it is easy and intuitive to rapidly retune the policy in order to achieve improved tracking performance in the other scenarios.

Planar biped. We model a planar biped based on Pratt's Spring Flamingo [171] with $n_q = 9$ configuration variables and $m = 7$ control inputs. The system is modeled with four contact points, one at the toe and heel of each foot.

The biped is tasked with moving to the right over three different terrains: flat, sinusoidal, and piecewise-linear surfaces (Fig. 8.5) using the same policy.

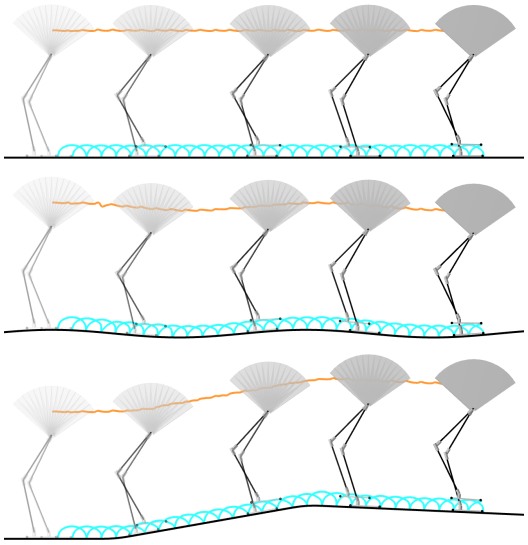


Figure 8.5: Biped walking from left to right across flat (top), sinusoidal (middle), and piecewise linear (bottom) terrain using the same policy.

In Table 8.4, we compare this to Pratt's policy [171], which relies on a state-machine architecture and a number of proportional-derivative controllers.

Table 8.4: Comparison between CI-MPC and Pratt state-machine [171] policies for flamingo system on flat and inclined terrains. We report the number of steps taken by the robot on the flat terrain and compare the maximum incline traversed by our policy in simulation with reported results† [172].

Policy	**Flat**	**Incline**
CI-MPC	+100	**10 deg.**
Pratt	+100†	5† deg.

Our CI-MPC policy—with no additional tuning—can easily walk on all of the terrains and reliably walks up inclines of up to ten degrees. Pratt reports that Spring Flamingo can only walk up inclines of five degrees without requiring the controllers to be re-tuned [172].

Monte Carlo. In order to assess the robustness of CI-MPC, we perform Monte Carlo analysis on two systems: the hopper and planar quadruped. The robots are tasked with tracking a reference gait and we initialize the systems with configurations that are randomly perturbed from the reference trajectory. We use 100 randomly sampled initial conditions for each system; the hopper recovers from significant orientation offsets and the quadruped is robust to large drops (Fig. 8.6).

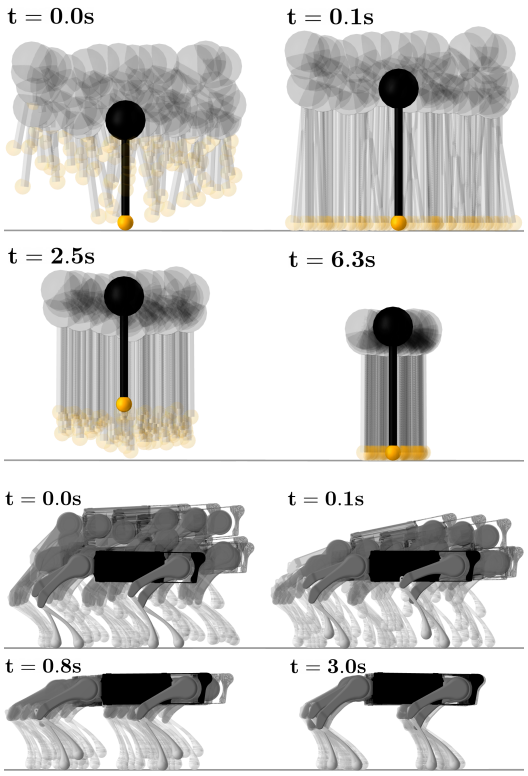


Figure 8.6: Monte Carlo simulations of initial conditions for systems tracking a reference trajectory. 100 initial configurations are randomly sampled for a hopper (top) and quadruped (bottom). Perturbations from the reference initial configuration include large translations, tilts, and joint angles offsets. For all the samples, and both systems, the policy successfully recovers to the reference gait.

8.5.2 Hardware

In order to assess the robustness of CI-MPC, we perform Monte Carlo analysis on two systems: the hopper and planar quadruped. The robots are tasked with tracking a reference gait and we initialize the systems with configurations that are randomly perturbed from the reference trajectory. We use 100 randomly sampled initial conditions for each system; the hopper recovers from significant orientation offsets and the quadruped is robust to large drops (Fig. 8.6).

Point-foot model. We utilize a simplified point-foot model of the quadruped that neglects leg dynamics. This model comprises 36 states which include the positions and velocities of the body and each foot. The orientation of the body is represented with Euler angles. The controls are three-dimensional forces applied to each foot, which is modeled as a point mass. Reference motions are generated offline with this model using contact-implicit trajectory optimization [37].

The primary reason for these modeling simplifications is to reduce the online computational

requirement when evaluating the dynamics and their derivatives while still being able to reason about new contact sequences online. Importantly, unlike traditional convex quadratic programming policies [173], which assume a fixed contact sequence for the feet, our model is contact-implicit and enables new foot-step sequences to be generated online.

Experimental setup. A reference trajectory generated offline is tracked online using CI-MPC. A pre-tuned low-level controller generates joint torques that aim to match the forces specified by the policy that should be applied by each leg at the foot. The CI-MPC policy is written in Julia and is precompiled in order to interface with an existing C++ low-level controller running at 1000Hz. State feedback to the CI-MPC policy and the low-level controller is provided by a 1000Hz Kalman filter which utilizes joint encoders, onboard IMU, and external motion-capture tracking to estimate the robot state. The policy, state estimator, and low-level controller run on a computer equipped with an Intel i9-12900KS CPU and 64GB of memory. The joint torque commands are sent to the quadruped via an ethernet connection. Additional information is provided in Table 8.5.

Table 8.5: Hardware experiments with CI-MPC tracking different reference trajectories on a Unitree Go1 quadruped.

Settings	**Trotting**	**Wall**	**Step**
reference trajectory length	0.8 s	19.75 s	7.0 s
reference time step	0.05 s	0.05 s	0.05 s
policy planning horizon	0.15 s	0.1 s	0.1 s
policy rate	100 Hz	100 Hz	100 Hz

Trotting. In this example, the specified behavior is trotting in place (Fig. 8.7).

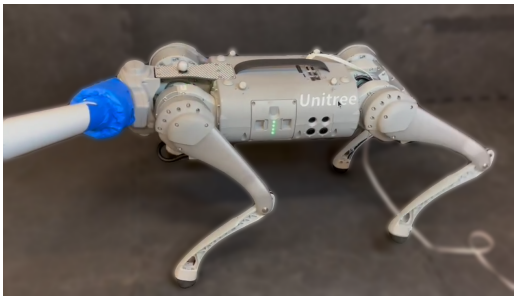


Figure 8.7: Unitree Go1 stable trotting while being pushed.

The reference trajectory is 0.8 seconds and is repeated to form a continuous gait. The policy planning horizon is 0.10 seconds and the model uses a time discretization of 0.05 seconds. The policy runs at an average rate 100 Hz.

A footstep sequence is shown in Fig. 8.8 for a scenario where a large disturbance is applied to the system during trotting.

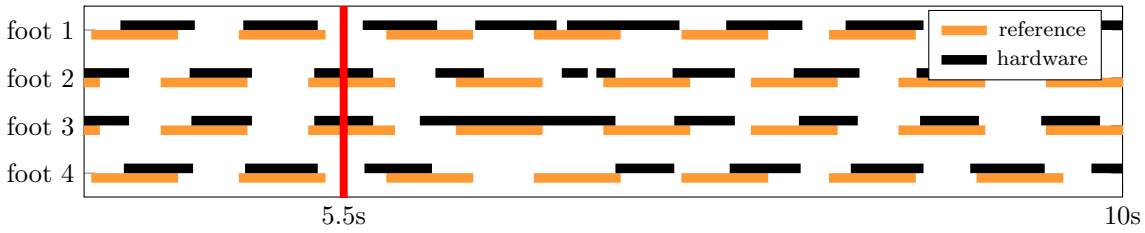


Figure 8.8: Contact sequence for quadruped trotting. An external disturbance is applied to the system (red) and the policy is able to generate a new contact sequence online (black) that differs significantly from the reference plan (orange).

Wall stand. In this example, the specified behavior is transitioning from four feet on the ground to standing against the wall with two feet (Fig. 8.9).

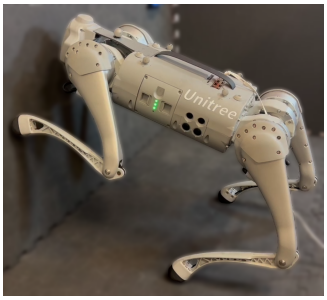


Figure 8.9: Unitree Go1 transitioning from ground to standing against a wall.

The reference trajectory is 19.75 seconds. The policy planning horizon is 0.15 seconds and the model uses a time discretization of 0.05 seconds. The policy runs at an average rate of 100 Hz.

Step. In this example, the specified behavior has the quadruped place its right foot onto a step, followed by its left foot (Fig. 8.10).

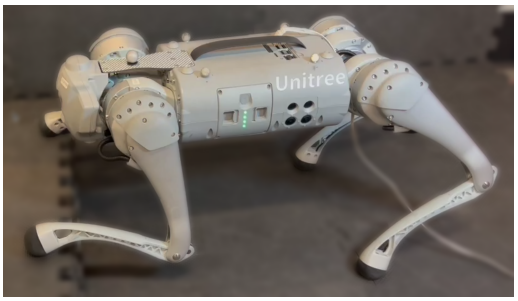


Figure 8.10: Unitree Go1 placing two feet onto a step.

The entire reference trajectory is 7.0. The policy planning horizon is 0.1 seconds and the model uses a time discretization of 0.05 seconds. The policy runs at an average rate of 100 Hz.

8.6 Limitations

We highlight important limitations including: approximations, contact model, and reliability, that should be considered before deploying CI-MPC policies.

Approximations. For the online trajectory optimization problem, strategically approximated contact dynamics are utilized for planning. This enables expensive gradient computations and partial matrix factorizations to be performed in an offline stage, in order to substantially reduce online computation. In the examples, we find that short planning horizons, typically between 0.1 and 0.25 seconds, are sufficient. Despite the approximations introduced by these simplifications, in practice, the controls optimized for the fast contact dynamics work well in simulation and on hardware. However, it remains to be seen how well these approximations work for control of highly dynamic behaviours or scenarios that require longer planning horizons, particularly when deployed on hardware.

Contact model. The physics of hard contact produces non-smooth and discontinuous gradients. With our custom interior-point method for the contact dynamics solver, we can efficiently compute smooth gradients in a principled way by exploiting intermediate results, parameterized by the central-path parameter. Hard contact is simulated by returning results from the contact dynamics solver with a central-path value $\kappa_{\text{sim}} = 1\text{e-}6$, whereas gradients are computed using intermediate results from the solve parameterized by $\kappa_{\text{grad}} \approx 1\text{e-}4$.

During online optimization, we prioritize fast updates by solving the trajectory-tracking problem to coarse tolerances. In this context, imposing highly accurate contact physics would be wasteful in terms of computational resources. As a result, the central-path value for the planning dynamics is fixed to the gradient central-path value in order to reduce online computation. This selection was

made to balance capturing accurate physics with producing usefully smooth gradients. Empirically, we observe that using these dynamics with slightly soften contact dynamics enhanced the convergence of the trajectory-tracking solver and likely enables the policy to more easily discover new contact sequences. Importantly, the allowed interpenetration with these tolerances is sub-millimeter—much less than allowed by MuJoCo's default settings—but, this raises the question, is simulation of perfectly hard contact actually necessary, or useful, for reliable planning and control of non-smooth systems?

Reliability. Generating high-quality reference trajectories is crucial for CI-MPC. Contact-implicit trajectory optimization [30, 37] is a powerful tool for generating these trajectories. However, it is notorious for poor convergence properties, despite relying on robust large-scale constrained solvers for non-convex problems and even good warm starting. This unreliability makes online optimization generally impractical—motivating this work. Ultimately, for CI-MPC to be of practical value, generation of references trajectories, even in the offline setting, must be improved. This may be possible with specialized solvers for contact-implicit trajectory optimization [25], alternative rollout-based methods that leverage the reliability of one-step contact dynamics [22], or learning-based approaches [153].

Additionally, our hardware experiments demonstrate the real-time capabilities of CI-MPC and its ability to be robust in many scenarios. However, in practice, we find that the classic convex MPC policy [174] is significantly more robust for general locomotion because that policy has an additional online foothold selection strategy (the Raibert strategy). Adding this strategy to CI-MPC will likely increase its speed and reliability to match the performance of the baseline convex MPC policy. Performance will likely be improved further with a more efficient C/C++ implementation that utilizes multi-threading for parallel evaluations of the fast contact dynamics.

8.7 Future Work

In summary, we have presented fast differentiable contact dynamics that can be utilized in an MPC framework that performs robust tracking for robotic systems that make and break contact with their environments. There remain many exciting avenues to explore in future work. First, it should be possible to perform higher-fidelity convex approximations of the contact dynamics that utilize second-order friction cones instead of its linearized approximation. This could enable tracking of highly dynamic behaviors that leverage accurate sliding contacts. Second, a natural extension of this work, which was focused on locomotion, is to the manipulation domain, potentially with quasi-static models, where control through contact is similarly an open problem, but where dynamics are slower and more amenable to online optimization. Third, in our hardware experiments, we utilized a simplified planning model for the quadruped. Similar point-contact models should extend

to bipeds and humanoid systems, potentially even dexterous hands. Lastly, a library of template behaviors, comprising CI-MPC policies, could be composed to enable more diverse behavior online with a high-level agent composing templates in a task-and-motion-planning framework in order to generate complex long-horizon plans.

Acknowledgments

This work was supported in part by Frontier Robotics, Innovative Research Excellence, Honda R&D Co., Ltd, ONR award N00014-18-1-2830, NSF NRI award 1830402, and DARPA YFA award D18AP00064. Toyota Research Institute (TRI) provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

Chapter 9

Predictive Sampling

We introduce MuJoCo MPC (MJPC), an open-source, interactive application and software framework for real-time predictive control, based on MuJoCo physics. MJPC allows the user to easily author and solve complex robotics tasks, and currently supports three shooting-based planners: derivative-based iLQG and Gradient Descent, and a simple derivative-free method we call Predictive Sampling. Predictive Sampling was designed as an elementary baseline, mostly for its pedagogical value, but turned out to be surprisingly competitive with the more established algorithms. This work does not present algorithmic advances, and instead, prioritizes performant algorithms, simple code, and accessibility of model-based methods via intuitive and interactive software.

Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo. Taylor A. Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. arXiv 2212.00541. 2022.

9.1 Introduction

Model-based approaches form the foundation of classical control and robotics. Since Kalman's seminal work [175], the *state* along with its dynamics and observation models has played a central role.

The classical approach is being challenged by *learning-based* methods, which forgo the explicit description of the state and associated models, letting internal representations emerge from the learning process [176, 177, 178, 179, 180]. The flexibility afforded by learned representations makes these methods powerful and general, but the requirement for large amounts of data and computation makes them slow. In contrast, pure model-based methods, like the ones described below, can synthesize behavior in real time [88].

Since both approaches ultimately generate behavior by optimizing an objective, there is reason to believe they can be effectively combined. Indeed, well-known discrete-domain breakthroughs like AlphaGo [181] are predicated on combining model-based search and learning-based value and policy approximation. We believe the same could happen for robotics and control, and describe our thinking on how this might happen in the Discussion (Section 9.5). However, before the community can rise to this challenge, a core problem must be overcome: Model-based optimization is difficult to implement, often depends on elaborate optimization algorithms, and is generally inaccessible.

To address this deficit, we present *MJPC*, an open-source interactive application and software framework for predictive control, based on MuJoCo physics [40], which lets the user easily author and solve complex tasks using predictive control algorithms in real time. The tool offers implementations of standard derivative-based algorithms: iLQG (second-order planner) and Gradient Descent (first-order planner). Additionally, it introduces *Predictive Sampling*, a simple zero-order, sampling-based algorithm that works surprisingly well and is easy to understand.

Importantly, the interactive simulation can be slowed down asynchronously, speeding up the planner with respect to simulation time. This means that behaviors can be generated on older, slower machines, leading to a democratization of predictive control tooling.

MJPC and Predictive Sampling advance our central goal of lowering the barriers to entry for predictive control in robotics research. An important secondary goal is to accelerate *research velocity*. When tweaking a parameter, a researcher should not need to wait hours or minutes, but should receive instantaneous feedback – which will measurably enhance their own cognitive performance [182]. We believe that flexible, interactive simulation with researcher-authored graphical user interface is not just a "nice to have", but a prerequisite for advanced robotics research.

9.2 Background

In this section we provide a general background on Optimal Control and Trajectory Optimization, then focus on Predictive Control and derivative-free optimization.

Optimal control. Optimal Control means choosing actions in order to minimize future costs (equivalently, to maximize future returns). A dynamical system with *state* $x \in \mathbf{R}^n$, which takes a user-chosen *control* (or *action*) $u \in \mathbf{R}^m$, evolves according to the discrete-time[1] dynamics:

$$x_{t+1} = f(x_t, u_t). \tag{9.1}$$

The behavior of the system is encoded via the running cost:

$$c(x_t, u_t), \tag{9.2}$$

a function of state and control, where explicit time-dependence can be realized by folding time into the state. Future costs (a.k.a *cost-to-go* or *value*), can be defined in several ways. The summation can continue to infinity, leading to the *average-cost* or *discounted-cost* formulations, favored in temporal-difference learning, which we discuss in Section 9.5. Here we focus on the *finite-horizon* formulation, whereby the optimization objective J is given by:

$$J(x_{0:T}, u_{0:T}) = \sum_{t=0}^{T} c(x_t, u_t). \tag{9.3}$$

Trajectory optimization. Solving the finite-horizon optimal control problem (9.1, 9.3), i.e., optimizing a fixed-length trajectory, is commonly known as *planning* or *trajectory optimization*. These algorithms [167, 2] have a rich history reaching back to the Apollo Program [183, 184]. An important distinction can be made between two classes of algorithms:

- *Direct* or *simultaneous* methods have both states and controls as decision variables and enforce the dynamics (9.1) as constraints. These methods (e.g., [102]) specify a large, sparse optimization problem, which is usually solved with general-purpose software [26, 27]. They have the important benefit that non-physically-realizable trajectories can be represented, for example in order to clamp a final state, without initially knowing how to get to it.

- *Shooting* methods like Differential Dynamic Programming [23] use only the controls $u_{0:T}$ as decision variables and enforce the dynamics via forward simulation. In the shooting approach only physically-realizable trajectories can be considered, but they benefit from the reduced search space and from the optimizer not having to enforce the dynamics. The latter benefit is especially important for stiff systems like those with contact, where the difference between a physical and non-physical trajectory can be very small[2]. Unlike *direct* methods which require dynamics derivatives, shooting methods can employ derivative-free optimization, as discussed below.

[1] The continuous-time formulation is generally equivalent, we choose the discrete-time for notation simplicity.

[2] For example, consider the physical scenario of a free rigid box lying flat on a plane under gravity, and then consider the non-physical scenario of the same box hovering above the plane or penetrating it by a few microns.

Predictive control. The key idea of Predictive Control, invented in the late 60s and first published in [46], is to use trajectory optimization in *real-time* as the system dynamics are evolving. This class of algorithm has been successfully deployed in numerous real-world settings including: chemical and nuclear process control [47, 48], navigation for autonomous vehicles [49], and whole-body control of humanoid robots [137].

In the real-time setting, the current state x needs to be estimated or measured, and the trajectory optimizer is required to return a set of optimal or near-optimal controls for the finite-horizon (here often called the *receding horizon*) problem, starting at x. We use $\mathbf{\Pi}$ to denote the *plan*, the finite-horizon policy. In the context of shooting methods $\mathbf{\Pi} = u_{0:T}$, though as we discuss later, in some cases it can be re-parameterized, rather than using the discrete-time control sequence directly. Predictive control is best thought of in terms of two asynchronous processes, the *agent* and the *planner*:

Algorithm 8 Predictive Control (asynchronous)

Agent: (repeat)

1: Read the current action u from the *nominal* plan $\mathbf{\Pi}$, apply it to the controlled system.

Planner: (repeat)

1: Measure the current state x.
2: Using the nominal $\mathbf{\Pi}$ to warm start, optimize the finite-horizon objective J.
3: Update $\mathbf{\Pi}$.

Predictive Control has the following notable properties:

- Faster computation improves performance. The reason for this is clear, the more optimization steps the planner can take in one unit of time, the better the optimized control sequences will be.
- Warm starting has a large beneficial effect. By reusing the plan from the previous planning step, the optimizer only needs to make small modifications in order to correct for the changes implied by the new state. Warm starting also leads to an amortization of the optimization process across multiple planning steps.
- The optimization is not required to converge, only to improve, see Section 9.5.
- Tasks with a behavior timescale much longer than the planning horizon T are often still solvable, though this property is task dependent.
- Predictive controllers can easily get stuck in local minima, especially those using derivatives. This is due to the myopic nature of the optimization, and can be addressed by terminating the rollout with a value function approximation, see Section 9.5.3.

Derivative-free optimization. One of the main reasons for our initial focus on shooting methods is their ability to make use of derivative-free optimization, also known as *sampling-based* optimization [185]. Despite not leveraging problem structure or gradient information, this class of algorithms can discover complex behaviors [178, 83].

Sampling-based methods maintain a search distribution over policy parameters and evaluate the objective at sampled points in order to find an improved solution. Popular algorithms include: random search [186], genetic algorithms [187], and evolutionary strategies [188], including CMA-ES [189].

This class of algorithms has a number of desirable properties. First, because derivative information is not required, they are well-suited for tasks with non-smooth and discontinuous dynamics. Second, these methods are trivially parallelizable.

Sampling-based methods have been used in the predictive control context, but not very widely. Notable exceptions are Hämäläinen's work [190, 191], which is an early precursor to MJPC, and the oeuvre of Theodorou including [192] and related papers. While these methods are usually considered to be sample inefficient, we explain below why, specifically in the predictive control context, they can be surprisingly competitive.

9.3 MuJoCo MPC (MJPC)

We introduce MJPC, an open-source interactive application and software framework for predictive control, that lets the user easily synthesize behaviors for complex systems using predictive control algorithms in real time. Behaviors are specified by simple, composable objectives that are risk-aware. The planners, including: Gradient Descent, Iterative Linear Quadratic Gaussian (iLQG), and Predictive Sampling are implemented in C++ and extensively utilize multi-threading for parallel rollouts. The framework is asynchronous, enabling simulation slow-down and emulation of a faster controller, allowing this tool to run on slow machines. An intuitive graphical user interface enables real-time interactions with the environment and the ability to modify task parameters, planner and model settings, and to instantly see the effects of the modifications. The tool is available at:

`https://github.com/deepmind/mujoco_mpc`

9.3.1 Physics simulation

We build MJPC using the API of the open-source physics engine MuJoCo [40]. MuJoCo is a good infrastructure for an interactive framework for robotics algorithms for two main reasons: first, MuJoCo supports simulating multiple candidate future trajectories in parallel by offering a thread-safe API, which maximizes the utilization of modern multi-core CPU architectures; second, MuJoCo affords faster-than-real-time simulation of high-dimensional systems with many contacts — for example, the humanoid (a 27-DoF system) can be simulated 4000 times faster than real-time on a

single CPU thread.

9.3.2 Objective

MJPC provides convenient utilities to easily design and compose costs in order to specify an objective (9.3); as well as automatically and efficiently compute derivatives.

Costs. We use a "base cost" of the form:

$$l(x, u) = \sum_{i=0}^{M} w_i \cdot \mathrm{n}_i\big(r_i(x, u)\big). \tag{9.4}$$

This cost is a sum of M terms, each comprising:

- A nonnegative weight $w \in \mathbf{R}_+$ determining the relative importance of this term.
- A twice-differentiable norm $\mathrm{n}(\cdot) : \mathbf{R}^p \to \mathbf{R}_+$, which takes its minimum at 0^p.
- The residual $r \in \mathbf{R}^p$ is a vector of elements that are "small when the task is solved".

Risk sensitivity. We augment the base cost (9.4) with a risk-aware exponential scalar transformation, $\rho : \mathbf{R}_+ \times \mathbf{R} \to \mathbf{R}$, corresponding to the classical risk-sensitive control framework [193, 194]. The final running cost c is given by:

$$c(x, u) = \rho(l(x, u); R) = \frac{e^{R \cdot l(x,u)} - 1}{R}. \tag{9.5}$$

The scalar parameter $R \in \mathbf{R}$ denotes risk-sensitivity. $R = 0$ (the default) is interpreted as risk-neutral, $R > 0$ as risk-averse, and $R < 0$ as risk-seeking. The mapping ρ (see Figure 9.1)

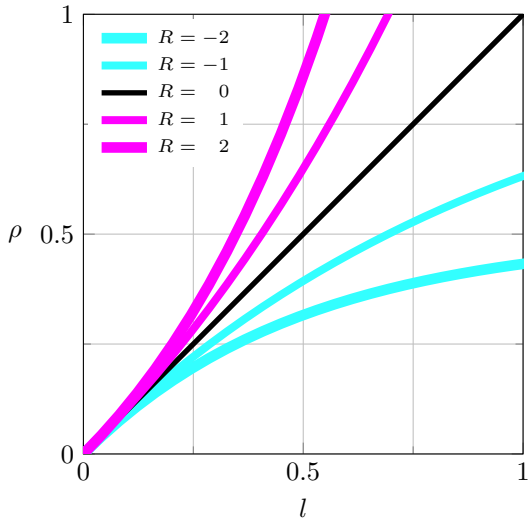


Figure 9.1: Risk transformation $\rho(l; R)$. The function is evaluated between 0 and 1 for different values of the risk parameter R.

has the following properties:

- Defined and smooth for any R.
- If $R = 0$, is the identity $\rho(l; 0) = l$ (in the limit).
- Zero is a fixed point: $\rho(0; R) = 0$.
- The derivative at 0 is 1: $\partial\rho(0; R)/\partial l = 1$.
- Non-negative: If $l \geq 0$ then $\rho(l; R) \geq 0$.
- Monotonic: $\rho(l; R) > \rho(z; R)$ if $l > z$.
- If $R < 0$ then ρ is bounded: $\rho(l; R) < -\frac{1}{R}$.
- $\rho(l; R)$ carries the same units as l.

Note that for negative R, the transformation ρ creates costs that are similar to the bounded rewards commonly used in reinforcement learning. For example, when using the quadratic norm $\mathrm{n}(r) = r^T W r$ for some SPD matrix $W = \Sigma^{-1}$ and a risk parameter $R = -1$, we get an inverted-Gaussian cost $c = 1 - e^{-r^T \Sigma^{-1} r}$, whose minimization is equivalent to maximum-likelihood maximization of the Gaussian. This leads to the interesting interpretation of the bounded rewards commonly used in RL as *risk-seeking*. We do not investigate this relationship further in this paper.

Derivatives. MuJoCo provides a utility for computing finite-difference (FD) Jacobians of the dynamics, which is efficient in two ways. First, by avoiding re-computation where possible, for example when differencing w.r.t. controls, quantities that depend only on positions and velocities are not recomputed. Second, because FD computational costs scale with the dimension of the *input*, outputs can be added cheaply. MuJoCo's step function $x_{t+1}, r_{t+1} = f(x_t, u_t)$, computes both the next state x_{t+1} and sensor values r_{t+1}, defined in the model. Because the FD approximation of the Jacobians,

$$\frac{\partial x_{t+1}}{\partial x_t}, \frac{\partial x_{t+1}}{\partial u_t}, \frac{\partial r_{t+1}}{\partial x_t}, \frac{\partial r_{t+1}}{\partial u_t}, \tag{9.6}$$

scales like the combined dimension of x_t and u_t, adding more sensors r_{t+1} is effectively "free". MJPC automatically and efficiently computes cost derivatives as follows.

Gradients. Cost gradients are computed with:

$$\frac{\partial c}{\partial x} = e^{Rl} \frac{\partial l}{\partial x} = e^{Rl} \sum_{i=0}^{M} w_i \frac{\partial \mathrm{n}_i}{\partial r} \frac{\partial r_i}{\partial x}, \tag{9.7a}$$

$$\frac{\partial c}{\partial u} = e^{Rl} \frac{\partial l}{\partial u} = e^{Rl} \sum_{i=0}^{M} w_i \frac{\partial \mathrm{n}_i}{\partial r} \frac{\partial r_i}{\partial u}. \tag{9.7b}$$

The norm gradients, $\partial \mathrm{n}/\partial r$, are computed analytically.

Hessians. Second-order derivatives use the Gauss-Newton approximation, ignoring second derivatives of r:

$$\frac{\partial^2 c}{\partial x^2} \approx e^{Rl} \left[\sum_{i=0}^{M} w_i \frac{\partial r_i}{\partial x}^T \frac{\partial^2 \mathrm{n}_i}{\partial r^2} \frac{\partial r_i}{\partial x} + R \frac{\partial l}{\partial x}^T \frac{\partial l}{\partial x} \right], \tag{9.8a}$$

$$\frac{\partial^2 c}{\partial u^2} \approx e^{Rl} \left[\sum_{i=0}^{M} w_i \frac{\partial r_i}{\partial u}^T \frac{\partial^2 \mathrm{n}_i}{\partial r^2} \frac{\partial r_i}{\partial u} + R \frac{\partial l}{\partial u}^T \frac{\partial l}{\partial u} \right], \tag{9.8b}$$

$$\frac{\partial^2 c}{\partial x \partial u} \approx e^{Rl} \left[\sum_{i=0}^{M} w_i \frac{\partial r_i}{\partial x}^T \frac{\partial^2 \mathrm{n}_i}{\partial r^2} \frac{\partial r_i}{\partial u} + R \frac{\partial l}{\partial x}^T \frac{\partial l}{\partial u} \right]. \tag{9.8c}$$

The norm Hessians, $\partial^2 \mathrm{n}/\partial r^2$, are computed analytically.

9.3.3 Splines

As we explain below, planners like iLQG require the *direct* control-sequence representation $u_{0:T}$ due to the requirements of the Bellman Principle. Without this constraint, controls can be "compressed" into a lower-dimensional object. There are many ways to do this, we picked the simplest: splines. Action trajectories are represented as a time-indexed set of knots, or control-points, parameterized

by a sequence of monotonic time points $\tau_{0:P}$ and parameter values $\theta_{0:P}$, where we use the shorthand $\theta = \theta_{0:P}$. Given a query point τ, the evaluation of the spline is given by:

$$u = s\big(\tau; (\tau_{0:P}, \theta)\big). \tag{9.9}$$

We provide three spline implementations: traditional cubic Hermite splines, piecewise-linear interpolation, and zero-order hold. See (Fig. 9.2) for an illustration.

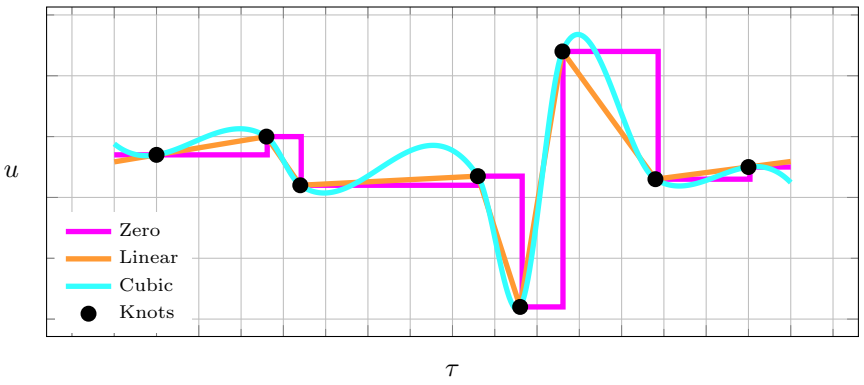


Figure 9.2: Time-indexed spline representation of the controls. Parameter points (black) utilized to construct: zero (magenta), linear (orange), and cubic (blue) interpolants.

The main benefit of compressed representations like splines is that they reduce the search space. They also smooth the control trajectory, which is often desirable. Spline functions belong to the class of linear bases, which includes the Fourier basis and orthogonal polynomials. These are useful because they allow easy propagation of gradients from the direct representation $\partial\mathbf{\Pi}$ back to the parameter values $\partial\theta$. In our case this amounts to computing:

$$\frac{\partial s}{\partial \theta}, \tag{9.10}$$

which has a simple analytic formula (see code for details). Unlike other linear bases, splines have the convenient property that bounding the values θ also bounds the spline trajectory. This is exactly true for the zero and linear interpolations, and mostly-true for cubic splines. Bounding is important as most physical systems clamp controls to bounds, and there is no point searching outside of them. Expressions for cubic, linear and zero interpolations are provided in Appendix A.

9.3.4 Planners

MJPC includes two derivative-based planners.

iLQG. The iLQG[3] planner [156], a Gauss-Newton approximation of the DDP algorithm [23], utilizes first- and second-order derivative information to take an approximate Newton step over the open-loop control sequence $u_{0:T}$ via dynamic programming [6], producing a time-varying linear feedback policy:

$$u_t = \bar{u}_t + K_t(x_t - \bar{x}_t) + \alpha k_t. \tag{9.11}$$

The *nominal*, or current best trajectory, is denoted with overbars ($^-$), K is a feedback gain matrix, and k is an improvement to the current action trajectory. A parallel line search over the step size $\alpha \in [\alpha_{\min}, 1]$ is performed to find the best improvement. Additional enhancements include a constrained backward pass [88] that enforces action limits and adaptive regularization. The details of iLQG are too involved to restate here, we refer the reader to the references above for details.

Algorithm 9 iLQG

Require: initial state x_0, nominal plan $\mathbf{\Pi} = u_{0:T}$

1: Roll out nominal trajectory from x_0 using $\mathbf{\Pi}$
2: Compute action improvements and feedback policy using Dynamic Programming.
3: Roll out parallel line search with feedback policy (9.11).
4: Best actions are new nominal actions.

Gradient descent. This first-order planner, known as Pontryagin's Maximum Principle [195], utilizes gradient information to improve action sequences, here represented as splines. The gradient of the total return is used to update the spline parameters, using a parallel line search over the step size $\alpha \in [\alpha_{\min}, \alpha_{\max}]$:

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}. \tag{9.12}$$

The total gradient is given by:

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial \mathbf{\Pi}} \frac{\partial \mathbf{\Pi}}{\partial \theta}, \tag{9.13}$$

where the spline gradient $\partial \mathbf{\Pi} / \partial \theta$ is given by (9.10), while $\partial J / \partial \mathbf{\Pi}$ is computed with the Maximum Principle. Letting λ denote the *co-state*, the gradients with respect to u are given by:

$$\lambda_j = \frac{\partial c}{\partial x_j} + \left(\frac{\partial f}{\partial x_j}\right)^T \lambda_{j+1}, \tag{9.14a}$$

$$\frac{\partial J}{\partial u_j} = \frac{\partial c}{\partial u_j} + \left(\frac{\partial f}{\partial u_j}\right)^T \lambda_{j+1}. \tag{9.14b}$$

The primary advantage of this first-order method compared to a computationally more expensive method like iLQG is that optimization is performed over the smaller space of spline parameters, instead of the entire (non-parametric) sequences of actions.

[3] Equivalently, "iLQR", since we don't make use of the noise-sensitive term for which iLQG was originally developed [81]. We keep the name "iLQG" due to its provenance.

Algorithm 10 Gradient Descent

Require: initial state x_0, nominal plan $\mathbf{\Pi}(\theta)$

1: Roll out nominal from x_0 using $\mathbf{\Pi}(\theta)$
2: Compute $\partial J / \partial \mathbf{\Pi}$ with (9.14)
3: Compute $\partial J / \partial \theta$ with (9.13)
4: Roll out parallel line-search with (9.12)
5: Pick the best one: $\theta \leftarrow \text{argmin}\big(J(\theta^{(i)})\big)$

Predictive Sampling. This is a trivial, zero-order, sampling-based Predictive Control method that works well and is easy to understand. Designed as an elementary baseline, this algorithm turned out to be surprisingly competitive with the more elaborate derivative-based algorithms.

Algorithm. A nominal sequence of actions, represented with spline parameters, is iteratively improved using random search [186]. At each iteration, N candidate splines are evaluated: the nominal itself and $N-1$ noisy samples from a Gaussian with the nominal as its mean and fixed standard deviation σ. After sampling, the actions are clamped to the control limits by clamping the spline parameters θ. Each candidate's total return is evaluated and the nominal is updated with the best candidate. See Algorithm 11 and pseudocode in Appendix C. Predictive Sampling is not innovative or performant, but is presented as a simple baseline, see Discussion below.

Algorithm 11 Predictive Sampling

Parameters: N rollouts, noise scale σ

Require: initial state x_0, nominal plan $\mathbf{\Pi}(\theta)$

1: Get $N-1$ samples $\theta_i \sim \mathcal{N}(\theta, \sigma^2)$
2: Including θ, roll out all N samples from x_0
3: Pick the best one: $\theta \leftarrow \text{argmin}\big(J(\theta^{(i)})\big)$

9.4 Results

We provide a short textual description of our graphical user interface (GUI) for three example tasks. They are best understood by viewing the associated video at: `https://dpmd.ai/mjpc`, or better yet, by downloading the software and interacting with it.

9.4.1 Graphical user interface

The MJPC GUI, shown and described in Fig. 9.3, provides an interactive simulation environment, as well as modules containing live plots and parameters that can be set by the researcher.

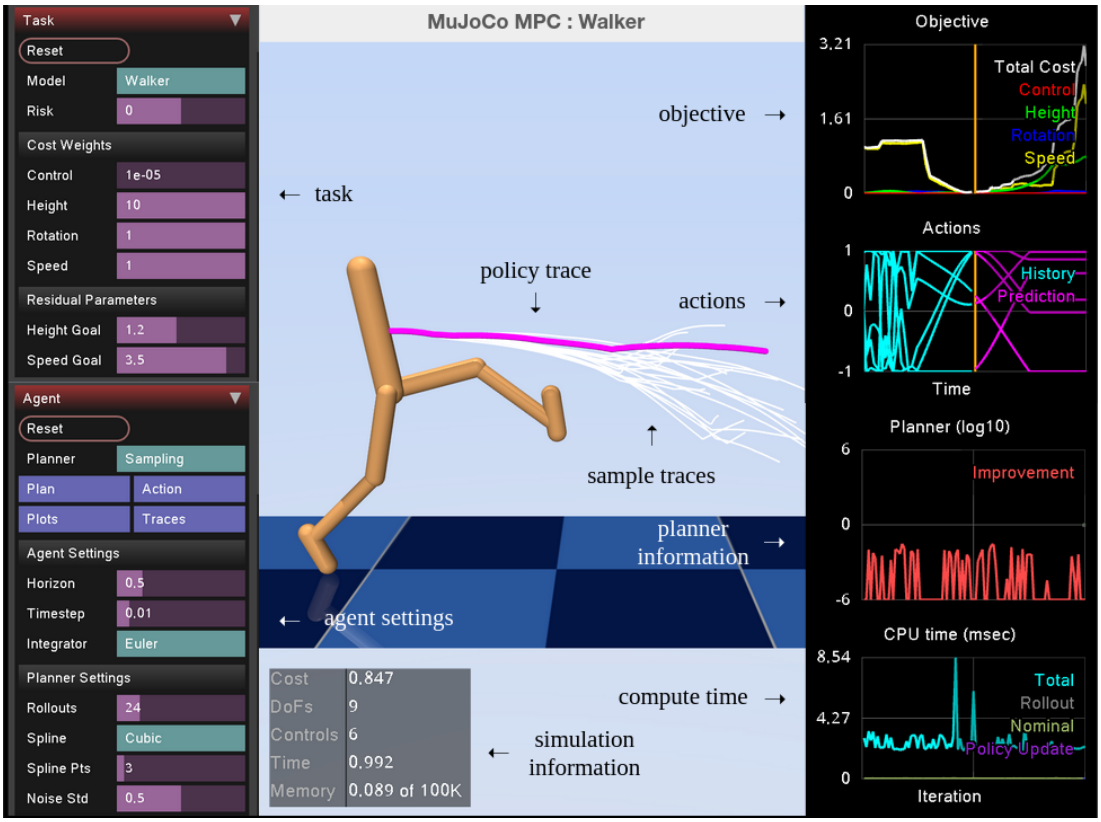


Figure 9.3: MuJoCo MPC graphical user interface. The left tab includes modules for Tasks and the Agent. In the Task module, models are selected from a drop-down menu and the risk value is set; cost weights and residual parameters are specified with interactive sliders. The Agent module provides access to a drop-down menu for planners, settings, and switches that toggle the planner and controller. The right tab includes simulation readings and predictions for the cost, including individual terms, and actions. Live, planner-specific information and various compute time results are shown below. Traces of the current policy and sampled trajectories are visualized and the user can interactively pause or slow down the simulation and apply external forces and moments to the system.

The intuitive interface makes policy design easy by enabling the researcher to interactively change cost parameters or planner settings and immediately see the results in both the simulation environment and live plots, allowing fast debugging and an enhanced understanding of the factors that influence behavior.

9.4.2 Examples

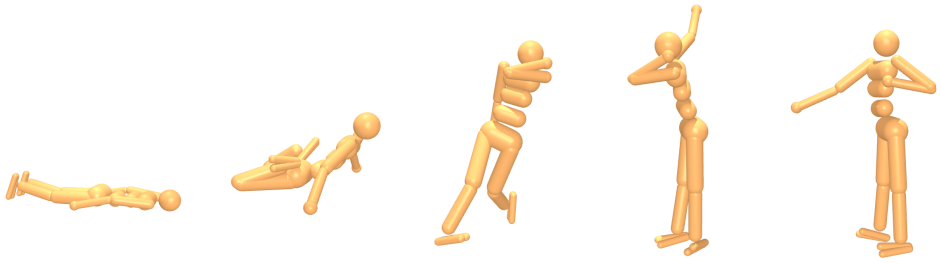
In the following examples, we demonstrate the ability to synthesize complex locomotion and manipulation behaviors for a variety of high-dimensional systems in simulation on a single CPU. Further, we demonstrate that the behaviors are robust to disturbances and mismatch between the simulation

and planning model, and can adapt extremely quickly in new scenarios. For all of the examples, the total planning time for a single update is between 1 and 20 milliseconds. We highlight three examples below and provide additional examples with the software. Experimental details for objectives and planner settings are found in the Appendix B.

Humanoid. This 27-DOF human-like system, from DeepMind Control Suite [196], has 21 actions and is tasked with standing. The system can be initialized on the floor and quickly stands in a manner that is robust to large disturbances. If a sufficiently large disturbance knocks the humanoid onto the floor, the system will stand back up (Fig. 9.4a).

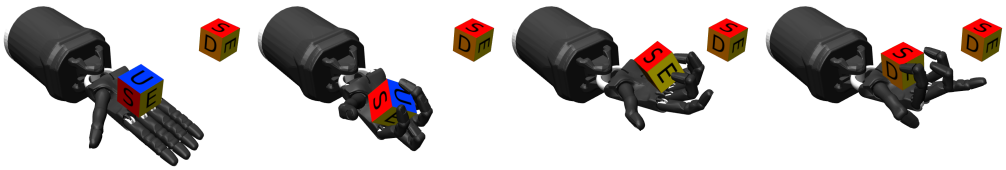
Quadruped. A Unitree A1 quadruped [80], from MuJoCo Menagerie [197], exhibits agile behavior to traverse uneven terrain which includes walking over a steep slope. On slower machines, the quadruped often struggles to ascend. In this scenario, the simulation slow down can be effectively utilized to provide the planner with addition simulation time to plan a successful climb. The system is also capable of rolling off its back and standing up (Fig. 9.4b). In order to perform long-horizon tasks like continuously navigating the terrain, a series of target poses are set. Once a goal is reached, an automatic transition occurs and the next target is set.

Hand. A Shadow Hand [198], also from MuJoCo Menagerie, performs in-hand manipulation of a cube to a desired orientation (Fig. 9.4c), where this goal can be set by the researcher in real-time by interactively setting the target orientation. In-hand reorientation—a high-DoF system with complex contact dynamics—is considered difficult to solve [199] and to the best of our knowledge has not previously been solved from scratch, in real time.



(a) Humanoid standing up off the floor.

(b) Quadruped rolling off its back to stand up.



(c) Hand manipulating a cube to a goal orientation.

Figure 9.4: Behaviors generated with MuJoCo MPC. Time progresses left to right.

9.5 Discussion

The thrust of this paper is to make predictive control accessible via customizable, interactive, open-source tooling. We believe that responsive, GUI-based tools are a prerequisite for accelerated robotics research, and that due to their importance, these tools should be modifiable and the inner workings transparent to the researcher. We hope that our MJPC project will be embraced by the community, and look forward to improving and extending it together.

9.5.1 Predictive Sampling

The effectiveness of this simple method suggests that fast, approximate optimization can be competitive with more sophisticated methods which return better solutions but at a lower rate. Does the higher planning rate completely explain this surprising effectiveness? We believe there is another, subtler reason. Predictive Control is not well-described by the tenets of traditional optimization. For example, it usually makes no sense to take more than one step of optimization. Once a single

iteration is complete, it is more important to measure a new value of the state and re-plan, than it is to continue to converge to the minimum of an already-outdated problem. The constant shifting of the optimization landscape makes Predictive Control a *qualitatively different problem*, more like surfing than mountain climbing. The goal is not to find the minimum, but to *remain in the basin-of-attraction of the minimum*. This is a different, weaker criterion, at which simple algorithms fair better than when measured by the traditional yardstick of convergence.

To be clear, Predictive Sampling is not a novel algorithm; instead, it is a baseline. A corner-case of many existing methods, it can variously be described as "MPPI with infinite temperature", "CEM with a non-adaptive distribution" or just "trivial random search". Better algorithms exist, but none are so easy to describe or implement. We are introducing Predictive Sampling not because it is good, but because it is *not good*. It is the simplest possible sampling-based shooting method, and therefore establishes a *lower bound* for performance baselines.

9.5.2 Use cases

Before we discuss limitations and their possible resolutions, it is worth asking how can MJPC be used *now*, as it is described above?

1. **Task design**. MJPC makes it easy to add new tasks, expose task parameters to the GUI, and quickly generate the desired behavior. The task can then be re-implemented in any other framework of choice. While we have not yet implemented time-dependent tasks, it is possible and easy; we expect MJPC to work especially well for motion-tracking tasks.

2. **Data generation**. MJPC can be used to generate data for learning-based approaches, i.e., it can act like an "expert policy". In this context, it is often the case that the model and task from which the data is generated do not have to exactly match the one used by the learner, and the data can likely be useful for a wide range of setups.

3. **Predictive Control research**. For researchers interested in Predictive Control itself, MJPC provides an ideal playground. MJPC can switch planners on-the-fly, and its asynchronous design affords a fair comparison by correctly accounting for and rewarding faster planners.

9.5.3 Limitations and future work

Can only control what MuJoCo can simulate. This is a general limitation of Predictive Control and is in fact stronger since one can only control what can be simulated *much faster than real-time*. For example, it is difficult to imagine any simulation of a very-high-DOF system, like fluid, cloth or soft bodies advancing so fast. One solution is improved simulation using a combination of traditional physics modeling and learning, e.g., [200]. Another possibility is to entirely learn the dynamics from observations. This approach, often termed Model Based Reinforcement Learning is

showing great promise [201, 202, 203]. We would recommend that where possible, when attempting Predictive Control using learned dynamics models, a traditional simulator be employed as a fallback as in [204], to disambiguate the effects of modeling errors.

Myopic. The core limitation of Predictive Control is that it is *myopic* and cannot see past the fixed horizon. This can be ameliorated in three conceptually straightforward ways:

1. **Learned policies**. By adding a learned policy, information from past episodes can propagate to the present via policy generalization [205]. This approach is attractive since it can only *improve* performance: when rolling out samples, one also rolls out the proposal policy. If the rollout is better, it becomes the new nominal. A learned policy is also expected to lead to more stereotypical, periodic behaviors, which are important in locomotion.

2. **Value functions**. Terminating the rollout with a learned value function which estimates the remaining cost-to-go is the obvious way by which to increase the effective horizon. Combining learned policies and value functions with model-based search would amount to an "AlphaGo for control" [181, 206].

3. **High-level agent**. A predictive controller could be used as the low-level module in a hierarchical control setup. In this scenario, the actions of the high-level agent have the semantics of setting the cost function of the predictive controller. The predictive controller remains myopic while the high-level agent contains the long-horizon "cognitive" aspects of the task. A benefit of this scenario is that the high-level actions have a much lower frequency than required for low-level control (e.g., torques).

Hardware. MJPC is aimed at robotics research, which raises the question, can it be used to control hardware?

1. **Transfer learning**. As mentioned in 9.5.2, using MJPC to generate data which can then be transferred to a real robot is already possible.

2. **Estimation**. The most obvious yet difficult route to controlling hardware is to follow in the footsteps of classic control and couple MJPC to an estimator providing real-time state estimates. In the rare cases where estimation is easy, for example with fixed-base manipulators and static objects, controlling a robot directly with MJPC would be a straightforward exercise. The difficult and interesting case involves free-moving bodies and contacts, as in locomotion and manipulation. For certain specific cases, like locomotion on flat, uniform terrain, reasonable estimates should not be difficult to obtain. For the general case, we believe that contact-aware estimation is possible following the approach of [207], but that remains to be seen. Similarly, we believe that high-quality estimators also require the same kind of interactive, GUI-driven interface used by MJPC.

9.6 Appendix A: Interpolation

This section provides analytical expressions for computing interpolations. The indices j and $j+1$ correspond to the indices of the domain variables which contain the query point τ. These values are efficiently found using binary search in $\mathbf{O}(\log(n))$ where n is the dimension of the domain variable set.

Zero. The zero-order interpolation simply returns the parameter values at the lower bound index:

$$\theta_j \leftarrow s. \tag{9.15}$$

Linear. The linear interpolation returns:

$$(1-q)\cdot\theta_j + q\cdot\theta_{j+1} \leftarrow s, \tag{9.16}$$

where $q = (\tau - \tau_j)/(\tau_{j+1} - \tau_j)$.

Cubic. The cubic interpolation leverages finite-difference approximations of the slope at the interval points:

$$\phi_j = \frac{1}{2}\left(\frac{\theta_{j+1}-\theta_j}{\tau_{j+1}-\tau_j} + \frac{\theta_j - \theta_{j-1}}{\tau_j - \tau_{j-1}}\right), \tag{9.17}$$

$$\phi_{j+1} = \frac{1}{2}\left(\frac{\theta_{j+2}-\theta_{j+1}}{\tau_{j+2}-\tau_{j+1}} + \frac{\theta_{j+1} - \theta_j}{\tau_{j+1} - \tau_j}\right), \tag{9.18}$$

and returns:

$$a\cdot\theta_j + b\cdot\phi_j + c\cdot\theta_{j+1} + d\cdot\phi_{j+1} \leftarrow s, \tag{9.19}$$

where:

$$a = 2q^3 - 3q^2 + 1, \tag{9.20}$$

$$b = (q^3 - 2q^2 + q)\cdot(\tau_{j+1} - \tau_j), \tag{9.21}$$

$$c = -2q^3 + 3q^2, \tag{9.22}$$

$$d = (q^3 - q^2)\cdot(\tau_{j+1} - \tau_j). \tag{9.23}$$

9.7 Appendix B: Tasks

This section provides additional information about the examples provided in Section 9.4, including objective formulations and planner settings.

Humanoid objective. This task comprises $M = 6$ cost terms:

- Term 0:

  r_0: lateral center-of-mass position and average lateral feet position alignment

  n_0: hyperbolic cosine

  w_0: 100

- Term 1:

  r_1: lateral torso position and lateral center-of-mass position alignment

  n_1: smooth absolute value

  w_1: 1

- Term 2:

  - r_2: head height and feet height difference minus target height difference
  - n_2: smooth absolute value
  - w_2: 100

- Term 3:

  r_3: lateral center-of-mass velocity

  n_3: smooth absolute value

  w_3: 10

- Term 4:

  r_4: joint velocity

  n_4: quadratic

  w_4: 0.01

- Term 5:

  r_5: control effort

  n_5: quadratic

  w_5: 0.025

Quadruped objective. This task comprises $M = 4$ cost terms:

- Term 0:

  r_0: body height and average feet height difference minus target height

  n_0: quadratic

  w_0: 1

- Term 1:

  r_1: body position minus goal position

  n_1: quadratic

  w_1: 5.0

- Term 2:

  r_2: body orientation minus goal orientation

  n_2: quadratic

  w_2: 1.0

- Term 3:

  r_3: control effort

  n_3: quadratic

  w_3: 0.25

Hand objective. This task comprises $M = 3$ cost terms:

- Term 0:

  r_0: cube position minus hand palm position

  n_0: quadratic

  w_0: 20

- Term 1:

  r_1: cube orientation minus goal orientation

  n_1: quadratic

  w_1: 3

- Term 2:

  r_2: cube linear velocity

  n_2: quadratic

  w_2: 10

Planner settings. We provide the settings used for Predictive Sampling in Table 9.1.

Table 9.1: Predictive Sampling settings

Task	P	N	T	σ
Humanoid	3	10	23	0.125
Quadruped	3	10	35	0.25
Hand	6	10	25	0.1

9.8 Appendix C: Predictive Sampling Algorithm

We provide a numerical algorithm for Predictive Sampling:

Algorithm 12 PredictiveSampling

procedure OPTIMIZEPOLICY
  task: f, c, R
  settings: T, N, σ, s
  initialise: (τ, θ)
  for $k = 0, \dots, \infty$
    $(x, \tau) \leftarrow$ get state
    $(\bar{\tau}, \theta) \leftarrow$ resample
    for $i = 0, \dots, N$ (parallel)
      $\tilde{\theta}^{(i)} = \theta + \begin{cases} 0, & i = 0, \\ \mathcal{N}(0, \sigma^2 \cdot I), & \text{else} \end{cases}$
      $x_0^{(i)} = x$, $\tau_0^{(i)} = \tau$,
      for $t = 0, \dots, T$
        $u_t^{(i)} = s(\tau_t^{(i)}; (\bar{\tau}, \tilde{\theta}^{(i)}))$
        $c_t^{(i)} = c(x_t^{(i)}, u_t^{(i)}; R)$
        $(x_{t+1}^{(i)}, \tau_{t+1}^{(i)}) = f(x_t^{(i)}, u_t^{(i)})$
      end for
    end for
    $\theta \leftarrow \text{argmin} \left(J(\tilde{\theta}^{(i)}) \right)$
  end for
procedure ACTIONFROMPOLICY
  $(x, \tau) \leftarrow$ get state
  $u = s(\tau; (\bar{\tau}, \theta))$
  return u

9.9 Appendix D: Compute Resources

Experiments were performed on a Lenovo ThinkStation P920 with 48GB of memory and an Intel Xeon Gold 6154 72-core CPU. Additional experiments were performed on an Apple MacBook Pro (2021) with 16 GB of memory and an M1 Pro CPU.

Bibliography

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

[2] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[3] J. E. Marsden and M. West, "Discrete mechanics and variational integrators," *Acta Numerica*, vol. 10, pp. 357–514, 2001.

[4] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

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

[6] R. E. Kalman, "When Is a Linear Control System Optimal?" *Journal of Basic Engineering*, vol. 86, no. 1, pp. 51–60, 1964.

[7] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[8] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction," *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.

[9] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 2014.

[10] Y. Nesterov, "Interior point polynomial methods in convex programming," *Theory and Applications*, 1994.

[11] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, 2017, pp. 136–145.

[12] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi, "Differentiating through a cone program," *arXiv:1904.09043*, 2019.

[13] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable physics engine for robotics," *arXiv:2203.00806*, 2022.

[14] J. Brüdigam and Z. Manchester, "Linear-time variational integrators in maximal coordinates," in *International Workshop on the Algorithmic Foundations of Robotics*, 2020, pp. 194–209.

[15] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[16] R. W. Cottle, J. Pang, and R. E. Stone, *The Linear Complementarity Problem.* SIAM, 2009.

[17] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

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

[19] G. Lantoine and R. P. Russell, "A hybrid differential dynamic programming algorithm for constrained optimal control problems. Part 1: Theory," *Journal of Optimization Theory and Applications*, vol. 154, no. 2, pp. 382–417, 2012.

[20] B. Plancher, Z. Manchester, and S. Kuindersma, "Constrained unscented dynamic programming," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 5674–5680.

[21] P. G. Kaminski, A. E. Bryson, and S. F. Schmidt, "Discrete square root filtering: A survey of current techniques," *IEEE Transactions on Automatic Control*, vol. 16, no. 6, pp. 727–736, 1971.

[22] T. A. Howell, S. Le Cleac'h, S. Singh, P. Florence, Z. Manchester, and V. Sindhwani, "Trajectory optimization with optimization-based dynamics," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6750–6757, 2022.

[23] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming.* Elsevier Publishing Company, 1970, no. 24.

[24] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[25] T. A. Howell, S. Le Cleac'h, K. Tracy, and Z. Manchester, "CALIPSO: A differentiable solver for trajectory optimization with conic and complementarity constraints," *arXiv:2205.09255*, 2022.

[26] 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.

[27] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[28] P. E. Gill and D. P. Robinson, "A primal-dual augmented Lagrangian," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 1–25, 2012.

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

[30] 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.

[31] A. F. Izmailov, M. V. Solodov, and E. I. Uskov, "Global convergence of augmented Lagrangian methods applied to optimization problems with degenerate constraints, including problems with complementarity constraints," *SIAM Journal on Optimization*, vol. 22, no. 4, pp. 1579–1606, 2012.

[32] M. Szmuk, T. P. Reynolds, and B. Açıkmeşe, "Successive convexification for real-time six-degree-of-freedom powered descent guidance with state-triggered constraints," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, pp. 1399–1413, 2020.

[33] T. A. Howell, C. Fu, and Z. Manchester, "Direct policy optimization using deterministic sampling and collocation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5324–5331, 2021.

[34] Z. Manchester and S. Kuindersma, "Robust direct trajectory optimization using approximate invariant funnels," *Autonomous Robots*, vol. 43, no. 2, pp. 375–387, 2019.

[35] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[36] S. Le Cleac'h, T. A. Howell, S. Yang, C. Lee, J. Zhang, A. Bishop, M. Schwager, and Z. Manchester, "Fast contact-implicit model-predictive control," *arXiv:2107.05616*, 2021.

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

[38] "Unitree Go1 Quadruped," https://m.unitree.com/products/go1/, 2022.

[39] T. A. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive Sampling: Real-time behaviour synthesis with MuJoCo," *arXiv:2212.00541*, 2022.

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

[41] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

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

[43] 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.

[44] S. P. Dirkse and M. C. Ferris, "The PATH Solver: A nommonotone stabilization scheme for mixed complementarity problems," *Optimization Methods and Software*, vol. 5, no. 2, pp. 123–156, 1995.

[45] E. R. Westervelt, J. W. Grizzle, and D. E. Koditschek, "Hybrid zero dynamics of planar biped walkers," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 42–56, 2003.

[46] J. Richalet, A. Rault, J. L. Testud, and J. Papon, "Model predictive heuristic control: Applications to industrial processes," *Automatica*, vol. 14, no. 5, pp. 413–428, 1978.

[47] M. G. Na, S. H. Shin, and W. C. Kim, "A model predictive controller for nuclear reactor power," *Nuclear Engineering and Technology*, vol. 35, no. 5, pp. 399–411, 2003.

[48] R. Lopez-Negrete, F. J. D'Amato, L. T. Biegler, and A. Kumar, "Fast nonlinear model predictive control: Formulation and industrial process applications," *Computers & Chemical Engineering*, vol. 51, pp. 55–64, 2013.

[49] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[50] Boston Dynamics. (2019) More Parkour Atlas. [Online]. Available: https://www.youtube.com/watch?v=_sBBaNYex3E

[51] R. Kuhlmann and C. Büskens, "A primal-dual augmented Lagrangian penalty-interior-point filter line search algorithm," *Mathematical Methods of Operations Research*, vol. 87, no. 3, pp. 451–483, 2018.

[52] M. Argáez and R. A. Tapia, "On the global convergence of a modified augmented Lagrangian line-search interior-point Newton method for nonlinear programming," *Journal of Optimization Theory and Applications*, vol. 114, no. 1, pp. 1–25, 2002.

[53] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *European Control Conference*, 2013, pp. 3071–3076.

[54] J. Domke, "Generic methods for optimization-based modeling," in *Artificial Intelligence and Statistics*, 2012, pp. 318–326.

[55] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[56] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving Rubik's cube with a robot hand," *arXiv:1910.07113*, 2019.

[57] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

[58] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," *arXiv:2107.04034*, 2021.

[59] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[60] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[61] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax–A differentiable physics engine for large scale rigid body simulation," *arXiv:2106.13281*, 2021.

[62] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics for articulated rigid bodies with contact," *arXiv:2103.16021*, 2021.

[63] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "ADD: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Transactions on Graphics*, vol. 39, no. 6, pp. 1–15, 2020.

[64] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "DiffTaichi: Differentiable programming for physical simulation," *International Conference on Learning Representations*, 2020.

[65] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "NeuralSim: Augmenting differentiable simulators with neural networks," pp. 9474–9481, 2021.

[66] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540*, 2016.

[67] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo," in *IEEE International Conference on Robotics and Automation*, 2014, pp. 6054–6061.

[68] H. J. T. Suh, T. Pang, and R. Tedrake, "Bundled gradients through contact via randomized smoothing," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4000–4007, 2022.

[69] R. Elandt, E. Drumwright, M. Sherman, and A. Ruina, "A pressure field model for fast, robust approximation of net contact force and moment between nominally rigid objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 8238–8245.

[70] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2004, pp. 2149–2154.

[71] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[72] Nvidia, "PhysX physics engine," 2022. [Online]. Available: https://developer.nvidia.com/physx-sdk

[73] J. Brüdigam, J. Janeva, S. Sosnowski, and S. Hirche, "Linear-time contact and friction dynamics in maximal coordinates using variational integrators," *arXiv:2109.07262*, 2021.

[74] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," 2016–2019.

[75] 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.

[76] B. E. Jackson, K. Tracy, and Z. Manchester, "Planning with attitude," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5658–5664, 2021.

[77] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 4397–4404.

[78] H. Sharma, M. Patil, and C. Woolsey, "Energy-preserving variational integrators for forced Lagrangian systems," *Communications in Nonlinear Science and Numerical Simulation*, vol. 64, pp. 159–177, 2018.

[79] 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.

[80] Unitree, "Unitree A1 Quadruped," https://m.unitree.com/products/a1, 2022.

[81] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *International Conference on Informatics in Control, Automation and Robotics*, 2004, pp. 222–229.

[82] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[83] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 1800–1809.

[84] S. Pfrommer, M. Halm, and M. Posa, "ContactNets: Learning discontinuous contact dynamics with smooth, implicit representations," in *Conference on Robot Learning*, 2021, pp. 2279–2291.

[85] D. Pardo, L. Möller, M. Neunert, A. W. Winkler, and J. Buchli, "Evaluating direct transcription and nonlinear optimization methods for robot motion planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 946–953, 2016.

[86] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[87] H. B. Keller, *Numerical Methods for Two-Point Boundary-Value Problems*. Courier Dover Publications, 2018.

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

[89] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 695–702.

[90] T. C. Lin and J. S. Arora, "Differential dynamic programming technique for constrained optimal control," *Computational Mechanics*, vol. 9, no. 1, pp. 27–40, 1991.

[91] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative Gauss-Newton shooting methods for nonlinear optimal control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 1–9.

[92] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 93–100.

[93] M. Toussaint, "A novel augmented Lagrangian approach for inequalities and convergent any-time non-central updates," *arXiv:1412.4329*, 2014.

[94] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for MIT 6.832)," 2022.

[95] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[96] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, 2018, pp. 8289–8300.

[97] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning convex optimization control policies," in *Learning for Dynamics and Control*, 2020, pp. 361–373.

[98] 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.

[99] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 3346–3351.

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

[101] 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 *IEEE International Conference on Robotics and Automation*, 2020, pp. 2536–2542.

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

[103] W. Jallet, N. Mansard, and J. Carpentier, "Implicit differential dynamic programming," in *International Conference on Robotics and Automation*, 2022, pp. 1455–1461.

[104] 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.

[105] 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.

[106] B. Landry, J. Lorenzetti, Z. Manchester, and M. Pavone, "Bilevel optimization for planning through contact: A semidirect method," in *The International Symposium of Robotics Research*, 2019, pp. 789–804.

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

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

[109] 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.

[110] 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.

[111] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for large convex problems," in *European Control Conference*, 2019, pp. 1951–1956.

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

[113] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," *Advances in Neural Information Processing Systems*, vol. 20, 2007.

[114] 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.

[115] 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.

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

[117] 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.

[118] T. Schlick, *Molecular Modeling and Simulation: An Interdisciplinary Guide*. Springer, 2010, vol. 2.

[119] R. J. Vanderbei and H. Yurttan, "Using LOQO to solve second-order cone programming problems," *Constraints*, vol. 1, p. 2, 1998.

[120] H. Scheel and S. Scholtes, "Mathematical programs with complementarity constraints: Stationarity, optimality, and sensitivity," *Mathematics of Operations Research*, vol. 25, no. 1, pp. 1–22, 2000.

[121] M. Szmuk, D. Malyuta, T. P. Reynolds, M. S. Mceowen, and B. Açikmeşe, "Real-time quad-rotor path planning using convex optimization and compound state-triggered constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 7666–7673.

[122] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. SIAM, 2010.

[123] T. A. Davis, "Algorithm 849: A concise sparse Cholesky factorization package," *ACM Transactions on Mathematical Software*, vol. 31, no. 4, pp. 587–591, 2005.

[124] S. Singh, J. Slotine, and V. Sindhwani, "Optimizing trajectories with closed-loop dynamic SQP," in *International Conference on Robotics and Automation*, 2022, pp. 5249–5254.

[125] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, "Constrained differential dynamic programming: A primal-dual augmented Lagrangian approach," 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03597630

[126] B. E. Jackson, T. Punnoose, D. Neamati, K. Tracy, R. Jitosho, and Z. Manchester, "ALTRO-C: A fast solver for conic model-predictive control," in *IEEE International Conference on Robotics and Automation*, 2021, pp. 7357–7364.

[127] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *International Journal of Control*, vol. 93, no. 1, pp. 13–29, 2020.

[128] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *International Conference on Robotics and Automation*, 2019, pp. 6741–6747.

[129] A. U. Raghunathan and L. T. Biegler, "Mathematical programs with equilibrium constraints (MPECs) in process engineering," *Computers & Chemical Engineering*, vol. 27, no. 10, pp. 1381–1392, 2003.

[130] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, 2019, pp. 9562–9574.

[131] S. Tu and R. Frostig, "Trajax: A Python library for differentiable optimal control on accelerators," 2022. [Online]. Available: https://github.com/google/trajax

[132] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[133] H. S. Najafi, S. A. Edalatpanah, and G. A. Gravvanis, "An efficient method for computing the inverse of arrowhead matrices," *Applied Mathematics Letters*, vol. 33, pp. 1–5, 2014.

[134] M. Palan, S. Barratt, A. McCauley, D. Sadigh, V. Sindhwani, and S. Boyd, "Fitting a linear control policy to demonstrations with a Kalman constraint," in *Learning for Dynamics and Control*, 2020, pp. 374–383.

[135] O. Hinder and Y. Ye, "A one-phase interior point method for nonconvex optimization," *arXiv:1801.03072*, 2018.

[136] D. P. Bertsekas and S. Shreve, *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 2004.

[137] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the Atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, 2016.

[138] J. L. Moore, "Robust post-stall perching with a fixed-wing UAV," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[139] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference*, 2005, pp. 300–306.

[140] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014.

[141] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[142] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 23–30.

[143] J. Bu, A. Mesbahi, M. Fazel, and M. Mesbahi, "LQR through the lens of first order methods: Discrete-time case," *arXiv:1907.08921*, 2019.

[144] M. Wytock and J. Z. Kolter, "A fast algorithm for sparse controller design," *arXiv:1312.4892*, 2013.

[145] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[146] J. K. Uhlmann, "Dynamic map building and localization: New theoretical foundations," Ph.D. dissertation, University of Oxford, 1995.

[147] H. M. T. Menegaz, J. Y. Ishihara, G. A. Borges, and A. N. Vargas, "A systematization of the unscented Kalman filter theory," *IEEE Transactions on Automatic Control*, vol. 60, no. 10, pp. 2583–2598, 2015.

[148] J. S. Meditch, "On the problem of optimal thrust programming for a lunar soft landing," *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 477–484, 1964.

[149] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[150] A. D. Ames, K. Galloway, K. Sreenath, and J. W. Grizzle, "Rapidly exponentially stabilizing control Lyapunov functions and hybrid zero dynamics," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 876–891, 2014.

[151] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for robust parameterized locomotion control of bipedal robots," in *IEEE International Conference on Robotics and Automation*, 2021, pp. 2811–2817.

[152] A. Aydinoglu, V. M. Preciado, and M. Posa, "Contact-aware controller design for complementarity systems," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 1525–1531.

[153] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv:1707.02286*, 2017.

[154] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[155] J. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified MPC framework for whole-body dynamic locomotion and manipulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4688–4695, 2021.

[156] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913.

[157] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of Honda humanoid robot," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1998, pp. 1321–1326.

[158] G. Bledt, "Regularized predictive control framework for robust dynamic legged locomotion," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[159] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 295–302.

[160] M. Chignoli, D. Kim, E. Stanger-Jones, and S. Kim, "The MIT humanoid robot: Design, motion planning, and control for acrobatic behaviors," pp. 1–8, 2021.

[161] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.

[162] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv:1909.06586*, 2019.

[163] E. Drumwright, "Rapidly computable viscous friction and no-slip rigid contact models," *arXiv:1504.00719*, 2015.

[164] M. Kojima, N. Megiddo, T. Noma, and A. Yoshise, "A unified approach to interior point algorithms for linear complementarity problems: A summary," *Operations Research Letters*, vol. 10, no. 5, pp. 247–254, 1991.

[165] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," *Advances in Neural Information Processing Systems*, vol. 31, pp. 7178–7189, 2018.

[166] M. Anitescu, "Optimization-based simulation of nonsmooth rigid multibody dynamics," *Mathematical Programming*, vol. 105, no. 1, pp. 113–143, 2006.

[167] O. Von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, vol. 37, no. 1, pp. 357–373, 1992.

[168] I. Yamazaki, S. Nooshabadi, S. Tomov, and J. Dongarra, "Structure-aware linear solver for realtime convex optimization for embedded systems," *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 61–64, 2017.

[169] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[170] T. Marcucci and R. Tedrake, "Warm start of mixed-integer programs for model predictive control of hybrid systems," *Transactions on Automatic Control*, vol. 66, no. 6, pp. 2433–2448, 2020.

[171] J. E. Pratt, "Exploiting Inherent Robustness and Natural Dynamics in the Control of Bipedal Walking Robots," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[172] J. Pratt and G. Pratt, "Intuitive control of a planar bipedal walking robot," in *IEEE International Conference on Robotics and Automation*, vol. 3, 1998, pp. 2014–2021.

[173] G. Bledt, P. M. Wensing, and S. Kim, "Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the MIT cheetah," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 4102–4109.

[174] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the MIT cheetah 3 through convex model-predictive control," in *International Conference on Intelligent Robots and Systems*, 2018, pp. 1–9.

[175] R. E. Kalman, "On the general theory of control systems," in *International Conference on Automatic Control*, 1960, pp. 481–492.

[176] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[177] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[178] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv:1703.03864*, 2017.

[179] L. Smith, I. Kostrikov, and S. Levine, "A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning," *arXiv:2208.07860*, 2022.

[180] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, 2022, pp. 91–100.

[181] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[182] Z. Lu and B. A. Dosher, "Current directions in visual perceptual learning," *Nature Reviews Psychology*, vol. 1, no. 11, pp. 654–668, 2022.

[183] NASA, *Apollo 11 Mission Report*. Scientific and Technical Information Office, National Aeronautics and Space Administration, 1971, vol. 238.

[184] R. E. Smith and J. W. Yound, "Trajectory optimization for an Apollo-type vehicle under entry conditions encountered during lunar return," Tech. Rep., 1967.

[185] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*. Springer, 2017, vol. 2.

[186] J. Matyas, "Random optimization," *Automation and Remote Control*, vol. 26, no. 2, pp. 246–253, 1965.

[187] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.

[188] I. Rechenberg, "Evolutionsstrategie," *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution*, 1973.

[189] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[190] P. Hämäläinen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen, "Online motion synthesis using sequential Monte Carlo," *ACM Transactions on Graphics*, 2014.

[191] P. Hämäläinen, J. Rajamäki, and C. K. Liu, "Online control of simulated humanoids using particle belief propagation," *ACM Transactions on Graphics*, 2015.

[192] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[193] D. Jacobson, "Optimal stochastic linear systems with exponential performance criteria and their relation to deterministic differential games," *IEEE Transactions on Automatic Control*, vol. 18, no. 2, pp. 124–131, 1973.

[194] P. Whittle, "Risk-sensitive linear/quadratic/Gaussian control," *Advances in Applied Probability*, vol. 13, no. 4, pp. 764–777, 1981.

[195] O. L. Mangasarian, "Sufficient conditions for the optimal control of nonlinear systems," *SIAM Journal on Control*, vol. 4, no. 1, pp. 139–152, 1966.

[196] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020.

[197] MuJoCo Menagerie Contributors, "MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo," 2022. [Online]. Available: http://github.com/deepmind/mujoco_menagerie

[198] P. Tuffield and H. Elias, "The Shadow robot mimics human actions," *Industrial Robot: An International Journal*, 2003.

[199] T. Chen, J. Xu, and P. Agrawal, "A system for general in-hand object re-orientation," in *Conference on Robot Learning*, 2022, pp. 297–307.

[200] L. Ladicky, S. Jeong, N. Bartolovic, M. Pollefeys, and M. Gross, "PhysicsForests: Real-time fluid simulation using machine learning," in *ACM SIGGRAPH Real Time Live!*, 2017, p. 22.

[201] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[202] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on Robot Learning*, 2020, pp. 1101–1112.

[203] P. Wu, A. Escontrela, D. Hafner, K. Goldberg, and P. Abbeel, "DayDreamer: World models for physical robot learning," *arXiv:2206.14176*, 2022.

[204] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[205] A. Byravan, L. Hasenclever, P. Trochim, M. Mirza, A. D. Ialongo, Y. Tassa, J. T. Springenberg, A. Abdolmaleki, N. Heess, J. Merel *et al.*, "Evaluating model-based planning and planner amortization for continuous control," *arXiv:2110.03363*, 2021.

[206] J. T. Springenberg, N. Heess, D. Mankowitz, J. Merel, A. Byravan, A. Abdolmaleki, J. Kay, J. Degrave, J. Schrittwieser, Y. Tassa *et al.*, "Local search for policy iteration in continuous control," *arXiv:2010.05545*, 2020.

[207] K. Lowrey, S. Kolev, Y. Tassa, T. Erez, and E. Todorov, "Physically-consistent sensor fusion in contact-rich behaviors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1656–1662.