Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact

Keenon Werling[1], Dalton Omens[1], Jeongseok Lee[2], Ioannis Exarchos[1] and C. Karen Liu[1]
[1]Stanford University: {keenon, domens, exarchos, ckliu38}@stanford.edu
[2]Robotics AI, Amazon: jeoslee@amazon.com

***Abstract*—We present a fast and feature-complete differentiable physics engine, Nimble (nimblephysics.org), that supports Lagrangian dynamics and hard contact constraints for articulated rigid body simulation. Our differentiable physics engine offers a complete set of features that are typically only available in non-differentiable physics simulators commonly used by robotics applications. We solve contact constraints precisely using linear complementarity problems (LCPs). We present efficient and novel analytical gradients through the LCP formulation of inelastic contact that exploit the sparsity of the LCP solution. We support complex contact geometry, and gradients approximating continuous-time elastic collision. We also introduce a novel method to compute *complementarity-aware gradients* that help downstream optimization tasks avoid stalling in saddle points. We show that an implementation of this combination in a fork of an existing physics engine (DART) is capable of a 87x single-core speedup over finite-differencing in computing analytical Jacobians for a single timestep, while preserving all the expressiveness of original DART.**

I. Introduction

Many modern robotics problems are optimization problems. Finding optimal trajectories, guessing the physical parameters of a world that best fits our observed data, or designing a control policy to optimally respond to a dynamic environment are all types of optimization problems. Optimization methods can be sorted into two buckets: gradient-based, and gradient-free. Despite the well-known drawbacks of gradient-free methods (high sample complexity and noisy solutions), they remain popular in robotics because physics engines with the necessary features to model complex robots are generally non-differentiable. When gradients are required, we typically approximate them using finite differencing [41].

Recent years have seen many differentiable physics engines published [11, 19, 15, 42, 38, 10, 34, 29, 12], but none has yet gained traction as a replacement for popular non-differentiable engines [9, 41, 26]. We hypothesize that an ideal differentiable physics engine needs to implement an equivalent feature set to existing popular non-differentiable engines, as well as provide excellent computational efficiency, in order to gain adoption. In this paper, we extend the existing physics engine DART [26], which is commonly used in robotics and graphics communities, and make it differentiable. Our resulting engine supports all features available to the forward simulation process, meaning existing code and applications will remain compatible while also enjoying new capabilities enabled by efficient analytical differentiability.

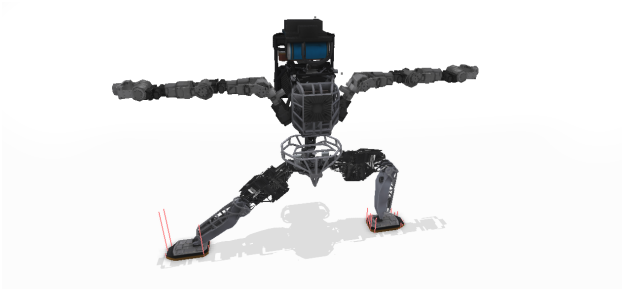

Fig. 1. Boston Dynamics' Atlas Robot learning to do yoga, being simulated by our engine. This robot has 34 mesh colliders and 32 degrees of freedom. This freeze frame contains 24 contact points, 12 per foot. Even with all that complexity, we are able to compute the Jacobians of dynamics on this robot **87x faster** on a single CPU core using the analytical methods introduced in this paper than by finite differencing (only **8.5ms** vs 749ms for finite differencing).

A fully-featured physics engine like DART is complex and has many components that must be differentiated. Some components (such as collision detection and contact force computation) are not naively differentiable, but we show that under very reasonable assumptions we can compute useful Jacobians regardless. In order to differentiate through contacts and collision forces, we introduce an efficient method for differentiating the contact Linear Complementarity Problem (LCP) that exploits sparsity, as well as novel contact geometry algorithms and an efficient continuous-time approximation for elastic collisions. As a result, our engine is able to compute gradients with hard contact constraints up to 87 times faster than finite differencing methods, depending on the size of the dynamic system. Our relative speedup over finite differencing grows as the system complexity grows. In deriving our novel method to differentiate through the LCP, we also gain an understanding of the nature of contact dynamics Jacobians that allows us to propose heuristic "complementarity-aware gradients" that may be good search directions to try, if a downstream optimizer is getting stuck in a saddle point.

We provide an open-source implementation of all of these ideas, as well as derivations from previous work [25, 7], in a fully differentiable fork of the DART physics engine, which we call Nimble. Code and documentation are available at nimblephysics.org.

To summarize, our contributions are as follows:

- A novel and fast method for local differentiability of LCPs that exploits the sparsity of the LCP solution, which gives us efficient gradients through static and sliding

Engine	Contact Force	Dynamic State	Collision Geometry	Gradients Method
MuJoCo	customized	**generalized**	**complete**	finite
Degrave	impulse	cartesian	primitives	auto
DiffTaichi	impulse	caresian	primitives	auto
Heiden	iter LCP	**generalized**	primitives	auto
de A. B.-P.	**direct LCP**	cartesian	primitives	**symbolic**
Geilinger	customized	**generalized**	primitives	**symbolic**
Ours	**direct LCP**	**generalized**	**complete**	**symbolic**

TABLE I
DIFFERENTIABLE ENGINES SUPPORTING ARTICULATED RIGID BODIES

contacts and friction without changing traditional forward-simulation formulations. *Section IV*

- A novel method that manipulates gradient computation to help downstream optimization problems escape saddle points due to discrete contact states. *Section IV-D*
- Fast geometric analytical gradients through collision detection algorithms which support various types of 3D geometry and meshes. *Section V*
- A novel analytical approximation of continuous-time gradients through elastic contact, which otherwise can lead to errors in discrete-time systems. *Section VI*
- An open-source implementation of all of our proposed methods (along with analytical gradients through Featherstone first described in GEAR [25]) in a fork of the DART physics engine which we call Nimble. We have created Python bindings and a PyPI package, `pip install nimblephysics`, for ease of use.

II. RELATED WORK

Differentiable physics simulation has been investigated previously in many different fields, including mechanical engineering [16], robotics [14], physics [23, 21] and computer graphics [32, 28]. Enabled by recent advances in automatic differentiation methods and libraries [31, 1, 18], a number of differentiable physics engines have been proposed to solve control and parameter estimation problems for rigid bodies [25, 11, 18, 15, 10, 12, 34] and non-rigid bodies [37, 27, 13, 20, 34, 17, 12]. While they share a similar high-level goal of solving "inverse problems", the features and functionality provided by these engines vary widely, including the variations in contact handling, state space parameterization and collision geometry support. Table II highlights the differences in a few differentiable physics engines that have demonstrated the ability to simulate articulated rigid bodies with contact. Based on the functionalities each engine intends to support, the approaches to computing gradients can be organized in following categories.

Finite-differencing is a straightforward way to approximate gradients of a function. For a feature-complete physics engine, where analytical gradients are complex to obtain, finite-differencing provides a simpler method. For example, a widely used physics engine, MuJoCo [41], supports gradient computation via finite differencing. However, finite-differencing tends to introduce round-off errors and performs poorly for a large number of input variables.

Automatic differentiation (auto-diff) is a method for computing gradients of a sequence of elementary arithmetic operations or functions automatically. However, the constraint satisfaction problems required by many existing, feature-complete robotic physics engines are not supported by auto-diff libraries. To avoid this issue, many recent differentiable physics engines instead implement impulse-based contact handling, which could lead to numerical instability and constraint violation if the contact parameters are not tuned properly for the specific dynamic system and the simulation task. Degrave et al. [11] implemented a rigid body simulator in the Theano framework [1], while DiffTaichi [18] implemented a number of differentiable physics engines, including rigid bodies, extending the Taichi programming language [19], both representing dynamic equations in Cartesian coordinates and handling contact with impulse-based methods. In contrast, Tiny Differentiable Simulator [15] models contacts as an LCP, but they solve the LCP iteratively via Projected Gauss Siedel (PGS) method [24], instead of directly solving a constraint satisfaction problem, making it possible to compute gradient using auto-diff libraries.

Symbolic differentiation is another way to compute gradients by directly differentiate mathematical expressions. For complex programs like Lagrangian dynamics with constraints formulated as a Differential Algebraic Equations, symbolic differentiation can be exceedingly difficult. Earlier work computed symbolic gradients for smooth dynamic systems [25], and [7] simplified the computation of the derivative of the forward dynamics exploiting the derivative of the inverse dynamics. Symbolic differentiation becomes manageable when the gradients are only required within smooth contact modes [42] or a specific contact mode is assumed [38]. Recently, Amos and Kolter proposed a method, Opt-Net, that back-propagates through the solution of an optimization problem to its input parameters [2]. Building on Opt-Net, de Avila Belbute-Peres et al. [10] derived analytical gradients through LCP formulated as a QP. Their method enables differentiability for rigid body simulation with hard constraints, but their implementation represents 2D rigid bodies in Cartesian coordinates and only supports collisions with a plane, insufficient for simulating complex articulated rigid body systems. More importantly, computing gradients via QP requires solving a number of linear systems which does not take advantage of the sparsity of the LCP structure. Qiao et al. [34] built on [2] and improved the performance of contact handling by breaking a large scene into smaller impact zones. A QP is solved for each impact zone to ensure that the geometry is not interpenetrating, but contact dynamics and conservation laws are not considered. Solving contacts for localized zones has been previously implemented in many existing physics engines [41, 9, 26]. Adapting the collision handling routine in DART, our method by default utilizes the localized contact zones to speed up the performance. Adjoint sensitivity analysis [35] has also been used for computing gradients of dynamics. Millard et

al. [29] combined auto-diff with adjoint sensitivity analysis to achieve faster gradient computation for higher-dof systems, but their method did not handle contact and collision. Geilinger et al. [12] analytically computed derivatives through adjoint sensitivity analysis and proposed a differentiable physics engine with implicit forward integration and a customized frictional contact model that is natively differentiable.

Approximating physics with neural networks is a different approach towards differentiable physics engine. Instead of forward simulating a dynamic system from the first principles of Newtonian mechanics, a neural network is learned from training data. Examples of this approach include Battaglia et. al [4], Chang et. al. [8], and Mrowca et. al [30].

Our engine employs symbolic differentiation to compute gradients through every part of the engine using hand-tuned C++ code. We introduce a novel method to differentiate the LCP analytically that takes advantage of the sparsity of the solution and is compatible with using direct methods to solve the LCP. In addition, our engine supports a richer set of geometry for collision and contact handling than has been previously available, including mesh-mesh and mesh-primitive collisions, in order to achieve a fully functional differentiable version of the DART physics engine for robotic applications.

III. Overview

A physics engine can be thought of as a simple function that takes the current position $\boldsymbol{q}_t$, velocity $\dot{\boldsymbol{q}}_t$, control forces $\boldsymbol{\tau}$ and inertial properties $\boldsymbol{\mu}$, and returns the position and velocity at the next timestep, $\boldsymbol{q}_{t+1}$ and $\dot{\boldsymbol{q}}_{t+1}$:

$$P(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\tau}, \boldsymbol{\mu}) = [\boldsymbol{q}_{t+1}, \dot{\boldsymbol{q}}_{t+1}]. \tag{1}$$

In an engine with simple explicit time integration, our next position $\boldsymbol{q}_{t+1}$ is a trivial function of current position and velocity, $\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \Delta t \dot{\boldsymbol{q}}_t$, where Δt is the descritized time interval.

The computational work of the physics engine comes from solving for our next velocity, $\dot{\boldsymbol{q}}_{t+1}$. We are representing our articulated rigid body system in generalized coordinates using the following Lagrangian dynamic equation:

$$\begin{aligned}\boldsymbol{M}(\boldsymbol{q}_t, \boldsymbol{\mu})\dot{\boldsymbol{q}}_{t+1} = \boldsymbol{M}(\boldsymbol{q}_t, \boldsymbol{\mu})\dot{\boldsymbol{q}}_t - \Delta t(\boldsymbol{c}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\mu}) - \boldsymbol{\tau}) \\ + \boldsymbol{J}^T(\boldsymbol{q}_t)\boldsymbol{f},\end{aligned} \tag{2}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{c}$ is the Coriolis and gravitational force, and $\boldsymbol{f}$ is the contact impulse transformed into the generalized coordinates by the contact Jacobian matrix $\boldsymbol{J}$. Note that multiple contact points and/or other constraint impulses can be trivially added to Equation 2.

Every term in Equation 2 can be evaluated given $\boldsymbol{q}_t$, $\dot{\boldsymbol{q}}_t$ and $\boldsymbol{\tau}$ except for the contact impulse $\boldsymbol{f}$, which requires the engine to form and solve an LCP:

$$\begin{aligned}&\text{find} \quad \boldsymbol{f}, \boldsymbol{v}_{t+1} \\ &\text{such that} \quad \boldsymbol{f} \geq \boldsymbol{0},\ \boldsymbol{v}_{t+1} \geq \boldsymbol{0},\ \boldsymbol{f}^T\boldsymbol{v}_{t+1} = 0.\end{aligned} \tag{3}$$

The velocity of a contact point at the next time step, $\boldsymbol{v}_{t+1}$, can be expressed as a linear function in $\boldsymbol{f}$

$$\begin{aligned}\boldsymbol{v}_{t+1} &= \boldsymbol{J}\dot{\boldsymbol{q}}_{t+1} = \boldsymbol{J}\boldsymbol{M}^{-1}\left(\boldsymbol{M}\dot{\boldsymbol{q}}_t - \Delta t(\boldsymbol{c} - \boldsymbol{\tau}) + \boldsymbol{J}^T\boldsymbol{f}\right) \\ &= \boldsymbol{A}\boldsymbol{f} + \boldsymbol{b},\end{aligned} \tag{4}$$

where $\boldsymbol{A} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^T$ and $\boldsymbol{b} = \boldsymbol{J}(\dot{\boldsymbol{q}}_t + \Delta t\boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{c}))$. The LCP procedure can then be expressed as a function that maps $(\boldsymbol{A}, \boldsymbol{b})$ to the contact impulse $\boldsymbol{f}$:

$$f_{\text{LCP}}\big(\boldsymbol{A}(\boldsymbol{q}_t, \boldsymbol{\mu}), \boldsymbol{b}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\tau}, \boldsymbol{\mu})\big) = \boldsymbol{f} \tag{5}$$

As such, the process of forward stepping is to find $\boldsymbol{f}$ and resulting $\dot{\boldsymbol{q}}_{t+1}$ that satisfy Equation 2 and Equation 5.

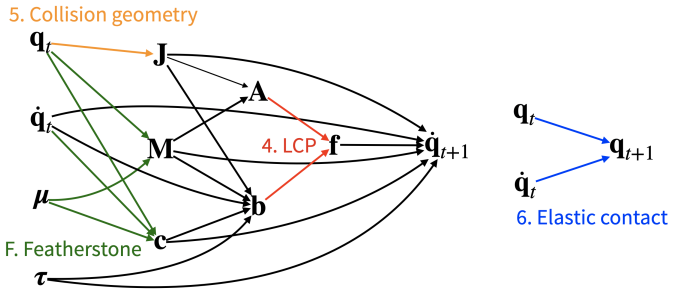

Fig. 2. Data flow during a forward simulation, visualizing equations 2 and 5. The inputs are $\boldsymbol{q}_t$ and $\dot{\boldsymbol{q}}_t$, the current position and velocity in generalized coordinates, μ, the inertial properties, and τ, the external (control) torques on the joints. The outputs are the generalized position and velocity at the next timestep, $\boldsymbol{q}_{t+1}$ and $\dot{\boldsymbol{q}}_{t+1}$. Every forward arrow represents a dependency in data flow, which must be differentiated during backpropagation. Challenging dependencies, and the relevant sections where we introduce analytical Jacobians, are labeled with colored arrows and text in the diagram.

The main task in developing a differentiable physics engine is to solve for the gradient of next velocity $\dot{\boldsymbol{q}}_{t+1}$ with respect to the input to the current time step, namely $\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\tau}_t$, and $\boldsymbol{\mu}$. The data flow is shown in Figure 2. For brevity we refer to the ouput of a function for a given timestep by the same name as the function with a subscript t (e.g. $\boldsymbol{J}_t = \boldsymbol{J}(\boldsymbol{q}_t)$). The velocity at the next step can be simplified to

$$\dot{\boldsymbol{q}}_{t+1} = \dot{\boldsymbol{q}}_t + \boldsymbol{M}_t^{-1}\boldsymbol{z}_t, \tag{6}$$

where $\boldsymbol{z}_t \equiv -\Delta t(\boldsymbol{c}_t - \boldsymbol{\tau}_t) + \boldsymbol{J}_t^T\boldsymbol{f}_t$. The gradients we need to compute at each time step are written as:

$$\begin{aligned}\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{q}_t} = \frac{\partial \boldsymbol{M}_t^{-1}\boldsymbol{z}_t}{\partial \boldsymbol{q}_t} + \boldsymbol{M}_t^{-1}\left(-\Delta t\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{q}_t} + \frac{\partial \boldsymbol{J}_t^T}{\partial \boldsymbol{q}_t}\boldsymbol{f}_t\right. \\ \left. + \boldsymbol{J}_t^T\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{q}_t}\right)\end{aligned} \tag{7}$$

$$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \dot{\boldsymbol{q}}_t} = \boldsymbol{I} + \boldsymbol{M}_t^{-1}\left(-\Delta t\frac{\partial \boldsymbol{c}_t}{\partial \dot{\boldsymbol{q}}_t} + \boldsymbol{J}_t^T\frac{\partial \boldsymbol{f}_t}{\partial \dot{\boldsymbol{q}}_t}\right) \tag{8}$$

$$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\tau}_t} = \boldsymbol{M}_t^{-1}\left(\Delta t\boldsymbol{I} + \boldsymbol{J}_t^T\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{\tau}_t}\right) \tag{9}$$

$$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\mu}} = \frac{\partial \boldsymbol{M}_t^{-1}\boldsymbol{z}_t}{\partial \boldsymbol{\mu}} + \boldsymbol{M}_t^{-1}\left(-\Delta t\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{\mu}} + \boldsymbol{J}_t^T\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{\mu}}\right) \tag{10}$$

We tackle the tricky intermediate Jacobians in sections that follow. In Section IV we will introduce a novel sparse

analytical method to compute the gradients of contact force $\boldsymbol{f}_t$ with respect to $\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\tau}_t, \boldsymbol{\mu}$. Section V will discuss $\frac{\partial \boldsymbol{J}_t}{\partial \boldsymbol{q}_t}$—how collision geometry changes with respect to changes in position. In Section VI we will tackle $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ and $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$, which is not as simple as it may at first appear, because naively taking gradients through a discrete time physics engine yields problematic results when elastic collisions take place. Additionally, the appendix gives a way to apply the derivations from [25] and [7] to analytically find $\frac{\partial \boldsymbol{M}_t^{-1}\boldsymbol{z}_t}{\partial \boldsymbol{q}_t}$, $\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{q}_t}$, and $\frac{\partial \boldsymbol{c}_t}{\partial \dot{\boldsymbol{q}}_t}$.

IV. Differentiating the LCP

This section introduces a method to analytically compute $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}}$ and $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}}$. It turns out that it is possible to efficiently get unambiguous gradients through an LCP in the vast majority of practical scenarios, without recasting it as a QP (which throws away sparsity information by replacing the complementarity constraint with an objective function). To see this, let us consider a hypothetical LCP problem parameterized by $\boldsymbol{A}, \boldsymbol{b}$ with a solution $\boldsymbol{f}^*$ found during the forward pass: $f_{\text{LCP}}(\boldsymbol{A}, \boldsymbol{b}) = \boldsymbol{f}^*$.

For brevity, we only include the discussion on normal contact impulses in this section and leave the extension to friction impulses in Appendix A. Therefore, each element in $\boldsymbol{f}^* \geq \boldsymbol{0}$ indicates the *normal* impulse of a point contact. By complementarity, we know that if some element $f_i^* > 0$, then $v_i = (\boldsymbol{A}\boldsymbol{f}^* + \boldsymbol{b})_i = 0$. Intuitively, the relative velocity at contact point i *must be 0* if there is any non-zero impulse being exerted at contact point i. We call such contact points "Clamping" because the impulse $f_i > 0$ is adjusted to keep the relative velocity $v_i = 0$. Let the set $\mathcal{C}$ to be all indices that are clamping. Symmetrically, if $f_j = 0$, then the relative velocity $v_j = (\boldsymbol{A}\boldsymbol{f}^* + \boldsymbol{b})_j \geq 0$ is free to vary without the LCP needing to adjust f_j to compensate. We call such contact points "Separating" and define the set $\mathcal{S}$ to be all indices that are separating. Let us call indices j where $f_j = 0$ *and* $v_j = 0$ "Tied." Define the set $\mathcal{T}$ to be all indices that are tied.

If no contact points are tied ($\mathcal{T} = \emptyset$), the LCP is strictly differentiable and the gradients can be analytically computed. When some contact points are tied ($\mathcal{T} \neq \emptyset$), the LCP has valid subgradients and it is possible to follow any in an optimization. The tied case is analogous to the non-differentiable points in a QP where an inequality constraint is active while the corresponding dual variable is also zero. In such a case, computing gradients via taking differentials of the KKT conditions will result in a low-rank linear system and thus non-unique gradients [2].

A. Strictly differentiable cases

Consider the case where $\mathcal{T} = \emptyset$. We shuffle the indices of $\boldsymbol{f}^*$, $\boldsymbol{v}$, $\boldsymbol{A}$ and $\boldsymbol{b}$ to group together members of $\mathcal{C}$ and $\mathcal{S}$. The LCP becomes:

$$\begin{aligned} &\text{find} && \boldsymbol{f}_{\mathcal{C}}^*, \boldsymbol{f}_{\mathcal{S}}^*, \boldsymbol{v}_{\mathcal{C}}, \boldsymbol{v}_{\mathcal{S}} \\ &\text{such that} && \begin{bmatrix} \boldsymbol{v}_{\mathcal{C}} \\ \boldsymbol{v}_{\mathcal{S}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{\mathcal{C}\mathcal{C}} & \boldsymbol{A}_{\mathcal{C}\mathcal{S}} \\ \boldsymbol{A}_{\mathcal{S}\mathcal{C}} & \boldsymbol{A}_{\mathcal{S}\mathcal{S}} \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_{\mathcal{C}}^* \\ \boldsymbol{f}_{\mathcal{S}}^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_{\mathcal{C}} \\ \boldsymbol{b}_{\mathcal{S}} \end{bmatrix} \\ & && \boldsymbol{f}_{\mathcal{C}}^* \geq \boldsymbol{0},\ \boldsymbol{f}_{\mathcal{S}}^* \geq \boldsymbol{0},\ \boldsymbol{v}_{\mathcal{C}} \geq \boldsymbol{0},\ \boldsymbol{v}_{\mathcal{S}} \geq \boldsymbol{0} \\ & && \boldsymbol{f}_{\mathcal{C}}^{*T} \boldsymbol{v}_{\mathcal{C}} = 0,\ \boldsymbol{f}_{\mathcal{S}}^{*T} \boldsymbol{v}_{\mathcal{S}} = 0. \end{aligned} \tag{11}$$

Since we know the classification of each contact that forms the valid solution $\boldsymbol{f}^*$, we rewrite the LCP constraints as follows:

$$\begin{aligned} &\text{find} && \boldsymbol{f}_{\mathcal{C}}^*, \boldsymbol{f}_{\mathcal{S}}^*, \boldsymbol{v}_{\mathcal{C}}, \boldsymbol{v}_{\mathcal{S}} \\ &\text{such that} && \begin{bmatrix} \boldsymbol{v}_{\mathcal{C}} \\ \boldsymbol{v}_{\mathcal{S}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{\mathcal{C}\mathcal{C}} & \boldsymbol{A}_{\mathcal{C}\mathcal{S}} \\ \boldsymbol{A}_{\mathcal{S}\mathcal{C}} & \boldsymbol{A}_{\mathcal{S}\mathcal{S}} \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_{\mathcal{C}}^* \\ \boldsymbol{f}_{\mathcal{S}}^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_{\mathcal{C}} \\ \boldsymbol{b}_{\mathcal{S}} \end{bmatrix} \\ & && \boldsymbol{f}_{\mathcal{C}}^* > \boldsymbol{0},\ \boldsymbol{f}_{\mathcal{S}}^* = \boldsymbol{0},\ \boldsymbol{v}_{\mathcal{C}} = \boldsymbol{0},\ \boldsymbol{v}_{\mathcal{S}} > \boldsymbol{0}. \end{aligned} \tag{12}$$

From here we can see how the valid solution $\boldsymbol{f}^*$ changes under infinitesimal perturbations $\boldsymbol{\epsilon}$ to $\boldsymbol{A}$ and $\boldsymbol{b}$. Since $\boldsymbol{f}_{\mathcal{S}}^* = \boldsymbol{0}$ and $\boldsymbol{v}_{\mathcal{C}} = \boldsymbol{0}$, the LCP can be reduced to three conditions on $\boldsymbol{f}_{\mathcal{C}}^*$:

$$\boldsymbol{0} = \boldsymbol{A}_{\mathcal{C}\mathcal{C}} \boldsymbol{f}_{\mathcal{C}}^* + \boldsymbol{b}_{\mathcal{C}} \tag{13}$$

$$\boldsymbol{f}_{\mathcal{C}}^* > \boldsymbol{0} \tag{14}$$

$$\boldsymbol{A}_{\mathcal{S}\mathcal{C}} \boldsymbol{f}_{\mathcal{C}}^* + \boldsymbol{b}_{\mathcal{S}} > \boldsymbol{0}. \tag{15}$$

We will show that these conditions will always be possible to satisfy under small enough perturbations $\boldsymbol{\epsilon}$ in the neighborhood of a valid solution. Let us first consider tiny perturbations to $\boldsymbol{b}_{\mathcal{S}}$ and $\boldsymbol{A}_{\mathcal{S}\mathcal{C}}$. If the perturbations are small enough, then Equation 15 will still be satisfied with our original $\boldsymbol{f}_{\mathcal{C}}^*$, because we know Equation 15 already holds *strictly* such that there is some non-zero room to decrease any element of $\boldsymbol{A}_{\mathcal{S}\mathcal{C}} \boldsymbol{f}_{\mathcal{C}}^* + \boldsymbol{b}_{\mathcal{S}}$ without violating Equation 15. Therefore,

$$\frac{\partial \boldsymbol{f}_{\mathcal{C}}^*}{\partial \boldsymbol{b}_{\mathcal{S}}} = \boldsymbol{0} \ \text{ and } \ \frac{\partial \boldsymbol{f}_{\mathcal{C}}^*}{\partial \boldsymbol{A}_{\mathcal{S}\mathcal{C}}} = \boldsymbol{0}. \tag{16}$$

Next let us consider an infinitesimal perturbation $\boldsymbol{\epsilon}$ to $\boldsymbol{b}_{\mathcal{C}}$ and the necessary change on the clamping force $\Delta \boldsymbol{f}_{\mathcal{C}}^*$ to satisfy Equation 13:

$$\boldsymbol{0} = \boldsymbol{A}_{\mathcal{C}\mathcal{C}} (\boldsymbol{f}_{\mathcal{C}}^* + \Delta \boldsymbol{f}_{\mathcal{C}}^*) + \boldsymbol{b}_{\mathcal{C}} + \boldsymbol{\epsilon}. \tag{17}$$

Setting $\boldsymbol{A}_{\mathcal{C}\mathcal{C}} \boldsymbol{f}_{\mathcal{C}}^* + \boldsymbol{b}_{\mathcal{C}} = \boldsymbol{0}$ and assuming $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is invertible, the change of the clamping force is given as $\Delta \boldsymbol{f}_{\mathcal{C}}^* = -\boldsymbol{A}_{\mathcal{C}\mathcal{C}}^{-1} \boldsymbol{\epsilon}$. Since $\boldsymbol{f}_{\mathcal{C}}^*$ is strictly greater than $\boldsymbol{0}$, it is always possible to choose an $\boldsymbol{\epsilon}$ small enough to make $\boldsymbol{f}_{\mathcal{C}}^* - \boldsymbol{A}_{\mathcal{C}\mathcal{C}}^{-1} \boldsymbol{\epsilon} > 0$ and $\boldsymbol{A}_{\mathcal{S}\mathcal{C}} (\boldsymbol{f}_{\mathcal{C}}^* + \Delta \boldsymbol{f}_{\mathcal{C}}^*) + \boldsymbol{b}_{\mathcal{S}} > 0$ remain true. Therefore,

$$\frac{\partial \boldsymbol{f}_{\mathcal{C}}^*}{\partial \boldsymbol{b}_{\mathcal{C}}} = -\boldsymbol{A}_{\mathcal{C}\mathcal{C}}^{-1}. \tag{18}$$

Note that $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is not always invertible because $\boldsymbol{A}$ is positive *semidefinite*. We will discuss the case when $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is not full rank in Section IV-C along with a method to stabilize the LCP when there exists multiple LCP solutions in Appendix B.

Lastly, we compute gradients with respect to $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$. In practice, changes to $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ only happen because we are differentiating with respect to parameters $\boldsymbol{q}$ or $\boldsymbol{\mu}$, which also changes $\boldsymbol{b}_{\mathcal{C}}$. As such, we introduce a new scalar variable, x, which could represent any arbitrary scalar quantity that effects both $\boldsymbol{A}$ and $\boldsymbol{b}$. Equation 13 can be rewritten as:

$$\boldsymbol{f}_{\mathcal{C}}^* = -\boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)^{-1} \boldsymbol{b}_{\mathcal{C}}(x). \tag{19}$$

Because $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)$ and $\boldsymbol{b}_{\mathcal{C}}(x)$ are continuous, and the original solution is valid, any sufficiently small perturbation to x will

not reduce $\boldsymbol{f}^*_{\mathcal{C}}$ below 0 or violate Equation 15. The Jacobian with respect to x can be expressed as:

$$\frac{\partial \boldsymbol{f}^*_{\mathcal{C}}}{\partial x} = \boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)^{-1}\frac{\partial \boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)}{\partial x}\boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)^{-1}\boldsymbol{b}_{\mathcal{C}}(x) + \boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)^{-1}\frac{\partial \boldsymbol{b}_{\mathcal{C}}(x)}{\partial x}. \quad (20)$$

Using $\boldsymbol{A} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^T$ and $\boldsymbol{b} = \boldsymbol{J}(\dot{\boldsymbol{q}}_t + \Delta t \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{c}))$, along with the derivations of Featherstone presented in Appendix F, it is possible to compute $\frac{\partial \boldsymbol{A}_{\mathcal{C}\mathcal{C}}}{\partial x}$ and $\frac{\partial \boldsymbol{b}_{\mathcal{C}}}{\partial x}$ for any specific x.

Remark: Previous methods [10, 27, 34] cast an LCP to a QP and solved for a linear system of size $n+m$ derived from taking differentials of the KKT conditions of the QP, where n is the dimension of the state variable and m is the number of contact constraints. Our method also solves for linear systems to obtain $\boldsymbol{A}^{-1}_{\mathcal{C}\mathcal{C}}$, but the size of $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is often much less than m due to the sparsity of the solution $(\boldsymbol{f}^*, \boldsymbol{v}^*)$.

B. Subdifferentiable case

Now let us consider when $\mathcal{T} \neq \emptyset$. Replacing the LCP constraints with linear constraints will no longer work because any perturbation will immediately change the state of the contact and the change also depends on the direction of perturbation. Including the class of "tied" contact points to Equation 11, we need to satisfy an additional linear system,

$$\begin{aligned}\boldsymbol{v}_{\mathcal{T}} &= \boldsymbol{A}_{\mathcal{T}\mathcal{T}}\boldsymbol{f}^*_{\mathcal{T}} + \boldsymbol{A}_{\mathcal{T}\mathcal{C}}\boldsymbol{f}^*_{\mathcal{C}} + \boldsymbol{A}_{\mathcal{T}\mathcal{S}}\boldsymbol{f}^*_{\mathcal{S}} + \boldsymbol{b}_{\mathcal{T}} \\ &= \boldsymbol{A}_{\mathcal{T}\mathcal{T}}\boldsymbol{f}^*_{\mathcal{T}} + \tilde{\boldsymbol{b}}_{\mathcal{T}}, \end{aligned} \quad (21)$$

where $\tilde{\boldsymbol{b}}_{\mathcal{T}} = \boldsymbol{A}_{\mathcal{T}\mathcal{C}}\boldsymbol{f}^*_{\mathcal{C}} + \boldsymbol{b}_{\mathcal{T}}$ and $\boldsymbol{v}_{\mathcal{T}}$ and $\boldsymbol{f}^*_{\mathcal{T}}$ are both zero at the solution. Let $i \in \mathcal{T}$ be the index of a tied contact point. Consider perturbing the i'th element of $\tilde{\boldsymbol{b}}_{\mathcal{T}}$ by ϵ. If $\epsilon > 0$, $\boldsymbol{A}_{\mathcal{T}\mathcal{T}}\boldsymbol{f}^*_{\mathcal{T}}$ cannot become negative to balance Equation 21 because $\boldsymbol{A}_{\mathcal{T}\mathcal{T}}$ is positive semidefinite and $\boldsymbol{f}^*_{\mathcal{T}}$ must be nonnegative. Therefore, $v_{\mathcal{T}i}$ must become positive, resulting contact point i being separated and $f^*_{\mathcal{T}i}$ remaining zero. If $\epsilon < 0$, then i is immediately bumped into the "clamping" set $\mathcal{C}$ because $v_{\mathcal{T}i}$ cannot be negative. Therefore, $f^*_{\mathcal{T}i}$ must become positive to balance Equation 21. The gradients for the clamping and separating cases are both valid subgraidents for a tied contact point. In an optimization, we can choose either of the two subgradients at random without impacting the convergence [6]. In practice, encountering elements in $\mathcal{T}$ is quite rare for practical numerical reasons.

C. When $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is not full rank

When $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is not full rank, the solution to $f_{\text{LCP}}(\boldsymbol{A}, \boldsymbol{b}) = \boldsymbol{f}^*$ is no longer unique. Nevertheless, once a solution is computed using any algorithm and the clamping set $\mathcal{C}$ is found, we can use the stabilization method proposed in Appendix B to find the least-squares minimal $\boldsymbol{f}^*$ that solves the LCP. The gradient of clamping forces can then be written as:

$$\frac{\partial \boldsymbol{f}^*_{\mathcal{C}}}{\partial x} = \boldsymbol{A}^{-1}_{\mathcal{C}\mathcal{C}}\frac{\partial \boldsymbol{A}_{\mathcal{C}\mathcal{C}}}{\partial x}\boldsymbol{A}^{-1}_{\mathcal{C}\mathcal{C}}\boldsymbol{b}_{\mathcal{C}} + \boldsymbol{A}^{-1}_{\mathcal{C}\mathcal{C}}\frac{\partial \boldsymbol{b}_{\mathcal{C}}}{\partial x} + (\boldsymbol{I} - \boldsymbol{A}^{+}_{\mathcal{C}\mathcal{C}}\boldsymbol{A}_{\mathcal{C}\mathcal{C}})\left(\frac{\partial \boldsymbol{A}^T_{\mathcal{C}\mathcal{C}}}{\partial x}\right)\boldsymbol{A}^{+T}_{\mathcal{C}\mathcal{C}}\boldsymbol{A}^{+}_{\mathcal{C}\mathcal{C}}, \quad (22)$$

where $\boldsymbol{A}^{+}_{\mathcal{C}\mathcal{C}}$ is the pseudo inverse matrix of the low-rank $\boldsymbol{A}$. For numerical stability, we can solve a series of linear systems instead of explicitly evaluating $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}(x)^{+}$.

D. Complementarity-aware gradients via contact constraints

Sometimes analytically correct Jacobians through the LCP can actually prevent an optimizer from finding a good solution. When a contact i is clamping ($i \in \mathcal{C}$), we effectively impose a constraint that the relative velocity at that contact point is zero no matter how we push *or pull* on the contact. This prevents the gradients from pointing towards any motions that require breaking contact, because our constraints will zero out gradients that lead to $\boldsymbol{v}_i > 0$.

This behavior is caused by the complementarity constraint requiring that at most one of v_i or f_i can be non-zero. This phenomenon grows out of the short-sightedness of the gradient (it only considers an infinitely small neighborhood ϵ around the current state). While it is true that small attempts to push $v_i > 0$ will result in no change to $\boldsymbol{v}$ as $\boldsymbol{f}$ compensates to keep $v_i = 0$, eventually we will reach a discontinuity where $\boldsymbol{f}$ can no longer prevent the contact from separating. However, a gradient-based optimizer may never take steps in that direction because gradients in that direction are 0. In this section we propose a heuristic to opportunistically explore "projecting forward" to the discontinuity where the complementarity constraint flips during backpropagation, depending on the gradient of loss function with respect to $\boldsymbol{v}$ and $\frac{\partial \ell}{\partial \boldsymbol{v}}$.

To make it more concrete, let us consider a simple example of a 2D circle attached to a linear actuator that can produce force along the vertical axis (Figure 3). Our goal is to lift the circle up to a target height above the ground by generating an upward velocity using the linear actuator. When the circle is resting on the ground under gravity, the contact point with the ground is classified as "clamping". This means that any tiny perturbation in the control force of linear actuator will be met with an exactly offsetting contact force to ensure that the relative velocity between the circle and the ground remains zero. As such, no matter what gradient of loss function with respect to the control force we try to backpropagate through the contact, we will always get $\frac{\partial l}{\partial \boldsymbol{\tau}} = \boldsymbol{0}$.

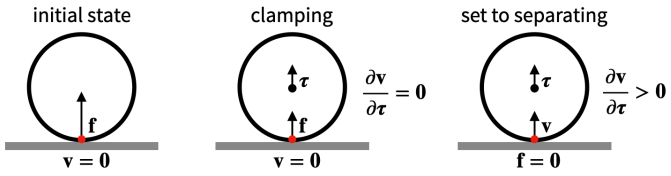

Fig. 3. Increasing force from the linear actuator ($\boldsymbol{\tau}$) is met by an equal and opposite decrease in ground reaction force $\boldsymbol{f}$, resulting in no change in velocity and thus no gain in height. Gradients of loss function will be zeroed out by $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\tau}} = \boldsymbol{0}$ when backpropgating through the contact. In contrast, if the contact is classified as "separating", the gradient will increase the linear actuator, resulting in an upward force $\boldsymbol{v}$.

Consider another example where the same 2D circle is uncontrolled and resting on a linearly actuated platform (Figure 4). To lift the circle up, we need to utilize the contact force

from the platform. If the initialization of the problem results in the contact point being classified as "separating", no constraint force will be applied on the contact point, as if the contact did not exist. Therefore, the gradient of contact force with respect to the platform control force is zero, which may cause the optimization to be trapped in a saddle point.

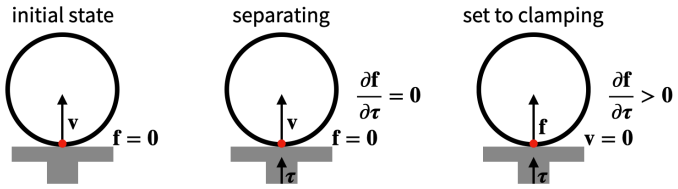

Fig. 4. If the initial contact state is "separating", the platform cannot apply contact force to the circle, resulting in zero gradient of the contact force with respect to the control force of the platform, $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\tau}} = \mathbf{0}$. However, if the contact is classified as "clamping", we can utilize the contact force from the platform to push the circle upwards by increasing the control force of the platform.

In both examples, the issues can be resolved by classifying the contact point differently. If we know that to solve this problem we need to actuate the circle and the platform separately, we would set the contact point to be "separating." This gives the optimizer a chance to explore a solution with contact breaking. On the other hand, if we know that we need to exploit the contact force between the circle and the platform, we would relabel a separating contact point to be "clamping". This lets the optimizer search for the most effective contact force that reduces the loss function.

Is there a way to know which strategy to use in advance? In the general case, the answer seems to be no, but we can do better than naive gradients which stick with whatever contact classification they were initialized into and do not explore to break out of resulting saddle points. We propose a heuristic to explore more broadly at a constant additional cost.

We propose that always picking the contact strategy that results in the largest $||\frac{\partial \ell}{\partial \dot{\boldsymbol{q}}_t}||_2^2 + \frac{||\frac{\partial \ell}{\partial \boldsymbol{\tau}_t}||_2^2}{\Delta t}$ is a good heuristic during learning for avoiding saddle points. While we could try backprop through all $\mathcal{O}(2^n)$ possible strategies and pick the best one, that gets expensive as the number of contacts becomes larger than a small handful. Instead of exhaustively searching for all possible combinations of contact states with exponential complexity, we propose to only check two classifications:

1) First, check the "correct" contact classification solved by the LCP.
2) Second, we compute $\frac{\partial \ell}{\partial \boldsymbol{v}}$ (the gradient of loss with respect to relative contact velocity), and use the elements of $\frac{\partial \ell}{\partial \boldsymbol{v}}$ to compute the "clamping" and "separating" sets as follows. Take any indices i that are trying to increase the relative velocity at that contact $\frac{\partial \ell}{\partial v_i} < 0$, which implies the optimizer is feebly trying to separate the objects, and put those indices into "separating" to remove the constraint that $v_i = 0$. Take any indices i where $\frac{\partial \ell}{\partial v_i} > 0$, which implies the optimizer is trying to move the objects closer together (which would violate contact constraints), and put them into "clamping" to impose the constraint that $v_i = 0$.

The contact strategy with the larger $||\frac{\partial \ell}{\partial \dot{\boldsymbol{q}}_t}||_2^2 + \frac{||\frac{\partial \ell}{\partial \boldsymbol{\tau}_t}||_2^2}{\Delta t}$ is used during backpropagation for learning. We call the gradient produced by this exploratory procedure a "Complementarity-aware Gradient," and find empirically that it can help avoid saddle points during trajectory optimization. We show an example of this in Section VII.

The complementarity-aware gradients do not guarantee to improve global convergence because the gradient is chosen for each contact point independently, which might not result in an aggregated gradient $\frac{\partial \ell}{\partial \boldsymbol{\tau}}$ that moves in a globally optimal direction. However, if an optimization problem currently gets stuck in a saddle point, complementarity-aware gradients provide another tool for practitioners to try.

V. Gradients through collision geometry

This section addresses efficient computation of $\frac{\partial \boldsymbol{J}_t^T \boldsymbol{f}}{\partial \boldsymbol{q}_t}$, the relationship between position and joint impulse. In theory, we could utilize auto-diff libraries for the derivative computation. However, using auto-diff and passing every gradient through a long kinematic chain of transformations is inefficient for complex articulated rigid body systems. In contrast, computing the gradients symbolically shortcuts much computation by operating directly in the world coordinate frame.

Let $\mathcal{A}_i \in se(3)$ be the screw axis for the i'th DOF, expressed in the world frame. Let the k'th contact point give an impulse $\mathcal{F}_k \in dse(3)$, also expressed in the world frame. Let $\psi_{i,k} \in \{-1, 0, 1\}$ be the relationship between contact k and joint i. $\psi_{i,k} = 1$ or $\psi_{i,k} = -1$ means contact k is on a child body of joint i. Otherwise, $\psi_{i,k} = 0$. The total joint impulse caused by contact impulses for the i'th joint is given by:

$$(\boldsymbol{J}_t^T \boldsymbol{f})_i = \sum_k \psi_{i,k} \mathcal{A}_i^T \mathcal{F}_k = \mathcal{A}_i^T \sum_k \psi_{i,k} \mathcal{F}_k. \tag{23}$$

Taking the derivative of Equation 23 gives

$$\frac{\partial (\boldsymbol{J}_t^T \boldsymbol{f})_i}{\partial \boldsymbol{q}_t} = \frac{\partial \mathcal{A}_i}{\partial \boldsymbol{q}_t}^T \sum_k \psi_{i,k} \mathcal{F}_k + \mathcal{A}_i^T \sum_k \psi_{i,k} \frac{\partial \mathcal{F}_k}{\partial \boldsymbol{q}_t}. \tag{24}$$

Evaluating $\frac{\partial \mathcal{A}_i}{\partial \boldsymbol{q}_t}$ is straightforward, but computing $\frac{\partial \mathcal{F}_k}{\partial q_t}$ requires understanding how the contact normal $\boldsymbol{n}_k \in \mathcal{R}^3$ and contact position $\boldsymbol{p}_k \in \mathcal{R}^3$ change with changes in $\boldsymbol{q}_t$.

Let $\mathcal{F}_k$ be a concatenation of torque and force in $dse(3)$. The derivative with respect to the current joint position $\boldsymbol{q}_t$ is:

$$\frac{\partial \mathcal{F}_k}{\partial \boldsymbol{q}_t} = \begin{bmatrix} \frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}_t} \times \boldsymbol{p}_k + \boldsymbol{n}_k \times \frac{\partial \boldsymbol{p}_k}{\partial \boldsymbol{q}_t} \\ \frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}_t} \end{bmatrix} \tag{25}$$

It turns out that computing $\frac{\partial \boldsymbol{n}_k}{\partial q_t}$ for curved primitives shape colliders (spheres and capsules) requires different treatment from meshes or polygonal primitives. To understand why, consider using a high polycount mesh to approximate a true sphere as shown in Figure 5.

On a mesh approximation of a sphere, infinitesimally moving the contact point $\boldsymbol{p}_k$ by ϵ (by perturbing $\boldsymbol{q}$ of the triangle) will

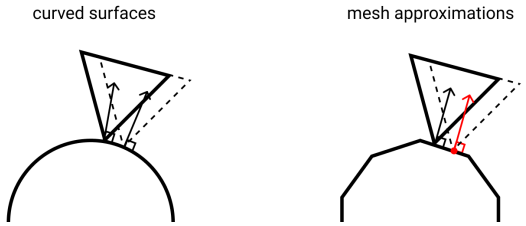

Fig. 5. $\frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}}$ can be set to zero for mesh or polygonal geometry, but has to be computed analytically for curved geometry.

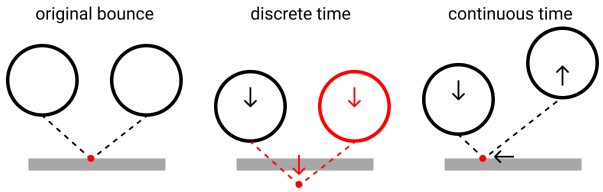

Fig. 6. Discrete-time Jacobians do not adequately describe the dynamics of an elastic collision. In the system pictured here, letting $\boldsymbol{q}$ be a scalar giving the distance from the ground, discrete-time would lead us to falsely believe that $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} = 1$. In continuous time, we would instead expect $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}} = -\sigma$, where σ is the coefficient of restitution. Modeling this requires understanding how the time of collision changes with initial conditions, and how that affects the final state after the step.

not cause the normal of the contact face $\boldsymbol{n}_k$ to change at all, because we remain on the same face of the mesh. So $\frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}}$ is always zero for a mesh or polygonal shape. However, on a true sphere, an infinitesimal perturbation of the contact point with the sphere (no matter how small) will cause the contact normal with the sphere $\boldsymbol{n}_k$ to change. The way the contact normal $\boldsymbol{n}_k$ changes with position (e.g. $\frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}}$) is often crucial information for the optimizer to have in order to solve complex problems, and mesh approximations to curved surfaces falsely set $\frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}} = 0$.

We refer the interested reader to Appendix C for a discussion of how we compute $\frac{\partial \boldsymbol{n}_k}{\partial \boldsymbol{q}_t}$ and $\frac{\partial \boldsymbol{p}_k}{\partial \boldsymbol{q}_t}$ for different combinations of collider types.

VI. Gradients through Elastic Contacts

DiffTaichi [18] pointed out an interesting problem that arises from discretization of time in simulating elastic bouncing phenomenon between two objects. The problems arise from the discrete time integration of position: $\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \Delta t \dot{\boldsymbol{q}}_t$. The Jacobians $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} = \boldsymbol{I}$ and $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t} = \Delta t \boldsymbol{I}$ are correct for most scenarios. However, when an elastic collision occurs, the discrete time integration scheme creates problems for differentiation. In a discrete time world, the closer the object is to the collision site at the beginning of the time step when collision happens, the closer to the collision site it ends up at the end of that time step. In continuous time, however, the closer the object begins to its collision site, the *further away it ends up*, because the object changes velocity sooner (Figure 6).

DiffTaichi [18] proposed using continuous-time collision detection and resolution during training, but noted that switching to discrete-time at test time did not harm performance. Since introducing full continuous-time collision detection is computationally costly for complex dynamic systems and scenes, we instead opt to find a $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ and $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$ that approximates the behavior of continuous-time Jacobians when elastic collisions occur.

Let $\boldsymbol{p}_{t,i}$ be the relative distance at contact i at time t and $\boldsymbol{v}_{t,i}$ be the relative velocity. We further define σ_i as the coefficient of restitution at contact i and $t + \boldsymbol{c}_i$ as the time of collision of contact i, where $\boldsymbol{c}_i \in \mathcal{R}$. Assuming the relative velocity is constant in this time step, we get $\boldsymbol{c}_i = -\boldsymbol{p}_i / \boldsymbol{v}_i$ and $\boldsymbol{p}_{t+1,i} = (\Delta t - \boldsymbol{c}_i)\sigma \boldsymbol{v}_{t,i}$. From there we have:

$$\frac{\partial \boldsymbol{p}_{i,t+1}}{\partial \boldsymbol{p}_{i,t}} = -\sigma_i, \quad \frac{\partial \boldsymbol{p}_{i,t+1}}{\partial \boldsymbol{v}_{i,t}} = -\sigma_i \Delta t. \tag{26}$$

TABLE II
Benchmarks against Finite Differencing

Environment	Analytical	Central Differences	Speedup
Atlas	8.53ms	749ms	87.84x
Half Cheetah	0.395ms	8.64ms	21.86x
Jump-Worm	0.26ms	3.82ms	14.8x
Catapult	0.27ms	4.36ms	16.4x

After finding the above gradients for each collision independently, we need to find a pair of Jacobians, $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ and $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$, that approximate our desired behavior at each of the contact points as closely as possible.

$$\begin{aligned} \frac{\partial \boldsymbol{p}_{i,t+1}}{\partial \boldsymbol{p}_{i,t}} &= \frac{\partial \boldsymbol{p}_{i,t+1}}{\partial \boldsymbol{q}_{t+1}} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \frac{\partial \boldsymbol{q}_t}{\partial \boldsymbol{p}_{i,t}} \\ &= \boldsymbol{J}_{i,t+1} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \boldsymbol{J}_{i,t}^{-1}, \text{ for } i = 1 \cdots m \end{aligned} \tag{27}$$

where $\boldsymbol{J}_{i,t+1}$ is the Jacobian matrix of contact i, $\boldsymbol{J}_{i,t}^{-1}$ is the pseudo inverse Jacobian, and m is the total number of contact points. Our goal is to find a Jacobian $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ that satisfies the above equations (Equation 27) as closely as possible (details in Appendix E). Once we find a satisfactory approximation for $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$, we can get

$$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t} = \Delta t \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}. \tag{28}$$

VII. Evaluation

First, we evaluate our methods by testing the performance of gradient computation. In addition, we compare gradient-based trajectory optimization enabled by our method to gradient-free stochastic optimization, and discuss implications. We also demonstrate the effectiveness of the complementarity-aware

gradients in a trajectory optimization problem. Finally, we show that our physics engine can solve optimal control problems for complex dynamic systems with contact and collision, including an Atlas humanoid jumping in the air.

A. Performance

Controlled comparison to existing methods can be challenging because they use different formulations for forward simulation, essentially simulating different physical phenomena. We therefore benchmark the performance of our method at the atomic level–measuring the computation time of a Jacobian for a single time step using a single-core CPU and comparing it to finite differencing methods, using the same forward simulation process provided by an existing non-differentiable physics engine (DART). This comparison removes factors due to differences in forward simulation, implementation techniques, and computation resources, and focuses on the speed gain solely contributed by our gradient computation method.

Table II contains abbreviated benchmark performance of our Jacobians against central differencing. We evaluate our results in four environments: the Atlas robot on the ground (33 DOFs, 12 contact points), Half Cheetah on the ground (9 DOFs, 2 contact points), Jump-Worm (5 DOFs, 2 contact points), and Catapult (5 DOFs, 2 contact points). For each environment, we compare the speed of evaluating all five primary Jacobians. For a complete table, including the speed of individual Jacobian evaluations, see Appendix D.

B. Gradient-based vs gradient-free trajectory optimization

To demonstrate the benefit of analytical Jacobians of dynamics, we compare trajectory optimization on our catapult trajectory problem between Multiple Shooting and Stochastic Search (SS) [5], as well as cartpole and the double-pendulum using both Differential Dynamic Programming (DDP) [39, 22, 40] and Stochastic Search [5]. The gradient-based methods (DDP and Multiple Shooting) are able to converge much more quickly, because the additional convergence gained from analytical Jacobians more than offsets the time to compute them. See Figure VII-B.

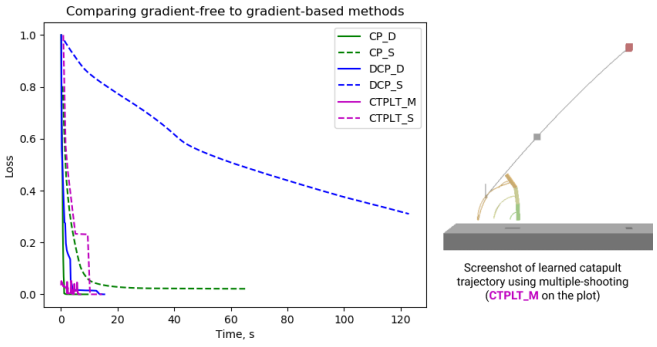

Fig. 7. Comparing wall clock time to find a single-pendulum cartpole (CP), double-pendulum cartpole (DCP), and catapult (CTPLT) trajectory, using DDP (D), SS (S), and Multiple Shooting (M). The results are unsurprising: gradient information speeds convergence tremendously.

C. Complementarity-aware gradients

To highlight the complementarity-aware gradients, we solve a trajectory optimization problem of a drone taking-off from the ground and reaching a fixed height in 500 timesteps.

Because the drone is initialized resting on the ground, it has a contact with the ground that is classified as "clamping." When we attempt to optimize the control on the drone using correct gradients, we get zero gradients and make no progress. By contrast, when we use our *complementarity aware gradient*, while we still see no change in loss for the first few iterations of SGD, we're able to get non-zero gradients and escape the saddle point (Figure 7). See the supplementary video for the resulting drone trajectories.

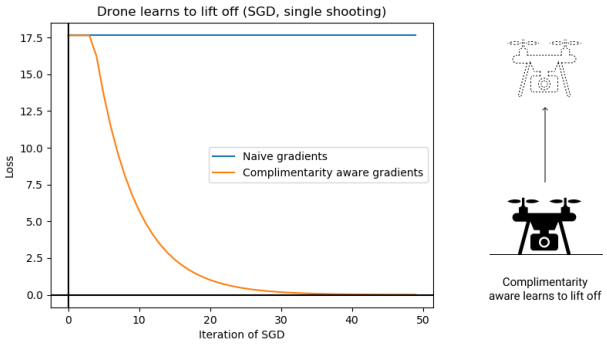

Fig. 8. Training a drone to lift off the ground and fly to a target height after 500 timesteps. Loss is the squared distance of the drone from the target at $t = 500$. The drone is initialized resting on the ground, which means the drone-ground contacts are classified as "clamping." That means Jacobians will show $\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\tau}_t} = 0$. While both standard and complementarity-aware training runs start in a saddle point, the complementarity-aware gradients are able to guide SGD to escape after several iterations of learning.

D. Optimal control with contact

We present several trajectory optimization problems that involve complex contact dynamics, optimized using multiple shooting. We demonstrate "Catapult", which is a 3-dof robot that is tasked with batting a free ball towards a target, in such a way that it exactly hits the target at the desired timestep (pictured in Figure 7). We also demonstrate "Jump-Worm", which is a 5-dof worm-shaped robot that is attempting to jump as high as possible at the end of the trajectory. Both of these problems involve complex contact switching throughout the course of the trajectory. We also optimize a trajectory where the "Atlas" robot learns to jump. See the supplementary video for the resulting trajectories.

VIII. Conclusions

We present a fast and feature-complete differentiable physics engine for articulated rigid body simulation. We introduce a method to compute analytical gradients through the LCP formulation of inelastic contact by exploiting the sparsity of the LCP solution. Our engine supports complex contact geometry and approximating continuous-time elastic collision. We also

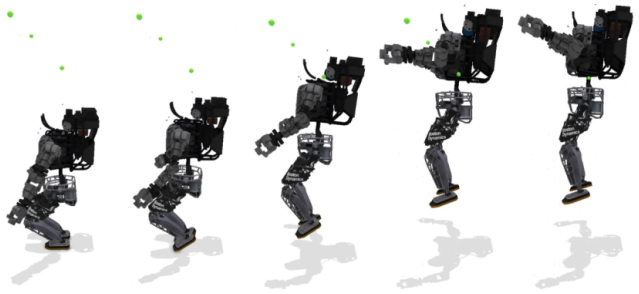

Fig. 9. A few snapshots of the trajectory that an Atlas robot learns when it is asked to jump up towards the green markers, starting from a crouch.

introduce a novel method to compute complementarity-aware gradients that help optimizers to avoid stalling in saddle points.

There a few limitations of our current method. Currently, computing $\frac{\partial \boldsymbol{J}_t \boldsymbol{f}}{\partial \boldsymbol{q}_t}$, the way the joint torques produced by the contact forces change as we vary joint positions (which in turn varies contact positions and normals) takes a large portion of the total time to compute all Jacobians of dynamics for a given timestep. Perhaps a more efficient formulation can be found.

Our engine also doesn't yet differentiate through the geometric properties (like link length) of a robot, or friction coefficients. Extending to these parameters is an important step to enable parameter estimation applications.

We are excited about future work that integrates gradients *and* stochastic methods to solve hard optimization problems in robotics. In running the experiments for this paper, we found that stochastic trajectory optimization methods could often find better solutions to complex problems than gradient-based methods, because they were able to escape from local optima. However, stochastic methods are notoriously sample inefficient, and have trouble fine-tuning results as they begin to approach a local optima. The authors speculate that there are many useful undiscovered techniques waiting to be invented that lie at the intersection of stochastic gradient-free methods and iterative gradient-based methods. It is our hope that an engine like the one presented in this paper will enable such research.

Acknowledgments

The authors of this paper received funding from NSF-1748067, Stanford HAI, and the Hoffman-Yee research grants. We would also like to thank Steve Collins for several excellent ideas, including renaming the engine. Also, thanks to Amy Bearman and Dylan Shell for invaluable editing.

References

[1] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv e-prints*, pages arXiv–1605, 2016.

[2] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017.

[3] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 23–34, 1994.

[4] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[5] George I Boutselis, Ziyi Wang, and Evangelos A Theodorou. Constrained sampling-based trajectory optimization using stochastic approximation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2522–2528. IEEE, 2020.

[6] Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. *lecture notes of EE392o, Stanford University, Autumn Quarter*, 2004:2004–2005, 2003.

[7] Justin Carpentier and Nicolas Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and Systems (RSS 2018)*, 2018.

[8] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[9] Erwin Coumans. Bullet physics engine. *Open Source Software: http://bulletphysics. org*, 1(3):84, 2010.

[10] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.

[11] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019.

[12] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[13] David Hahn, Pol Banzet, James M. Bern, and Stelian Coros. Real2sim: Visco-elastic parameter estimation from dynamic motion. *ACM Trans. Graph.*, 38(6), November 2019.

[14] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019.

[15] Eric Heiden, David Millard, Erwin Coumans, and Gaurav S Sukhatme. Augmenting differentiable simulators with neural networks. *arXiv preprint arXiv:2007.06045*, 2020.

[16] Michiel Hermans, Benjamin Schrauwen, Peter Bienstman,

and Joni Dambre. Automated design of complex dynamic systems. *PLOS ONE*, 9(1):1–11, 01 2014.

[17] Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2020.

[18] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[19] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6): 1–16, 2019.

[20] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[21] A. Iollo, M. Ferlauto, and L. Zannetti. An aerodynamic optimization method based on the inverse problem adjoint equations. *Journal of Computational Physics*, 173(1):87–115, 2001.

[22] David H Jacobson and David Q Mayne. Differential dynamic programming. 1970.

[23] Y. Jarny, M.N. Ozisik, and J.P. Bardon. A general optimization method using adjoint equation for solving multidimensional inverse heat conduction. *International Journal of Heat and Mass Transfer*, 34(11):2911–2919, 1991.

[24] Franck Jourdan, Pierre Alart, and Michel Jean. A gauss-seidel like algorithm to solve frictional contact problems. *Computer Methods in Applied Mechanics and Engineering*, 155(1):31–47, 1998.

[25] Junggon Kim. Lie group formulation of articulated rigid body dynamics. Technical report, Technical Report. Carnegie Mellon University, 2012.

[26] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3 (22):500, 2018.

[27] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[28] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, August 2004.

[29] David Millard, Eric Heiden, Shubham Agrawal, and Gaurav S. Sukhatme. Automatic differentiation and continuous sensitivity analysis of rigid body dynamics, 2020.

[30] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in neural information processing systems*, pages 8799–8810, 2018.

[31] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[32] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 209–217. ACM Press/Addison-Wesley Publishing Co., 2000.

[33] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007.

[34] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.

[35] Christopher Rackauckas, Yingbo Ma, Vaibhav Dixit, Xingjian Guo, Mike Innes, Jarrett Revels, Joakim Nyberg, and Vijay Ivaturi. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions, 2018.

[36] C.J.F. Ridders. Accurate computation of f'(x) and f'(x) f"(x). *Advances in Engineering Software (1978)*, 4(2): 75–76, 1982.

[37] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 317–335. PMLR, 29–31 Oct 2018.

[38] Changkyu Song and Abdeslam Boularias. Learning to slide unknown objects with differentiable physics simulations. In *Robotics: Science and Systems*, Oregon State University at Corvallis, Oregon, USA, 14–16 July 2020.

[39] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[40] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[42] Marc Toussaint, Kelsey R. Allen, K. Smith, and J. Tenenbaum. Differentiable physics and stable modes for tool-

use and manipulation planning. In *Robotics: Science and Systems*, 2018.

APPENDIX

A. Frictional impulse

In DART, friction impulses are solved by the boxed LCP method, using the same implementation of the boxed variant of the Dantzig algorithm found in the Open Dynamics Engine, and originally proposed in [3]. Boxed LCPs are not theoretically guaranteed to be solvable, but are quite common in practice because of their speed and high-quality results. We therefore extend our formulation in Section IV to compute the gradient of frictional impulse magnitudes found in a boxed LCP solver with respect to $\boldsymbol{A}$ and $\boldsymbol{b}$. Similar to normal impulses, each frictional impulse is classified into one of the two states:

a) Clamping ($\mathcal{C}$): If the relative velocity along the frictional impulse direction is zero, and friction impulse magnitude is below its bound, then any attempt to push this contact will be met with an increase in frictional impulse holding the point in place. This means the contact point is clamping, which behaves will be treated in the same way as clamped normal forces.

b) Bounded ($\mathcal{B}$): If the frictional impulse magnitude is at its bound (either positive or negative) then the contact point is sliding or about to slide along this friction direction. Bounded frictional impulse $\mathcal{B}$ is quite like "Separating" $\mathcal{S}$ in normal impulse. The difference is that frictional impulses in $\mathcal{B}$ are not zero but at a non-zero bound based on the corresponding normal impulses.

Bounded frictional impulse can be expressed by $\boldsymbol{f}^*_{\mathcal{B}} = \boldsymbol{E}\boldsymbol{f}^*_{\mathcal{C}}$, where each row of $\boldsymbol{E} \in \mathcal{R}^{|\mathcal{B}|\times|\mathcal{C}|}$ contains a single non-zero value with the friction coefficient, μ_s, for the corresponding normal impulse index in $\mathcal{C}$.

We define $\boldsymbol{J}_{\mathcal{C}}$ to be the matrix with just the columns of $\boldsymbol{J}$ corresponding to the indices in $\mathcal{C}$. Likewise, we define $\boldsymbol{J}_{\mathcal{B}}$ containing just the columns of $\boldsymbol{J}$ that are bounded. If we multiply $\boldsymbol{J}^T_{\mathcal{C}}\boldsymbol{f}_{\mathcal{C}}$ we get the joint torques due to the clamping constraint forces. Similarly, if we multiply $\boldsymbol{J}^T_{\mathcal{B}}\boldsymbol{f}_{\mathcal{B}}$ we get the joint torques due to bounded friction impulses. Since $\boldsymbol{f}_{\mathcal{B}} = \boldsymbol{E}\boldsymbol{f}_{\mathcal{C}}$, we can modify $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ to take bounded frictional impulses into account:

$$\boldsymbol{A}_{\mathcal{C}\mathcal{C}} = \boldsymbol{J}_{\mathcal{C}}\boldsymbol{M}^{-1}(\boldsymbol{J}^T_{\mathcal{C}} + \boldsymbol{J}^T_{\mathcal{B}}\boldsymbol{E})$$

With this one small change, all the formulation in Section IV works with frictional forces. An interesting observation is that the bounded frictional cases are analogous to the separating cases for the normal forces, in that the force value is constrained at $\mu_s f_i$ (or $-\mu_s f_i$), where f_i is the corresponding normal force for the same contact. The only way the bounded force values will change is through the change of corresponding clamping normal force, which needs to be accounted for when computing the gradients.

Just like the overall boxed LCP problem is not solvable, $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is no longer guaranteed to be exactly invertible. In order to support this, we need to use the pseudoinverse of $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ during the forward pass, and the gradient of the pseudoinverse when computing Jacobians.

B. LCP stabilization

When $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is not full rank, the solution to $f_{\text{LCP}}(\boldsymbol{A}, \boldsymbol{b}) = \boldsymbol{f}^*$ is no longer unique. To grasp this intuitively, consider a 2D case where a box of unit mass that cannot rotate is resting on a plane. The box has two contact points, with identical contact normals, and because the box is not allowed to rotate, the effect of an impulse at each contact point is exactly the same (it causes the box's upward velocity to increase). This means that both columns of $\boldsymbol{A}$ (one per contact) are identical. That means that $\boldsymbol{A}_{\mathcal{C}\mathcal{C}} \in \mathcal{R}^{2\times 2}$ is actually only rank one. Let's assume we need a total upward impulse of $-mg$ to prevent the box from interpenetrating the floor. Because $\boldsymbol{A}_{\mathcal{C}\mathcal{C}}$ is a low rank, we're left with one equation and two unknowns:

$$\boldsymbol{A}_{\mathcal{C}\mathcal{C}}\boldsymbol{f}^*_{\mathcal{C}} = \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}f^*_{\mathcal{C}_1}\\ f^*_{\mathcal{C}_2}\end{bmatrix} = \begin{bmatrix}f^*_{\mathcal{C}_1} + f^*_{\mathcal{C}_2}\\ f^*_{\mathcal{C}_1} + f^*_{\mathcal{C}_2}\end{bmatrix} = \begin{bmatrix}-mg\\ -mg\end{bmatrix}$$

It is easy to see that this is just $f^*_{\mathcal{C}_1} + f^*_{\mathcal{C}_2} = -mg$. And that means that we have an infinite number of valid solutions to the LCP.

In order to have valid gradients, our LCP needs to have predictable behavior when faced with multiple valid solutions. Thankfully, our analysis in the previous sections suggests a quite simple and efficient (and to the authors' knowledge novel) LCP stabilization method. Once an initial solution is computed using any algorithm, and the clamping set $\mathcal{C}$ is found, we can produce a least-squares-minimal (and numerically exact) solution to the LCP by setting:

$$\boldsymbol{f}^*_{\mathcal{C}} = \boldsymbol{A}^{-1}_{\mathcal{C}\mathcal{C}}\boldsymbol{b}_{\mathcal{C}}, \qquad \boldsymbol{f}^*_{\mathcal{S}} = 0$$

This is possible with a single matrix inversion because the hard part of the LCP problem (determining which indices belong in which classes) was already solved for us by the main solver. Once we know which indices belong in which classes, solving the LCP exactly reduces to simple linear algebra.

As an interesting aside, this "LCP stabilization" method doubles as an extremely efficient LCP solver for iterative LCP problems. In practical physics engines, most contact points do not change from clamping ($\mathcal{C}$) to separating ($\mathcal{S}$) or back again on most time steps. With that intuition in mind, we can opportunistically attempt to solve a new $f_{\text{LCP}}(\boldsymbol{A}_{t+1}, \boldsymbol{b}_{t+1}) = \boldsymbol{f}^*_{t+1}$ at a new timestep by simply guessing that the contacts will sort into $\mathcal{C}$ and $\mathcal{S}$ in exactly the same way they did on the last time step. Then we can solve our stabilization equations for $\boldsymbol{f}^*_{t+1}$ as follows:

$$\boldsymbol{f}_{t+1}{}^*_{\mathcal{C}} = \boldsymbol{A}^{-1}_{t+1\mathcal{C}\mathcal{C}}\boldsymbol{b}_{t+1\mathcal{C}}, \qquad \boldsymbol{f}_{t+1}{}^*_{\mathcal{S}} = 0$$

If we guessed correctly, which we can verify in negligible time, then $\boldsymbol{f}^*_{t+1}$ is a valid, stable, *and perfectly numerically exact* solution to the LCP. When that happens, and in our experiments this heuristic is right $> 95\%$ of the time, we can skip the expensive call to our LCP solver entirely. As an added

TABLE III
FULL BENCHMARKS AGAINST FINITE DIFFERENCING

ENVIRONMENT	JACOBIAN	ANALYTICAL TIME	CENTRAL DIFFERENCES TIME	SPEEDUP	RIDDERS TIME	SPEEDUP
ATLAS	ALL	8.53MS	749MS	**87.84X**	2229MS	261X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$	0.0204MS	160MS	7850X	581MS	28480X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{q}_t}$	6.75MS	161MS	23.8X	581MS	86X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.016MS	107.2MS	6570.9X	304MS	19000X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.778MS	160MS	205.7X	487MS	626X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\tau}_t}$	0.981MS	161MS	163.9X	524MS	471X
HALF CHEETAH	ALL	0.395MS	8.64MS	**21.86X**	24MS	60X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$	0.0059MS	1.69MS	284X	3.17MS	537X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{q}_t}$	0.254MS	1.874MS	7.35X	7.92MS	31.1X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.00358MS	1.228MS	342X	2.42MS	675.9X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.053MS	1.98MS	37.1X	5.63MS	106.2X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\tau}_t}$	0.077MS	1.87MS	24.1X	5.75MS	74.6X
JUMP-WORM	ALL	0.256MS	3.82MS	**14.89X**	9.727MS	37.9X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$	0.00433MS	0.732MS	168.8X	1.58MS	364.8X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{q}_t}$	0.174MS	0.836MS	4.78X	2.84MS	16.32X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.00307MS	0.532MS	173.1X	0.81MS	263.8X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.0343MS	0.872MS	25.4X	2.17MS	63.2X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\tau}_t}$	0.040MS	0.851MS	21.2X	2.31MS	57.75X
CATAPULT	ALL	0.265MS	4.36MS	**16.45X**	12.0MS	45.2X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$	0.0050MS	0.845MS	167.8X	1.79MS	358X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{q}_t}$	0.181MS	0.972MS	5.36X	3.71MS	20.49X
	$\frac{\partial \boldsymbol{q}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.0036MS	0.574MS	156.7X	0.958MS	266X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \dot{\boldsymbol{q}}_t}$	0.035MS	1.01MS	28.42X	2.72MS	77.7X
	$\frac{\partial \dot{\boldsymbol{q}}_{t+1}}{\partial \boldsymbol{\tau}_t}$	0.039MS	0.960MS	24.3X	2.85MS	73.1X

bonus, because $\mathcal{C}$ is usually not all indices, inverting $\boldsymbol{A}_{t+1\mathcal{C}\mathcal{C}}$ can be considerably cheaper than inverting all of $\boldsymbol{A}_{t+1}$, which can be necessary in an algorithm to solve the full LCP.

When our heuristic does not result in a valid $\boldsymbol{f}^*_{t+1}$, we can simply return to our ordinary LCP solver to get a valid set $\mathcal{C}$ and $\mathcal{S}$, and then re-run our stabilization.

As long as results from our LCP are stabilized, the gradients through the LCP presented in this section are valid even when $\boldsymbol{A}$ is not full rank.

C. Contact point and normal stabilization and gradients

To compute gradients through each contact point and normal with respect to $\boldsymbol{q}_t$, we need to provide specialized routines for each type of collision geometry, including collisions between spheres, capsules, boxes, and arbitrary convex meshes. Deriving the routines is a matter of straightforward algebra. However in order for well-defined gradients to exist at all, each collision must have precisely specified contact points and normals, so that we can compute well-behaved gradients.

We describe in this appendix how we produce well-defined contact points and normals, since this is a design choice. The gradients of these methods are left as an exercise to the reader, who is invited to check their work against our open-source implementation.

1) Mesh-mesh collisions: Mesh-mesh collisions only have two types of contacts: vertex-face and edge-edge collisions. The other cases (vertex-edge, vertex-vertex, edge-face, face-face) are degenerate and easily mapped into vertex-face and edge-edge collisions.

Mesh-mesh collision detection algorithms like Gilbert-Johnson-Keerthi or Minkowski Portal Refinement are iterative, and so produce imprecise collision points and normals that can vary depending on initialization. Both algorithms produce a separating "witness plane" as part of their output, which is a 3D plane that approximately separates the two meshes (as much as possible, if they're intersecting). Going from an approximate separating witness plane to precisely specified vertex-face and edge-edge collisions in the general case is complex. Take all the points on object A that lie within some tiny ϵ of the witness plane, and map them into 2D coordinates within the witness plane. Do likewise with the points on object B. Now we have two convex 2D shapes, because any linear slice of a convex shape is itself convex. Throw out any vertices that are not on the 2D convex hull of the point cloud for A and

B respectively. Call the resulting convex shapes "witness hulls" of A and B. Now any vertices on the witness hull of A that lie within the witness hull of B are vertex-face collisions from A to B. Analogously, any vertices on the witness hull of B that lie within the witness hull of A are face-vertex collisions from A to B. Finally, any edges of the witness hulls A and B that intersect are edge-edge collisions.

For all vertex-face collisions, during the forward simulation, the collision detector places a collision at the point of the vertex, with a normal dictated by the face under collision. The body providing the vertex can only influence the collision location $\boldsymbol{p}$, and the body providing the face can only influence the collision normal $\boldsymbol{n}$.

For all edge-edge collisions, the collision detector first finds the nearest point on edge A to edge B (call that $\boldsymbol{a}$), and the nearest point on edge B to edge A (call that $\boldsymbol{b}$). Then the contact point is set to the average of $\boldsymbol{a}$ and $\boldsymbol{b}$, which is $\frac{\boldsymbol{a}+\boldsymbol{b}}{2}$. The normal is given by the cross product of the two edges. That means changing $\boldsymbol{q}$ of Object A can affect the contact normal and the contact location along the other edge from Object B. For this reason, we need to construct our Jacobians globally.

2) Sphere-sphere collisions: These are straightforward. We have sphere A, with center c_a and radius r_a, and sphere B, with center c_b and radius r_b. The contact normal is the normalized vector pointing from c_b to c_a. The contact point is a weighted combination of the two sphere centers, $\frac{r_b * c_a + r_a * c_b}{r_a + r_b}$.

3) Mesh-sphere collisions: These can be divided into three categories: sphere-face collisions, sphere-edge collisions, and sphere-vertex collisions. Starting from the simplest, a sphere-vertex collision places the contact point at the vertex and the normal points from the vertex to the sphere center.

A sphere-edge collision places the contact point on the closest point to the sphere center along the edge. The contact normal points from the collision point to the sphere center.

A sphere-face collision gets the contact normal from the normal of the colliding face. Call the contact normal $\boldsymbol{n}$, and define the sphere center as $\boldsymbol{c}$ and radius as $\boldsymbol{r}$. For the simplicity of downstream gradient computation, we then place the contact point at the point on the sphere, projected along with the contact normal towards the contact: $\boldsymbol{c} + \boldsymbol{r} * \boldsymbol{n}$.

4) Pipe-pipe collisions: In a capsule-capsule collision, when both capsule cylinders are colliding, we call that a pipe-pipe collision. Mechanically, this is very similar to an edge-edge collision, only with added radius variables r_a and r_b. Let c_a be the nearest point on the centerline of pipe A to the centerline of pipe B. Let c_b be the nearest point on the centerline of pipe B to the centerline of pipe A. Then we say the contact point is $\frac{r_b * c_a + r_a * c_b}{r_a + r_b}$. The contact normal points from c_b to c_a.

5) Pipe-sphere collisions: These look a lot like pipe-pipe collisions, except that c_b is now fixed to the center of the sphere. Let c_a be the nearest point on the centerline of the pipe to c_b. Then we say the contact point is $\frac{r_b * c_a + r_a * c_b}{r_a + r_b}$. The contact normal points from c_b to c_a.

6) Pipe-mesh collisions: This breaks down into two types of contact: vertex-pipe, and edge-pipe. Face-pipe contacts reduce to two edge-pipe contacts.

For vertex-pipe contacts, we put the contact point at the vertex, and point the normal towards the nearest point on the centerline of the pipe.

For edge-pipe contacts, we treat them as pipe-pipe contacts where the radius of the first pipe is 0.

7) Gradients: Once all the contact behavior is firmly established, it's just a matter of rote calculus to compute derivatives. We refer the reader to our open source code for the implementation of those derivatives. Once a stable collision detection system and its derivatives are implemented that, it's possible to efficiently compute $\frac{\partial J_t^T f}{\partial q_t}$.

D. Jacobian Benchmark Evaluation

In addition to speed, we are interested in the accuracy of our Jacobians, as gradients computed via finite differencing become more inaccurate as the system becomes more complex, which can lead to instability in optimization. As another baseline, we apply Ridders' method [36] to efficiently calculate Jacobians to a higher-order error than central differencing, using the stopping criterion given in [33].

Table III contains the full benchmark performance of our analytical Jacobians against central differencing and the accurate Ridders' extrapolated finite differences. For each environment, we compare the speed of evaluation of each individual component Jacobian as well as the total time.

E. Calculating the Bounce Approximation Jacobian

Recall the matrix $\boldsymbol{J}^T$ that transforms contact forces to joint forces. By conservation of momentum $\boldsymbol{J}\dot{\boldsymbol{q}}_t = \boldsymbol{v}_t$, leading to:

$$\frac{\partial \boldsymbol{p}_{t+1}}{\partial \boldsymbol{p}_t} \approx \boldsymbol{J} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \boldsymbol{J}^{-1} \tag{29}$$

Our matrix notation is somewhat misleading here, because we do not want our approximation to capture off-diagonals of $\frac{\partial \boldsymbol{p}_{t+1}}{\partial \boldsymbol{p}_t}$. Because we construct our approximate $\frac{\partial \boldsymbol{p}_{t+1}}{\partial \boldsymbol{p}_t}$ by assuming each bounce is independent, we end up with 0s on the off-diagonals, but we know that assumption to be false. We just want to find a $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ where the above equation matches the diagonals as closely as possible, ignoring other elements. The following derives this relation in closed form.

Since we are only interested in enforcing the diagonal entries, so we can write out a series of linear constraints we would like to attempt to satisfy. Let $\boldsymbol{J}_{(i)}$ denote the i'th column of $\boldsymbol{J}$. Recall that $\frac{\partial \boldsymbol{p}_{i,t+1}}{\partial \boldsymbol{p}_{i,t}} = -\sigma_i$.

$$\boldsymbol{J}_{(i)} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \boldsymbol{J}_{(i)}^{-1} \approx -\sigma_i \tag{30}$$

We would like to find some approximate matrix $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ that satisfies all n of the above constraints as closely as possible. Stated directly as a least-squares optimization object:

$$\min \sum_i \left(\boldsymbol{J}_{(i)} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \boldsymbol{J}_{(i)}^{-1} - (-\sigma_i) \right)^2 \tag{31}$$

The solution to this optimization can be found without iterative methods.

Let $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}_{(i)}$ be the i'th column of $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$, and $\boldsymbol{J}_{(j,i)}$ be the j'th row and the i'th column of $\boldsymbol{J}$. Note that:

$$\boldsymbol{J}_{(i)} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \boldsymbol{J}_{(i)}^{-1} = \sum_j \underbrace{\boldsymbol{J}_{(j,i)}}_{\text{scalar}} (\boldsymbol{J}_{(i)}^{-T} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}_{(j)}) \tag{32}$$

$$\underbrace{\boldsymbol{J}_{(j,i)}}_{\text{scalar}} (\boldsymbol{J}_{(i)}^{-T} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}_{(j)}) = \sum_j \underbrace{(\boldsymbol{J}_{(j,i)} \boldsymbol{J}_{(i)}^{-T})}_{\text{vector}}^T \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}_{(j)}) \tag{33}$$

It becomes clear that we could construct a long vector $\boldsymbol{v}$, which will map to every column of $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ placed end to end. We can also construct a matrix $\boldsymbol{W}$ where every column $\boldsymbol{W}_{(i)}$ is the vectors $\boldsymbol{J}_{(j,i)} \boldsymbol{J}_{(i)}^{-1}$ placed end to end (iterating over j). Now if we take the values of σ_i as entries of a vector $\boldsymbol{r} \in \mathcal{R}^n$, we can write our optimization problem as a linear equation:

$$\min \sum_i \left(\boldsymbol{J}_{(i)} \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \boldsymbol{J}_{(i)}^{-1} - (-\sigma_i) \right)^2 = \min ||\boldsymbol{W}^T \boldsymbol{v} + \boldsymbol{r}||_2^2 \tag{34}$$

This is a standard least squares problem, and is solved when:

$$\boldsymbol{v} = -\boldsymbol{W}^{T\dagger} \boldsymbol{r} \tag{35}$$

Once we have a value of $\boldsymbol{v}$, we can reconstruct the original matrix $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ by taking each column of $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ the appropriate segment of $\boldsymbol{v}$.

This is almost always an under-determined system, and we want to default to having $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t}$ as close to I as possible, rather than as close to 0 as possible. We can slightly reformulate our optimization problem where the least square objective tries to keep the diagonals at 1, rather than 0. If we define an arbitrary $\boldsymbol{c}$ vector (for "center"), we can use the identity:

$$\boldsymbol{W}^T (\boldsymbol{v} - \boldsymbol{c}) = -\boldsymbol{r} - \boldsymbol{W}^T \boldsymbol{c} \tag{36}$$

$$\boldsymbol{v} = \boldsymbol{c} - \boldsymbol{W}^{T\dagger} (\boldsymbol{r} + \boldsymbol{W}^T \boldsymbol{c}) \tag{37}$$

If we set $\boldsymbol{c}$ to the mapping for $\frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} = I$, then we get a solution that minimizes the distance to the identity while satisfying the constraints, measured as the sum of the squares of all the terms.

Also, remember that to get our Jacobian with respect to velocity, we simply:

$$\frac{d\theta_{t+1}}{d\dot{\theta}_t} = \Delta t \frac{\partial \boldsymbol{q}_{t+1}}{\partial \boldsymbol{q}_t} \tag{38}$$

And that should approximately solve the "gradient bounce" problem. On timesteps where there is only a single bouncing contact, this will provide an exact solution. With more than one bounce in a single frame, this may be approximate.

F. Analytical derivatives through Featherstone

We present the derivation of analytical derivatives computation through the Featherstone algorithm. Although the intellectual computation should be credited entirely to [25] and [7], we specialized the derivations to obtain $\frac{\partial \boldsymbol{M}_t^{-1} \boldsymbol{z}_t}{\partial \boldsymbol{q}_t}$, $\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{q}_t}$, and $\frac{\partial \boldsymbol{c}_t}{\partial \dot{\boldsymbol{q}}_t}$ for our implementation, rather than the entire inverse and forward dynamics. The detailed derivation might be of interest to some readers.

The partial derivative of the inverse of joint space inertia matrix can be computed from the partial derivative of the joint space inertia matrix through the relation [7]:

$$\frac{\partial \boldsymbol{M}^{-1} \boldsymbol{z}}{\partial \boldsymbol{q}} = \boldsymbol{M}^{-1} \frac{\partial \boldsymbol{M}}{\partial \boldsymbol{q}} \boldsymbol{M}^{-1} \boldsymbol{z}. \tag{39}$$

Algorithm 1 and 2 show the recursive algorithms to compute $\boldsymbol{M}^{-1}$ and $\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{q}} \boldsymbol{M}^{-1} \boldsymbol{z}$, respectively. In Algorithm 3, the derivatives of the Coriolis force with respect to the joint position and the joint velocity are given.

Algorithm 1 Recursive forward dynamics for $\boldsymbol{M}^{-1}$

for $j = 1$ to n **do**
  for $i = n$ to 1 **do**
    $\hat{\boldsymbol{\mathcal{I}}}_i = \boldsymbol{\mathcal{I}}_i + \sum_{l \in \mu(i)} \text{Ad}^*_{\boldsymbol{T}_{i,l}^{-1}} \boldsymbol{\Pi}_l \text{Ad}_{\boldsymbol{T}_{i,l}^{-1}}$
    $\hat{\boldsymbol{\mathcal{B}}}_i = \sum_{l \in \mu(i)} \text{Ad}^*_{\boldsymbol{T}_{i,l}^{-1}} \boldsymbol{\beta}_l$
    $\boldsymbol{\Psi}_i = \left(\boldsymbol{S}_i^T \hat{\boldsymbol{\mathcal{I}}}_i \boldsymbol{S}_i \right)^{-1}$
    $\boldsymbol{\Pi}_i = \hat{\boldsymbol{\mathcal{I}}}_i - \hat{\boldsymbol{\mathcal{I}}}_i \boldsymbol{S}_i \Psi_i \boldsymbol{S}_i^T \hat{\boldsymbol{\mathcal{I}}}_i$
    $\boldsymbol{\alpha}_i = \delta_{i,j} - \boldsymbol{S}_i^T \hat{\boldsymbol{\mathcal{B}}}_i$, where $\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$
    $\boldsymbol{\beta}_i = \hat{\boldsymbol{\mathcal{B}}}_i + \hat{\boldsymbol{\mathcal{I}}}_i \boldsymbol{S}_i \boldsymbol{\Psi}_i \boldsymbol{\alpha}_i$
  end for
  for $i = 1$ to n **do**
    $\left[\boldsymbol{M}^{-1} \right]_{i,j} = \boldsymbol{\Psi}_i \left(\boldsymbol{\alpha}_i - \boldsymbol{S}_i^T \hat{\boldsymbol{\mathcal{I}}}_i \text{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \dot{\boldsymbol{V}}_{\lambda(i)} \right)$
    $\dot{\boldsymbol{V}}_i = \text{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \dot{\boldsymbol{V}}_{\lambda(i)} + \boldsymbol{S}_i \left[\boldsymbol{M}^{-1} \right]_{i,j}$
  end for
end for

Algorithm 2 Derivative of the recursive inverse dynamics for $\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{z}$

for $i = 1$ to n **do**
  $\dot{\boldsymbol{V}}_i = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \dot{\boldsymbol{V}}_{\lambda(i)} + \boldsymbol{S}_i \left[\boldsymbol{M}^{-1}\boldsymbol{z}\right]_i$
  for $j = 1$ to n **do**
    $\frac{\partial \dot{\boldsymbol{V}}_i}{\partial \boldsymbol{q}_j} = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \frac{\partial \dot{\boldsymbol{V}}_{\lambda(i)}}{\partial \boldsymbol{q}_j} - \mathrm{ad}_{\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{q}_j}} \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \dot{\boldsymbol{V}}_{\lambda(i)}$
      $+ \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_j} \left[\boldsymbol{M}^{-1}\boldsymbol{z}\right]_i$
  end for
end for
for $i = n$ to 1 **do**
  $\boldsymbol{F}_i = \boldsymbol{\mathcal{I}}_i \dot{\boldsymbol{V}}_i + \sum_{l \in \mu(i)} \mathrm{Ad}^*_{\boldsymbol{T}^{-1}_{i,l}} \boldsymbol{F}_l$
  for $j = 1$ to n **do**
    $\frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{q}_j} = \boldsymbol{\mathcal{I}}_i \frac{\partial \dot{\boldsymbol{V}}_i}{\partial \boldsymbol{q}_j} - \frac{\partial \boldsymbol{F}^{\mathrm{g}}}{\partial \boldsymbol{q}_j}$
      $+ \sum_{l \in \mu(i)} \mathrm{Ad}^*_{\boldsymbol{T}^{-1}_{i,l}} \left(\frac{\partial \boldsymbol{F}_l}{\partial \boldsymbol{q}_j} - \mathrm{ad}^*_{\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{q}_j}} \boldsymbol{F}_l \right)$
    $\left[\frac{\partial \boldsymbol{M}}{\partial \boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{z}\right]_{i,j} = \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_j}^T \boldsymbol{F}_i + \boldsymbol{S}_i^T \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{q}_j}$
  end for
end for

Algorithm 3 Derivative of the recursive inverse dynamics for $\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{q}}$ and $\frac{\partial \boldsymbol{c}}{\partial \dot{\boldsymbol{q}}}$

for $i = 1$ to n **do**
  $\boldsymbol{V}_i = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \boldsymbol{V}_{\lambda(i)} + \boldsymbol{S}_i \dot{\boldsymbol{q}}_i$
  $\dot{\boldsymbol{V}}_i = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \dot{\boldsymbol{V}}_{\lambda(i)} + \mathrm{ad}_{\boldsymbol{V}_i} \boldsymbol{S}_i \dot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i$
  for $j = 1$ to n **do**
    $\frac{\partial \boldsymbol{V}_i}{\partial \boldsymbol{q}_j} = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \frac{\partial \boldsymbol{V}_{\lambda(i)}}{\partial \boldsymbol{q}_j} - \mathrm{ad}_{\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{q}_j}} \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \boldsymbol{V}_{\lambda(i)} + \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_j} \dot{\boldsymbol{q}}_i$
    $\frac{\partial \boldsymbol{V}_i}{\partial \dot{\boldsymbol{q}}_j} = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \frac{\partial \boldsymbol{V}_{\lambda(i)}}{\partial \dot{\boldsymbol{q}}_j} + \boldsymbol{S}_i^k$
    $\frac{\partial \dot{\boldsymbol{V}}_i}{\partial \boldsymbol{q}_j} = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \frac{\partial \dot{\boldsymbol{V}}_{\lambda(i)}}{\partial \boldsymbol{q}_j} - \mathrm{ad}_{\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{q}_j}} \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \dot{\boldsymbol{V}}_{\lambda(i)}$
      $+ \mathrm{ad}_{\frac{\partial \boldsymbol{V}_i}{\partial \boldsymbol{q}_j}} \boldsymbol{S}_i \dot{\boldsymbol{q}}_i + \mathrm{ad}_{\boldsymbol{V}_i} \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_j} \dot{\boldsymbol{q}}_i + \frac{\partial \dot{\boldsymbol{S}}_i}{\partial \boldsymbol{q}_j} \dot{\boldsymbol{q}}_i$
    $\frac{\partial \dot{\boldsymbol{V}}_i}{\partial \dot{\boldsymbol{q}}_j} = \mathrm{Ad}_{\boldsymbol{T}^{-1}_{\lambda(i),i}} \frac{\partial \dot{\boldsymbol{V}}_{\lambda(i)}}{\partial \dot{\boldsymbol{q}}_j}$
      $+ \mathrm{ad}_{\frac{\partial \boldsymbol{V}_i}{\partial \dot{\boldsymbol{q}}_j}} \boldsymbol{S}_i \dot{\boldsymbol{q}}_i + \mathrm{ad}_{\boldsymbol{V}_i} \boldsymbol{S}_i^k + \frac{\partial \dot{\boldsymbol{S}}_i}{\partial \dot{\boldsymbol{q}}_j} \dot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i^k$
  end for
end for
for $i = n$ to 1 **do**
  $\boldsymbol{F}_i = \boldsymbol{\mathcal{I}}_i \dot{\boldsymbol{V}}_i - \mathrm{ad}^*_{\boldsymbol{V}_i} \boldsymbol{\mathcal{I}}_i \boldsymbol{V}_i - \boldsymbol{F}_i^g + \sum_{l \in \mu(i)} \mathrm{Ad}^*_{\boldsymbol{T}^{-1}_{i,l}} \boldsymbol{F}_l$
  for $j = 1$ to n **do**
    $\frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{q}_j} = \boldsymbol{\mathcal{I}}_i \frac{\partial \dot{\boldsymbol{V}}_i}{\partial q_j} - \mathrm{ad}^*_{\frac{\partial \boldsymbol{V}_i}{\partial q_j}} \boldsymbol{\mathcal{I}} \boldsymbol{V}_i - \mathrm{ad}^*_{\boldsymbol{V}_i} \boldsymbol{\mathcal{I}}_i \frac{\partial \boldsymbol{V}_i}{\partial \boldsymbol{q}_j} - \frac{\partial \boldsymbol{F}^{\mathrm{g}}}{\partial \boldsymbol{q}_j}$
      $+ \sum_{l \in \mu(i)} \mathrm{Ad}^*_{\boldsymbol{T}^{-1}_{i,l}} \left(\frac{\partial \boldsymbol{F}_l}{\partial \boldsymbol{q}_j} - \mathrm{ad}^*_{\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{q}_j}} \boldsymbol{F}_l \right)$
    $\frac{\partial \boldsymbol{F}_i}{\partial \dot{\boldsymbol{q}}_j} = \boldsymbol{\mathcal{I}}_i \frac{\partial \dot{\boldsymbol{V}}_i}{\partial \dot{\boldsymbol{q}}_j} - \mathrm{ad}^*_{\frac{\partial \boldsymbol{V}_i}{\partial \dot{\boldsymbol{q}}_j}} \boldsymbol{\mathcal{I}} \boldsymbol{V}_i - \mathrm{ad}^*_{\boldsymbol{V}_i} \boldsymbol{\mathcal{I}}_i \frac{\partial \boldsymbol{V}_i}{\partial \dot{\boldsymbol{q}}_j}$
      $+ \sum_{l \in \mu(i)} \mathrm{Ad}^*_{\boldsymbol{T}^{-1}_{i,l}} \frac{\partial \boldsymbol{F}_l}{\partial \boldsymbol{q}_j}$
    $\left[\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{q}}\right]_{i,j} = \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_j}^T \boldsymbol{F}_i + \boldsymbol{S}_i^T \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{q}_j}$
    $\left[\frac{\partial \boldsymbol{c}}{\partial \dot{\boldsymbol{q}}}\right]_{i,j} = \boldsymbol{S}_i^T \frac{\partial \boldsymbol{F}_i}{\partial \dot{\boldsymbol{q}}_j}$
  end for
end for