

Dojo: A Differentiable Physics Engine for Robotics

Taylor A. Howell^{1*}, Simon Le Cleac'h^{1*}, Jan Brüdigam², Qianzhong Chen¹, Jiankai Sun¹, J. Zico Kolter³, Mac Schwager¹, and Zachary Manchester⁴

Abstract—We present Dojo, a differentiable physics engine for robotics that prioritizes stable simulation, accurate contact physics, and differentiability with respect to states, actions, and system parameters. Dojo models hard contact and friction with a nonlinear complementarity problem with second-order cone constraints. We introduce a custom primal-dual interior-point method to solve the second order cone program for stable forward simulation over a broad range of sample rates. We obtain smooth gradient approximations with this solver through the implicit function theorem, giving gradients that are useful for downstream trajectory optimization, policy optimization, and system identification applications. Specifically, we propose to use the central path parameter threshold in the interior point solver as a user-tunable design parameter. A high value gives a smooth approximation to contact dynamics with smooth gradients for optimization and learning, while a low value gives precise simulation rollouts with hard contact. We demonstrate Dojo’s differentiability in trajectory optimization, policy learning, and system identification examples. We also benchmark Dojo against MuJoCo, PyBullet, Drake, and Brax on a variety of robot models, and study the stability and simulation quality over a range of sample frequencies and accuracy tolerances. Finally, we evaluate the sim-to-real gap in hardware experiments with a Ufactory xArm 6 robot. Dojo is an open source project implemented in Julia with Python bindings, with code available at <https://github.com/dojo-sim/Dojo.jl>.

Index Terms—Contact Dynamics, Differentiable Optimization, Simulation, Robotics

I. INTRODUCTION

The last decade has seen immense advances in learning-based methods for policy optimization and trajectory optimization in robotics, e.g., for dexterous manipulation [1, 2], quadrupedal locomotion [3, 4], and pixels-to-torques control [5]. These advances have largely hinged on innovations in learning architectures, large scale optimization algorithms, and large datasets. In contrast, there has been comparatively little work on the lowest level of the robotics reinforcement learning stack: the *physics engine*. We argue that core improvements in physics engines can enable future advancements in robotics, and we present Dojo as a physics engine that embodies several such advances.

Physics engines that simulate rigid-body dynamics with contact are utilized for trajectory optimization, reinforcement

¹ Stanford University, Stanford, CA 94305, USA. {thowell, simonlc, qchen23, jksun, schwager}@stanford.edu

² School of Computation, Information and Technology, Technical University of Munich, Munich, 80333, Germany. jan.bruedigam@tum.de

³ Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA. zkoltzer@cs.cmu.edu

⁴ The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. zacm@cmu.edu

(Corresponding author: S. Le Cleac'h)

* These authors contributed equally to this work.

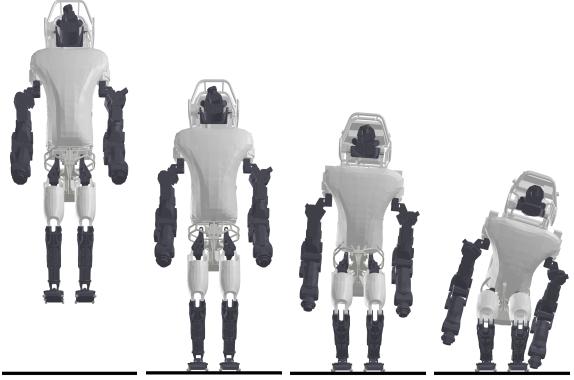


Fig. 1: Atlas drop simulation. Dojo simulates this system with 403 maximal-coordinates states, 30 joint constraints, 36 inputs, and 8 contact points in real-time at 65 Hz. Dojo respects floor-feet penetration constraints to machine precision. Other simulators struggle to maintain the floor contact constraint, especially at low simulation rates.

learning, system identification, and dataset generation for domains ranging from locomotion to manipulation. To overcome the sim-to-real gap [6] and to be of practical value in real-world applications, an engine should provide stable simulation, accurately reproduce a robot’s dynamics, and ideally, be differentiable to enable the use of efficient gradient-based optimization methods.

In recent years, a number of physics engines [7, 8, 9, 10, 11, 12, 13, 14] have been developed and utilized for robotics. These works have advanced the state of the art in robot simulation, particularly offering differentiability [8, 11, 9, 10], multi-physics simulation [13, 14, 11, 15], and the incorporation of learnable dynamics residuals [12]. Recent work is also moving towards a convergence of fully learning based world models and traditional physics-based simulators [16, 17], representing an exciting new frontier in robot simulation.

In this work, we propose to further advance the state of the art by focusing on the underlying numerics of the physics engine, introducing features that can be adopted throughout the existing simulation ecosystem to improve performance in a variety of ways. We package these numerical improvements in a new simulation engine, Dojo, to highlight their favorable properties over exiting numerical techniques commonly used in physics simulators.

Specifically, we introduce two new contributions specific to Dojo: (i) We propose a custom primal-dual interior point solver for stably solving the complementary problem for forward simulation. This solver allows for low sample rates while maintaining stable and accurate contact simulation,

thereby alleviating the *vanishing/exploding* gradient problem that appears when differentiating through high sample rate rollouts common in other differentiable simulators. (ii) We obtain tunable gradient information through implicit differentiation of the interior point solver, obtaining user-defined smoothness of the gradients through contact events by tuning the central path parameter. This gives informative gradients through contact events for policy and trajectory optimization (with a high central path parameter), while also enabling sharp, physically accurate simulation rollouts (with a low central path parameter). This is in contrast to existing methods that either deliver non-smooth gradients that are not informative for trajectory or policy optimization, or expensive sampling-based gradient approximations that require a large number of calls to the engine, leading to slow computation.

Additionally, we also incorporate the following features that have been presented in previous works, but are not typically implemented in existing physics engines: (a) We use a variational integration scheme for strong energy and momentum conservation in non-contact regimes. (b) We use full nonlinear complementarity constraints with nonlinear friction cones to avoid numerical artifacts like interpenetration of rigid bodies (e.g., a robot foot sinking through the floor) and creep (e.g., objects that should be at rest incorrectly sliding) commonly seen in robotics simulators [18]. (c) We use maximal coordinates, explicitly representing the 6 degree of freedom pose of each rigid link, and the internal constraint forces that bind them together.

The Dojo physics engine is designed around these core numerical innovations with the goal of advancing robot simulation for trajectory optimization and motion planning, control, reinforcement learning, system identification, and for generating high-quality datasets for learning and validation.

To demonstrate the advantages of this combination of numerical features, we benchmark Dojo against MuJoCo [19], Drake [7], and Brax [8] for computational speed in simulations with four different robot platforms, showing Dojo delivers a comparable performance as other differentiable simulators. We demonstrate Dojo’s numerical stability and accuracy over sample rates down to 20Hz, allowing for lower sample rates than other simulators, leading to fewer rollout steps, and therefore less severe vanishing/exploding gradients. We also compare Dojo’s primal-dual solver versus a more common primal only interior point solver, showing improved convergence speed and constraint satisfaction. We demonstrate Dojo’s differentiability in trajectory optimization, policy optimization, and system identification examples in comparison with sampling-based gradient approximations. Finally, we show Dojo’s low sim-to-real gap in hardware experiments with a UFactory xArm 6.

In summary, the key contributions of this paper as integrated into the Dojo simulator include:

- (i) a custom primal-dual interior-point method for giving stable, accurate simulation over sample rates as low as 20Hz,
- (ii) analytic gradients through contact efficiently computed via implicit differentiation of the interior-point solver

with a user-defined smoothness approximation set by the central path parameter,

- (iii) incorporation of variational integration and a nonlinear complementarity problem (NCP) model for accurate contact dynamics (previously introduced in literature, but not yet integrated in a simulator).

In the remainder of this paper, we first provide an overview of related state-of-the-art physics engines in Section II. We then summarize important technical background in Section III. Next, we present Dojo, and its key features in Section IV. Simulation, planning, policy optimization, and system identification examples, and hardware sim-to-real gap evaluation are presented in Section V. Finally, we conclude with a discussion of limitations and future work in Section VI.

II. RELATED WORK

In this section, we provide an overview of several popular physics engines, focusing on their physical fidelity, underlying optimization algorithms, and capability to compute gradients.

A. Mainstream Physics Engines

1) *MuJoCo*: In the learning community, *MuJoCo* [19] has become a standard for benchmarking reinforcement learning algorithms using the OpenAI Gym environments [20]. *MuJoCo* utilizes minimal-coordinates representations, and employs both semi-implicit Euler and explicit fourth-order Runge-Kutta integrators to simulate multi-body systems. These integrators often require small time steps, particularly for systems experiencing contact, and typically sample rates of hundreds to thousands of Hertz are required for stable simulation, which a mature and efficient implementation is able to achieve at much faster than real-time rates. However, these high rates can prove a challenge for control tasks, such as reinforcement learning settings where vanishing or exploding gradients are exacerbated over long horizons with many time steps [21, 22, 23].

Impact and friction are modeled using a smooth, convex contact model [24]. While this approach reliably computes contact forces, it introduces unphysical artifacts, and contact forces at a distance (i.e., while not in contact) [25], and the default friction model introduces creep and velocity drift during sliding. Additionally, achieving good simulation behavior often requires system-specific tuning of multiple solver parameters. Further, the “soft” contact model is computed using a *primal* optimization method, meaning that as parameters are set to produce “hard” or more realistic contact, the underlying optimization problem becomes increasingly ill-conditioned and difficult to solve. For RL methods, where obtaining a large volume of rollouts quickly is more important than preserving physical fidelity of each rollout, these compromises have been effective. However, for control and planning with real robot hardware they can be problematic. For example, it is often not possible to eliminate unphysical artifacts from the simulation to produce realistic results. The lack of smooth gradients is also a major challenge in deep learning [18], where contact dynamics must often be smoothed unrealistically in order to make learning progress. Analytical gradients are not provided

TABLE I: Comparison of physics engines used for robotics. Several simulators like MuJoCo and Drake have a selection of integrators that users can choose. We demonstrate their default/most-widely-used integrators.

Engine	Application	Integrator	State	Contact Softness	Contact Model/Solver	Gradients
MuJoCo	robotics	RK4	minimal	soft	Newton	finite difference
Drake	robotics	implicit Euler	minimal	soft/hard	penalty-based/SAP	randomized smoothing
Bullet	graphics	implicit Euler	minimal	soft/hard	iterative	finite difference
DART	robotics	implicit Euler	minimal	hard	LCP	subgradient
PhysX	graphics	explicit	minimal	soft	iterative	finite difference
Brax	graphics	explicit	maximal	soft	iterative	analytical
Dojo	robotics	variational	maximal	hard	NCP	implicit gradient

by the engine, and instead require finite-difference schemes [26] that are computationally expensive. This approach requires multiple calls to the engine, which can be expensive if not performed in parallel.

2) *Drake*: *Drake* [7], designed for robotics applications, prioritizes physical accuracy and flexibility in simulation. Its contact modeling primarily employs a penalty method, where contact forces are approximated using stiff springs, based on the Hunt-Crossley model [27]. This approach provides a compliant contact model but requires small time steps to ensure stability, as the stiffness can lead to numerical challenges such as instability and gradient explosion [21]. Recently, *Drake* has introduced the soft articulated-body dynamics with compliant contact (SAP) method [28], which formulates contact as a time-stepping optimization problem. SAP introduces compliance to relax the contact model, making it robust for simulating articulated systems, similar in spirit to methods used in MuJoCo [19]. *Drake* offers flexibility in integrator choices, including advanced error-controlled methods, ensuring stable and accurate simulation for various dynamics. Gradients in *Drake* are currently computed using Eigen’s autodiff framework through the penalty method, but this approach is less ideal for stiff systems. While SAP could theoretically utilize the implicit function theorem for gradient computation, this feature has not been implemented yet. Additionally, methods like randomized smoothing for returning gradients [29] provide alternative strategies for handling contact-rich dynamics outside of *Drake*’s native capabilities.

3) *Other Engines*: The popular robotics simulator *Gazebo* [30] can utilize several different physics engines to simulate multi-body contact dynamics. *Bullet* [31] and *DART* [32] are common choices. These engines model hard contact dynamics with an LCP formulation. Automatic differentiation tools have been utilized to compute gradients [12], [33], [34]. However, because of the discontinuous nature of contact dynamics, this approach will return discontinuous gradients at contact events, which are less useful for gradient based trajectory or policy optimization. Heuristics have been proposed to enumerate contact modes in order to select informative gradients [9]. However, this approach scales poorly with the number of contact mode switches.

Engines designed for hardware accelerators (e.g., GPUs), including *Brax* [8] and *PhysX* [35], typically utilize simplified contact dynamics. Additionally, these engines usually require system-specific tuning and their simulation results typically prioritize speed over physical fidelity, so a large number of rollouts can be obtained quickly to train learning based

policies.

B. Differentiable Physics Engines

In contrast to traditional physics engines, differentiable physics engines present promising opportunities for robotics by incorporating physical models into auto-differentiation frameworks [36]. Prior research has explored differentiation across various domains, such as contact and friction models [37, 38, 39, 40, 41, 42], latent state models [43, 44, 45, 46], volumetric soft bodies [47, 48, 11], and particle dynamics [44, 49]. Additionally, system identification using parameterized physics models [50, 51, 52, 53, 54, 55, 56, 57, 58, 59] and inverse simulation techniques [60] have also been explored. Furthermore, there has been considerable research on smooth gradient computation [61], which aids in reducing noise in gradient-based model explanations and ensures stable gradient calculations. Examples of such systems include Warp [62] and Brax [8], both of which utilize the XPBD [63] model for contact simulation. The GradSim framework [64] combines a physics simulator with a differentiable rendering pipeline. Key components of these simulators include gradient calculation, dynamics models, contact models, and integrators. These simulators are capable of leveraging gradient-based optimization techniques to enhance real-to-sim transfer capabilities.

Gradients of rigid body dynamics can be useful in robotics for various purposes. Applications include system identification [65], controller design [66], controller tuning [67], trajectory optimization [68], and policy optimization [69]. However, gradients are typically computed through autodifferentiating through a simulation rollout by expressing the rollout as a computation graph, making use of mature auto-diff capabilities, e.g. in PyTorch. Unfortunately, this approach often leads to numerical instability as a long chain of differentiation tends toward infinity or zero as determined by the eigenvalues of the Jacobians in the computation graph: the so called vanishing/exploding gradients problem.

Dojo uses an optimization solver to propagate a trajectory in time, making it impossible to apply back propagation to compute gradients. Instead, we use implicit differentiation, which has been explored recently to obtain gradients of the solution to optimization problems with respect to problem parameters [38, 70]. Implicit gradients are computed using the implicit function theorem, rather than directly auto-diffing through a simulation rollout, in hopes of producing more stable gradient computations.

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

Building on prior work [71], Dojo utilizes the open-source maximal-coordinates dynamics library `ConstrainedDynamics.jl` and efficient graph-based linear-system solver `GraphBasedSystems.jl`. However, unlike this previous work, Dojo has an improved contact model, specifically with regard to friction; and utilizes a more efficient, reliable, and versatile interior-point solver for the NCP.

III. BACKGROUND

Here we review existing methods of complementarity-based contact models, implicit differentiation, maximal coordinates, and variational integrators that are used in Dojo.

A. Complementarity-Based Contact Models

Impacts and friction can be modeled through constraints on the system's configuration and the applied contact impulses.

Impact: For a system with P contact points, we define a signed-distance function, $\phi : \mathbf{Z} \rightarrow \mathbf{R}^P$, subject to the following element-wise constraint

$$\phi(z) \geq 0. \quad (1)$$

Where z represents the system configuration (i.e., the pose of robot and other objects in the simulation environment). Impact forces with magnitude $\gamma \in \mathbf{R}^P$ are applied to the bodies' contact points in the direction of their surface normals in order to enforce (1) and prevent interpenetration. Collision points¹ are checked for constraint satisfaction at each iteration of our primal-dual interior-point solver that will be introduced in IV-B. A non-negative constraint

$$\gamma \geq 0, \quad (2)$$

enforces physical behavior that impulses are repulsive (e.g., the floor does not attract bodies), and the complementarity condition

$$\gamma \circ \phi(z) = 0, \quad (3)$$

where \circ is an element-wise product operator, enforces zero force if the body is not in contact and allows non-zero force during contact.

Friction: Coulomb friction instantaneously maximizes the dissipation of kinetic energy between two objects in contact. For a single contact point, this physical phenomenon can be modeled by the following optimization problem:

$$\begin{aligned} & \underset{b}{\text{minimize}} && v^T b \\ & \text{subject to} && \|b\|_2 \leq c_f \gamma, \end{aligned} \quad (4)$$

¹Collision geometries are currently limited to simple shape primitives (e.g., four points on the foot of a humanoid instead of using a full mesh). While this is a limitation of the current implementation, it is common in other simulators to manually edit contact geometries for critical parts of the robot, such as the feet of a humanoid or quadruped.



Fig. 2: Friction-cone comparison. Linearized double-parameterized (left) and nonlinear second-order (right) cones.

where $v \in \mathbf{R}^2$ is the tangential velocity at the contact point, $b \in \mathbf{R}^2$ is the friction force, and $c_f \in \mathbf{R}_+$ is the coefficient of friction between the two objects [72].

This problem is naturally a convex second-order cone program, and can be efficiently and reliably solved [73]. Classically, other works solve an approximate version of (4),

$$\begin{aligned} & \underset{\beta}{\text{minimize}} && [v^T \ -v^T] \beta \\ & \text{subject to} && \beta^T \mathbf{1} \leq c_f \gamma, \\ & && \beta \geq 0, \end{aligned} \quad (5)$$

which satisfies the LCP formulation. Here, the friction cone is linearized (Fig. 2) and the friction vector, $\beta \in \mathbf{R}^4$, is correspondingly overparameterized and subject to additional positivity constraints [74].

The optimality conditions of (5) and constraints used in the LCP are

$$[v^T \ -v^T]^T + \psi \mathbf{1} - \eta = 0, \quad (6)$$

$$c_f \gamma - \beta^T \mathbf{1} \geq 0, \quad (7)$$

$$\psi \cdot (c_f \gamma - \beta^T \mathbf{1}) = 0, \quad (8)$$

$$\beta \circ \eta = 0, \quad (9)$$

$$\beta, \psi, \eta \geq 0, \quad (10)$$

where $\psi \in \mathbf{R}$ and $\eta \in \mathbf{R}^4$ are the dual variables associated with the friction cone and positivity constraints, respectively. In practice, ψ acts as a flag to judge if the object has relative movement to the ground, while η is used to enforce the constraint that friction forces align with the vertices of the linearized friction cone approximation, and $\mathbf{1}$ is a vector of ones.

The primary drawback of this formulation is that the optimized friction force will naturally align with the vertices of the cone approximation, which may not align with the velocity vector of the contact point. Thus, the friction force does not perfectly oppose the movement of the system at the contact point. Unless the pyramidal approximation is improved with a finer discretization, incurring increased computational cost, unphysical velocity drift will occur. Additionally, the LCP contact model requires a linearized form of the dynamics (12) and a linear approximation of the signed-distance functions (1-3), both of which negatively impact physical accuracy. To avoid these problems, in Dojo we solve the full NCP with nonlinear friction cone, and develop a new primal-dual interior point method to reliably solve this more difficult optimization problem.

B. Implicit Differentiation

An implicit function, $r : \mathbf{R}^{n_w} \times \mathbf{R}^{n_\theta} \rightarrow \mathbf{R}^{n_w}$, is defined as $r(w^*; \theta) = 0$ for solution $w^* \in \mathbf{R}^{n_w}$ and problem data $\theta \in \mathbf{R}^{n_\theta}$. At a solution point, the sensitivities of the solution with respect to the problem data, i.e., $\partial w^*/\partial \theta$, can be computed using the implicit function theorem [75]

$$\frac{\partial w^*}{\partial \theta} = -\left(\frac{\partial r}{\partial w}\right)^{-1} \frac{\partial r}{\partial \theta}. \quad (11)$$

Newton's method is typically employed to find solutions w^* . When the method succeeds, the sensitivity (11) can be computed and the factorization of $\partial r/\partial w$ used to find the solution is reused to efficiently compute sensitivities at very low computational cost, using only back-substitution. Additionally, each element of the sensitivity can be computed in parallel. Dojo uses implicit method to obtain gradients for simulation rollouts with respect to control inputs, dynamics parameters, or initial conditions.

C. Maximal-Coordinates State Representation

Most multi-body physics engines utilize minimal- or joint-coordinate representations for dynamics because of the small number of states and convenience of implementation. This results in small, but dense, systems of equations. In contrast, maximal-coordinates explicitly represent the position, orientation, and velocities of each body in a multi-body system. This produces large, sparse systems of equations that can be efficiently solved, including in the contact setting. We provide an overview, largely based on prior work [76], of this representation.

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

$$F(x_-, z, x_+) = 0, \quad (12)$$

where we indicate the previous and next time steps with minus ($-$) and plus ($+$) subscripts, respectively, and the current time step without decoration. We employ a variational integrator that has desirable energy and momentum conservation properties [77]. Linear and angular velocities are handled implicitly via finite-difference approximations.

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

$$k^{ab}(x_+^a, x_+^b) = 0. \quad (13)$$

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

$$\begin{bmatrix} F^a(x_-^a, x^a, x_+^a) + K^a(x^a, x^b)^T j^{ab} \\ F^b(x_-^b, x^b, x_+^b) + K^b(x^a, x^b)^T j^{ab} \\ k^{ab}(x_+^a, x_+^b) \end{bmatrix} = 0, \quad (14)$$

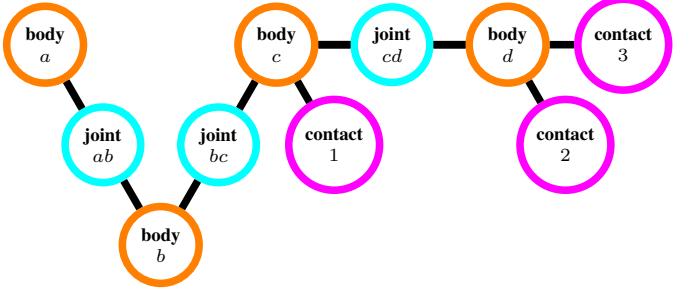


Fig. 3: Graph structure for maximal-coordinates system with 4 bodies, 3 joints, and 3 points of contact.

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

We can generalize (14) to include additional bodies and joints. For a multi-body system with N bodies and M joints we define a maximal-coordinates configuration $z = (x^{(1)}, \dots, x^{(N)}) \in \mathbf{Z}$ and joint impulse $j = (j^{(1)}, \dots, j^{(M)}) \in \mathbf{J}$. We define the implicit discrete-time dynamics of the maximal-coordinates system as

$$F(z_-, z, z_+, j) = 0, \quad (15)$$

where $F : \mathbf{Z} \times \mathbf{Z} \times \mathbf{Z} \times \mathbf{J} \rightarrow \mathbf{R}^{6N}$. In order to simulate the system we find z_+ and j that satisfy (15) for a provided z_- and z using Newton's method.

By exploiting the mechanism's structure, we can efficiently perform root finding on (15) (see [76] for additional details). This structure is manifested as a graph of the mechanism, where each body and joint is considered a node, and joints have edges connecting bodies (Fig. 3). Because the mechanism structure is known *a priori*, a permutation matrix can be precomputed and used to perform efficient sparse linear algebra during simulation. For instance, in the case where the joint constraints form a system without loops, the resulting sparse system can be solved in linear time with respect to the number of links.

D. Variational Integrator

We use a specialized implicit integrator that natively handles quaternions and alleviates spurious artifacts that commonly arise from contact interactions. The dynamics are derived by approximating Hamilton's Principle of Least-Action using a simple midpoint scheme [77, 78]. This approach produces *variational* integrators.

Each body has a linear

$$m \frac{p_+ - 2p + p_-}{h} - hmg - A(p)^T j - hf = 0, \quad (16)$$

and rotational

$$\begin{aligned} & \sqrt{1 - \psi_+^T \psi_+} J \psi_+ + \psi_+ \times J \psi_+ \\ & - \sqrt{1 - \psi^T \psi} J \psi + \psi \times J \psi - B(q)^T j - h^2 \frac{\tau}{2} = 0, \end{aligned} \quad (17)$$

dynamics specified by mass $m \in \mathbf{R}_{++}$, inertia $J \in \mathbf{S}_{++}^3$, gravity $g \in \mathbf{R}^3$, and time step $h \in \mathbf{R}_{++}$. Equations (16, 17)

Algorithm 1 Analytical Line Search For Cones

```

1: procedure SEARCH( $w, \Delta, \tau^{\text{ort}}, \tau^{\text{soc}}$ )
2:    $\alpha_y^{\text{ort}} \leftarrow \alpha(y^{(1)}, \tau^{\text{ort}} \Delta^{y^{(1)}})$   $\triangleright 41$ 
3:    $\alpha_z^{\text{ort}} \leftarrow \alpha(z^{(1)}, \tau^{\text{ort}} \Delta^{z^{(1)}})$   $\triangleright 41$ 
4:    $\alpha_y^{\text{soc}} \leftarrow \min_{i \in \{2, \dots, n\}} \alpha(y^{(i)}, \tau^{\text{soc}} \Delta^{y^{(i)}})$   $\triangleright 46$ 
5:    $\alpha_z^{\text{soc}} \leftarrow \min_{i \in \{2, \dots, n\}} \alpha(z^{(i)}, \tau^{\text{soc}} \Delta^{z^{(i)}})$   $\triangleright 46$ 
6:   Return  $\min(\alpha_y^{\text{ort}}, \alpha_z^{\text{ort}}, \alpha_y^{\text{soc}}, \alpha_z^{\text{soc}})$ 

```

are essentially second-order centered-finite-difference approximations of Newton's second law and Euler's equation for the rotational dynamics, respectively, where

$$q_+ = q \cdot \left[\sqrt{1 - \psi_+^T \psi_+} \right] \quad (18)$$

is recovered from a three-parameter representation $\psi \in \mathbf{R}^3$ [79]. We refer to Appendix A for quaternion conventions and algebra. Joint impulses $j \in \mathbf{J}$ have linear $A : \mathbf{R}^3 \rightarrow \mathbf{R}^{\dim(\mathbf{J}) \times 3}$ and rotational $B : \mathbf{H} \rightarrow \mathbf{R}^{\dim(\mathbf{J}) \times 3}$ mappings into the dynamics. The configuration of a body $x^{(i)} = (p^{(i)}, q^{(i)}) \in \mathbf{R}^3 \times \mathbf{H}$ comprises a position and orientation represented as a quaternion. Forces and torques $f, \tau \in \mathbf{R}^3$ can be applied to the bodies.

IV. METHOD

We now introduce Dojo's contact model and custom primal-dual interior-point solver, as well as the implicit differentiation method for obtaining gradients through the solver.

A. Contact Dynamics Model

Impact and friction behaviors are modeled, along with the system's dynamics, as an NCP. This model simulates hard contact without requiring system-specific solver tuning. Additionally, contacts between a system and the environment are treated as a single graph node connected to a rigid body (Fig 3). As a result, the engine retains efficient linear-time complexity for open-chain mechanical systems.

Dojo uses the rigid impact model (1-3) and in the following section we present its Coulomb friction model that utilizes an exact nonlinear friction cone.

Nonlinear friction cone: In contrast to the LCP approach, we utilize the optimality conditions of (4) in a form amenable to a primal-dual interior-point solver. The associated cone program is

$$\begin{aligned} & \underset{\xi}{\text{minimize}} && v^T \xi_{(2:3)} \\ & \text{subject to} && \xi_{(1)} = c_f \gamma, \\ & && \|\xi_{(2:3)}\|_2 \leq \xi_{(1)}, \end{aligned} \quad (19)$$

where ξ is an auxiliary vector under the nonlinear friction cone model, with elements 2 and 3 representing the friction force components, and subscripts indicate vector indices. The

relaxed optimality conditions for (19) in interior-point form are

$$v - \eta_{(2:3)} = 0, \quad (20)$$

$$\xi_{(1)} - c_f \gamma = 0, \quad (21)$$

$$\xi \circ \eta = \kappa \mathbf{e}, \quad (22)$$

$$\|\xi_{(2:3)}\|_2 \leq \xi_{(1)}, \|\eta_{(2:3)}\|_2 \leq \eta_{(1)}, \quad (23)$$

with dual variable $\eta \in \mathbf{R}^3$ associated with the second-order-cone constraints, and central-path parameter, $\kappa \in \mathbf{R}_+$. The second-order-cone product is

$$\xi \circ \eta = (\xi^T \eta, \xi_{(1)} \eta_{(2:3)} + \eta_{(1)} \xi_{(2:3)}), \quad (24)$$

and

$$\mathbf{e} = (1, 0, \dots, 0), \quad (25)$$

is its corresponding identity element [80]. Friction is recovered from the solution: $b = \xi_{(2:3)}^*$. The benefits of this model are increased physical fidelity and fewer optimization variables, without substantial increase in computational cost.

Nonlinear complementarity problem: Systems comprising N bodies and a single contact point are simulated using a time-stepping scheme that solves the feasibility problem

$$\begin{aligned} & \text{find} && z_+, j, \gamma, \xi, \eta \\ & \text{subject to} && F(z_-, z, z_+, j, \gamma, \xi, u) = 0, \\ & && \gamma \circ \phi(z_+) = \kappa \mathbf{1}, \\ & && \xi \circ \eta = \kappa \mathbf{e}, \\ & && v(z, z_+) - \eta_{(2:3)} = 0, \\ & && \xi_{(1)} - c_f \gamma = 0, \\ & && \gamma, \phi(z_+) \geq 0, \\ & && \|\xi_{(2:3)}\|_2 \leq \xi_{(1)}, \|\eta_{(2:3)}\|_2 \leq \eta_{(1)}. \end{aligned} \quad (26)$$

The system's smooth dynamics $F : \mathbf{Z} \times \mathbf{Z} \times \mathbf{Z} \times \mathbf{J} \times \mathbf{R}_+ \times \mathbf{R}^2 \times \mathbf{U} \rightarrow \mathbf{R}^{6N}$ comprise linear and rotational dynamics (16-17) for each body which are subject to inputs $u = (f^{(1)}, \tau^{(1)}, \dots, f^{(N)}, \tau^{(N)}) \in \mathbf{U}$. The contact-point tangential velocity $v : \mathbf{Z} \times \mathbf{Z} \rightarrow \mathbf{R}^2$ is a function of the current and next configurations (i.e., a finite-difference velocity). The central-path parameter $\kappa \in \mathbf{R}_+$ and target \mathbf{e} [80] are utilized by the interior-point solver in the following section. This formulation extends to multiple contacts.

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

To simulate a system forward in time one step, given a control input and state comprising the previous and current configurations, solutions to a sequence of barrier problems (26) are found with $\kappa \rightarrow 0$. The central-path parameter has a physical interpretation as being the softness of the contact model. A value $\kappa = 0$ corresponds to exact "hard" or inelastic contact, whereas a relaxed value produces soft contact where

Algorithm 2 Primal-Dual Interior-Point Solver

```

1: procedure OPTIMIZE( $a_0, b_0, c_0, \theta, \mathcal{K}$ )
2:   Parameters:  $\tau_{\max} = 0.99, \tau_{\min} = 0.95$ 
3:    $r_{\text{tol}} = 10^{-5}, \kappa_{\text{tol}} = 10^{-5}, \beta = 0.5$ 
4:   Initialize:  $a = a_0, b = b_0 \in \mathcal{K}, c = c_0 \in \mathcal{K}$ 
5:    $r_{\text{vio}}, \kappa_{\text{vio}} \leftarrow \text{VIOLATION}(w)$   $\triangleright (49, 50)$ 
6:   Until  $r_{\text{vio}} < r_{\text{tol}}$  and  $\kappa_{\text{vio}} < \kappa_{\text{tol}}$  do
7:      $\Delta_{\text{aff}} \leftarrow -\bar{R}^{-1}(w; \theta)r(w; \theta, 0)$ 
8:      $\alpha_{\text{aff}} \leftarrow \text{CONESEARCH}(w, \Delta_{\text{aff}}, 1, 1)$ 
9:      $\mu, \sigma \leftarrow \text{CENTER}(b, c, \alpha_{\text{aff}}, \Delta_{\text{aff}})$   $\triangleright (51-54)$ 
10:     $\kappa \leftarrow \max(\sigma\mu, \kappa_{\text{tol}}/5)$ 
11:     $\Delta \leftarrow -\bar{R}^{-1}(w; \theta)r(w; \theta, \kappa)$ 
12:     $\tau^{\text{ort}} \leftarrow \max(\tau_{\min}, 1 - \max(r_{\text{vio}}, \kappa_{\text{vio}})^2)$ 
13:     $\tau^{\text{soc}} \leftarrow \min(\tau_{\max}, \tau^{\text{ort}})$ 
14:     $\alpha \leftarrow \text{CONESEARCH}(w, \Delta, \tau^{\text{ort}}, \tau^{\text{soc}})$ 
15:     $c_{\text{vio}}^*, \kappa_{\text{vio}}^* \leftarrow r_{\text{vio}}, \kappa_{\text{vio}}$ 
16:     $\hat{w} \leftarrow \text{UPDATE}(w, \Delta, \alpha)$   $\triangleright (47, 48)$ 
17:     $r_{\text{vio}}, \kappa_{\text{vio}} \leftarrow \text{VIOLATION}(\hat{w})$   $\triangleright (49, 50)$ 
18:    Until  $r_{\text{vio}} \leq c_{\text{vio}}^*$  or  $\kappa_{\text{vio}} \leq \kappa_{\text{vio}}^*$  do
19:       $\alpha \leftarrow \beta\alpha$ 
20:       $\hat{w} \leftarrow \text{UPDATE}(w, \Delta, \alpha)$   $\triangleright (47, 48)$ 
21:       $r_{\text{vio}}, \kappa_{\text{vio}} \leftarrow \text{VIOLATION}(\hat{w})$   $\triangleright (49, 50)$ 
22:    end
23:     $w \leftarrow \hat{w}$ 
24:  end
25:   $\partial w^*/\partial \theta \leftarrow -\bar{R}^{-1}(w^*; \theta)\bar{D}(w^*; \theta)$   $\triangleright (11)$ 
26:  Return  $w, \partial w^*/\partial \theta$ 

```

contact forces can occur at a distance. The primal-dual interior-point solver described in the next section adaptively decreases this parameter in order to efficiently and reliably converge to hard contact solutions. In practice, the engine is set to converge to small values (i.e. $\kappa \rightarrow 0$) for simulation in order to simulate accurate physics. Intermediate solutions (i.e., $\kappa > 0$) are cached and later utilized to compute smooth gradients in order to provide useful information through contact events.

B. Primal-Dual Interior-Point Solver

To efficiently and reliably satisfy (26), we developed a custom primal-dual interior-point solver for NCPs with support for cone constraints and quaternions. The algorithm is largely based upon Mehrotra's predictor-corrector algorithm [81, 82], while implementing non-Euclidean optimization techniques to handle quaternions [83] and borrowing features from CVX-OPT [80] to handle cones.

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

The solver functions as a systematic procedure that iteratively refines the solution while maintaining feasibility within the prescribed cone constraints. The primal-dual framework provides a structured pathway to the solution, and the interplay

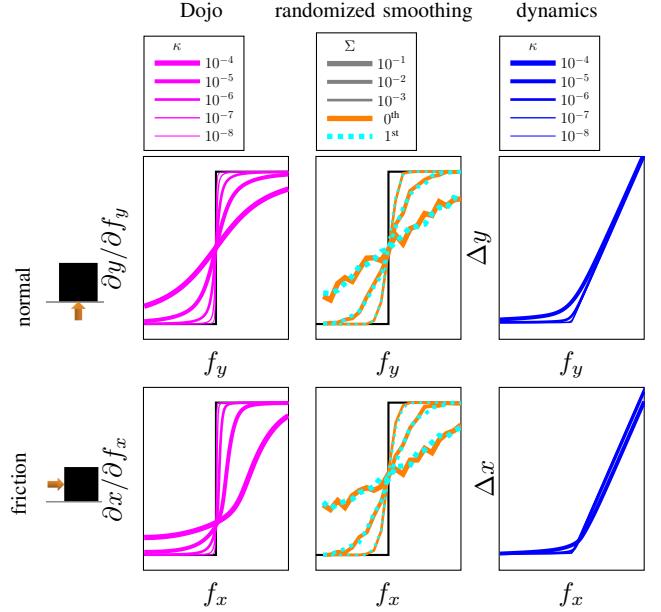


Fig. 4: Gradient and dynamics comparison between point-wise gradients (black), randomized-smoothing gradients [29] (orange, blue) and Dojo's analytic gradients (magenta). The dynamics for a box in the XY plane that is resting on a flat surface and displaced an amount Δ by a force f (top left). Randomized smoothing gradients (right column) are computed using 500 samples with varying covariances Σ . Dojo's gradients (middle column) are computed for different values of central-path parameter κ . Compared to Dojo, the randomized smoothing method produces noisy derivatives that are many times more expensive to compute. Simulated dynamics comparisons were conducted using Dojo's results under varying values of the central-path parameter, κ . The dynamics involve a box of 1 kg mass resting on a flat surface in the XY plane, displaced by a force Δ (top left). The applied force was gradually increased from 0 N to 20 N. Throughout the simulation process, considering two separate cases—one involving impact and the other involving friction—the maximum Δ_x and Δ_y differences between $\kappa = 1 \times 10^{-4}$ and $\kappa = 1 \times 10^{-8}$ were 1.52×10^{-3} m and 2.56×10^{-3} m, respectively.

between the affine (predictor) and corrector steps ensures steady progress towards eliminating both constraint and complementarity violations. The analytical line search offers a principled means of selecting step sizes that keep the iterates strictly within the cone, while the specialized handling of non-Euclidean variables ensures stable and accurate updates. By continuously monitoring violations and adjusting parameters accordingly, the approach reliably converges to a solution that meets predefined tolerances. As a result, this solver is capable of addressing a broad range of problems—from those with simple linear conditions to complex, nonlinear scenarios—while providing accurate solutions and informative gradients.

1) *Problem formulation:* The solver aims to satisfy instantiations of the following problem

$$\begin{aligned}
 & \text{find} && a, b, c \\
 & \text{subject to} && E(a, b, c; \theta) = 0, \\
 & && b \circ c = \kappa \mathbf{e}, \\
 & && b, c \in \mathcal{K},
 \end{aligned} \tag{27}$$

TABLE II: Contact violation for Atlas drop (Fig. 1). Comparison between Dojo and MuJoCo for foot contact penetration (millimeters) with the floor for different time steps (seconds). Dojo strictly enforces no penetration. When Atlas lands, its feet remains above the ground by an infinitesimal amount. In contrast, MuJoCo exhibits significant penetration through the floor (i.e., negative values).

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

with decision variables $a \in \mathbf{R}^{n_a}$ and $b, c \in \mathbf{R}^{n_K}$, equality-constraint set $E : \mathbf{R}^{n_a} \times \mathbf{R}^{n_K} \times \mathbf{R}^{n_K} \times \mathbf{R}^{n_\theta} \rightarrow \mathbf{R}^{n_a+n_K}$, problem data $\theta \in \mathbf{R}^{n_\theta}$; and where K is the Cartesian product of positive-orthant and second-order cones [84].

Interior-point methods aim to satisfy a sequence of relaxed problems with $\kappa > 0$ and $\kappa \rightarrow 0$ in order to reliably converge to a solution of the original problem (i.e., $\kappa = 0$). This continuation approach, though it makes our interior point solver hard to warm-start, helps avoid premature ill-conditioning and is the basis for numerous convex and nonconvex interior-point solvers [82].

The LCP formulation is a special-case instantiation of (27) where the constraint set is affine in the decision variables and the cone is the positive orthant. Most general-purpose solvers for LCP problems rely on active-set methods that strictly enforce $\kappa = 0$ at each iteration. Consequently, these solvers generate non-informative gradient information (see Section IV-C). In contrast our interior point solver can give informative gradients with a smoothing effect related to the size of κ .

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

$$r(w; \theta, \kappa) = \begin{bmatrix} E(w; \theta) \\ b^{(1)} \circ c^{(1)} - \kappa \mathbf{1} \\ \vdots \\ b^{(n_K)} \circ c^{(n_K)} - \kappa \mathbf{e} \end{bmatrix}, \quad (28)$$

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

$$R(w; \theta) = \frac{\partial r(w; \theta, \cdot)}{\partial w}, \quad (29)$$

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

$$D(w; \theta) = \frac{\partial r(w; \theta, \cdot)}{\partial \theta}, \quad (30)$$

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

The non-Euclidean properties of quaternion variables are handled with modifications to these Jacobians (29) and (30) by right multiplying each with a matrix H containing attitude Jacobians [83] corresponding to the quaternions in x and θ , respectively

$$\bar{R}(w; \theta) = R(w; \theta)H_R(w), \quad (31)$$

$$\bar{D}(w; \theta) = D(w; \theta)H_D(\theta). \quad (32)$$

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

3) *Cones*: The generalized inequality, cone-product operator, and target for the n -dimensional positive orthant are

$$\mathbf{R}_{++}^n = \{a \in \mathbf{R}^n \mid a_{(i)} > 0, i = 1, \dots, n\}, \quad (33)$$

$$a \circ b = (a_{(1)}b_{(1)}, \dots, a_{(n)}b_{(n)}), \quad (34)$$

$$\mathbf{e} = \mathbf{1}. \quad (35)$$

For the second-order cone they are

$$\mathcal{Q}^n = \{(a_{(1)}, a_{(2:n)}) \in \mathbf{R} \times \mathbf{R}^{n-1} \mid \|a_{(2:n)}\|_2 \leq a_{(1)}\}, \quad (36)$$

$$a \circ b = (a^T b, a_{(1)}b_{(2:n)} + a_{(1)}b_{(2:n)}), \quad (37)$$

$$\mathbf{e} = (1, 0, \dots, 0). \quad (38)$$

The solver utilizes the Cartesian product

$$\mathcal{K} = \mathbf{R}_+^n \times \mathcal{Q}_1^{l_1} \times \dots \times \mathcal{Q}_j^{l_j}, \quad (39)$$

of the n -dimensional positive orthant and j second-order cones, each of dimension l_i .

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

$$y \leftarrow y + \alpha \Delta, \quad (40)$$

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

$$\alpha = \min \left(1, \max_{k|\Delta_{(k)} < 0} \left\{ -\frac{y_{(k)}}{\Delta_{(k)}} \right\} \right), \quad (41)$$

and second-order cone

$$\nu = y_{(1)}^2 - y_{(2:k)}^T y_{(2:k)}, \quad (42)$$

$$\zeta = y_{(1)} \Delta_{(1)} - y_{(2:k)}^T \Delta_{(2:k)}, \quad (43)$$

$$\rho_{(1)} = \frac{\zeta}{\nu}, \quad (44)$$

$$\rho_{(2:k)} = \frac{\Delta_{(2:k)}}{\sqrt{\nu}} - \frac{\frac{\zeta}{\sqrt{\nu}} + \Delta_{(1)}}{y_{(1)} \sqrt{\nu} + \nu} y_{(2:k)}, \quad (45)$$

$$\alpha = \begin{cases} \min \left(1, \frac{1}{\|\rho_{(2:k)}\|_2 - \rho_{(1)}} \right), & \|\rho_{(2:k)}\|_2 > \rho_{(1)}, \\ 1, & \text{otherwise.} \end{cases} \quad (46)$$

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

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

$$a^{(1)} \leftarrow a^{(1)} + \alpha \Delta^{(1)}. \quad (47)$$

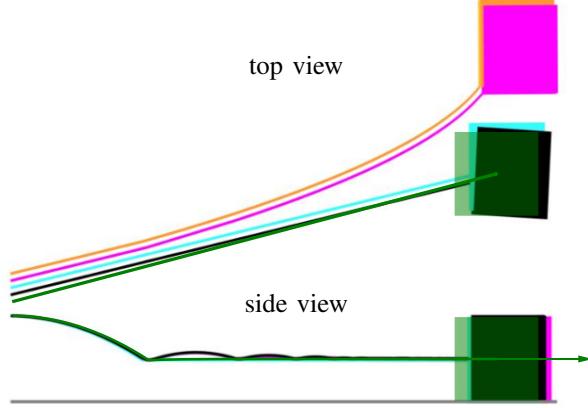


Fig. 5: Velocity drift resulting from friction-cone approximation. Comparison between a box sliding with approximate cones having four vertices implemented in MuJoCo (magenta) and Dojo (orange) versus MuJoCo’s (black) and Dojo’s (blue) and PyBullet’s (green) nonlinear friction cones. Dojo’s nonlinear friction cone gives the physically correct straight line motion, while linear friction-cone approximations lead to lateral drift. MuJoCo’s nonlinear friction cone exhibits a minor rotational drift. PyBullet’s nonlinear friction cone delivers correct straight line motion, but exerts greater friction force on the box leading to a shorter trajectory than Dojo and MuJoCo.

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

$$a^{(i)} \leftarrow L(a^{(i)})\varphi(\alpha\Delta^{(i)}), \quad (48)$$

where $L : \mathbf{H} \rightarrow \mathbf{R}^{4 \times 4}$ is a matrix representing a left-quaternion matrix multiplication, and $\varphi : \mathbf{R}^3 \rightarrow \mathbf{H}$ is a mapping to a unit quaternion. The standard update (47) is used for the remaining decision variables b and c .

6) *Violation metrics:* Two metrics are used to measure progress: (i) the constraint violation

$$r_{\text{vio}} = \|r(w; \theta)\|_\infty, \quad (49)$$

and (ii) complementarity violation

$$\kappa_{\text{vio}} = \max_i \{\|b^{(i)} \circ c^{(i)}\|_\infty\}. \quad (50)$$

The problem (27) is considered solved when $r_{\text{vio}} < r_{\text{tol}}$ and $\kappa_{\text{vio}} < \kappa_{\text{tol}}$.

7) *Centering:* The solver adaptively relaxes (27) by computing the centering parameters μ and σ . These values provide an estimate of the cone-constraint violation and determine the value of the central-path parameter that a correction step will aim to satisfy. These values rely on the degree of the cone [80],

$$\deg(\mathcal{K}) = \sum_{i=1}^{n\kappa} \deg(\mathcal{K}^{(i)}) = \dim(\mathcal{K}^{(1)}) + n\kappa - 1, \quad (51)$$

the complementarity violations,

$$\mu = \frac{1}{\deg(\mathcal{K})} \sum_{i=1}^{n\kappa} (b^{(i)})^T c^{(i)}, \quad (52)$$

and affine complementarity violations,

$$\mu^{\text{aff}} = \frac{1}{\deg(\mathcal{K})} \sum_{i=1}^{n\kappa} (b^{(i)} + \alpha\Delta b^{(i)})^T (c^{(i)} + \alpha\Delta c^{(i)}), \quad (53)$$

as well as their ratio,

$$\sigma = \min \left(1, \max \left(0, \frac{\mu^{\text{aff}}}{\mu} \right) \right)^3. \quad (54)$$

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

8) *Algorithm:* The interior-point algorithm used to solve (27) is summarized in Algorithm 2. Additional tolerances $\tau \in [0.9, 1]$ are used to improve numerical reliability of the solver. The algorithm parameters include $\tau_{\text{max}}^{\text{SOC}}$ to prevent the iterates from reaching the boundaries of the cones too rapidly during the solve, τ_{min} to ensure we are aiming at sufficiently large steps, and β is the decay rate of the step size α during the line search. In practice, r_{tol} and κ_{tol} are the only parameters the user might want to tune.

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

For an instance of problem (27), the algorithm is provided problem data and an initial point, which is projected to ensure that the cone variables are initially feasible with some margin. Next, an affine search direction (i.e., predictor) is computed that aims for zero complementarity violation. Using this direction, a cone line search is performed followed by a centering step that computes a target relaxation for the computation of the corrector search direction. A second cone line search is then performed for this new search direction. A subsequent line search is performed until either the constraint or complementarity violation is reduced. The current point is then updated, a new affine search direction is computed, and the procedure repeats until the violations satisfy the solver tolerances.

C. Gradients

Dojo simulates a user-tunable smoothed approximation of hard contact dynamics. This approach allows us to compute gradients that are more informative in the presence of contacts by enabling a force-from-a-distance mechanism, as discussed in references [85] and [29]. As previously discussed, interior-point methods optimize a sequence of smooth barrier subproblems, where the degree of smoothing is parameterized by the central-path parameter κ . Differentiating at a large value of κ gives more contact smoothing, with more informative gradients but less accurate solutions, while differentiating at small κ values gives less smoothing, less informative gradients, but better physical fidelity. The chosen intermediate solution, $w^*(\theta, \kappa > 0)$, is differentiated using the implicit function theorem (11) to compute smooth *implicit gradients*. In practice, we find that these gradients greatly improve the performance of gradient-based optimization methods, consistent with the long history of interior-point methods. Dojo’s gradients are

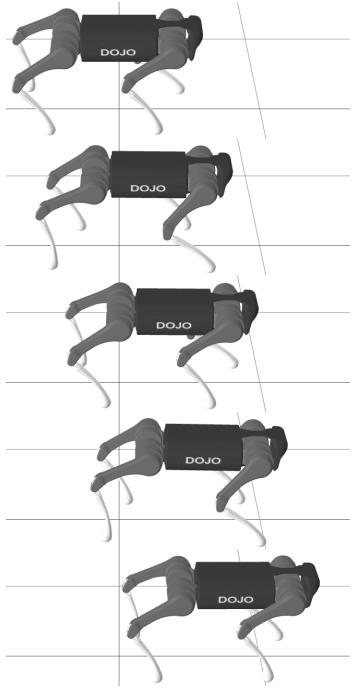


Fig. 6: Locomotion plan for quadruped generated using trajectory optimization. Time progresses top to bottom.

compared with point-wise gradients and randomized smoothing in Fig. 4. Since κ represents a tradeoff between simulation accuracy and gradient smoothness, we evaluate the effect of κ on simulation accuracy. We compare the simulation results of different κ values in Fig. 4.

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

D. Implementation

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

<https://www.github.com/dojo-sim/Dojo.jl>.

V. RESULTS

Dojo’s capabilities are highlighted through a collection of examples, including: simulating physical phenomena, gradient-based planning with trajectory optimization, policy optimization, system identification, and sim-to-real gap evaluation with robot hardware. The current implementation supports point, sphere, and capsule collisions with flat surfaces with a pre-existing collision detection module (which is outside the scope of this work). All of the experiments were performed on a computer with an Intel Core i9-10885H processor and 32GB of memory.

A. Simulation

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

TABLE III: Planning results. Comparison of final cost value, goal constraint violation, and total number of iterations for a collection of systems optimized with iterative LQR [86] using Dojo (D) with implicit gradients or MuJoCo (M) with finite-difference gradients.

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

Impact constraints comparison: The Atlas humanoid is simulated dropping onto a flat surface (Fig. 1). The system comprises 31 bodies, resulting in 403 maximal-coordinates states, and has 36 actuated degrees-of-freedom. Each foot has four contact points. A comparison with MuJoCo is performed measuring penetration violations with the floor for different simulation rates (Table II). The current implementation of Dojo simulates this system in real time at 65 Hz.

Friction-cone comparison: The effect of friction-cone approximation is demonstrated by simulating a box that is initialized with lateral velocity before impacting and sliding along a flat surface. The complementarity problem with P contact points requires $2P(1+2d)$ decision variables for contact and a corresponding number of constraints, where d is the degree of parameterization (e.g., double parameterization: $d = 2$). For a pyramidal approximation, in the probable scenario where its vertices are not aligned with the direction of motion, velocity drift occurs for a linearized cone implemented in Dojo and MuJoCo. Meanwhile, though MuJoCo and PyBullet also can use nonlinear friction cones, compared to Dojo, MuJoCo’s nonlinear friction cone exhibits a minor rotational drift and PyBullet’s nonlinear friction cone exerts greater friction force on box that leads to a shorter trajectory (Fig. 5). While it is possible to reduce such artifacts by increasing the number of vertices in the approximation of the second-order cone, this increases the computational complexity. Such approximation is unnecessary in Dojo as we handle the exact nonlinear cone constraint efficiently and reliably with optimization tools from cone programming; the result is accurate sliding.

Simulation Stability at Low Frequencies To validate the stability of the primal-dual interior point solver under different simulation frequencies, we conduct Atlas drop (Fig. 1) and similar quadrupedal drop experiments under different simulation frequencies. During the simulations, we recorded the robots’ torso heights over time. The simulation results can be found in Fig. 7. Under a large span of simulation frequency from 20-500 Hz, both Atlas drop and quadrupedal drop deliver similar simulation results with small reasonable deviations. The test results demonstrate that Dojo preserves simulation fidelity at low frequency, even through contact events.

Computation time: One of the primary challenges with differentiable simulators is their computation speed. Table V presents a benchmark comparing the computation time for forward simulation and gradient calculation across four simulators on four different robot types. Each test involved

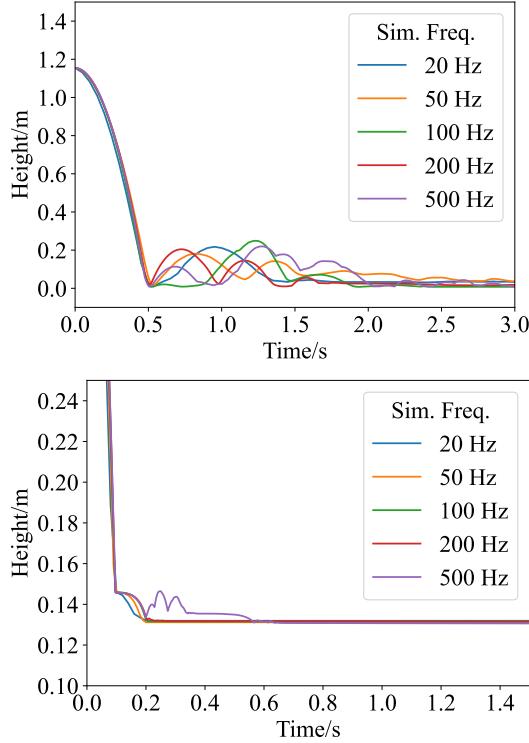


Fig. 7: Torso height over time under different simulation frequency for Atlas (top) drop and A1 (bottom) drop experiments.

simulating 1000 steps with a time step of 0.01s, using randomly generated actions. MuJoCo’s gradients are calculated with built-in finite differentiation function, Drake utilizes its randomized smoothing gradients calculation function, while Brax calculates gradients through auto-differentiation.

Among all four simulators, MuJoCo exhibited a clear advantage in computation speed. However, Dojo significantly outperforms another differentiable simulator, Brax, and delivers a comparable performance as Drake. This advantage stems from (i) the maximal-coordinate dynamics model, which is highly effective in handling complex systems with multiple links and joints, as discussed in Section III-C, and (ii) Dojo’s implicit differentiation method, which avoids the $O(n^2)$ complexity of finite difference techniques in the action and observation spaces.

Sim-to-Real gap evaluation: To assess the sim-to-real transfer capabilities and fidelity of Dojo, we conducted a series of box pushing experiments using a 6-axis xArm manipulator and a 0.5 kg rectangular box (11 cm by 13 cm by 21 cm) positioned at initial x-axis distances of 30, 35, 40, 45, and 50 cm from the manipulator’s base, the simulation and real experimental scenario can be seen in Fig. 8. In both the simulated and real-world trials, identical proportional-derivative (PD) gains and joint-space commands were applied. We record the box’s final location and flipping status of each experiment (real world) or simulation (Dojo), the results can be found in Table IV. After the pushing action, the positional discrepancy in the box’s final location between simulation and reality averaged approximately 0.5 cm (1.25%). Apart from positional

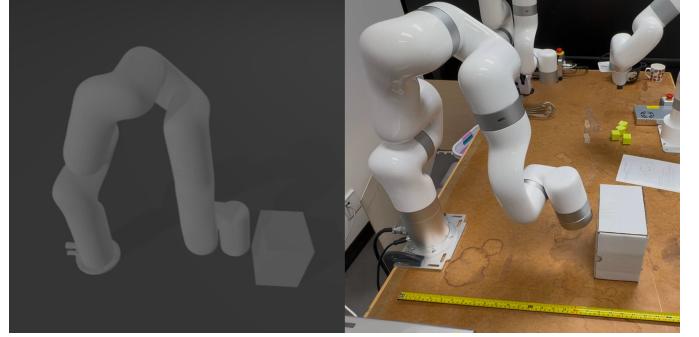


Fig. 8: Simulation and real robot comparison under physical contacts and frictions. Robot arm pushing box experiment scenarios.

TABLE IV: Robot arm pushing box experiment results for Sim-to-Real gap evaluation. Box was placed at different locations on the table with different initial x-direction distances to robot arm’s base. Both simulation and real robot are controlled by the same PD controller to track the same joint space at 100 Hz.

	Initial Distance (cm)	30	35	40	45	50
Real	Box Flip	Yes	Yes	Yes	Yes	No
	Final Distance (cm)	46.9	52.7	55.1	61.2	50
Sim	Box Flip	Yes	Yes	Yes	Yes	No
	Final Distance (cm)	45.7	52.3	55.2	62	50
Sim-to-Real Distance Gap (cm)		1.2	0.4	0.1	0.8	0

accuracy, we also examined the flipping behavior of the box, which is sensitive to the precise point of contact and influenced by complex frictional and contact forces. Dojo’s predictions of flipping outcomes closely matched those observed in the physical experiments, indicating that the simulator can capture the subtle and intricate dynamics involved in frictional contact scenarios. Taken together, these results give evidence of Dojo’s ability to reproduce physical phenomena, thereby supporting robust sim-to-real transfer in robotic manipulation tasks.

Convergence study: Dojo’s solver reduces the constraint violation r_{vio} and complementarity violation κ_{vio} until both residual values are smaller than prescribed tolerances. As the problem is nonconvex, it is important to analyze Dojo’s convergence performance across different robots under different conditions. We simulate three robots in Dojo and record their κ_{vio} and r_{vio} as well as the solver’s condition numbers over iterations under different tolerance settings. Meanwhile, to further substantiate Dojo’s ability to avoid ill-conditioning, we also compare Dojo with a primal-only interior point solver [28] on condition numbers over iterations under different tolerance settings. The convergence study results are shown in Fig. 9 and 10.

When testing with different κ_{tol} , we fix r_{tol} at 1×10^{-8} , while as κ_{vio} is the main bottleneck of the solver, we relax κ_{tol} to 0.1 when testing with different r_{tol} . We do the same for both residual values and condition number experiments. The experiment results demonstrate that Dojo’s primal-dual interior-point solver can converge within 15 iterations for all three robots. The residual decreases reliably for different tolerance settings, showing that Dojo’s solver has strong numerical stability. For condition number experiments, the

TABLE V: Computation time benchmark results. Comparison of computation time of forward simulation plus gradient calculation for different simulators on different types of robots. Simulation time step is 0.01 s, each test was simulated for 1000 steps, with randomly generated actions. MuJoCo’s gradients are calculated with built-in finite differentiation function, Drake utilizes its randomized smoothing gradients calculation function, while Brax calculates gradients through auto-differentiation.

Simulator	Simulation Time of Different Robots [s]			
	Humanoid	Unitree A1	Franka Panda	Skydio X2
MuJoCo	1.512 \pm 0.045	1.114 \pm 0.007	0.335 \pm 0.001	0.047 \pm 0.002
Drake	5.463 \pm 0.078	3.870 \pm 0.024	2.352 \pm 0.055	0.571 \pm 0.011
Brax	6.975 \pm 0.485	11.064 \pm 0.521	10.954 \pm 0.395	7.953 \pm 0.484
Dojo	1.750 \pm 0.135	5.235 \pm 0.071	1.159 \pm 0.077	0.807 \pm 0.003

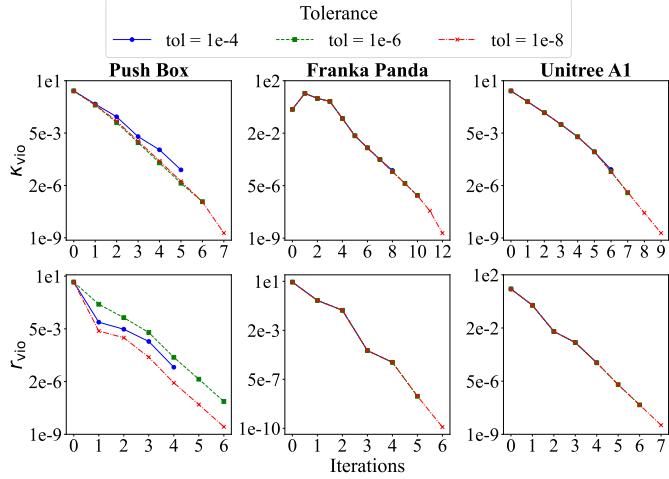


Fig. 9: Plots of residual values κ_{vio} and r_{vio} versus iteration under different tolerance for three different robots.

general trend is that condition number increases with the iteration for both methods, as expected. Specifically, for Dojo, under relaxed κ_{tol} (0.1), the condition number stays low (smaller than 1×10^{-4}) with iterations for all three robots. Meanwhile, under rigorous κ_{tol} (1×10^{-8}), the condition number was higher than that under relaxed κ_{tol} , but still lies at a reasonable level for a nonconvex optimization problem (smaller than 1×10^9). Moreover, the primal-only solver’s condition number is much higher than the proposed method. Under different tolerance settings, at the last iteration of each experiment, the primal method’s condition number is 2 to 1×10^4 times of Dojo’s condition number, showing that Dojo’s primal-dual interior-point solver has an advantage in avoiding numerical ill-conditioning.

B. Planning

We utilize iterative LQR by providing implicit gradients from Dojo [87] to perform trajectory optimization on three systems: planar box, hopper, and quadruped. A comparison is performed with MuJoCo and finite-difference gradients. The results are visualized for the quadruped in Fig. 6 and summarized for all of the systems in Table III.

Box: Inputs are optimized to move a stationary rigid body that is resting on a flat surface (Fig. 4) to a goal location that is either to the right or up in the air 1 meter. The planning horizon is 1 second and the controls are initialized with zeros.

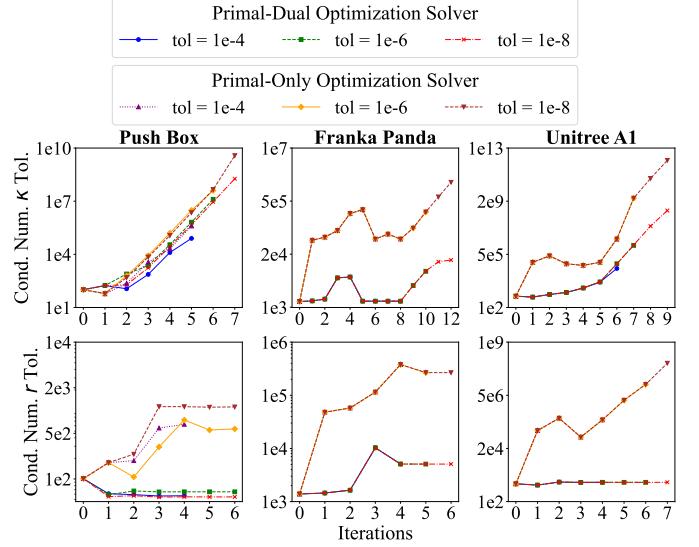


Fig. 10: Plots of condition numbers versus iteration under different κ and r tolerance for three different robots.

Dojo uses a time step $h = 0.1$, whereas MuJoCo uses $h = 0.01$ to prevent significant contact violations with the floor. MuJoCo fails in the scenario with the goal in the air, while Dojo succeeds at both tasks.

Hopper: The hopping robot [88] with $m = 3$ controls and $n = 14$ degrees-of-freedom is tasked with moving to a target pose over 1 second. Similar, although not identical, models and costs are used. Dojo uses a time step $h = 0.05$ whereas MuJoCo uses $h = 0.01$. The hopper is initialized with controls that maintain its standing configuration. Quadratic costs are used to penalize control effort and perform cost shaping on an intermediate state in the air and the goal pose. The optimizer typically finds a single-hop motion.

Quadruped: The Unitree A1 with $m = 12$ controls and $n = 36$ degrees-of-freedom is tasked with moving to a goal location over a planning horizon $T = 41$ with time step $h = 0.05$. Controls are initialized to compensate for gravity and there are costs on tracking a target kinematic gait and control inputs. The optimizer finds a dynamically feasible motion that closely tracks the kinematic plan (Fig. 6).

Overall, we find that final results from both engines are similar. However, importantly, MuJoCo is enforcing soft contact whereas Dojo simulates hard contact. Dojo’s gradients are computed with $\kappa = 3e-4$. Further, for systems with contact, MuJoCo requires a time step $h = 0.01$ for successful

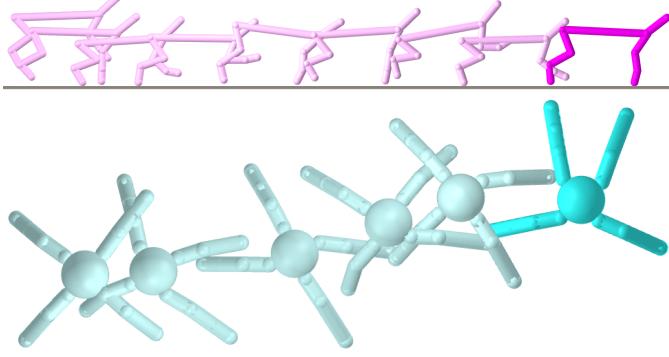


Fig. 11: Learned policy rollouts for half-cheetah (top) and ant (bottom). Time progresses left to right.

optimization, whereas Dojo succeeds with $h = 0.05$.

C. Policy Optimization

Gym-like environments [20, 89]: ant and half-cheetah are implemented in Dojo and we train static linear policies for locomotion. As a baseline, we employ Augmented Random Search (ARS) [90], a gradient-free approach coupling random search with a number of simple heuristics. For comparison, we train the same policies using augmented gradient search (AGS) which replaces the stochastic-gradient estimation of ARS with Dojo’s implicit gradients. Policy rollouts are visualized in Fig. 11 and results are summarized in Table VI.

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

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

First, we are able to successfully train policies using this simple learning algorithm in Dojo’s hard contact environments. Second, MuJoCo requires smaller $h = 0.01$ time steps for stable simulation, whereas Dojo is stable with $h = 0.05$. Third, our initial results indicate that it is possible to train comparable polices in Dojo with 5 to 10 times less samples by utilizing implicit gradients compared to the gradient-free method.

D. System Identification

System identification is performed on an existing real-world dataset of trajectories collected by throwing a box on a table with different initial conditions [91]. We learn a set of parameters $\theta = (c_f, p^{(1)}, \dots, p^{(8)})$ that include the friction coefficient c_f , and 3-dimensional vectors $p^{(i)}$ that represent the position of vertex i of the box with respect to its center of mass.

Each trajectory is decomposed into $T - 2$ triplets of consecutive configurations: $Z = (z_-, z, z_+)$, where T is the number of time steps in the trajectory. Using the initial conditions z_- , z from a tuple, and an estimate of the system’s parameters θ ,

TABLE VI: Policy optimization results. Comparison of total reward, number of simulation-step and gradient evaluations for a collection of policies trained with Augmented Random Search (ARS) [90] and Augmented Gradient Search (AGS). For AGS, we test with both implicit gradients and randomized gradients. The results are averaged over the best 3 out of 5 runs with different random seeds. Optimizing with implicit gradients from Dojo reaches similar performance levels while being 5 to 10 times more sample efficient, while optimizing with randomized finite difference gradients is 3 to 5 times more sample efficient.

Environment	Method	Reward	Simulation Evals
Half-Cheetah	ARS	46 ± 24	3e+4
	AGS (Dojo)	44 ± 24	5e+3
	AGS (random)	39 ± 27	8e+3
Ant	ARS	64 ± 15	2e+5
	AGS (Dojo)	54 ± 28	2e+4
	AGS (random)	59 ± 24	4e+4

Dojo performs one-step simulation to predict the next state, \hat{z}_+ . Implicit gradients are utilized by a Gauss-Newton method to perform gradient-based learning of the system parameters.

The parameters are learned by minimizing the following loss:

$$\mathcal{L}(\mathcal{D}, \theta) = \sum_{Z \in \mathcal{D}} L(Z, \theta) = \sum_{Z \in \mathcal{D}} \frac{1}{2} \|\mathbf{Dojo}(z_-, z; \theta) - z_+\|_W^2, \quad (55)$$

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

$$\frac{\partial L}{\partial \theta} = \frac{\partial \mathbf{Dojo}}{\partial \theta}^T W (\mathbf{Dojo}(z_-, z; \theta) - z_+), \quad (56)$$

and approximate Hessians

$$\frac{\partial^2 L}{\partial \theta^2} \approx \frac{\partial \mathbf{Dojo}}{\partial \theta}^T W \frac{\partial \mathbf{Dojo}}{\partial \theta}. \quad (57)$$

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

After training, the learned parameters are within 5% of the true geometry and friction coefficient for the box from the dataset. We complete the real-to-sim transfer and simulate the learned system in Dojo, comparing it to the ground-truth dataset trajectories. Results are visualized in Fig. 12.

VI. CONCLUSION

Dojo is designed from physics- and optimization-first principles to enable better gradient-based optimization for planning, control, policy optimization, and system identification.

A. Contributions

The engine makes several advancements over previous state-of-the-art engines for robotics: First, the variational integrator enables stable simulation at low sample rates. Second, the contact model includes an improved friction model that eliminates artifacts like creep, particularly for sliding, and hard contact for impact is achieved to machine precision. This enables sim-to-real transfer for implementation on real robot hardware. The underlying primal-dual interior point solver,

developed specifically for solving NCPs, is numerically robust and minimizes user hyperparameter tuning, while offering good performance across numerous systems, and handling cone and quaternion variables. Third, the engine efficiently returns implicit gradients whose smoothness through contact are tuned by the user to trade off gradient smoothness with simulation accuracy, providing useful information through contact events. Fourth, in addition to building and providing Dojo as an open-source tool, the physics and optimization algorithms presented can be ported into existing simulation engines.

B. Limitations

In terms of features, reliability, and wall-clock time, MuJoCo—the product of a decade of excellent software engineering—is impressive. As development of Dojo continues, we expect to make significant progress in all of these areas. However, fundamentally, Dojo’s approach of solving an NCP with a primal-dual interior point method requires more computation per time step compared to existing simulators that use a soft-contact model (e.g. MuJoCo and Drake), but allows for accurate simulation with a lower sample rate, making wall-clock comparisons between the two simulators difficult. This is the fundamental trade-off Dojo makes for robotics applications: greater computational cost per time step for accurate physics and smooth gradients over fewer total time steps.

Additionally, it should be acknowledged that Dojo’s interior point solver is solving a nonconvex optimization problem at each simulation time step, which may have a danger of reaching poor local minima or not converging within the allotted time window. Although in practice, we do not find this to be a problem, but for time- or safety-critical applications this should be a consideration. Moreover, the same is true in using Dojo’s gradients for trajectory optimization, policy optimization, or system identification. These require solving inherently nonconvex optimization problems, which may converge to poor local solutions, or fail to converge, regardless of the smoothness or quality of Dojo’s gradient information. Although smoother gradients may facilitate optimization for these problems, they do not fully resolve the complexities introduced by nonconvex optimization landscapes.

C. Future Work

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

Perhaps the most important remaining question is whether the physics and optimization improvements from this work

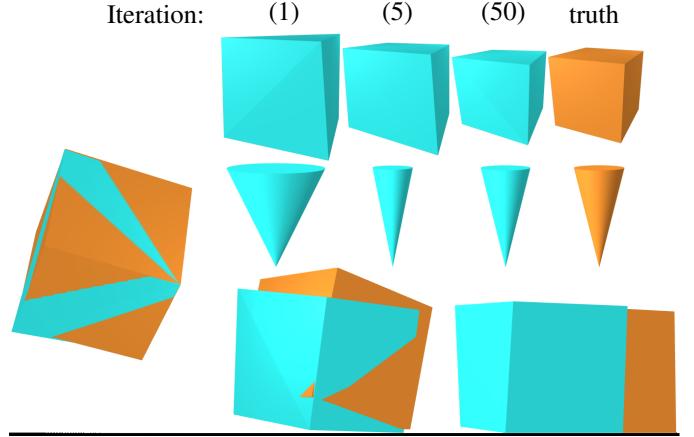


Fig. 12: System identification. Top right: Learning box geometry and friction cone to less than 5% error. Bottom: Simulated trajectory of the box using the learned properties (blue) compared to ground truth (orange).

translate into better transfer of simulation results to successes on real-world robotic hardware. In this thrust, future work will explore the transfer of control policies trained in Dojo to hardware and deployment of the engine in model predictive control frameworks.

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

ACKNOWLEDGEMENTS

The authors would like to thank Suvansh Sanjeev for assistance with the Python interface. Toyota Research Institute provided funds to support this work.

REFERENCES

- [1] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozeffowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.
- [2] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, “Solving Rubik’s cube with a robot hand,” *arXiv:1910.07113*, 2019.
- [3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadrupedal locomotion over challenging terrain,” *Science Robotics*, vol. 5, no. 47, 2020.
- [4] A. Kumar, Z. Fu, D. Pathak, and J. Malik, “RMA: Rapid motor adaptation for legged robots,” *arXiv:2107.04034*, 2021.
- [5] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

- [6] W. Zhao, J. P. Queraltà, and T. Westerlund, “Sim-to-real transfer in deep reinforcement learning for robotics: a survey,” in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.
- [7] R. Tedrake and the Drake Development Team, “Drake: Model-based design and verification for robotics,” 2019.
- [8] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, “Brax—A differentiable physics engine for large scale rigid body simulation,” *arXiv:2106.13281*, 2021.
- [9] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, “Fast and feature-complete differentiable physics for articulated rigid bodies with contact,” *arXiv:2103.16021*, 2021.
- [10] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, “ADD: Analytically differentiable dynamics for multi-body systems with frictional contact,” *ACM Transactions on Graphics*, vol. 39, no. 6, pp. 1–15, 2020.
- [11] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, “DiffTaichi: Differentiable programming for physical simulation,” *International Conference on Learning Representations*, 2020.
- [12] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, “NeuralSim: Augmenting differentiable simulators with neural networks,” *IEEE International Conference on Robotics and Automation*, pp. 9474–9481, 2021.
- [13] NVIDIA Corporation, “Isaac Sim: Simulation for Robotics,” 2025. Accessed: 2025-02-19.
- [14] G. Authors, “Genesis: A universal and generative physics engine for robotics and beyond,” December 2024.
- [15] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, “Unified particle physics for real-time applications,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–12, 2014.
- [16] N. Agarwal, A. Ali, M. Bala, Y. Balaji, E. Barker, T. Cai, P. Chattopadhyay, Y. Chen, Y. Cui, Y. Ding, et al., “Cosmos world foundation model platform for physical ai,” *arXiv preprint arXiv:2501.03575*, 2025.
- [17] G. Zhou, H. Pan, Y. LeCun, and L. Pinto, “Dino-wm: World models on pre-trained visual features enable zero-shot planning,” *arXiv preprint arXiv:2411.04983*, 2024.
- [18] M. Parmar, M. Halm, and M. Posa, “Fundamental challenges in deep learning for stiff contact dynamics,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5181–5188, 2021.
- [19] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.
- [20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” *arXiv:1606.01540*, 2016.
- [21] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake, “Do differentiable simulators give better policy gradients?,” in *International Conference on Machine Learning*, pp. 20668–20696, PMLR, 2022.
- [22] P. Parmas, C. E. Rasmussen, J. Peters, and K. Doya, “Pipps: Flexible model-based policy search robust to the curse of chaos,” in *International Conference on Machine Learning*, pp. 4065–4074, PMLR, 2018.
- [23] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman, “Gradients are not all you need,” *arXiv preprint arXiv:2111.05803*, 2021.
- [24] E. Todorov, “Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo,” in *IEEE International Conference on Robotics and Automation*, pp. 6054–6061, 2014.
- [25] Legged Robotics Group, “SimBenchmark: Benchmarking Simulation Performance for Legged Robots,” 2025. Accessed: 2025-02-17.
- [26] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.
- [27] K. H. Hunt and F. R. E. Crossley, “Coefficient of restitution interpreted as damping in vibroimpact,” 1975.
- [28] A. M. Castro, F. N. Permenter, and X. Han, “An unconstrained convex formulation of compliant contact,” *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1301–1320, 2022.
- [29] H. J. T. Suh, T. Pang, and R. Tedrake, “Bundled gradients through contact via randomized smoothing,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4000–4007, 2022.
- [30] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2149–2154, 2004.
- [31] E. Coumans and Y. Bai, “PyBullet, a Python module for physics simulation for games, robotics and machine learning,” 2016–2019.
- [32] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, “DART: Dynamic animation and robotics toolkit,” *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.
- [33] J. K. Murthy, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben, et al., “gradsim: Differentiable simulation for system identification and visuomotor control,” in *International conference on learning representations*, 2020.
- [34] E. Heiden, M. Macklin, Y. Narang, D. Fox, A. Garg, and F. Ramos, “Disect: A differentiable simulation engine for autonomous robotic cutting,” *arXiv preprint arXiv:2105.12244*, 2021.
- [35] Nvidia, “PhysX physics engine,” 2022.
- [36] R. Newbury, J. Collins, K. He, J. Pan, I. Posner, D. Howard, and A. Cosgun, “A review of differentiable simulators,” *IEEE Access*, 2024.
- [37] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, “Differentiable physics and stable modes for tool-use and manipulation planning,” 2018.
- [38] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, “End-to-end differentiable

- physics for learning and control,” *Advances in neural information processing systems*, vol. 31, 2018.
- [39] C. Song and A. Boularias, “Learning to slide unknown objects with differentiable physics simulations,” *arXiv preprint arXiv:2005.05456*, 2020.
- [40] C. Song and A. Boularias, “Identifying mechanical models through differentiable simulations,” *arXiv preprint arXiv:2005.05410*, 2020.
- [41] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels, “A differentiable physics engine for deep learning in robotics,” *Frontiers in neurorobotics*, vol. 13, p. 6, 2019.
- [42] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum, “Learning to see physics via visual de-animation,” *Advances in neural information processing systems*, vol. 30, 2017.
- [43] V. L. Guen and N. Thome, “Disentangling physical dynamics from unknown factors for unsupervised video prediction,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11474–11484, 2020.
- [44] C. Schenck and D. Fox, “Spnets: Differentiable fluid dynamics for deep neural networks,” in *Conference on Robot Learning*, pp. 317–335, PMLR, 2018.
- [45] M. Jaques, M. Burke, and T. Hospedales, “Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video,” *arXiv preprint arXiv:1905.11169*, 2019.
- [46] E. Heiden, D. Millard, H. Zhang, and G. S. Sukhatme, “Interactive differentiable simulation,” *arXiv preprint arXiv:1905.10706*, 2019.
- [47] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, “Chainqueen: A real-time differentiable physical simulator for soft robotics,” in *2019 International conference on robotics and automation (ICRA)*, pp. 6265–6271, IEEE, 2019.
- [48] J. Liang, M. Lin, and V. Koltun, “Differentiable cloth simulation for inverse problems,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [49] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, “Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids,” *arXiv preprint arXiv:1810.01566*, 2018.
- [50] M. Salzmann and R. Urtasun, “Physically-based motion models for 3d tracking: A convex formulation,” in *2011 International Conference on Computer Vision*, pp. 2064–2071, IEEE, 2011.
- [51] M. A. Brubaker, D. J. Fleet, and A. Hertzmann, “Physics-based person tracking using the anthropomorphic walker,” *International journal of computer vision*, vol. 87, no. 1, pp. 140–155, 2010.
- [52] K. R. Kozlowski, *Modelling and identification in robotics*. Springer Science & Business Media, 2012.
- [53] P. M. Wensing, S. Kim, and J.-J. E. Slotine, “Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution,” *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 60–67, 2017.
- [54] M. A. Brubaker, L. Sigal, and D. J. Fleet, “Estimating contact dynamics,” in *2009 IEEE 12th International Conference on Computer Vision*, pp. 2389–2396, IEEE, 2009.
- [55] K. S. Bhat, C. D. Twigg, J. K. Hodgins, P. Khosla, Z. Popovic, and S. M. Seitz, “Estimating cloth simulation parameters from video,” 2003.
- [56] C. K. Liu, A. Hertzmann, and Z. Popović, “Learning physics-based motion style with nonlinear inverse optimization,” *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 1071–1081, 2005.
- [57] R. Grzeszczuk, D. Terzopoulos, and G. Hinton, “Neuroanimator: Fast neural network emulation and control of physics-based models,” in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 9–20, 1998.
- [58] G. Sutanto, A. Wang, Y. Lin, M. Mukadam, G. Sukhatme, A. Rai, and F. Meier, “Encoding physical constraints in differentiable newton-euler algorithm,” in *Learning for Dynamics and Control*, pp. 804–813, PMLR, 2020.
- [59] K. Wang, M. Aanjaneya, and K. Bekris, “A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines,” in *Learning for Dynamics and Control*, pp. 651–665, PMLR, 2020.
- [60] D. Murray-Smith, “The inverse simulation approach: a focused review of methods and applications,” *Mathematics and computers in simulation*, vol. 53, no. 4-6, pp. 239–247, 2000.
- [61] G. Kim, D. Kang, J.-H. Kim, and H.-W. Park, “Contact-implicit differential dynamic programming for model predictive control with relaxed complementarity constraints,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11978–11985, 2022.
- [62] M. Macklin, “Warp: A high-performance python framework for gpu simulation and graphics,” in *NVIDIA GPU Technology Conference (GTC)*, 2022.
- [63] M. Macklin, M. Müller, and N. Chentanez, “Xpbd: position-based simulation of compliant constrained dynamics,” in *Proceedings of the 9th International Conference on Motion in Games*, pp. 49–54, 2016.
- [64] K. M. Jatavallabhula, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben, et al., “gradsim: Differentiable simulation for system identification and visuomotor control,” *arXiv preprint arXiv:2104.02646*, 2021.
- [65] J. Carpentier and N. Mansard, “Analytical derivatives of rigid body dynamics algorithms,” in *Robotics: Science and systems (RSS 2018)*, 2018.
- [66] M. Giftthaler, M. Neunert, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli, “Automatic differentiation of rigid body dynamics for optimal control and estimation,” *Advanced Robotics*, vol. 31, no. 22, pp. 1225–1237, 2017.
- [67] S. Cheng, M. Kim, L. Song, C. Yang, Y. Jin, S. Wang, and N. Hovakimyan, “DiffTune: Auto-tuning through auto-differentiation,” *IEEE Transactions on Robotics*, 2024.

- [68] Q. Chen, S. Cheng, and N. Hovakimyan, “Simultaneous spatial and temporal assignment for fast uav trajectory optimization using bilevel optimization,” *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3860–3867, 2023.
- [69] M. A. Z. Mora, M. Peychev, S. Ha, M. Vechev, and S. Coros, “Pods: Policy optimization via differentiable simulation,” in *International Conference on Machine Learning*, pp. 7805–7817, PMLR, 2021.
- [70] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, “Differentiable simulation for physical system identification,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3413–3420, 2021.
- [71] J. Brüdigam, J. Janeva, S. Sosnowski, and S. Hirche, “Linear-time contact and friction dynamics in maximal coordinates using variational integrators,” *arXiv:2109.07262*, 2021.
- [72] J. J. Moreau, “On unilateral constraints, friction and plasticity,” in *New Variational Techniques in Mathematical Physics*, pp. 171–322, Springer, 2011.
- [73] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, “Applications of second-order cone programming,” *Linear Algebra and its Applications*, vol. 284, no. 1-3, pp. 193–228, 1998.
- [74] D. E. Stewart and J. C. Trinkle, “An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction,” *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.
- [75] U. Dini, *Lezioni di analisi infinitesimale*, vol. 1. Fratelli Nistri, 1907.
- [76] J. Brüdigam and Z. Manchester, “Linear-time variational integrators in maximal coordinates,” in *International Workshop on the Algorithmic Foundations of Robotics*, pp. 194–209, 2020.
- [77] J. E. Marsden and M. West, “Discrete mechanics and variational integrators,” *Acta Numerica*, vol. 10, pp. 357–514, 2001.
- [78] Z. Manchester and S. Kuindersma, “Variational contact-implicit trajectory optimization,” in *Robotics Research*, pp. 985–1000, Springer, 2020.
- [79] Z. R. Manchester and M. A. Peck, “Quaternion variational integrators for spacecraft dynamics,” *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 1, pp. 69–76, 2016.
- [80] L. Vandenberghe, “The CVXOPT linear and quadratic cone program solvers,” *Online*: <http://cvxopt.org/documentation/coneprog.pdf>, 2010.
- [81] S. Mehrotra, “On the implementation of a primal-dual interior point method,” *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.
- [82] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, second ed., 2006.
- [83] B. E. Jackson, K. Tracy, and Z. Manchester, “Planning with attitude,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5658–5664, 2021.
- [84] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [85] M. Posa, C. Cantu, and R. Tedrake, “A direct method for trajectory optimization of rigid bodies through contact,” *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.
- [86] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems,” in *International Conference on Informatics in Control, Automation and Robotics*, pp. 222–229, 2004.
- [87] T. A. Howell, S. Le Cleac'h, S. Singh, P. Florence, Z. Manchester, and V. Sindhwani, “Trajectory optimization with optimization-based dynamics,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6750–6757, 2022.
- [88] M. H. Raibert, H. B. Brown Jr., M. Chepponis, J. Koehling, J. K. Hodgins, D. Dustman, W. K. Brennan, D. S. Barrett, C. M. Thompson, J. D. Hebert, W. Lee, and B. Lance, “Dynamically stable legged locomotion,” tech. rep., Massachusetts Institute of Technology Cambridge Artificial Intelligence Lab, 1989.
- [89] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” in *International Conference on Machine Learning*, pp. 1329–1338, 2016.
- [90] H. Mania, A. Guy, and B. Recht, “Simple random search of static linear policies is competitive for reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 1800–1809, 2018.
- [91] S. Pfrommer, M. Halm, and M. Posa, “ContactNets: Learning discontinuous contact dynamics with smooth, implicit representations,” in *Conference on Robot Learning*, pp. 2279–2291, 2021.

APPENDIX A QUATERNION ALGEBRA

In this section we introduce a set of conventions for notating standard quaternion operations, adopted from [76, 83], and employed in the rotational part of our variational integrator (17).

Quaternions are written as four-dimensional vectors:

$$q = (s, v) = (s, v_1, v_2, v_3) \in \mathbf{H}, \quad (58)$$

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

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

$$q^a \cdot q^b = \begin{bmatrix} s^a s^b - (v^a)^T v^b \\ s^a v^b + s^b v^a + v^a \times v^b \end{bmatrix} = L(q^a)q^b = R(q^b)q^a, \quad (59)$$

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

$$L(q) = \begin{bmatrix} s & -v^T \\ v & sI_3 + \text{skew}(v) \end{bmatrix} \in \mathbf{R}^{4 \times 4}, \quad (60)$$

$$R(q) = \begin{bmatrix} s & -v^T \\ v & sI_3 - \text{skew}(v) \end{bmatrix} \in \mathbf{R}^{4 \times 4}, \quad (61)$$

where:

$$\text{skew}(x) = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}, \quad (62)$$

is defined such that:

$$\text{skew}(x)y = x \times y, \quad (63)$$

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

$$v = Vq, \quad (64)$$

is extracted using the matrix:

$$V = [\mathbf{0} \ I_3] \in \mathbf{R}^{3 \times 4}, \quad (65)$$

and quaternion conjugate:

$$q^\dagger = \begin{bmatrix} s \\ -v \end{bmatrix} = Tq, \quad (66)$$

is computed using:

$$T = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & -I_3 \end{bmatrix} \in \mathbf{R}^{4 \times 4}. \quad (67)$$