Dojo: A Differentiable Simulator for Robotics

Taylor A. Howell*
Stanford University
thowell@stanford.edu

Simon Le Cleac'h*
Stanford University
simonlc@stanford.edu

J. Zico Kolter
Carnegie Mellon University
zkolter@cs.cmu.edu

Mac Schwager
Stanford University
schwager@stanford.edu

Zachary Manchester
Carnegie Mellon University
zacm@cmu.edu

***Abstract*—We present a differentiable rigid-body-dynamics simulator for robotics that prioritizes physical accuracy and differentiability: Dojo. The simulator utilizes an expressive maximal-coordinates representation, achieves stable simulation at low sample rates, and conserves energy and momentum by employing a variational integrator. A nonlinear complementarity problem, with nonlinear friction cones, models hard contact and is reliably solved using a custom primal-dual interior-point method. The implicit-function theorem enables efficient differentiation of an intermediate relaxed problem and computes smooth gradients from the contact model. We demonstrate the usefulness of the simulator and its gradients through a number of examples including: simulation, trajectory optimization, reinforcement learning, and system identification.**

I. Introduction

Simulators are vital tools across robotics domains, ranging from manipulation to locomotion, with a myriad of applications including: training policies with reinforcement-learning methods, identifying system parameters via gradient-based regression, generating datasets for learning, differentiable model representations in model-predictive-control frameworks, and general Monte Carlo testing and validation. In order to overcome the sim-to-real gap and be of practical value in real-world applications, a simulator should emulate physics to an appropriate fidelity, including energy and momentum conservation and the impact and friction behaviors of contact interactions. Additionally, simulation should be reliable, fast, and ideally, differentiable.

In recent years, a number of simulators [31, 11, 36, 12, 15, 13] have been proposed with potential application to robotics. The most popular and predominant among them is MuJoCo [34], which has become a *de facto* standard for reinforcement learning. In this work, we systematically address key deficiencies of these prior tools—specifically: dynamics and contact models, numerical optimization routines, and gradient computations—by taking a physics- and optimization-first approach to simulator design, prioritizing *physical accuracy* and *differentiability* that are useful across robotics applications. The result is Dojo, a differentiable rigid-body-dynamics-with-contact simulator designed for robotics applications like motion planning, control, and reinforcement learning. Key attributes of Dojo include:

* These authors contributed equally to this work.

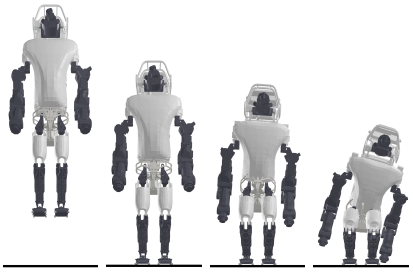

Fig. 1: Atlas drop. The current implementation of Dojo simulates this system with 403 maximal-coordinates states, 36 actuated degrees of freedom, and four contact points on each foot with real-time rates at 65 Hz. Dojo satisfies penetration constraints to machine precision, while MuJoCo allows for tens of millimeters of penetration between rigid bodies and becomes unstable at low simulation rates.

- Maximal-coordinates representation for multi-rigid-body dynamics
- Variational integrator for energy and momentum conservation
- Nonlinear complementarity problem (NCP) model for contact, including exact nonlinear friction cones
- Primal-dual interior-point method for reliably satisfying the NCP and handling cone constraints and quaternions
- Smooth gradients efficiently computed via the implicit-function theorem

In the remainder of this paper, we first provide an overview of existing state-of-the-art simulators, including their capabilities and deficiencies, in Section II. Then, we provide important technical background in Section III. Next, we present an overview of our simulator, Dojo, and its key subroutines in Section IV. A collection of examples including: simulation, trajectory optimization, reinforcement learning, and system identification is presented to highlight Dojo's capabilities and compare to existing tools in Section V. Finally, we conclude with a summary of our results and avenues for future work in Section VI.

II. Existing State-of-the-Art

Many simulators being developed and used in practice today were not designed for real-world robotics applications, and it is common for these tools to prioritize the *appearance* of realism

TABLE I: Comparison of popular simulators with potential application to robotics (adapted from [17]).

simulator	year	application	integrator	state	contact	solver	language	gradients
MuJoCo	2015	robotics	implicit Euler/RK4	minimal	soft	Newton/PGS/CG	C	finite-difference
Drake	2019	robotics	implicit Euler/RADAU5	minimal	soft/hard	LCP/Newton	C++	gradient-bundle
Bullet	2006	graphics	implicit Euler	minimal	soft/hard	LCP	C/C++	sub-gradient
DART	2012	robotics	implicit Euler	minimal	hard	LCP	C++	sub-gradient
Brax	2021	graphics	explicit Euler	maximal	soft	N/A	Python	sub-gradient
RaiSim	unreleased	robotics	implicit Euler	minimal	hard	bisection	C++	-
Dojo	2022	robotics	variational	maximal	hard	NCP	Julia	smooth gradient

over actual physical accuracy. Additionally, many of these simulators were designed primarily for graphics and animation applications where fast simulation rates are prioritized and general-purpose numerical-optimization routines that do not natively support key elements from robotics domains, like cone constraints or quaternions, are commonplace. In this section, we provide background on a collection of popular existing state-of-the-art simulators, including: discussion of physical fidelity, underlying optimization routines, and ability to return gradients.

MuJoCo, which has recently been made open-source, utilizes minimal-coordinates representations, and employs both semi-implicit Euler and explicit fourth-order Runge-Kutta integrators to simulate multi-rigid-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 real-time rates. However, these high rates can prove a challenge for control tasks, such as model-predictive control applications where real-time re-planning is required, or reinforcement-learning settings where vanishing or exploding gradients are exacerbated over long horizons with many time steps.

Impact and friction are modeled using a smooth, convex contact model [33]. While this approach reliably computes contact forces (e.g., the projected Gauss-Seidel (PGS) or conjugate-gradient (CG) solver does not fail to return a solution), it introduces artificial damping, and the system experiences unphysical interpenetration and forces at a distance (i.e., while not in contact). The default friction model employs a pyramidal approximation of the friction cone, which can introduce additional artifacts like velocity drift during sliding. Additionally, achieving good simulation behavior often requires system-specific tuning of multiple solver parameters. Further, the "soft" contact model is computed using a *primal* optimization method, meaning that as parameters are set to produce "hard," or more realistic contact, the underlying optimization problem becomes increasingly ill-conditioned and difficult to solve. As a result, it is often not possible to eliminate unphysical artifacts from the simulation and produce realistic results.

For applications that require them, MuJoCo returns gradients computed using finite-difference methods [29]. This approach requires multiple calls to the simulator, which can be expensive if not performed in parallel. Additionally, in the contact setting, the perturbations to the current state required by a finite-difference approach may violate contact constraints, giving spurious results.

Drake [31] was designed for robotics applications and its dynamics primarily rely on a classic time-stepping contact model that solves a linear complementarity problem (LCP) at each time step [28]. To satisfy the LCP problem formulation, a number of approximations are made to the dynamics and contact model, including the use of an approximate friction cone. To ensure stability of the simulation, small time steps are used where linearizations of the dynamics are valid, but importantly, the simulator can achieve accurate hard contact. General-purpose LCP solvers that are typically used rely on a pivoting method like Lemke's algorithm [6]. Randomized smoothing has been proposed as a method for returning gradients through contact [32] with this model. An alternative soft-contact model is also available for patch contacts [9], but it is more computationally expensive, requiring sophisticated higher-order implicit integrators, and does not natively provide gradients.

Bullet [13] similarly relies on an LCP contact model, but was originally designed for graphics applications. The simulator also has an alternative soft-contact model that trades off physical accuracy for computational speed. Gradients are computed using general-purpose automatic-differentiation tools.

DART [36] has similar dynamics and contact models to Bullet, but employs the implicit-function theorem to differentiate through the solver in order to return gradients. However, because the general-purpose solver relies on an active-set method, sub-gradients are returned that do not convey useful information through contact events. A heuristic [36] has been proposed to overcome this limitation, but it is unclear whether this approach can scale to the large number of contact transitions required in many robotics applications.

Brax [11] employs maximal-coordinates representations and has an impulse-correction contact model that utilizes explicit integrators. System-specific tuning of springs and dampers that connect rigid bodies is required for stable simulation and a relatively simple contact model is employed that does not require certain linear-algebra routines in order to deploy the simulator on hardware designed for parallel computation.

Similar to previous simulator work [5], Dojo utilizes the open-source maximal-coordinates dynamics library `ConstrainedDynamics.jl` and efficient graph-based linear-system solver `GraphBasedSystems.jl`. However, unlike this related work, Dojo has an improved contact model,

specifically with regard to friction and graph representation; utilizes a more efficient, reliable, and versatile interior-point solver for the NCP; efficiently returns smooth gradients; and includes improved integrator, joint, and internal friction support.

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

III. Technical Background

In this section we provide technical background on maximal-coordinates state representations, complementarity-based contact models, and implicit differentiation.

A. Maximal Coordinates

Most robotics simulators utilize minimal- or joint-coordinates representations for dynamics because of the small number of states and convenience of implementation. This results in small, but dense systems of linear equations. In contrast, maximal-coordinates explicitly represent the position, orientation, and velocities of each body in a multi-rigid-body system. This produces large, sparse systems of linear equations that can be efficiently optimized, including in the contact setting, and provides more information about a system at each simulation step. We provide an overview, largely based on prior work [4], 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 $d : \mathbf{X} \times \mathbf{X} \times \mathbf{X} \to \mathbf{R}^6$ as:

$$d(x_-, x, x_+) = 0, \tag{1}$$

where we indicate the previous and next time steps with minus $(-)$ and plus $(+)$ subscripts, respectively, and the current time step without decoration. A variational integrator is employed that has desirable energy and momentum conservation properties [21]. Linear and angular velocities are handled implicitly via finite-difference approximations.

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

$$g^{ab}(x_+^a, x_+^b) = 0. \tag{2}$$

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

$$\begin{bmatrix} d^a(x_-^a, x^a, x_+^a) + G^a(x^a, x^b)^T y^{ab} \\ d^b(x_-^b, x^b, x_+^b) + G^b(x^a, x^b)^T y^{ab} \\ g^{ab}(x_+^a, x_+^b) \end{bmatrix} = 0, \tag{3}$$

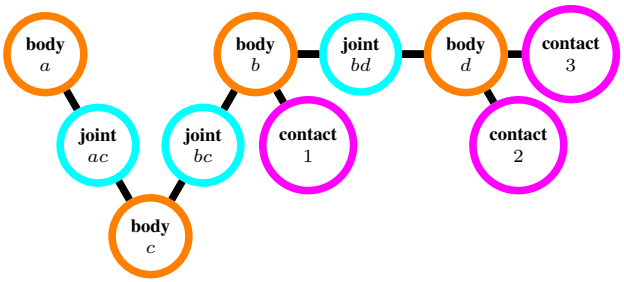

Fig. 2: Graph structure for maximal-coordinates system with 4 bodies, 3 joints, and 3 points of contact.

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

We can generalize (3) to include additional bodies and joints. For a system with N bodies and M joints we define a maximal-coordinates configuration $z = (x^{(1)}, \ldots, x^{(N)}) \in \mathbf{Z}$ and joint impulse $w = (y^{(1)}, \ldots, y^{(M)}) \in \mathbf{W}$. We define the implicit discrete-time dynamics of the maximal-coordinates system as:

$$f(z_-, z, z_+, w) = 0, \tag{4}$$

where $f : \mathbf{Z} \times \mathbf{Z} \times \mathbf{Z} \times \mathbf{W} \to \mathbf{R}^{6N}$. In order to simulate the system we find z_+ and w that satisfy (4) for a provided z_- and z using Newton's method.

By exploiting the mechanism's structure, we can efficiently perform root finding on (4) (see [4] for additional details). This structure is manifest as a graph of the mechanism, where each body and joint is considered a node, and joints have edges connecting bodies (Fig. 2). 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.

B. Complementarity-Based Contact Models

Contact is modeled via constraints on the system's configuration and the applied contact forces.

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

$$\phi(z) \geq 0. \tag{5}$$

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 (5) and prevent interpenetration. A non-negative constraint,

$$\gamma \geq 0, \tag{6}$$

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

$$\gamma \circ \phi(z) = 0, \tag{7}$$

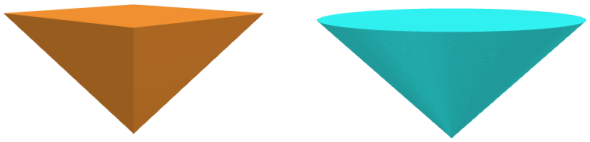

Fig. 3: Friction-cone comparison. Linearized double-parameterized (left) and nonlinear second-order (right) cones.

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,

$$\begin{aligned} \underset{b}{\text{minimize}} \quad & v^T b \\ \text{subject to} \quad & \|b\|_2 \leq \mu\gamma, \end{aligned} \tag{8}$$

where $v \in \mathbf{R}^2$ is the tangential velocity at the contact point, $b \in \mathbf{R}^2$ is the friction force, and $\mu \in \mathbf{R}_+$ is the coefficient of friction between the two objects [23].

This problem is naturally a convex second-order cone program, and can be efficiently and reliably solved [18]. However, classically, an approximate version of (8):

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & \begin{bmatrix} v^T & -v^T \end{bmatrix} \beta, \\ \text{subject to} \quad & \beta^T \mathbf{1} \leq \mu\gamma, \\ & \beta \geq 0, \end{aligned} \tag{9}$$

which satisfies the LCP formulation, is instead solved. Here, the friction cone is linearized (Fig. 3) and the friction vector, $\beta \in \mathbf{R}^4$, is correspondingly overparameterized and subject to additional non-negative constraints [28].

The optimality conditions of (9) and constraints used in the LCP are:

$$\begin{bmatrix} v^T & -v^T \end{bmatrix}^T + \psi\mathbf{1} - \eta = 0, \tag{10}$$

$$\mu\gamma - \beta^T\mathbf{1} \geq 0, \tag{11}$$

$$\psi \cdot (\mu\gamma - \beta^T\mathbf{1}) = 0, \tag{12}$$

$$\beta \circ \eta = 0, \tag{13}$$

$$\beta, \psi, \eta \geq 0, \tag{14}$$

where $\psi \in \mathbf{R}$ and $\eta \in \mathbf{R}^4$ are the dual variables associated with the friction cone and positivity constraints, respectively, and $\mathbf{1}$ is a vector of ones.

The primary drawback to this formulation is that the optimized friction force will naturally align with the vertices of the cone approximation, which may not align with the velocity vector of the contact point. 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 (1) and a linear approximation of the signed-distance functions (5-7), both of which negatively impact physical accuracy.

C. Implicit-Function Theorem

An implicit function, $r : \mathbf{R}^{n_w} \times \mathbf{R}^{n_\theta} \to \mathbf{R}^{n_w}$, is defined as

$$r(w^*; \theta) = 0, \tag{15}$$

for solution $w^* \in \mathbf{R}^{n_w}$ and problem data $\theta \in \mathbf{R}^{n_\theta}$. At a solution point of (15) the sensitivities of the solution with respect to the problem data, i.e., $\partial w^*/\partial\theta$, can be computed under certain conditions [7]. First, we approximate (15) to first order:

$$\frac{\partial r}{\partial w}\delta w + \frac{\partial r}{\partial \theta}\delta\theta = 0, \tag{16}$$

and then solve for the relationship:

$$\frac{\partial w^*}{\partial \theta} = -\left(\frac{\partial r}{\partial w}\right)^{-1}\frac{\partial r}{\partial \theta}. \tag{17}$$

In case $(\partial r/\partial w)^{-1}$ is not well defined, (e.g., not full rank) we can either apply regularization or approximately solve (17) with, for example, a least-squares approach.

Often, Newton's method is employed to find solutions to (15) and custom linear-system solvers can efficiently compute search directions for this purpose. Importantly, the factorization of $\partial r/\partial w$ used to find a solution can be reused to compute (17) at very low computational cost using only back-substitution. Additionally, each element of the problem-data sensitivity can be computed in parallel.

IV. Dojo

This section presents the key algorithms and subroutines, including: variational integrators, contact model, primal-dual interior-point solver, and smooth gradients implemented for Dojo. An open-source implementation of the simulator is also provided.

A. Variational Integrator

We use a specialized implicit integrator that preserves energy and momentum, natively handles quaternions, and alleviates spurious artifacts that commonly arise from contact interactions. This integrator is based on prior work [21, 4], but we make a number of key modifications to improve its numerical properties for simulation: First, we utilize impulses instead of forces. In practice we find that this subtle change greatly improves the convergence and reliability of a simulation step. Second, we utilize position and quaternion trajectories for simulation, with linear and angular velocities defined implicitly in the integrator via finite-difference schemes. The benefit to this approach is that velocities in the contact setting are *discontinuous*, whereas position and quaternion trajectories, while non-smooth, are *continuous*.

The integrator (1) used by Dojo for each body has linear:

$$m(p_+ - 2p + p_-)/h - hmg = 0, \tag{18}$$

and rotational:

$$\begin{aligned} VL(q)^T\Big(&TR(q_+)^T V^T JVL(q)^T q_+ \\ &+ L(q_-)V^T JVL(q_-)^T q\Big)/h = 0, \end{aligned} \tag{19}$$

components with mass $m \in \mathbf{R}_{++}$, inertia $J \in \mathbf{S}^3_{++}$, gravity $g \in \mathbf{R}^3$, and time step $h \in \mathbf{R}_{++}$. Equation (18) is essentially a second-order centered-finite-difference approximation of Newton's second law, while (19) is a similar second-order finite-difference approximation of Euler's equation for the rotational dynamics of a rigid body expressed with quaternions. The matrix notation we use for quaternion operations is defined in Appendix B.

Both (18) and (19) are derived by approximating Hamilton's Principle of Least-Action using a simple midpoint scheme [21, 19]. This approach produces *variational* integrators that automatically conserve momentum and energy [21].

B. Contact 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 2). As a result, the simulator retains efficient linear-time complexity for open-chain mechanical systems.

Dojo uses the classic impact model (5-7) 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 (8) in a form amenable to a primal-dual interior-point solver. The associated cone program is,

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & \begin{bmatrix} 0 & v^T \end{bmatrix} \beta \\ \text{subject to} \quad & \beta_{(1)} = \mu\gamma, \\ & \beta \in \mathcal{Q}^3, \end{aligned} \tag{20}$$

where subscripts indicate vector indices and the n-dimensional second-order cone $\mathcal{Q}^n$ is defined by:

$$\mathcal{Q}^n = \{(a_{(1)}, a_{(2:n)}) \in \mathbf{R} \times \mathbf{R}^{n-1} \mid \|a_{(2:n)}\|_2 \leq a_{(1)}\}. \tag{21}$$

The relaxed optimality conditions for (20) in interior-point form are:

$$v - \eta_{(2:3)} = 0, \tag{22}$$

$$\beta_{(1)} - \mu\gamma = 0, \tag{23}$$

$$\beta \circ \eta = \kappa \mathbf{e}, \tag{24}$$

$$\beta, \eta \in \mathcal{Q}^3, \tag{25}$$

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

$$\beta \circ \eta = (\beta^T \eta, \beta_{(1)}\eta_{(2:n)} + \eta_{(1)}\beta_{(2:n)}), \tag{26}$$

and,

$$\mathbf{e} = (1, 0, \ldots, 0), \tag{27}$$

is its corresponding identity element [35]. Friction is recovered from the solution: $b = \beta^*_{(2:3)}$. The benefits of this model are increased physical fidelity and fewer optimization variables, without substantial increase in computational cost.

Nonlinear complementarity problem: To simulate a system represented in maximal coordinates that experiences contact, a solver aims to satisfy the following relaxed feasibility problem:

$$\begin{aligned} \text{find} \quad & z_+, w, \gamma, \beta^{(1:P)}, \eta^{(1:P)}, s \\ \text{s.t.} \quad & f(z_-, z, z_+, w) + B(z)u + C(z)^T\lambda = 0 \\ & s - \phi(z_+) = 0, \\ & \gamma \circ s = \kappa \mathbf{1}, \\ & \beta^{(i)} \circ \eta^{(i)} = \kappa \mathbf{e}, \quad i = 1, \ldots, P, \\ & v^{(i)}(z, z_+) - \eta^{(i)}_{(2:3)} = 0, \quad i = 1, \ldots, P, \\ & \beta^{(i)}_{(1)} - \mu^{(i)}\gamma^{(i)} = 0, \quad i = 1, \ldots, P, \\ & \gamma, s \geq 0, \\ & \beta^{(i)}, \eta^{(i)} \in \mathcal{Q}^3, \quad i = 1, \ldots, P, \end{aligned} \tag{28}$$

where $u \in \mathbf{R}^m$ is the control input at the current time step, $\lambda = (\beta^{(1)}_{(2:3)}, \gamma^{(1)}, \ldots, \beta^{(P)}_{(2:3)}, \gamma^{(P)}) \in \mathbf{\Lambda}$ is the concatenation of impact and friction impulses, $B : \mathbf{Z} \to \mathbf{R}^{6N \times m}$ is the input Jacobian mapping control inputs into maximal coordinates, $C : \mathbf{Z} \to \mathbf{R}^{\dim(\mathbf{\Lambda}) \times 6N}$ is a contact Jacobian mapping between maximal coordinates and contact surfaces, $s \in \mathbf{R}^P$ is a slack variable introduced for convenience, and $v^{(i)} : \mathbf{Z} \times \mathbf{Z} \to \mathbf{R}^2$ is the tangential velocity at contact point i. Joint limits and internal friction are readily incorporated into this problem formulation.

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 relaxed problems (28) are found with $\kappa \to 0$. The central-path parameter has a physical interpretation as being the softness of the contact model. A value $\kappa = 0$ corresponds to exact "hard" or inelastic contact, whereas a relaxed value produces soft contact where contact forces can occur at a distance. The primal-dual interior-point solver described in the next section adaptively decreases this parameter in order to efficiently and reliably converge to hard contact solutions. In practice, the simulator is set to converge to small values for simulation in order to simulate accurate physics, while relaxed values are used to compute smooth gradients in order to provide useful information through contact events.

C. Primal-Dual Interior-Point Solver

To efficiently and reliably satisfy (28), we developed a custom primal-dual interior-point solver for NCPs with cone constraints and quaternions. The algorithm is largely based upon Mehrotra's predictor-corrector algorithm [22, 24], while borrowing practical numerical features from CVXOPT [35] to handle cones and non-Euclidean optimization to handle quaternions [16]. We also introduce heuristics that further improve reliability and overall performance of the solver for our simulation-step NCPs.

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

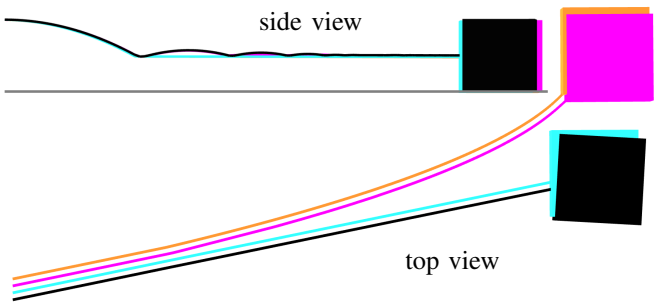

Fig. 4: Velocity drift resulting from friction-cone approximation. Comparison between a box sliding with approximate cones having four vertices implemented in MuJoCo (magenta) and Dojo (orange) versus MuJoCo's (black) and Dojo's (blue) nonlinear friction cones. Dojo's nonlinear friction cone gives the physically correct straight line motion, while linear friction-cone approximations lead to lateral drift. MuJoCo's nonlinear friction cone exhibits a minor rotational drift.

number of iterations), and feedback on the problem's central-path parameter that helps avoid premature ill-conditioning and adaptively drives the complementarity violation to zero in order to reliably simulate hard contact.

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

$$\begin{aligned} &\text{find} && x, y, z \\ &\text{subject to} && c(x, y, z; \theta) = 0, \\ &&& y^{(i)} \circ z^{(i)} = \kappa \mathbf{e}, \quad i = 1, \ldots, n, \\ &&& y^{(i)}, z^{(i)} \in \mathcal{K}, \quad i = 1, \ldots, n, \end{aligned} \tag{29}$$

with decision variables $x \in \mathbf{R}^k$ and $y, z \in \mathbf{R}^m$, equality-constraint set $c : \mathbf{R}^k \times \mathbf{R}^m \times \mathbf{R}^m \times \mathbf{R}^l \to \mathbf{R}^h$, problem data $\theta \in \mathbf{R}^l$; and where $\mathcal{K}$ is the Cartesian product of n total positive-orthant and second-order cones [1]. The variables are partitioned: $x = (x^{(1)}, \ldots, x^{(p)})$, where $i = 1$ are Euclidean variables and $i = 2, \ldots, p$ are each quaternion variables; and $y = (y^{(1)}, \ldots, y^{(n)})$, $z = (z^{(1)}, \ldots, z^{(n)})$, where $j = 1$ is the positive-orthant and the remaining $j = 2, \ldots, n$ are second-order cones. For convenience, we denote $w = (x, y, z)$.

The algorithm aims to satisfy a sequence of relaxed problems with $\kappa > 0$ and $\kappa \to 0$ in order to reliably converge to a solution of the original problem (i.e., $\kappa = 0$). This continuation approach helps avoid premature ill-conditioning and is the basis for numerous convex and non-convex general-purpose interior-point solvers [24].

The LCP formulation is a special-case of instantiation (29) where the constraint set is affine in the decision variables, the cone is the positive orthant, and where most general-purpose solvers rely on active-set methods that strictly enforce $\kappa = 0$ at each iteration.

Cones: The generalized inequality (21), cone-product operator (26), and the identity element (27) were previously defined for the second-order cone. For the n-dimensional positive orthant, these terms are:

$$\mathbf{R}^n_{++} = \{u \in \mathbf{R}^n \,|\, u_{(i)} > 0,\, i = 1, \ldots, n\}, \tag{30}$$

$$u \circ v = (u_{(1)} v_{(1)}, \ldots, u_{(n)} v_{(n)}), \tag{31}$$

$$\mathbf{e} = \mathbf{1}. \tag{32}$$

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

$$c_{\text{vio}} = \|c(w; \theta)\|_\infty, \tag{33}$$

and complementarity violation,

$$\kappa_{\text{vio}} = \max_i \{\|y^{(i)} \circ z^{(i)}\|_\infty\}. \tag{34}$$

The problem (29) is considered solved when $c_{\text{vio}} < c_{\text{tol}}$ and $\kappa_{\text{vio}} < \kappa_{\text{tol}}$.

Residual and Jacobians: The solver aims to drive the residual vector,

$$r(w; \theta, \kappa) = \begin{bmatrix} c(w; \theta) \\ y^{(1)} \circ z^{(1)} - \kappa \mathbf{1} \\ \vdots \\ y^{(n)} \circ z^{(n)} - \kappa \mathbf{e} \end{bmatrix}, \tag{35}$$

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

$$R(w; \theta) = \frac{\partial r(w; \theta, \cdot)}{\partial w}, \tag{36}$$

is used to compute a search direction. After a solution w^* is found, the Jacobian of the residual with respect to the problem data,

$$D(w; \theta) = \frac{\partial r(w; \theta, \cdot)}{\partial \theta}, \tag{37}$$

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 (36) and (37) by right multiplying each with a matrix H containing attitude Jacobians [16] corresponding to the quaternions in x and θ, respectively:

$$\bar{R}(w; \theta) = R(w; \theta) H_R(w), \tag{38}$$

$$\bar{D}(w; \theta) = D(w; \theta) H_D(\theta). \tag{39}$$

Euclidean variables have corresponding identity blocks. This modification accounts for the implicit unit-norm constraint

TABLE II: Contact violation for Atlas drop. Comparison between Dojo and MuJoCo for foot contact penetration (millimeters) with the floor for different time steps h (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).

	$h = 0.1$	$h = 0.01$	$h = 0.001$
MuJoCo	failure	–28	–46
Dojo	+1e-12	+1e-7	+8e-6

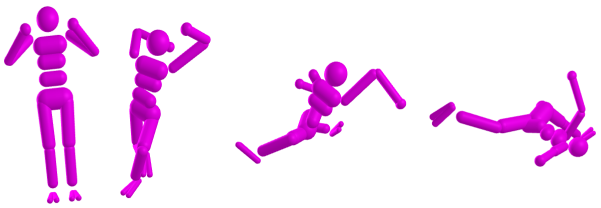

Fig. 5: Astronaut simulation for energy and momentum conservation test. Joints are initialized with zero velocities and randomly actuated for 1 second. The simulation is visualized, from left to right.

on each quaternion variable and improves the convergence behaviour of the solver.

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

$$y \leftarrow y + \alpha\Delta, \tag{40}$$

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:

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

and second-order cone:

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

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

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

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

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

The line search over all individual cones is summarized in Appendix A.

Update: For a given search direction, updates for Euclidean and quaternion variables are performed. The Euclidean variables in x use a standard update:

$$x^{(1)} \leftarrow x^{(1)} + \alpha\Delta^{(1)}, \tag{47}$$

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

$$x^{(i)} \leftarrow L(x^{(i)})\varphi(\alpha\Delta^{(i)}), \tag{48}$$

where $L : \mathbf{H} \rightarrow \mathbf{R}^{4\times4}$ is a matrix representing a left-quaternion matrix multiplication, and $\varphi : \mathbf{R}^3 \rightarrow \mathbf{H}$ is a mapping to a unit quaternion. The standard update is used for the remaining decision variables y and z.

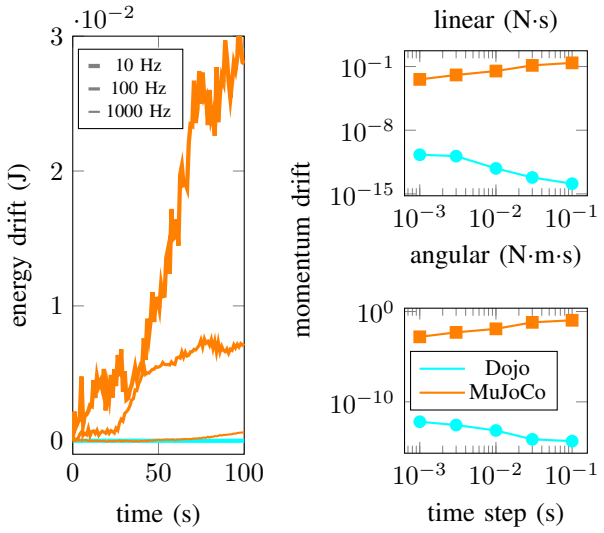

Fig. 6: Energy and momentum conservation comparison between MuJoCo and Dojo for the astronaut simulation (Fig 5) using time steps ranging from 0.001 to 0.1 second. Momentum drift is measured after actuating the astronaut for 1 second with random controls. Energy drift is measured over a 100 second simulation after 1 second of random actuation. Dojo achieves drift values near machine precision and is insensitive to large time steps.

Centering: The solver adaptively relaxes (29) 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 [35]:

$$\mathbf{deg}(\mathcal{K}) = \sum_{i=1}^{n} \mathbf{deg}(\mathcal{K}^{(i)}) = \mathbf{dim}(\mathcal{K}^{(1)}) + n - 1, \tag{49}$$

the complementarity violations:

$$\mu = \frac{1}{\mathbf{deg}(\mathcal{K})}\sum_{i=1}^{n}(y^{(i)})^T z^{(i)}. \tag{50}$$

and affine complementarity violations:

$$\mu^{\text{aff}} = \frac{1}{\mathbf{deg}(\mathcal{K})}\sum_{i=1}^{n}(y^{(i)} + \alpha\Delta^{y^{(i)}})^T(z^{(i)} + \alpha\Delta^{z^{(i)}}), \tag{51}$$

as well as their ratio:

$$\sigma = \min\left(1, \max\left(0, \mu^{\text{aff}}/\mu\right)\right)^3, \tag{52}$$

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

Solver: For a problem instance (29), 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

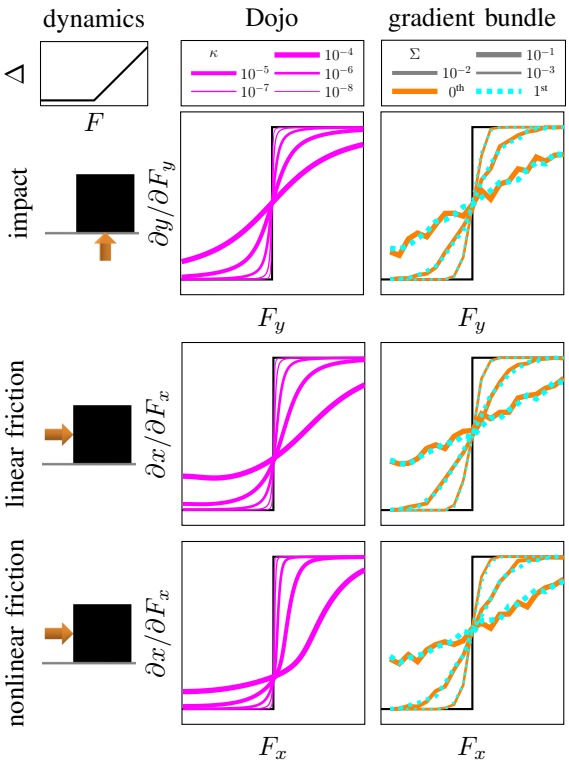

Fig. 7: Gradient comparison between randomized smoothing [32] and Dojo's smooth gradients. The dynamics for a box in the XY plane that is resting on a flat surface and displaced an amount Δ by an input F (top left). Its corresponding exact gradients are shown in black. Gradient bundles (right column) are computed using sampling schemes with varying covariances Σ and 500 samples. Dojo's gradients (middle column) are computed for different values of κ, corresponding to the smoothness of the contact model. Compared to the 500-sample gradient bundle, Dojo's gradients are not noisy and are a 100 times faster to compute with a single worker.

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. Further details are provided in Appendix A.

D. Gradients

At a solution point, $w^*(\theta, \kappa)$, the sensitivity of the solution with respect to the problem data, i.e., $\partial w^*/\partial\theta$, is efficiently computed using the implicit-function theorem (17) to differentiate through the solver's residual (35).

The efficient linear-system solver used for the simulator, as well as the computation and factorization of $\partial r/\partial w$, is used to compute the sensitivities for each element of the problem data. Calculations over the individual columns of $\partial r/\partial\theta$ can be performed in parallel.

The problem data for each simulation step include: the previous and current configurations, control input, and additional terms like the time step, friction coefficients, and parameters of each body. The chain rule is utilized to compute gradients with respect to the finite-difference velocities as well as transformations between minimal- and maximal-coordinate representations.

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

	Atlas	**humanoid**	**quadruped**	**ant**	**half-cheetah**
Dojo	3.7	4.9	2.5	2.3	1.2
FD	472.6	194.7	170.3	197.0	94.8

In many robotics scenarios, we are interested in gradient information through contact events. Instead of computing gradients for hard contact with zero or very small central-path parameters, we use a relaxed value from intermediate solutions $w^*(\theta, \kappa > 0)$ corresponding to a soft contact model. In practice, we find that these smooth gradients greatly improve the performance of gradient-based optimization methods.

E. Open-Source Tools

An open-source implementation of the simulator, `Dojo.jl`, written in Julia, a high-level programming language that has C-like performance, is provided. Additionally, a Python interface, `dojopy`, is included. These tools, and the following examples, are available at: https://github.com/dojo-sim.

V. Examples

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

A. Simulation

Dojo's simulation fidelity is demonstrated with a number of illustrative physical scenarios, including comparisons with MuJoCo.

Friction-cone comparison: The effect of friction-cone approximation is demonstrated by simulating a box that is initialized with lateral velocity before impacting and sliding along a flat surface. For a pyramidal approximation, in the probable scenario where its vertices are not aligned with the direction of motion, velocity drift occurs for a linearized cone implemented in Dojo and MuJoCo (Fig. 4). The complementarity problem with P contact points requires $2P(1 + 2d)$ decision variables for contact and a corresponding number of

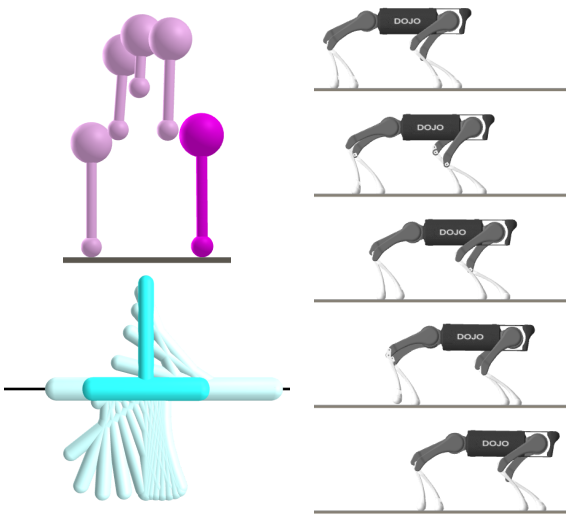

Fig. 8: Motion plans for hopper (top left), cart-pole (bottom left), and quadruped (right) generated using trajectory optimization.

TABLE IV: Trajectory-optimization results. Comparison of final cost value, goal constraint violation, and total number of iterations for a collection of systems systems with maximal (max.) and minimal (min.) representations, optimized with iterative LQR. While exhibiting similar performance, MuJoCo's more mature implementation is faster than the current version of Dojo. However, on the box-up example, MuJoCo's gradients cause the optimizer to fail whereas Dojo's smooth gradients enable it to succeed at the task.

system	cost	con. viol.	iter.
cart-pole (max.)	**30.2**	1.5e-3	100
cart-pole (min.)	35.5	**2.5e-5**	100
cart-pole (MuJoCo)	37.0	5.2e-4	**80**
box right (max.)	14.5	3.3e-3	**30**
box right (MuJoCo)	**13.5**	**3.2e-3**	95
box up (max.)	**14.5**	**3.1e-3**	**106**
box up (MuJoCo)	failure	1.0	-
hopper (max.)	10.2	3.7e-3	**57**
hopper (min.)	**8.9**	**1.2e-3**	96
hopper (MuJoCo)	26.7	1.8e-3	66
quadruped (min.)	1.8e-2	2.6e-4	20

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

Impact constraints comparison: The Atlas humanoid is simulated dropping on to 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. The current implementation of Dojo simulates this system in real time at 65 Hz. A comparison with MuJoCo is performed measuring penetration violations with the floor for different simulation rates (Table II).

Energy and momentum conservation: An accurate robotics simulator conserves important physical quantities like energy and momentum. Following the methodology from [10], we simulate "astronaut," a free-floating humanoid, and measure the drift of these quantities (Fig. 6). There is no internal damping or springs, joint limits, or contact, and gravity is turned off. The astronaut is initialized with no linear or angular velocity and momentum drift is computed after one second of uniformly sampled actuation: $u \sim \mathcal{U}(0, 0.05)$. Energy drift is computed over a 100 second period after 1 second of random actuation. MuJoCo exhibits drift in all scenarios. Characteristic of its variational integrator, Dojo conserves both linear and angular momentum to machine precision. Energy does not drift for Dojo but exhibits small bounded oscillations that decrease in amplitude as the time step decreases (Fig. 6). Conservation of energy to machine precision with variational integrators is possible and is a topic of current research [27].

Gradient comparison: Most simulators for robotics in the contact setting return discontinuous gradients. This poses significant difficulties to gradient-based optimization methods that assume smooth and continuously differentiable models. MuJoCo overcomes such difficulties and returns continuous gradients by employing a soft-contact model and a finite difference scheme. For LCP-based simulators that return sub-gradients, randomized smoothing via gradient bundles has been proposed [32]. We compare Dojo's smooth gradients with the latter approach.

Analysis is performed on an actuated box in two dimensions that is resting on a surface. The sensitivity of the position with respect to an input force is computed. We compare gradient bundles with 500 samples to Dojo's smooth gradients computed with different relaxed central-path parameters (Fig. 7). We remark that, despite the relatively large number of samples, the bundle still results in a noisy estimation for a 1-dimensional gradient. For higher dimensions, typical in robotics, this phenomenon will be exacerbated and more samples might be required to generate useful gradients. In contrast, Dojo's gradient is smooth and continuous and only requires an inexpensive back-substitution on a pre-factorized matrix. For comparison, we measure the ratio between the compute times for Dojo's gradient and simulation-step evaluation (Table III). Our experiments indicate that gradient evaluations are typically five times, or less, expensive compared to a step evaluation. This indicates that Dojo's gradient evaluation is 100 times faster than a serial bundle evaluation or requires the same time as a bundle evaluated with 100 parallel workers.

B. Trajectory Optimization

Iterative LQR [14] uses smooth gradients from Dojo to perform trajectory optimization on three underactuated systems. *Cart-pole:* This classic system [30] with two degrees-of-freedom and one control input is tasked with performing a swing-up over a planning horizon $T = 26$ with time step $h = 0.1$. The pendulum starts in the downward position and the controls are initialized with random values. Quadratic costs are used to penalize control effort and displacement from the

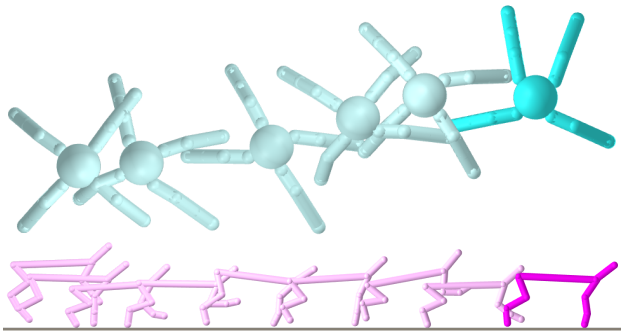

Fig. 9: Trajectories for ant (top) and half-cheetah (bottom) generated using policies trained with reinforcement learning.

goal state. The optimizer finds similar swing-up trajectories.

Box: Inputs are optimized to move a stationary rigid body that is resting on a flat surface (Fig. 7) to a goal location that is either to the right or up in the air 1 meter. The planning horizon is 1 second and the controls are initialized with zeros. Dojo uses a time step $h = 0.1$, whereas MuJoCo uses $h = 0.01$ to prevent significant contact violations with the floor. MuJoCo fails in the scenario with the goal in the air, while Dojo succeeds at both tasks.

Hopper: The robot [26] 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 displacement from an intermediate state in the air and the goal pose. The optimizer typically finds a single-hop motion.

Quadruped: The system 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.

In addition to optimizing in the maximal-coordinates space, we also transform this state to a minimal representation for comparison. Gradients are computed with $\kappa = 3e{-}4$. The results are visualized (Fig. 8) and summarized in Table IV.

C. Reinforcement Learning

Using Dojo, we implement Gym-like environments [2, 8]: ant and half-cheetah (Fig. 9). We train static linear policies for locomotion using Augmented Random Search (ARS) [20]. This is a gradient-free approach coupling random search with a number of simple heuristics. Additionally, we train the same policies using augmented gradient search (AGS) which replaces the stochastic-gradient estimation of ARS with the simulator's analytical gradient.

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 staying alive and is penalized for control effort and contact over a horizon $T = 150$ with time step $h = 0.05$.

The Gym environments use MuJoCo with time steps $h = 0.01$, since larger steps result is significant interpenetration of the robots with the floor. In contrast, Dojo is able to train reliably in environments with hard contact using a larger time step. Training is performed for a fixed number of steps and the policy is compared over the 3 best out of 5 runs for different random seeds. Results are summarized in Table V. The gradient-based approach closely matches ARS in terms of accumulated reward while requiring 5 to 10 times fewer dynamics evaluations, where for each simulation step, AGS computes the dynamics gradient with a computational cost comparable to one dynamics evaluation.

TABLE V: Reinforcement-learning results. Comparison of total reward, number of simulation-step and gradient evaluations for a collection of policies trained with Augmented Random Search (ARS) [20] and Augmented Gradient Search (AGS). The results are averaged over the best 3 out of 5 runs with different random seeds. AGS which relies on Dojo's gradients reaches similar performance levels as ARS while being 5 to 10 times more sample efficient.

system	reward	sim. eval.	grad. eval.
half-cheetah (ARS)	46 ± 24	2.9e+4	0
ant (ARS)	64 ± 15	1.6e+5	0
half-cheetah (AGS)	44 ± 24	**4.8e+3**	4.8e+3
ant (AGS)	54 ± 28	**1.5e+4**	1.5e+4

D. Real-To-Sim

System identification is performed on an existing real-world dataset of trajectories collected by throwing a box on a table with different initial conditions [25]. We aim to recover a set of parameters $\theta = (\mu, p^{(1)}, \ldots, p^{(8)})$ that include the friction coefficient μ, and 3-dimensional vectors $p^{(i)}$ that represent the position of vertex i of the box with respect to its center of mass.

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

$$\hat{z}_+ = s(z_-, z, \theta), \tag{53}$$

to predict the next state, $\hat{z}_+$. Additionally, Dojo computes a smooth Jacobian, $\partial \hat{z}_+ / \partial \theta$ for the dynamics with respect to the parameters using smoothness: $\kappa = 3e{-}4$.

The parameters are learned by minimizing the following loss:

$$\mathcal{L}(\mathcal{D}, \theta) = \sum_{Z \in \mathcal{D}} L(Z, \theta) = \sum_{Z \in \mathcal{D}} \frac{1}{2} ||s(z_-, z, \theta) - z_+||_W^2, \tag{54}$$

where $||\cdot||_W$ is a weighted norm, which aims to minimize the difference between the ground-truth trajectories and simulator

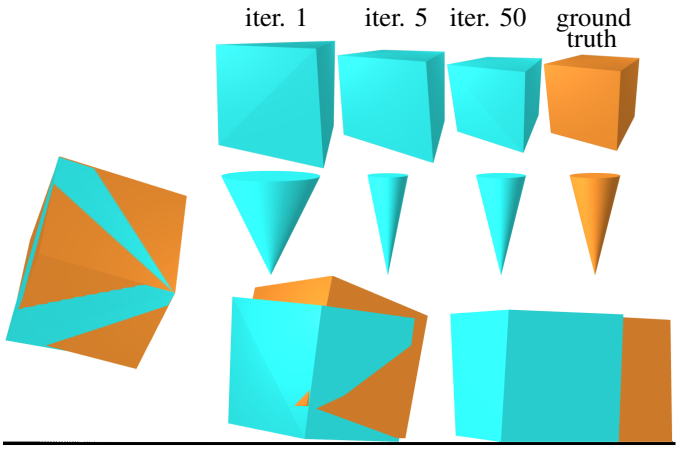

Fig. 10: System identification. Top right: Learning box geometry and friction cone to less than 5% error. Bottom: Simulated trajectory of the box using the learned properties (blue) compared to ground truth (orange).

predictions. We use a Gauss-Newton method that utilizes gradients:

$$\frac{\partial L}{\partial \theta} = \frac{\partial s}{\partial \theta}^T W \left(s(z_-, z, \theta) - z_+ \right), \tag{55}$$

and approximate Hessians:

$$\frac{\partial^2 L}{\partial \theta^2} \approx \frac{\partial s}{\partial \theta}^T W \frac{\partial s}{\partial \theta}. \tag{56}$$

We successfully learn parameters that match to within 5% of the actual geometry and friction properties of the box. Using these values, we simulate the system and compare it to the ground-truth simulation (Fig. 10).

VI. Discussion

Summary: Dojo makes several advancements over previous robotics simulators: First, a maximal-coordinates state representation is coupled with a numerically improved variational integrator. This large, sparse representation can be efficiently optimized and provides more information from the simulator compared to minimal-coordinate representations [3]. As demonstrated in the examples, the variational integrator better conserves energy and momentum compared to higher-order explicit and lower-order implicit counterparts while being stable using relatively large time steps. Further, the improved friction model requires less decision variables and produces higher-quality simulation results, particularly in sliding scenarios. Next, a custom interior-point solver was developed specifically for solving NCPs that is numerically robust, requires practically no hyperparameter tuning for good performance across numerous systems, and handles special cone and quaternion variables. Finally, we present an efficient approach for returning gradients from the simulator. The smoothness of these gradients is easily set using a single intuitive parameter and they provide useful information through contact events. In addition to building and providing an open-source tool, the physics and optimization improvements presented can improve many existing simulators.

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

Future directions: A number of future improvements to Dojo are planned. First, Dojo currently implements collision detection between capsules and flat surfaces. Natural extensions include support for self-collision and general curved surfaces. The former requires additional care to handle the cycles introduced to the system's graph structure and efficiently solve the resulting linear system, while the latter is possible by solving and differentiating through a lower-level minimum-distance optimization problem. 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, greater exploitation of algorithmic parallelism in Dojo would enable faster simulation and optimization.

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

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

Acknowledgements

The authors would like to thank Jan Brüdigam for his contributions to the open-source libraries `ConstrainedDynamics.jl` and `GraphBasedSystems.jl` which served as a foundation for Dojo, as well as early technical discussions and support; and Suvansh Sanjeev for assistance with the Python interface. Toyota Research Institute provided funds to support this work.

References

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

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

[3] J. Brüdigam and Z. Manchester. Linear-Quadratic optimal control in maximal coordinates. *arXiv preprint arXiv:2010.05886*, 2020.

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

[5] J. Brüdigam, J. Janeva, S. Sosnowski, and S. Hirche. Linear-time contact and friction dynamics in maximal coordinates using variational integrators. *arXiv preprint arXiv:2109.07262*, 2021.

[6] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The linear complementarity problem*. SIAM, 2009.

[7] U. Dini. *Lezioni di analisi infinitesimale*, volume 1. Fratelli Nistri, 1907.

[8] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338. PMLR, 2016.

[9] R. Elandt, E. Drumwright, M. Sherman, and A. Ruina. A pressure field model for fast, robust approximation of net contact force and moment between nominally rigid objects. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8238–8245. IEEE, 2019.

[10] T. Erez, Y. Tassa, and E. Todorov. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404. IEEE, 2015.

[11] 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 preprint arXiv:2106.13281*, 2021.

[12] 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 (TOG)*, 39(6):1–15, 2020.

[13] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme. NeuralSim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://github.com/google-research/tiny-differentiable-simulator.

[14] T. A. Howell, S. Le Cleac'h, S. Singh, P. Florence, Z. Manchester, and V. Sindhwani. Trajectory optimization with optimization-based dynamics. *arXiv preprint arXiv:2109.04928*, 2021.

[15] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020.

[16] B. E. Jackson, K. Tracy, and Z. Manchester. Planning with attitude. *IEEE Robotics and Automation Letters*, 6 (3):5658–5664, 2021.

[17] D. Kang and J. Hwangho. SimBenchmark, 2021. URL https://leggedrobotics.github.io/SimBenchmark/.

[18] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1-3):193–228, 1998.

[19] Z. Manchester and S. Kuindersma. Variational contact-implicit trajectory optimization. In *Robotics Research*, pages 985–1000. Springer, 2020.

[20] 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*, pages 1800–1809, 2018.

[21] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, 10:357–514, 2001.

[22] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2 (4):575–601, 1992.

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

[24] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

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

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

[27] H. Sharma, M. Patil, and C. Woolsey. Energy-preserving variational integrators for forced lagrangian systems. *Communications in Nonlinear Science and Numerical Simulation*, 64:159–177, 2018.

[28] 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*, 39(15):2673–2691, 1996.

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

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

[31] R. Tedrake and the Drake Development Team. Drake:

Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

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

[33] E. Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061. IEEE, 2014.

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

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

[36] 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 preprint arXiv:2103.16021*, 2021.

Algorithm 1 Analytical Line Search For Cones

procedure CONELINESEARCH($w, \Delta, \tau^{\text{ort}}, \tau^{\text{soc}}$)
  $\alpha_y^{\text{ort}} \leftarrow \alpha(y^{(1)}, \tau^{\text{ort}} \Delta^{y^{(1)}})$ ▷ Eq. 41
  $\alpha_z^{\text{ort}} \leftarrow \alpha(z^{(1)}, \tau^{\text{ort}} \Delta^{z^{(1)}})$ ▷ Eq. 41
  $\alpha_y^{\text{soc}} \leftarrow \min_{i \in \{2,\ldots,n\}} \alpha(y^{(i)}, \tau^{\text{soc}} \Delta^{y^{(i)}})$ ▷ Eq. 46
  $\alpha_z^{\text{soc}} \leftarrow \min_{i \in \{2,\ldots,n\}} \alpha(z^{(i)}, \tau^{\text{soc}} \Delta^{z^{(i)}})$ ▷ Eq. 46
  Return $\min(\alpha_y^{\text{ort}}, \alpha_z^{\text{ort}}, \alpha_y^{\text{soc}}, \alpha_z^{\text{soc}})$

APPENDIX A
SOLVER ALGORITHMS

The cone line search for all individual cones is summarized in Algorithm 1. Additional tolerances $\tau \in [0.9, 1]$ are used to improve numerical reliability of the solver. The primal-dual interior-point algorithm used to solve (29) is summarized in Algorithm 2.

Parameters: 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, c_{tol} and κ_{tol} are the only parameters the user might want to tune. Larger tolerance values leads to softer contact models, which might be desirable when computing gradients of the dynamics.

Output: The algorithm outputs a solution w that satisfies the solver tolerance levels and, optionally, the Jacobian of the solution with respect to the problem parameters θ.

APPENDIX B
QUATERNION ALGEBRA

We use a set of conventions for notating standard quaternion operations adopted from [4, 16] in the rotational part of our variational integrator (19). Quaternions are written as four-dimensional vectors:

$$q = (s, v) = (s, v_1, v_2, v_3) \in \mathbf{H}, \tag{57}$$

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

$$q^a \otimes q^b = \begin{bmatrix} s^a s^b - (v^a)^T v^b \\ s^a v^b + s^b v^a + v^a \times v^b \end{bmatrix} \tag{58}$$

$$= L(q^a) q^b = R(q^b) q^a, \tag{59}$$

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

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

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

Algorithm 2 Primal-Dual Interior-Point Solver

procedure SOLVER(x_0, y_0, z_0, θ)
  Parameters: $\tau_{\text{max}}^{\text{soc}} = 0.99, \tau_{\text{min}} = 0.95$
  $c_{\text{tol}} = 10^{-6}, \kappa_{\text{tol}} = 10^{-6}, \beta = 0.5$
  Initialize: $x = x_0, y = y_0 \succeq 0, z = z_0 \succeq 0$
  $c_{\text{vio}}, \kappa_{\text{vio}} \leftarrow$ VIOLATION(w) ▷ (33, 34)
  Until $c_{\text{vio}} < c_{\text{tol}}$ **and** $\kappa_{\text{vio}} < \kappa_{\text{tol}}$ **do**
    $\Delta^{\text{aff}} \leftarrow -\bar{R}^{-1}(w;\theta) r(w;\theta,0)$
    $\alpha^{\text{aff}} \leftarrow$ CONESEARCH($w, \Delta^{\text{aff}}, 1, 1$)
    $\mu, \sigma \leftarrow$ CENTER($y, z, \alpha^{\text{aff}}, \Delta^{\text{aff}}$) ▷ (49-52)
    $\kappa \leftarrow \max(\sigma\mu, \kappa_{\text{tol}}/5)$
    $\Delta \leftarrow -\bar{R}^{-1}(w;\theta) r(w;\theta,\kappa)$
    $\tau^{\text{ort}} \leftarrow \max(\tau_{\text{min}}, 1 - \max(c_{\text{vio}}, \kappa_{\text{vio}})^2)$
    $\tau^{\text{soc}} \leftarrow \min(\tau_{\text{max}}^{\text{soc}}, \tau^{\text{ort}})$
    $\alpha \leftarrow$ CONESEARCH($w, \Delta, \tau^{\text{ort}}, \tau^{\text{soc}}$)
    $c_{\text{vio}}^*, \kappa_{\text{vio}}^* \leftarrow c_{\text{vio}}, \kappa_{\text{vio}}$
    $\hat{w} \leftarrow$ UPDATE(w, Δ, α) ▷ (47, 48)
    $c_{\text{vio}}, \kappa_{\text{vio}} \leftarrow$ VIOLATION($\hat{w}$) ▷ (33, 34)
    Until $c_{\text{vio}} \leq c_{\text{vio}}^*$ **or** $\kappa_{\text{vio}} \leq \kappa_{\text{vio}}^*$ **do**
      $\alpha \leftarrow \beta\alpha$
      $\hat{w} \leftarrow$ UPDATE(w, Δ, α) ▷ (47, 48)
      $c_{\text{vio}}, \kappa_{\text{vio}} \leftarrow$ VIOLATION($\hat{w}$) ▷ (33, 34)
    end
    $w \leftarrow \hat{w}$
  end
  $\partial w^*/\partial\theta \leftarrow -\bar{R}^{-1}(w^*;\theta)\bar{D}(w^*;\theta)$ ▷ (17)
  Return $w, \partial w^*/\partial\theta$

where,

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

is defined such that,

$$\mathbf{skew}(x) y = x \times y, \tag{63}$$

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

$$v = Vq, \tag{64}$$

is extracted using the matrix:

$$V = \begin{bmatrix} \mathbf{0} & I_3 \end{bmatrix} \in \mathbf{R}^{3\times 4}, \tag{65}$$

and quaternion conjugate:

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

is computed using:

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