DIFFERENTIABLE SIMULATION OF HARD CONTACTS WITH SOFT GRADIENTS FOR LEARNING AND CONTROL

Anselm Paulus[1*], **A. René Geist**[1*],
Pierre Schumacher[*2], **Vít Musil**[3], **Simon Rappenecker**[1],
Georg Martius[1]

[1] University of Tübingen, Tübingen, Germany
[2] MyoLab.AI, New York City, USA
[3] Masaryk University, Brno, Czechia

{anselm.paulus,rene.geist,georg.martius}@uni-tuebingen.de

ABSTRACT

Contact forces introduce discontinuities into robot dynamics that severely limit the use of simulators for gradient-based optimization. Penalty-based simulators such as MuJoCo, soften contact resolution to enable gradient computation. However, realistically simulating hard contacts requires stiff solver settings, which leads to incorrect simulator gradients when using automatic differentiation. Contrarily, using non-stiff settings strongly increases the sim-to-real gap. We analyze penalty-based simulators to pinpoint why gradients degrade under hard contacts. Building on these insights, we propose DiffMJX, which couples adaptive time integration with penalty-based simulation to substantially improve gradient accuracy. A second challenge is that contact gradients vanish when bodies separate. To address this, we introduce contacts from distance (CFD) which combines penalty-based simulation with straight-through estimation. By applying CFD exclusively in the backward pass, we obtain informative pre-contact gradients while retaining physical realism.
Project page: https://github.com/martius-lab/diffmjx

1 INTRODUCTION

Gradients have powered major advances in machine learning ranging from video language models to robot control. In robotics, imitation and reinforcement learning widely rely on gradient-based optimization. Yet, despite the dominance of sim-to-real techniques that leverage robot simulators for policy learning (Tan et al., 2018; Lee et al., 2020; Andrychowicz et al., 2020; Radosavovic et al., 2024; Li et al., 2022), most methods conspicuously avoid using simulator gradients. This is a missed opportunity considering that simulator gradients offer a direct route to updating actions and learning model parameters. If these gradients were accurate, we could fit simulators to real data significantly narrowing the sim-to-real gap and accelerate policy learning – enabling policies for new tasks to be trained in seconds rather than hours. *Given the utility of simulator gradients, what prevents their use in robot learning?* In practice, two fundamental issues hinder the use of simulator gradients: (i) discontinuities arising from contacts yield **erroneous gradients**, and (ii), if objects do not touch, **contact gradients are zero**. In this work, we chart a path that tackles both challenges.

Differentiating through contacts. The choice of contact model has significant implications on a simulator's differentiability. Complementary-based solvers, such as Taylor et al. (2022), compute contact forces exactly, such that sudden jumps in the dynamics aggravate gradient computation. Hence, recent literature on differentiating through complementary-based solvers proposes analytical reformulations of the dynamics, e.g. based on the implicit function theorem (Werling et al., 2021; Taylor et al., 2022), or resorts to randomized smoothing (Tassa & Todorov, 2010; Duchi et al., 2012; Suh et al., 2022; Bouyarmane et al., 2009; Xu et al., 2010). While differing in computational cost, both approaches have similar empirical performance Pang et al. (2023); Schwarke et al. (2024). To improve computational efficiency, MuJoCo (Todorov, 2014) reformulates complementary constraints

[*] These authors contributed equally.

as a convex optimization problem, where a constraint's ability to generate force grows proportionally to the constraint violation. While such a penalty-based simulator can be made smooth, gradient inaccuracies increase with the *contact stiffness*, the *relative contact velocity*, and the *integration step size*. While common advice points to reducing the step size to improve gradient accuracy, using sufficiently small step sizes results in prohibitively slow simulation.

Computing gradients of stiff simulations. In the sequel, we show that gradient errors in penalty-based simulators arise from computing a discrete approximation of continuous physics – put simply, from numerically integrating stiff differential equations. Consequently, the gradients obtained via automatic differentiation are "incorrect" insofar as they do not align with those of the underlying continuous system. Although adaptive integration is well studied, it's rarely used in robotics and to the best of our knowledge its utility for differentiable simulation has not been explored. This perspective complements prior proposals for time-of-impact correction (Hu et al., 2020; Schwarke et al., 2024) in impulse-based simulators. We incorporate **adaptive timestep integration** into MuJoCo XLA, incurring a slight computational overhead while obtaining correct gradients in the presence of hard contacts, and remaining compatible with existing MuJoCo libraries.

Computing gradients between non-colliding objects. Another obstacle for gradient-based optimization for policy generation and system identification is the non-informativeness of gradients about unrealized contacts. For example, when a robot's hand is not in contact with an object, then there is no gradient directing it to make contact for task facilitation. Therefore, drawing inspiration from prior works on contact-invariant optimization (Mordatch et al., 2012b;a), we propose **Contacts From Distance** (CFD) to address this challenge. However, naively introducing artificial contact forces considerably alters the simulation, resulting in a too large sim-to-real gap. In order to preserve the simulation realism, we propose using the *straight-through-trick* to introduce CFD solely in the gradient computation. We implement our changes in Mujoco XLA, and additionally fix some low-level collision routines to be truly differentiable. CFD complements position-based costs often used in robot learning by removing the need to specify exact contact locations.

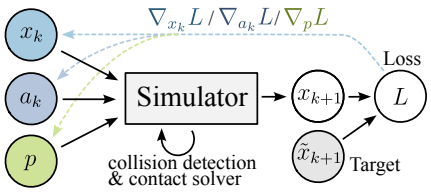


Figure 1: Common computational graph for robot control synthesis.

In summary, DiffMJX and CFD provide additional tuning knobs to enable practitioners to set gradient correctness and inform an optimization on how non-colliding objects would need to move to influence each other's state. In turn, this work provides a new perspective on how to obtain useful gradients in penalty-based differentiable simulators thereby enabling parameter estimation and policy synthesis for collision-rich and high-dimensional systems.

2 Robot Simulation

As illustrated in Fig. 1, we want to use automatic differentiation to obtain the *correct* gradients of a loss functional $L(\tilde{x}_{k+1}, x_k, a_k, p)$ where the next state of the robotic system is governed by the discrete-time dynamics $x_{k+1} = \text{step}(x_k, a_k, p)$ with the state $x_k := x(t_k) = [q_k, v_k]$ at time t_k consisting of the system's generalized position $q_k \in \mathbb{R}^{n_q}$ and velocity $v_k \in \mathbb{R}^{n_v}$, control actions $a_k \in \mathbb{R}^{n_a}$, and model parameters $p \in \mathbb{R}^{n_p}$. Typically, multi-body dynamics simulators consist of *forward dynamics model* and a *numerical integration method*. The forward dynamics govern the system's acceleration $\dot{v}$ via the equations of motion

$$\dot{v} = M^{-1}\left(\tau - c + J^\top f\right) \tag{1}$$

with the joint-space inertia matrix $M(q) \in \mathbb{R}^{n_v \times n_v}$, applied forces $\tau(x, a) \in \mathbb{R}^{n_v}$, bias force $c(x) \in \mathbb{R}^{n_v}$, constraint space Jacobian $J(q) \in \mathbb{R}^{n_c \times n_v}$, and constraint forces $f(x) = f_\mathcal{E} + f_\mathcal{F} + f_\mathcal{C} \in \mathbb{R}^{n_c}$ consisting of the equality constraint, the generalized friction, and the contact constraint forces. The contact-free dynamics are typically derived via recursive multi-body algorithms (Featherstone, 2014) while computing contacts are resolved through an intricate interplay of collision detection and contact force optimization. As we will see throughout this work, numerical integration plays a pivotal role in understanding how contact forces may hinder correct gradient computation. For that, we will resort to MuJoCo XLA as a concrete example of a penalty-based simulator.

2.1 CONSTRAINT RESOLUTION IN MUJOCO XLA

MuJoCo XLA (MJX) is a reimplementation of MuJoCo using the Python library JAX (Bradbury et al., 2018), which enables GPU-parallelizable gradient computation via automatic differentiation. MuJoCo has become the de facto standard in robotics, alongside other widely used simulators such as Bullet (Coumans & Bai, 2016), Drake (Russ Tedrake, 2019), DiffTaichi (Hu et al., 2020) and Nvidia PhysX (Liang et al., 2018). MuJoCo 's importance is underpinned by NVIDIA and Google recently releasing MuJoCo Warp (Howell, 2025). In what follows, we provide a brief overview of how MuJoCo resolves contacts, which serves as a hands-on example of a penalty-based simulator.

Collision detection. Given the state x and geometry parameterizations of two bodies, a collision detector returns the *signed distance* r between potential contact points alongside with their surface normals. A contact is only considered *active* – that is, a contact point can exert contact force – if $r < 0$. While collision detection is not the primary focus of this work, we later show how to remove discontinuities from its functions to ensure differentiability.

Contact force solver. A detailed description of MuJoCo is given in its documentation, here we provide a summary of key equations. Given Eq. (1), MuJoCo resolves constraints via a relaxation of Gauss' principle (Gauß, 1829) which in its primal formulation reads

$$(\dot{v}, \dot{\omega}) = \underset{(x,y)}{\arg\min} \left\| x - M^{-1}(\tau - c) \right\|_M^2 + \|y - a_{\text{ref}}\|_{R^{-1}}^{\text{H}} \quad \text{subject to } J_{\mathcal{E}} x - y_e = 0,\ J_{\mathcal{F}} x - y_{\mathcal{F}} = 0,\ J_{\mathcal{C}}\, x - y_{\mathcal{C}} \in \mathcal{K}^* \tag{2}$$

with the friction cone (dual) $\mathcal{K}^*$, the regularizer $R > 0$, and Huber norm $\|\cdot\|^{\text{H}}$. The reference acceleration a_{ref} denotes the solver's target for the constraint space acceleration $\dot{\omega}$. Drawing inspiration from Baumgarte (1972), a_{ref} follows a damped harmonic oscillator

$$a_{\text{ref},i} = -b_i (Jv)_i - k_i r_i = -\frac{2}{d_w \cdot t_c} (Jv)_i - \frac{d(r_i)}{d_w^2 \cdot t_c^2 \cdot \phi_d^2} r_i \tag{3}$$

whose dynamics are determined by the impedance $d(r)$, and the *solref* parameters consisting of the time constant t_c and the damping ratio ϕ_d. The *impedance* $d(r)$ is the central function for determining constraint forces (such as contact forces). As illustrated in Fig. 2, the impedance $d \in [0, 1]$ is a function of the constraint violation r and is specified by the *solimp* parameters $(d_o, d_w, w, \text{midpoint}, \text{power})$ which define its shape as a polynomial spline. MuJoCo's documentation refers to the *impedance* as a "constraints ability to generate force" as it determines a_{ref} and weights the cost for applying constraint forces in Eq. (2) via the diagonal matrix R, which is computed as $R_{ii} = \hat{A}_{ii}(1 - d_i)/d_i$, where $\hat{A}$ denotes an approximation to $A = \text{diag}(JM^{-1}J^\top)$. In turn, the constraint is hard if $R \to 0$, and approaches an infinitely soft (i.e. non-existent) constraint in the limit $R \to \infty$. If $\mathcal{K}^*$ solely contains pyramidal or elliptic cone constraints, then problem (2) reduces to a convex problem that is solved efficiently via an exact Newton method.

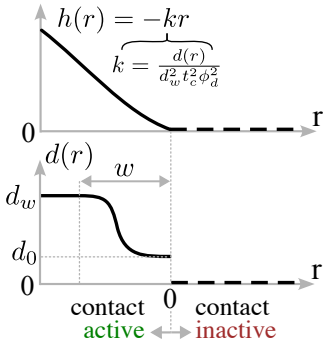


Figure 2: The position-level reference acceleration $h(r)$ and impedance $d(r)$ determine the contact force magnitudes that the solver can apply.

3 CORRECTING THE CONTACT GRADIENTS OF PENALTY-BASED SIMULATION

To evaluate the correctness of gradients in the presence of contacts, as illustrated in the top row of Fig. 3, we unroll the trajectory of several primitives bouncing against a plane to observe the final position and its gradient with respect to the initial velocity. In Fig. 3 (MJX row), at the default integration step of 0.002 s, we observe that the loss is oscillating at a high frequency, which results in large fluctuations of the gradient. Similar oscillations have been discussed in previous works (Hu et al., 2020; Schwarke et al., 2024). In particular, Hu et al. (2020) emphasizes that the "time-of-impact" (TOI) causes gradient oscillations when using ideal elastic or complementary collisions. Yet, TOI gradient errors differ in nature from errors observed in penalty-based simulators; as illustrated below.

Example – Point collision: As illustrated in Fig. 4 (left), a point mass starts at height q_0 with velocity $v_0 = -1$ and is integrated for N steps with semi-implicit Euler in the absence of gravity. It collides with a flat surface and bounces back up. Contact collision is either resolved via a minimal version of a penalty-based contact model as found in MuJoCo or an ideal elastic collision

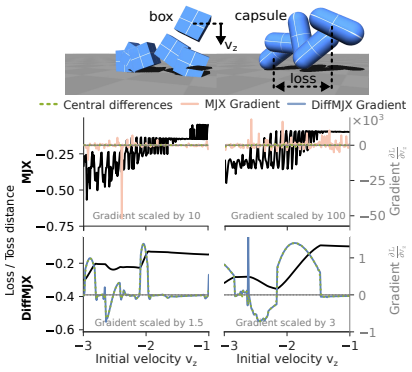


Figure 3: Simulation of primitives thrown onto a surface. For stiff contacts, MJX's gradients of the toss distance deviate from central differences, while DiffMJX maintains agreement.

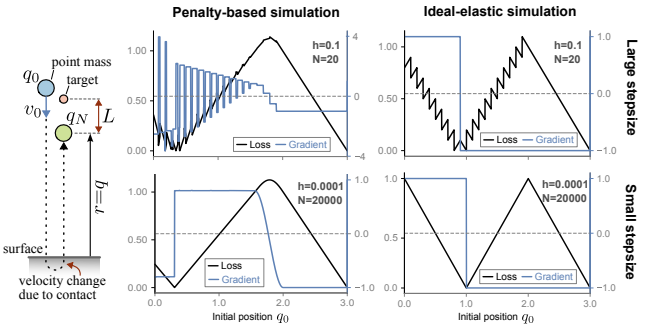


Figure 4: Toy simulation of a point mass colliding with a surface that either resorts to an ideal-elastic contact model used in DiffTaichi or a penalty-based contact model similar to MuJoCo. Decreasing the integration stepsize h solely reduces errors in the penalty-based simulation, in the ideal-elastic simulation the gradient sign remains wrong.

as in (Hu et al., 2020). The corresponding JAX code is shown in Fig. 16 in the Appendix. The loss $L = |q_N - q_T|$ is the distance between the point's final state q_N and the target height $q_T = 1$. For ideal-elastic collisions, we observe sawtooth-like loss oscillations in L, resulting in the gradient $\nabla_{q_0} L$ with *the wrong sign*, independently of the stepsize as observed by Hu et al. (2020). For the penalty-based simulation, gradient oscillations *notably reduce when lowering the stepsize*.

We observe that state oscillations and the corresponding gradient artifacts are not a characteristic of the specific simulator. Instead, more fundamentally, they arise from time-discretization errors in the ODE integration. While this affects both penalty-based and ideal-elastic simulators, we will see that the issue has to be addressed differently.

TOI correction does not fix gradients for penalty-based simulators, but small stepsizes do. For an ideal elastic collision, the ODE of our minimal example is piecewise linear: The dynamics are linear (due to the absence of gravity) before and after the contact, at which the velocity is inverted (see also Fig. 16 in the Appendix). The TOI approach proposed by Hu et al. (2020) exploits this structure by dynamically splitting the ODE into two linear segments at the time of contact. Integrating those separately thereby eliminates the discretization error and yields correct gradients. In penalty-based simulation, the ODE is linear before and after the collision, but is non-linear with variable stiffness over the time of the collision. Therefore, it cannot be easily divided into large linear segments.

Luckily, for penalty-based simulation we can instead rely on a different technique for reducing the gradient error. The simple solution is to reduce the stepsize, which does not work in the ideal elastic case, as seen in Fig. 4. From an ODE perspective, this works because the penalty-based ODE is smooth, allowing us to continuously control the integration error by reducing the stepsize. Unfortunately, simply reducing the step size is not a practical solution, as it necessitates extremely small steps that substantially increase the computational and memory demands of gradient computation. This trade-off raises a critical question: *Can we retain correct gradients of realistic contacts while maintaining practical simulation speeds?*

3.1 Adaptive stepsize integration: Numerical precision on demand

A standard method for integrating ODEs with variable stiffness is *adaptive integration*. The idea behind adaptive integration is elegant: Two numerical integrators of different orders compute the next state. Their difference provides an estimate of the error. If the error is smaller than a given threshold, the step is accepted; otherwise, the step is rejected and the procedure is repeated with a different stepsize chosen by a feedback controller. For further details on the rich history of adaptive stepsize integration, see e.g. Hairer et al. (2008); Hairer & Wanner (2002); Söderlind (2002; 2003).

We use Diffrax (Kidger, 2021) for efficient numerical integration in JAX, taking advantage of its solver flexibility and multiple backpropagation modes. *Notably, as detailed in Appendix C.2, we*

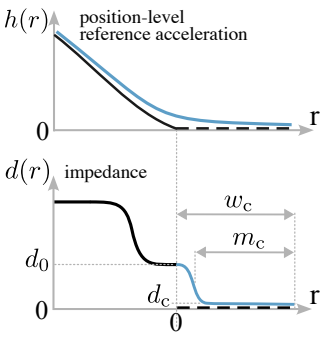


Figure 6: Contacts from distance (CFD): To let MuJoCo create small contact forces between non-colliding objects, reference acceleration $h(r)$ and impedance $d(r)$ are adjusted to be nonzero for positive signed distances $r > 0$.

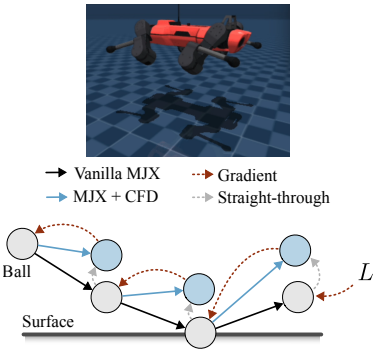


Figure 7: **Top:** Applying contact forces for $r > 0$ in the forward pass of the simulation causes a robot to hover. **Bottom:** The straight-through-trick is used to replace the original MJX derivative with the derivative of MJX + CFD, evaluated at the unaltered trajectory.

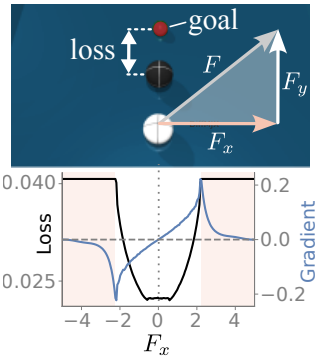


Figure 8: Billiard simulation. **Top:** Force F acts on the white ball affecting the loss. **Bottom:** Despite the loss derivative being zero if the balls do not collide, DiffMJX with CFD provides informative non-zero gradients.

devoted substantial effort to seamlessly integrating quaternions and stateful actuators, which enables seamless compatibility between Diffrax and MJX while ensuring efficient adaptive integration and backwards compatibility with other MuJoCo libraries.

Testing our implementation on the cube bounce toy example, we observe in Fig. 5 that adaptive integration reduces the error in the loss and gradient by multiple orders of magnitude given the same computational budget; more detailed analysis is in Fig. 17 in the Appendix. Importantly, the adaptive stepsize selection in Diffrax is independent across parallelized environments. Therefore, when vmapping across environments, contacts in one simulation do not slow down other simulations. For experimental verification, see Fig. 20 in Appendix D.

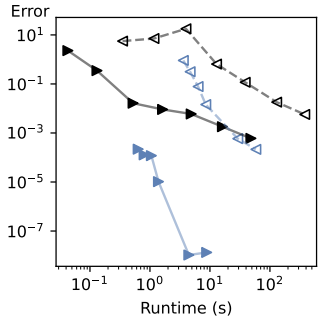


Figure 5: Pareto front of gradient error vs forward runtime (▶) and gradient error vs backward runtime (◁) for standard semi-implicit Euler in MJX (**black**) and adaptive integration in DiffMJX (**blue**) on the cube bounce toy-example.

Resolving problems of Collision Detection. Using an adaptive integrator with MJX eliminates oscillations in the bounce example. However, gradients for some object primitives (capsule, cylinder, box) experience gradient artifacts due to non-differentiable operations in the collision detector arising from discrete case distinctions. We smoothed them with standard proxies, finally leading to the results in the bottom row of Fig. 3, where analytical gradients nearly match central differences. Henceforth, we refer to MJX with the Diffrax integrator and smoothed collision detection as DiffMJX.

4 Contacts from Distance with Straight-Through Estimation

While adaptive integration improves gradient accuracy, we now shift attention to computing *informative gradients* between objects that are not in contact. To illustrate why the computation of such gradients is of fundamental importance for robot learning consider the following example.

Example – Billiard shot: A billiard table is set up as shown in Fig. 8 (top). At the first timestep, a force F is exerted on the white ball such that it may hit the black ball. The optimization objective is the distance L between the black ball and the target position. If the balls collide, MJX with

adaptive integration yields non-zero gradients $\nabla_F L$. However, if F does not cause the balls to touch, then $\nabla_F L$ is zero and therefore uninformative for optimization.

In the following, we propose *contacts from distance* (CFD), a method for computing contact forces for positive signed distances r in the gradient computation of a penalty-based simulation that yields informative gradients even if objects are not in contact. This is accomplished by ***(i)* applying artificial contact forces** between non-colliding objects, and ***(ii)* using artificial forces only in the gradient computation** to maintain simulation realism.

Creating artificial contact forces. How to generate artificial contact forces in a penalty-based simulator depends on the respective contact model. Below, as a concrete example, we propose a method tailored to MuJoCo. As discussed in Section 2, the magnitude of contact forces is determined by the impedance $d(r)$ and position-level reference acceleration $h(r)$. To enable the solver to apply CFD, $d(r)$ is augmented as shown in Fig. 6. Here, $d(r)$ remains unaltered for $r < 0$ and is extended by an additional spline for $r > 0$. This continuation is parametrized by *solimp-CFD* parameters $(d_c, d_0, w_c, m_c, p_c)$. By default, the curve smoothly continues MuJoCo's impedance at d_0 and tapers off to $d_c = 0$ to ensure smooth differentiability. The CFD-width w_c specifies the distance for which artificial contact forces are generated. Moreover, we soften the reference acceleration $h(r)$ by replacing the ReLU function on the signed distance with a softplus (Fig. 6). This yields modified contact forces f_{CFD}.

Designing a surrogate gradient estimator. Naively adding CFD to a simulation produces non-physical behaviors. As shown in Fig. 7 (top), the artificial contact forces would cause a quadruped to hover above the ground as if a soft foam mat of thickness w_c had been placed on the surface. As significantly altering simulation realism is not an option, we are faced with the question: *Can CFD be used to obtain informative contact gradients without affecting simulation realism?*

To positively answer this question, we resort to the *straight-through-trick* on the ODE level:

$$\dot{x}(t) = \text{sg}(F_\theta(t, x(t))) + \tilde{F}_\theta(t, x(t)) - \text{sg}(\tilde{F}_\theta(t, x(t))), \tag{4}$$

where sg is the stop-gradient operator of an automatic differentiation library. Here F denotes the original ODE obtained from MJX or DiffMJX and $\tilde{F}$ denotes the ODE using CFD via $\dot{v}_{\text{CFD}} = M^{-1}\left(\tau - c + J^\top f_{\text{CFD}}\right)$. In JAX this reads as:

```
from jax.lax import stop_gradient
def forward(m: Model, d: Data) -> Data:
    d_mjx = _forward(m, d, cfd=False)  # Compute system acceleration without CFD
    d_cfd = _forward(m, d, cfd=True)   # Compute system acceleration using CFD
    grad_replace_fn = lambda x_mjx, x_cfd: stop_gradient(x_mjx) + x_cfd - stop_gradient(x_cfd)
    return jax.tree.map(grad_replace_fn, d_mjx, d_cfd)  # Reroute gradient computation
```

As illustrated in Fig. 7 (bottom), the above code ensures that the forward pass uses $F_\theta(t, x(t))$, whereas the backward pass deploys $\frac{\partial \tilde{F}_\theta}{\partial(x,\theta)}(t, x(t))$. Crucially, the derivatives are evaluated at the unmodified forward trajectory $x(t)$. While our approach requires the simulator's forward pass to be computed twice, the gradient is only evaluated once. As the gradient computation dominates the computational cost, CFD forms a practical method for improving contact gradients. Revisiting the billiard toss example (Fig. 8), using CFD with the straight-through-trick yields informative gradients while keeping the loss unaltered.

Note that the straight-through-trick is inspired by prior work on straight-through estimators (Bengio et al., 2013); see also the related work in Appendix A. DiffMJX and CFD introduce easy-to-use tuning knobs in MJX to control gradient quality. Appendix B explains how to tune these knobs. In Appendix C.2, we evaluate the straight-through-trick with different autodiff techniques, including "Discretize-then-optimize" and "Optimize-then-discretize."

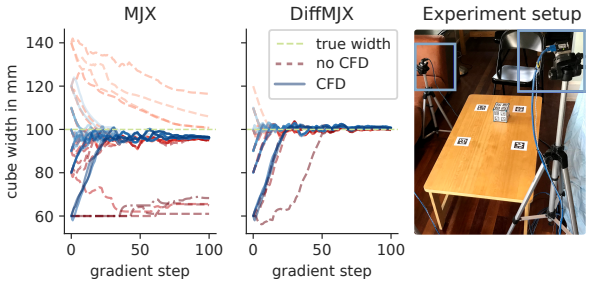


Figure 9: **Left:** Estimation of a cube's side length in MJX via gradient descent using multi-step ahead predictions. **Right:** Experimental setup for collecting cube toss data. Image adapted from Pfrommer et al. (2021).

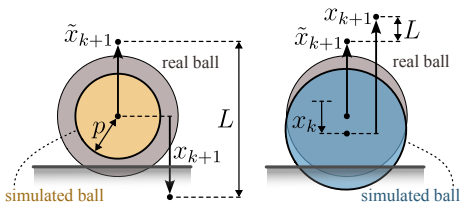
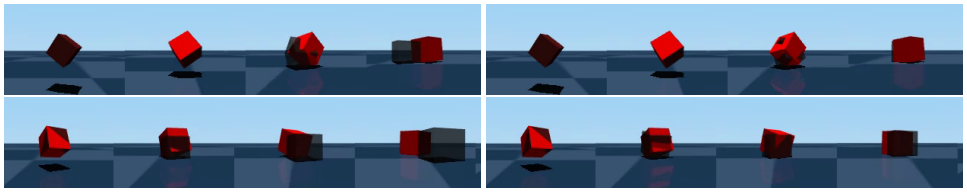


Figure 10: **Left:** If the initial ball radius p is set too small, then the gradient $\nabla_p L$ resulting from a one-step-ahead prediction is zero. **Right:** Large initial constraint penetration results in large gradients $\nabla_{x_k} L$.



Before training After training

Figure 11: Comparison between real-world cube tosses (**red**) and DiffMJX cube simulations (**black**).

5 Evaluation

In the following, we demonstrate the use of DiffMJX and CFD for learning physics parameters and computing robot control actions using gradient descent with simulator gradients.

5.1 Parameter Identification

Parameter identification in the presence of hard contacts remains a laborious task. If contacts are hard, learning the dynamics of a cube requires impractical amounts of data for "naive" neural network regression (Parmar et al., 2021). In comparison, penalty-based simulators can capture hard contacts, but the lack of correct gradients hinders efficient parameter estimation (Acosta et al., 2022). Therefore, recent work introduced intricate analytical pipelines for cube geometry estimation (Pfrommer et al., 2021; Bianchini et al., 2023) and graph-based networks for learning contact dynamics (Allen et al., 2023). In what follows, we use the same real-world data as used in Pfrommer et al. (2021); Bianchini et al. (2023); Allen et al. (2023). We demonstrate that DiffMJX with CDF enables simulator parameter estimation via standard gradient-based optimization.

Dataset and training setup. We use the Contactnets dataset (Pfrommer et al., 2021) which consists of 550 trajectories of a 10 cm acrylic cube that has been repeatedly tossed onto a wooden table. For training, trajectories are split into segments of length five such that the simulator is tasked to unroll four future steps starting from the initial state. Each segment and its prediction are fed to an L_2 loss whose gradient is used for gradient-based optimization using Adam (Kingma & Ba, 2015). For systems with stiff dynamics, we favor multi-step-ahead predictions over one-step-ahead predictions, as they capture the cumulative effects of prediction errors over time. This setup enables a fairer analysis of MJX without CFD, as even for too small side length estimates as illustrated in Fig. 10, future state predictions can make contact to inform the optimization.

Training results. The training results are shown in Fig. 9. Our version of MJX with smooth collision detection and MJX with CFD both achieve good estimation results with an error of around 5% relative to the ground truth. If the side length is initialized at 60 mm or 140 mm, training either stalls fully or convergence is severely limited for MJX. The incorporation of CFD into MJX addresses convergence issues arising from poor initial parameters, while the integration of adaptive integration via DiffMJX significantly enhances estimation accuracy. DiffMJX improves estimation accuracy

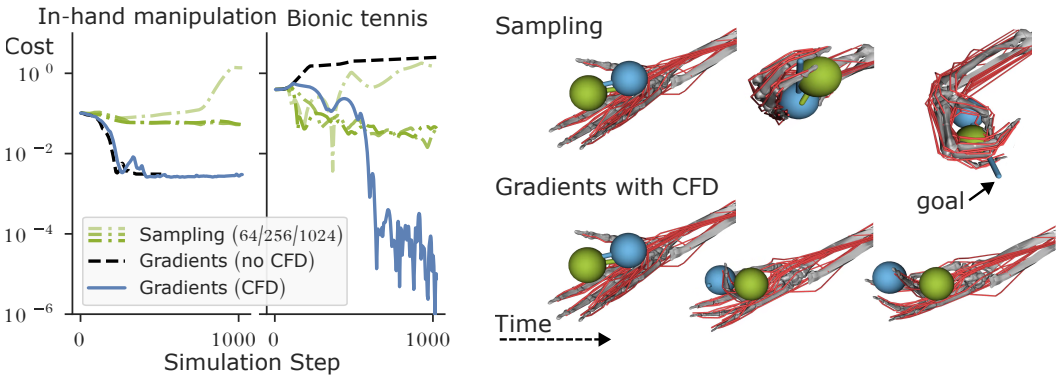


Figure 12: Autodiff-driven MPC with and without CFD on the in-hand manipulation and bionic tennis tasks (Fig. 13). **In both tasks, only the distance between the ball and the respective goal is used as cost. Left:** Simulation cost evolution of gradient-based MPC, with and without contacts from distance (CFD), vs sampling-based MPC. The number for sampling indicates the number of samples used per planning step. Sampling has difficulties solving the dexterous in-hand manipulation task; gradients without CFD cannot solve the bionic tennis task. **Right:** Rendering of sampling-based MPC (1024 samples) vs gradient-based MPC with CFD on the in-hand manipulation task. The goal is to swap the balls, with the cost computed as L2 distance of the ball centers to positions fixed in the frame of the hand. The MyoHand model is actuated by 39 muscle-tendon units.

by dynamically adjusting the time steps during collisions, thereby mitigating time discretization errors. Further details and additional experiments in which also the contact parameters were identified are provided in E.2. Fig. 11 provides a comparison of DiffMJX's predictions after *estimating contact and geometry parameters*. To the best of our knowledge, we are the first to demonstrate parameter estimation of real-world cube dynamics using an automatically differentiable penalty-based simulator. While this represents a promising step forward, further experimentation is necessary to fully characterize the scope and limitations of this approach.

5.2 Model Predictive Control

Next, we conduct experiments on gradient-based model-predictive control. At every plan step in the MPC loop, we refine a sequence of controls over a 256-step horizon. In the gradient-based planner, we compute gradients by backpropagating the differentiable cost computed on the rollout of the current plan through the MJX simulator. The plan is then iteratively optimized using the Adam optimizer with a learning rate of 0.01 for 32 iterations. Finally, the resulting plan is executed for 16 steps in simulation, after which the planning procedure is repeated with the previous plan as a warm start. As a baseline, we include a version of the predictive sampling planner from Mujoco MPC (Howell et al., 2022), which at every plan step samples $k = \{64, 256, 1024\}$ trajectories, and executes the lowest-cost plan. For enabling a fair comparison, *we significantly improved the performance of this planner for muscular systems by resorting to brown noise for sampling* (Pinneri et al., 2020).

Models. As physical systems, we resort to state-of-the-art muscle-tendon models provided by MyoSuite (Caggiano et al., 2022; Wang et al., 2022). Models include the MyoHand (Fig. 12, right) adapted from the MyoChallenge 2022, which is comprised of 29 bones, 23 joints, and 39 muscle-tendon units. We also use a bionic model (Fig. 13) modified from the MyoChallenge 2024, which is comprised of the MyoArm with 27 degrees of freedom and 63 muscle-tendon units, and the simulated modular prosthetic limb with 26 degrees of freedom and 17 motor control units.

Dexterous in-hand manipulation. First, we consider an in-hand manipulation task, where the goal is to swap two balls in the MyoHand. The cost is given by the Euclidean distance between each of the balls and the respective target location, fixed in the frame of the hand. Note that the muscle actuator implementation caused gradient errors, which we corrected using smooth functions as surrogates. The results are reported in Fig. 12. We find that gradient-based MPC can reliably solve this task, in contrast to the sampling-based planner. Overparameterization in the muscle-tendon model with at least two muscles per joint actually benefits the gradient-based planner by helping

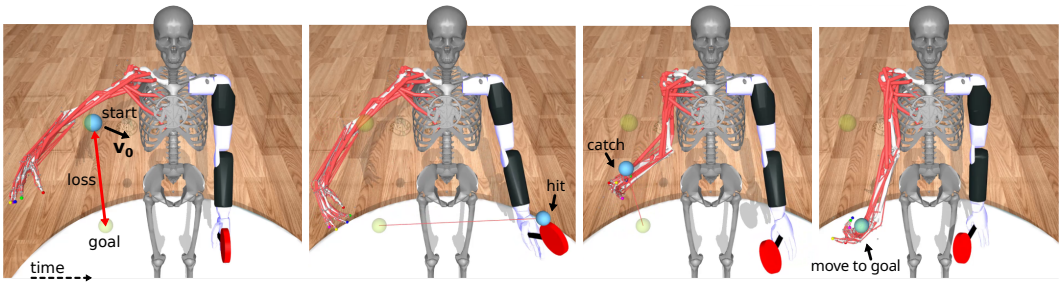


Figure 13: Autodiff-driven MPC with CFD on the bionic tennis task. Task completion requires the racket to deflect the ball towards the MyoArm with 63 muscle-tendon actuators, which then catches the ball and moves it to the goal position. **Only the distance between ball and goal is used as cost.**

escape local minima, similar to its role in optimizing overparametrized neural networks. In contrast, RL and sampling-based planners struggle with scaling in overparametrized higher-dimensional systems (Schumacher et al., 2023). Thus, first-order methods using differentiable simulation should be able to tackle more complex control problems.

Notably, this task does not require contacts from distance because hand-ball interactions are frequent due to gravity. Moreover, we identify two crucial components of the gradient-based MPC loop: First is gradient clipping, which is important as the scale of gradients changes massively in the presence of contacts, as illustrated in Fig. 10 (right). This technique has also been reported to be effective in previous works on differentiable simulation (Xu et al., 2022; Georgiev et al., 2024). Second, we store the rollout cost of all gradient iterations and select the one with minimal cost. This is important as the cost landscape is highly non-convex, which is reflected in the non-monotonic cost evolution between the iterations of a planning step.

Bionic tennis: Using CFD to solve complex control tasks with minimal task supervision. Next, we test a more complex custom tennis task on the bionic model. The task is to move a ball that is initially moving sideways to a target location below. This can be achieved by bouncing it back using a racket that is welded to the prosthetic hand, and then catching it at the target location with the muscle hand. In this task, *the only cost supervision is again the Euclidean distance of the ball to the target*, the complicated sequential movement has to be discovered purely from this signal.

We report our findings in Fig. 12 (left), see Fig. 13 for a rendering. By design, the task initialization is such that the ball misses both hands, hence this task is not solvable by purely gradient-based MPC using vanilla MJX. On the other hand, we observe that adding the CFD mechanism allows solving this task. The sampling-based planner is a strong baseline in this task and gets close to solving it. Initially, bouncing the ball back to the target only requires controlling the prosthetic arm, which is relatively low-dimensional. Hence, the sampling-based planner achieves this part easily. However, as seen in the in-hand manipulation task, it struggles with precise control of the high-dimensional MyoHand, leading to sub-optimal results in balancing the ball at the goal position.

Cube reorientation. Finally, we tackle another manipulation task of reorienting a cube to a desired target orientation, while staying at a target position. We include both an in-hand version and an on-the-table version of this task.

We report our findings in Fig. 14. We observe that in the in-hand reorientation task, the sampling-based planner fails, while the gradient-based approach solves it. Similar to the ball-swap task, CFD does not make a big difference here, as contacts are automatically discovered from gravity. In the on-the-table reorientation task, only the gradient-based planner with CFD activated solves the task perfectly. When deactivating CFD or using sampling, the planner still manages to reorient the cube to a satisfactory level, but fails to keep it in the target location. Here, gravity alone does not result in all fingers making contact with the cuber, therefore CFD allows higher-fidelity control in comparison to CFD disabled.

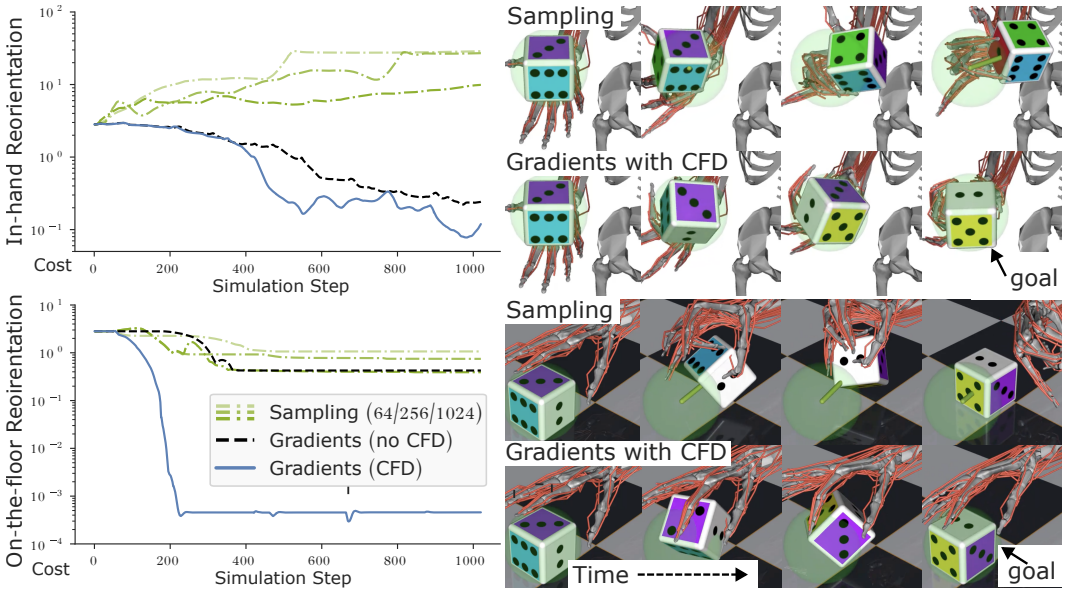


Figure 14: Autodiff-driven MPC with and without CFD on the in-hand and on-the-floor cube reorientation tasks. In both tasks, the loss consists of the L2 distance between the cube center and the target location, as well as a squared loss on the difference of quaternions describing the cube and target orientation. **Top:** Simulation cost evolution of gradient-based MPC, with and without contacts from distance (CFD), vs sampling-based MPC. The number for sampling indicates the number of samples used per planning step. Sampling has difficulties solving the the in-hand reorientation task; only gradients with CFD can solve the on-the-table reorientation task perfectly. **Right:** Rendering of sampling-based MPC (1024 samples) vs gradient-based MPC with CFD on the in-hand and on-the-table reorientation task. The goal is to reorient the cubes while staying at the target location (green sphere). The MyoHand model is actuated by 39 muscle-tendon units.

6 Conclusion

In this work, we tackle two standing challenges of computing gradients of penalty-based simulations with hard contacts, namely gradient errors due to time discretization and zero contact gradients between non-colliding objects. A 1D toy example illustrates how gradient errors are a consequence of time discretization and can be mitigated by reducing the integration stepsize. While reducing the stepsize improves gradient accuracy, it comes at the cost of increased simulation time and GPU memory usage. To address these limitations, we introduce DiffMJX, a JAX library that incorporates the adaptive integration library Diffrax into MuJoCo XLA. The use of adaptive time integration enables the reduction of discretization errors while significantly reducing memory overhead through checkpointing. Complementing DiffMJX, we propose Contacts from Distance (CFD), an extension of penalty-based contact resolution that improves gradient utility by introducing small virtual contact forces between near-interacting bodies, without affecting the forward simulation due to the straight-through-trick. We validated our contributions in a real-world system identification study, where DiffMJX accurately recovered a cube's geometric parameters via gradient-based optimization. Beyond this, we present the first evidence that automatic differentiation through CFD can outperform sampling-based predictive planning on musculoskeletal manipulation tasks. We hope these results will guide future work on mitigating discretization errors in differentiable simulators and establish DiffMJX with CFD as a helpful toolbox for gradient-based robot learning.

ACKNOWLEDGEMENTS

The authors thank Onur Beker, Thomas Rupf, and Nico Gürtler for providing valuable feedback.

We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Anselm Paulus.

This work was supported by the ERC - 101045454 REAL-RL and the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B). Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645.

The work on this paper was supported by the Czech Science Foundation (GAČR) grant no. 26-23981S.

REFERENCES

Brian Acosta, William Yang, and Michael Posa. Validating robotics simulators on real-world impacts. *IEEE Robotics and Automation Letters*, 7(3):6471–6478, 2022.

Kelsey R Allen, Tatiana Lopez Guevara, Yulia Rubanova, Kim Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1157–1167. PMLR, 14–18 Dec 2023.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, January 2020. ISSN 0278-3649. doi: 10.1177/0278364919887447.

Joachim Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

Onur Beker, Nico Gürtler, Ji Shi, A René Geist, Amirreza Razmjoo, Georg Martius, and Sylvain Calinon. A smooth analytical formulation of collision detection and rigid body dynamics with contact. *arXiv preprint arXiv:2503.11736*, 2025.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Bibit Bianchini, Mathew Halm, and Michael Posa. Simultaneous learning of contact and continuous dynamics. In *Conference on Robot Learning*, pp. 3966–3978. PMLR, 2023.

J. Frédéric Bonnans and Julien Laurent-Varin. Computation of order conditions for symplectic partitioned runge-kutta schemes with application to optimal control. *Numer. Math.*, 103(1):1–10, March 2006. ISSN 0029-599X. doi: 10.1007/s00211-005-0661-y. URL https://doi.org/10.1007/s00211-005-0661-y.

Karim Bouyarmane, Adrien Escande, Florent Lamiraux, and Abderrahmane Kheddar. Potential field guide for humanoid multicontacts acyclic motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pp. 1165–1170. IEEE, 2009.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Romain Brégier. Deep regression on manifolds: a 3d rotation case study. In *2021 International Conference on 3D Vision (3DV)*, pp. 166–174. IEEE, 2021.

Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. Myosuite: A contact-rich simulation suite for musculoskeletal motor control. In Roya Firoozi, Negar Mehr, Esen Yel, Rika Antonova, Jeannette Bohg, Mac Schwager, and Mykel J. Kochenderfer (eds.), *Learning for Dynamics and Control Conference, L4DC 2022, 23-24 June 2022, Stanford University, Stanford, CA, USA*, volume 168 of *Proceedings of Machine Learning Research*, pp. 492–507. PMLR, 2022.

M. Calvo, S. González-Pinto, and J.I. Montijano. Global error estimation based on the tolerance proportionality for some adaptive runge–kutta codes. *Journal of Computational and Applied Mathematics*, 218(2):329–341, 2008. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2007.02.034. URL https://www.sciencedirect.com/science/article/pii/S0377042707001240. The Proceedings of the Twelfth International Congress on Computational and Applied Mathematics.

Erin Catto. Box2d is a 2d physics engine for games. 2018.

Erin Catto et al. Modeling and solving constraints. In *Game Developers Conference*, 2009.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/google-deepmind.

J. R. Dormand and P. J. Prince. A family of embedded Runge–Kutta formulae. *J. Comp. Appl. Math*, 6:19–26, 1980.

Evan Drumwright and Dylan A Shell. Modeling contact friction and joint friction in dynamic robotic simulation using the principle of maximum dissipation. In *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*, pp. 249–266. Springer, 2011.

John C Duchi, Peter L Bartlett, and Martin J Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, 2012.

Ryan Elandt, Evan Drumwright, Michael Sherman, and Andy 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)*, pp. 8238–8245. IEEE, 2019.

Roy Featherstone. Rigid body dynamics algorithms. 2014.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

Carl Friedrich Gauß. Über ein neues allgemeines Grundgesetz der Mechanik. *Journal für die reine und angewandte Mathematik*, 4:232–235, 1829.

Andreas René Geist, Jonas Frey, Mikel Zhobro, Anna Levina, and Georg Martius. Learning with 3D rotations, a hitchhiker's guide to SO(3). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15331–15350. PMLR, 21–27 Jul 2024.

Ignat Georgiev, Krishnan Srinivasan, Jie Xu, Eric Heiden, and Animesh Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

Christian Gumbsch, Martin V Butz, and Georg Martius. Sparsely changing latent states for prediction and planning in partially observable domains. *Advances in Neural Information Processing Systems*, 34:17518–17531, 2021.

William Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87, 02 2001. doi: 10.1007/s002110000178.

E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II Stiff and Differential-Algebraic Problems*. Springer, Berlin, second revised edition edition, 2002.

E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I Nonstiff Problems*. Springer, Berlin, second revised edition edition, 2008.

Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. NeuralSim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

Michael Hinze, René Pinnau, Michael Ulbrich, and Stefan Ulbrich. Optimization with pde constraints. In *Mathematical Modelling*, 2008. URL https://api.semanticscholar.org/CorpusID:1278773.

Coşku Can Horuz, Geoffrey Kasenbacher, Saya Higuchi, Sebastian Kairat, Jendrik Stoltz, Moritz Pesl, Bernhard A Moser, Christoph Linse, Thomas Martinetz, and Sebastian Otte. The resurrection of the relu. *arXiv preprint arXiv:2505.22074*, 2025.

Taylor Howell. Mujoco warp (mjwarp). https://github.com/google-deepmind/mujoco_warp, March 2025. NVIDIA GPU Technology Conference (GTC).

Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo. dec 2022.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019a.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pp. 6265–6271. IEEE, 2019b.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2020.

Danny M Kaufman, Shinjiro Sueda, Doug L James, and Dinesh K Pai. Staggered projections for frictional contact in multibody systems. In *ACM SIGGRAPH Asia 2008 papers*, pp. 1–11. 2008.

Patrick Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Vittorio La Barbera, Steven Bohez, Leonard Hasenclever, Yuval Tassa, and John R Hutchinson. Motion tracking with muscles: Predictive control of a parametric musculoskeletal canine model. *arXiv preprint arXiv:2506.23768*, 2025.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, October 2020. doi: 10.1126/scirobotics.abc5986.

C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius. Learning agile skills via adversarial imitation of rough partial demonstrations. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, December 2022.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning, 2018.

Min Liu, Gang Yang, Siyuan Luo, and Lin Shao. Softmac: Differentiable soft body simulation with forecast-based contact model and two-way coupling with articulated rigid bodies and clothes. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12008–12015. IEEE, 2024.

Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

Igor Mordatch, Zoran Popović, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pp. 137–144, 2012a.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (ToG)*, 31(4):1–8, 2012b.

Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. A Review of Differentiable Simulators. *IEEE Access*, 12:97581–97604, 2024. ISSN 2169-3536.

Tao Pang, HJ Terry Suh, Lujie Yang, and Russ Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *IEEE Transactions on robotics*, 39(6): 4691–4711, 2023.

Mihir Parmar, Mathew Halm, and Michael 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. IEEE, 2021.

Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In *Conference on Robot Learning*, pp. 2279–2291. PMLR, 2021.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. In *Conference on Robot Learning (CoRL) 2020*, volume 155, pp. 1049–1065. PMLR, 2020. URL https://corlconf.github.io/paper_217.

Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*, 9(89):eadi9579, April 2024. doi: 10.1126/scirobotics.adi9579.

Russ Tedrake, and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

Subham Sekhar Sahoo, Anselm Paulus, Marin Vlastelica, Vít Musil, Volodymyr Kuleshov, and Georg Martius. Backpropagation through combinatorial algorithms: Identity with projection works. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=JZMR727O29.

Adrian Sandu. On the properties of runge-kutta discrete adjoints. In *Proceedings of the 6th International Conference on Computational Science - Volume Part IV*, ICCS'06, pp. 550–557, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3540343857. doi: 10.1007/11758549_76. URL https://doi.org/10.1007/11758549_76.

Pierre Schumacher, Daniel F.B. Haeufle, Dieter Büchler, Syn Schmitt, and Georg Martius. Dep-rl: Embodied exploration for reinforcement learning in overactuated and musculoskeletal systems. In *The Eleventh International Conference on Learning Representations (ICLR)*, May 2023.

Clemens Schwarke, Victor Klemm, Jesus Tordesillas, Jean-Pierre Sleiman, and Marco Hutter. Learning quadrupedal locomotion via differentiable simulation. *arXiv preprint arXiv:2404.02887*, 2024.

Gustaf Söderlind. Automatic control and adaptive time-stepping. *Numerical Algorithms*, 31:281–310, 2002.

Gustaf Söderlind. Digital Filters in Adaptive Time-Stepping. *ACM Transactions on Mathematical Software*, 20(1):1–26, 2003.

Yunlong Song, Sang bae Kim, and Davide Scaramuzza. Learning quadruped locomotion using differentiable simulation. In *8th Annual Conference on Robot Learning*, 2024.

Hyung Ju Terry Suh, Tao Pang, and Russ Tedrake. Bundled gradients through contact via randomized smoothing. *IEEE Robotics and Automation Letters*, 7(2):4000–4007, 2022.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.010.

Yuval Tassa and Emo Todorov. Stochastic complementarity for local control of discontinuous dynamics. In *Robotics: Science and Systems*, volume 6, 2010.

A. Howell Taylor, Simon Le Cleac'h, Zico Kolter, Mac Schwager, and Zachary Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 2022.

Emanuel Todorov. A convex, smooth and invertible contact model for trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, pp. 1071–1076, May 2011. ISSN: 1050-4729.

Emanuel 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)*, pp. 6054–6061. IEEE, 2014.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, October 2012. ISSN: 2153-0866.

Ch Tsitouras. Runge–kutta pairs of order 5 (4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, 2011.

Andrea Walther. Automatic differentiation of explicit runge-kutta methods for optimal control. *Computational Optimization and Applications*, 36:83–108, 11 2007. doi: 10.1007/s10589-006-0397-3.

Huawei Wang, Vittorio Caggiano, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. Myosim: Fast and physiologically realistic mujoco models for musculoskeletal and exoskeletal studies. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pp. 8104–8111. IEEE, 2022.

Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints. In *Robotics: Science and Systems*, 2021.

Jie Xu, Viktor Makoviychuk, Yashraj S. Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Jijie Xu, Tak-Kuen John Koo, and Zexiang Li. Sampling-based finger gaits planning for multifingered robotic hand. *Autonomous Robots*, 28(4):385–402, 2010.

Gang Yang, Siyuan Luo, Yunhai Feng, Zhixin Sun, Chenrui Tie, and Lin Shao. Jade: A differentiable physics engine for articulated rigid bodies with intersection-free frictional contact. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16915–16922. IEEE, 2024.

Yunbo Zhang, Deepak Gopinath, Yuting Ye, Jessica Hodgins, Greg Turk, and Jungdam Won. Simulation and retargeting of complex multi-character interactions. In *ACM SIGGRAPH 2023 Conference Proceedings*, pp. 1–11, 2023.

A Related Work

Differentiable simulators are an active area of research spanning multiple fields of physics, including elastic object and fluids modeling Hu et al. (2019b); Macklin (2022); Hu et al. (2020); Liu et al. (2024) and modeling rigid body collisions Todorov et al. (2012); Hu et al. (2020); Taylor et al. (2022); Howell (2025) (see Newbury et al. (2024) for a recent overview). As outlined in Fig. 15, we focus on robotics simulators involving hard contact collisions, with the hierarchical objectives of **(i)** accurately simulating dynamics, **(ii)** computing correct simulator gradients, and **(iii)** obtaining informative gradients between non-colliding objects.

Simulating contact dynamics. Commonly deployed robot simulators can be categorized according to their contact model into impulse-based Catto (2018); Freeman et al. (2021); Hu et al. (2020), complementary-based Taylor et al. (2022); Werling et al. (2021), and penalty-based Todorov et al. (2012) approaches. Due to their simplicity and speed, impulse-based simulators (Catto et al., 2009), such as those using DiffTaichi (Hu et al., 2019a; 2020), are widely used in game development but remain currently uncommon in robot controller synthesis. Complementary-based contact models compute constraint forces as a solution to a constrained optimization problem either in the form of a nonlinear-complementary problem (NCP) (Taylor et al., 2022) or linear-complementary problem (LCP) (Russ Tedrake, 2019; Coumans & Bai, 2016; Heiden et al., 2021; Yang et al., 2024). Exactly solving NCP is an NP-hard problem Kaufman et al. (2008) that is considerably difficult to solve even approximately. The NCP formulation in Dojo is physically more accurate than LCP formulations Taylor et al. (2022), but lacks support for parallel computation limiting its utility for controller synthesis. That said, it is currently not clear which complementary formulation is best suited for robotics.

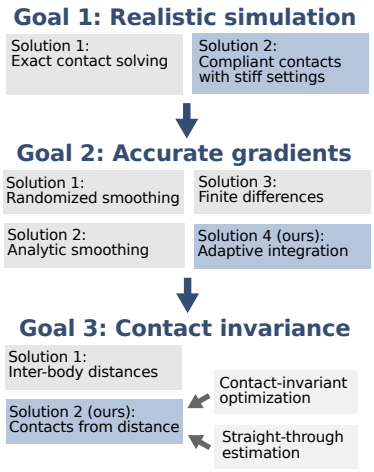


Figure 15: Overview on hierarchical goals that need to be accomplished to use simulator gradients for robot controller synthesis.

To improve computational efficiency, MuJoCo (Todorov et al., 2012; Todorov, 2014) reformulates the complementary constraint problem as a convex optimization problem. This reformulation builds on prior work on complementarity-free approaches (Todorov, 2011; Drumwright & Shell, 2011). Briefly, contact forces are computed by solving a global optimization problem, in which the ability of a constraint to generate force is modulated by the geometric penetration depth at the contact interface. As an alternative, Drake Russ Tedrake (2019) also adopts a soft patch contact model Elandt et al. (2019); Pang et al. (2023). In turn, these approaches to contact resolution soften the dynamics.

Contact-invariant optimization. To guide an optimizer toward using body collisions for task facilitation, recent works in controller synthesis and path planning incorporate inter-body distances into the optimization objective. In RL, Zhang et al. (2023) added inter-point distances to rewards using a softmax function. Alternatively, the algorithms in MuJoCo MPC (Howell et al., 2022) rely on inter-point distances combined with trajectory samples establishing contact to inform the optimization. However, depending on the task at hand, setting up distance-based loss terms to inform an optimizer about the necessity for specific object collisions can quickly become cumbersome. As an alternative approach, Mordatch et al. (2012a) proposed the framework of *"contact-invariant optimization"* (CIO) in which a simulation is adjusted to also apply contact forces between non-colliding bodies. While the simulation can apply non-physical contact forces, the optimization is encouraged through the addition of several loss terms to minimize the usage of these forces. In Mordatch et al. (2012b), CIO was extended to in-hand manipulation of objects. Our proposed extension of *contacts from distance* (CFD) is inspired by CIO. Yet, while CIO applies artificial contact forces in the forward simulation, CFD resorts to the straight-through-trick to only exert contact forces in a simulation that is solely used for gradient computation. In turn, CFD does not require the addition of regularization terms to a loss when used for planning. Recently, Beker et al. (2025) proposed soft signed distance fields as an alternative geometry representation, while also adjusting collision detection and contact force computation to be inherently soft. This approach which routes at the geometry level of robot simulation can be seen as an alternative avenue towards CIO.

CFD's mechanism to create "forces from a distance" (FFD) is also conceptually similar to Pang et al. (2023). Drawing inspiration from works on creating FFD, this work smoothens the contact model of a quasi-static penalty-based simulator using log-barrier functions and analyzes FFD in the context of quasi-static dynamics and implicit differentiation. In comparison, CFD uses MuJoCo's smooth contact model based on polynomial splines to obtain forces from a distance. We note that both quasi-static dynamics and implicit differentation would be useful extensions to DiffMJX and CFD.

Straight-through estimation. The use of the straight-through-trick in CFD is inspired by a plethora of works in robotics and beyond (Bengio et al., 2013; Sahoo et al., 2023). The code underlying the straight-through-trick is reported in JAX (DeepMind et al., 2020) documentation under the collective heading of *straight-through estimation*. Straight-through estimation has been used in Gumbsch et al. (2021), where the backward pass of a Heaviside function is replaced with an identity function to obtain sparse RNNs for model-based RL. Horuz et al. (2025) demonstrate that using a ReLu activation with a custom backward function rivals smooth ReLu surrogates. Recently, Song et al. (2024) replaced simulator dynamics obtained from IsaacGym (Liang et al., 2018) with single-rigid body dynamics for the gradient computation in the backward pass. This approach is closely related to our work, but unlike CFD, Liang et al. (2018) does not utilize the forward pass simulation in the backward pass, limiting its utility for complex control synthesis tasks and system identification.

```python
def step_ideal(q, v, h):  # Ideal elastic collision simulation
    a = 0.0  # Contact-free acceleration
    q, v = jax.lax.cond(q < 0.0, lambda: (q, -v), lambda: (q, v))
    v = v + a * h  # Semi-implicit Euler
    q = q + v * h
    return q, v

def step_penalty(q, v, h):  # Penalty based collision simulation
    a = 0.0  # Contact-free acceleration
    a_ref = jax.lax.cond(q < 0.0, lambda: -1.0*v -50.0*q, lambda: 0.0)
    v = v + (a + a_ref) * h  # Semi-implicit Euler
    q = q + v * h
    return q, v

@jax.value_and_grad
def unroll(q0, v0, n_steps, step_fn, h):
    def body(carry, _):
        q, v = carry
        q, v = step_fn(q, v, h)
        return (q, v), (q, v)

    (qf, vf), _ = jax.lax.scan(body, (q0, v0), length=n_steps)
    q_target = 1.0
    loss = jnp.abs(qf - q_target)
    return loss
```

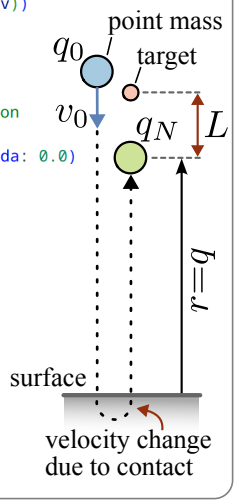


Figure 16: **Jax code for minimal collision simulation.** The code simulates a minimal version of a penalty-based or ideal-elastic simulator and is used in Fig. 4 to illustrate the TOI discretization errors as shown in Fig. 4.

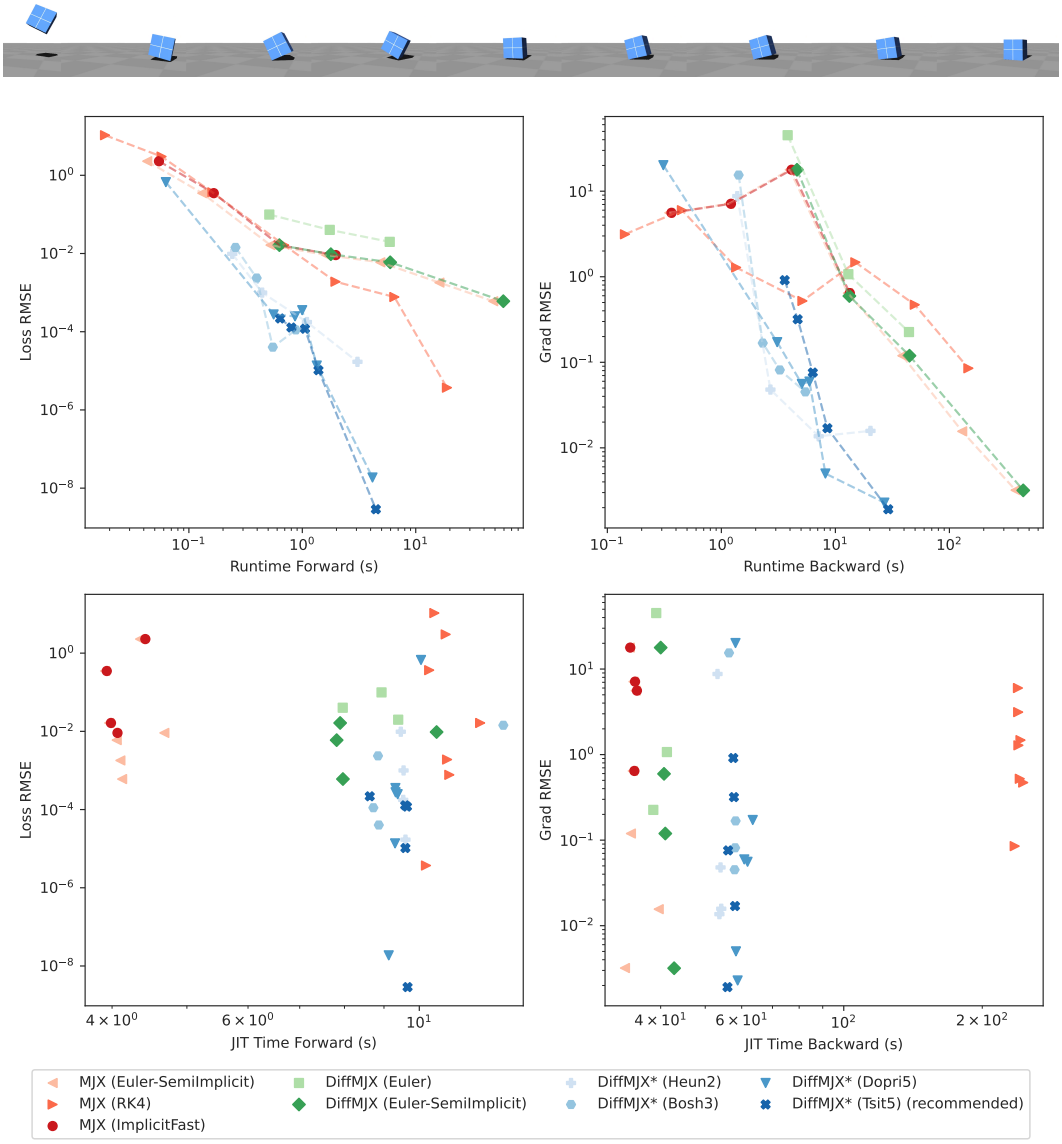


Figure 17: **Error of loss and gradient vs runtime and compilation time for different integrators.** The loss and gradient are computed for the cube toss at an initial velocity of $v_x = -2.0$ with contact settings `solref=[0.005 1.0]` and `solimp=[0.0 0.95 0.001 0.5 2]` as depicted in Fig. 21 and the top of this figure. The ground-truth gradients are computed from finite-differences using an adaptive solver with very low tolerance (10^{-12}) to simulate the rollout. Standard fixed-stepsize MJX integrators have red colors, fixed-stepsize integrators in Diffrax have green color, and adaptive-stepsize integrators in Diffrax have blue color (also marked by $*$). For adaptive stepsize control we use the Diffrax PID controller with $P = 0.2, I = 0.4, D = 0.0$, as recommended for stiff ODEs. We observe that the pareto-front of adaptive solvers is shifted, allowing lower loss and gradient errors with less runtime. The best-performing solver is the Tsit5 solver (Tsitouras, 2011), which is a 5th order explicit Runge–Kutta method with an embedded 4th order method for adaptive step sizing. This is also the default solver used in other experiments.

B TUNING DIFFMJX

Through Diffrax, DiffMJX provides a whole suite of additional tuning knobs that can be used to improve gradient computation as shown in Fig. 18.

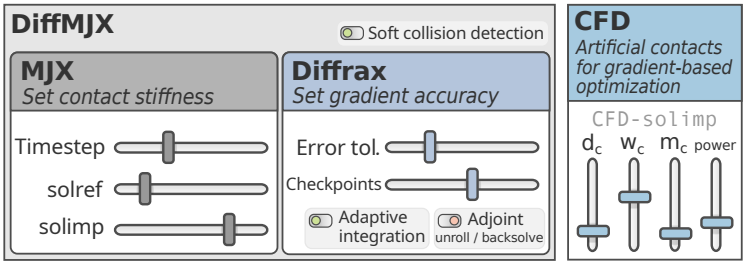


Figure 18: **DiffMJX and CFD add tuning knobs to MJX's gradient computation.** DiffMJX adds support for adaptive integration atop MJX. The integrator's error tolerances trades computational speed for improved gradient accuracy. The number of checkpointing steps trades computation speed with GPU memory consumption. CFD adds parameters that set the distance and magnitude at which artificial contact forces between non-colliding objects are applied in the gradient computation.

Error tolerances set gradient accuracy. In this work, we showed that the errors of penalty-based simulator gradients depends on

1. the **simulation stiffness**, aka the hardness of contacts and joint limits,
2. the **contact velocities** in normal direction, aka how fast objects collide into each other,
3. and the **integration stepsize**.

While simulation stiffness and contact velocities are system dependent, the integration stepsize is a tuning parameter. To obtain accurate gradients, one can use a constant stepsize integrator with a sufficiently small stepsize which inevitably results in slow integration.

To speed up computation, an adaptive integrator solely reduces the stepsize if necessary as a function of the integrator's error tolerances. In turn, as the integrator's error tolerances directly determines the gradient errors, it forms the most important hyperparameter.

> *To speed up computation, use the largest error tolerance that still produces sufficiently accurate gradients.*

Figure 17 shows the error vs runtime trade-off for different integrators in MJX and DiffMJX. We observe that in the case of the cube toss, **the adaptive integrators reduce the runtime required to achieve gradients with low error**.

Number of checkpoint trades GPU memory for compute time. Thanks to Diffrax's advanced techniques for checkpointing, DiffMJX memory consumption is significantly reduced compared to MJX. As noted in Diffrax's documentation, the memory used approximately equals the number of checkpoints multiplied by the size of the ODE's state.

> *Increasing the number of checkpoints speeds up computation. Therefore, use as many checkpoints as possible given the available GPU memory.*

By reducing the number of checkpoints, we were able to run DiffMJX simulations that MJX could not, because MJX exceeded the available GPU memory.

Maximum number of ODE steps affects JIT compilation time. As shown in Fig. 17, there is a notable increase in JIT compilation time when using Diffrax compared to MuJoCo's semi-implicit Euler integrator. This is due to the program being compiled for the maximum number of ODE steps that is set by the user. Therefore, this parameter should be chosen as small as possible.

C METHOD DETAILS

C.1 REMOVING DISCONTINUITIES FROM COLLISION DETECTION AND MUSCLE ACTUATORS

The collision checking in MJX is implemented for pairs of several different geometries. Many of the collision check implementations rely on several case distinctions, e.g. the collision of a cylinder and a plane is separated into the case of the cylinder being parallel to the plane vs not. These hard case distinctions can lead to errors in the gradients, as they introduce discontinuities in the dynamics and its gradient. We adopt the typical approach for smoothing such non-differentiabilities by replacing any hard case distinctions by smoothly interpolating between the cases with weights computed from a sigmoid. Specifically, we end up modifying the collision detection between plane-cylinder, sphere-capsule, capsule-capsule, plane-capsule and plane-box pairs. More complicated collisions like mesh-mesh collisions are currently not modified as they are not relevant in our experiments, but we plan on supporting these in a future version.

As an example, consider our soft version of a function for plane-box collision detection. More details on other smoothed functions will be available in the released codebase.

```
def _plane_box(plane: GeomInfo, convex: ConvexInfo) -> Collision:
  """Calculates contacts between a plane and a convex object."""
  vert = convex.vert

  # get points in the convex frame
  plane_pos = convex.mat.T @ (plane.pos - convex.pos)
  n = convex.mat.T @ plane.mat[:, 2]
  support = (plane_pos - vert) @ n
  # search for manifold points within a epsilon skin depth
  threshold = jp.maximum(0, support.max() - 1e-3)
  soft_poly_mask = math.sigmoid(support, low=threshold - 1e-3, high=threshold)
  support_masked = math.soft_where(soft_poly_mask, support, 0)

  dist_neg, idx = jax.lax.top_k(support_masked, 4)
  dist = -dist_neg
  pos = vert[idx]

  # convert to world frame
  pos = convex.pos + pos @ convex.mat.T
  n = plane.mat[:, 2]

  frame = jp.stack([math.make_frame(n)] * 4, axis=0)
  pos = pos - 0.5 * dist[:, None] * n
  return dist, pos, frame

def soft_where(condition, x, y):
  return condition * x + (1 - condition) * y
```

We also encountered differentiation errors in MJX environments with muscle–tendon units. The source of these issues is that the functions for computing wrapping of muscles around objects rely on nonlinearities such as arcsin, which has an undefined gradient at some points. A practical remedy is the "double-where" trick, which prevents gradient flow at those singularities.

Concurrently with our work, La Barbera et al. (2025) employed MJPC Howell et al. (2022) – a library for CPU-based MPC in MuJoCo via finite differences – to compute ideal controls of a musculoskeletal dog model. This seminal work also reports that modifying the muscle dynamics to be differentiable was a key contribution for improving MPC performance.

C.2 ODE DIFFERENTIATION IN DIFFRAX

Diffrax: Numerical integration with JAX The ODE solved by the Mujoco simulator can be written as

$$x(0) = x_0 \qquad \dot{x}(t) = F_\theta(t, x(t)), \tag{5}$$

where θ incorporates all model parameters and x is the full state of the system.

In this work, we resort to the Diffrax library (Kidger, 2021; Kidger & Garcia, 2021) for numerical integration in Jax to solve the above ODE. This powerful library provides efficient implementations of many fixed and adaptive timestep solvers (Tsitouras, 2011; Dormand & Prince, 1980). It also allows easily switching between different modes for backpropagation: Discretize-then-optimize (also referred to as unrolling) and optimize-then-discretize (also referred to as backsolving) (Chen et al., 2018; Kidger, 2021). Overall, the choice of solver, stepsize controller and differentiation

technique is typically application-dependent. Hence, we implement the general Diffrax integrator as an easy-to-use alternative for the existing fixed-stepsize integrators in MJX, offering the full flexibility of the Diffrax library to the user. Notably, *we adjust the Diffrax solvers to accommodate for exact integration of quaternions and stateful actuators*, similar to the Runge-Kutta implementation of MJX, which reduces the number of required integration steps.

Discretize-then-optimize. The first notable differentiation mode is discretize-then-optimize. It is the result of discretizing the forward ODE with a numerical integrator and then computing the gradients of the discretized forward ODE by unrolling the computation graph. This approach aligns with the paradigm of differentiable programming used in autodifferentiation libraries such as JAX, hence this is also the approach taken in standard MJX. The problem here is that the memory requirements scale linearly with the number of solver steps. For many applications this is a serious bottleneck in standard MJX, and the problem is exacerbated when relying on adaptive integration, as it typically involves storing even more substeps. Fortunately, Diffrax implements an optimal gradient checkpointing scheme that allows reducing memory requirements from $O(n)$ to $O(1)$ in exchange for increasing runtime from $O(n)$ to $O(n \log n)$, where n is the maximum number of solver steps.

Another potential issue is that discretize-then-optimize inherently computes gradients of the discretized dynamics, rather than the true continuous dynamics. This can lead to interesting failure cases in which even in the limit of the stepsize going to zero, the computed gradient is different from the true gradient of the continuous dynamics. This is the result of the basic fact that the convergence of a parameterized function (the discretized ODE) to a limit (the continuous ODE) does not imply the convergence of its gradient. Note that this is not a mere mathematical corner case, instead it is the underlying reason for time-of-impact oscillations as described in Fig. 4 and Hu et al. (2020); Schwarke et al. (2024).

Optimize-then-discretize. Fixing this requires (approximately) computing the gradients of the true continuous forward ODE. This is achieved by analytically computing the gradient as the solution to the continuous adjoint equations. Assuming a loss $L = L(x(T))$ only on the final state w.l.o.g., the continuous adjoint equations (Kidger, 2021, Theorem 5.2) can be written as:

$$a_x(T) = \frac{dL}{dx(T)} \qquad \dot{a}_x(t) = -a_x(t)^\top \frac{\partial F_\theta}{\partial x}(t, x(t)) \tag{6}$$

$$a_\theta(T) = 0 \qquad \dot{a}_\theta(t) = -a_x(t)^\top \frac{\partial F_\theta}{d\theta}(t, x(t)). \tag{7}$$

Solving these using any numerical solver on the backward pass, as offered readily by the Diffrax library, yields the desired gradients $\frac{dL}{dx(t)} = a_x(t)$ and $\frac{dL}{d\theta} = a_\theta(0)$.

This approach is called "optimize-then-discretize" or BacksolveAdjoint in Diffrax. Note also that the memory requirements can again be reduced to constant in the number of solver steps, by solving the forward ODE "backwards in time" together with the adjoint ODE. Optimize-then-discretize, in contrast to unrolling, allows to directly specify tolerances on the error in the gradients.

One of the intuitive potential benefits of optimize-then-discretize is that the adaptive stepsize controller for solving the adjoint ODE now has the ability to adapt the stepsize when computing the adjoint derivatives, rather than being constrained to unrolling the steps that were taken during the forward solve. However, in practice it is typically the case that the forward ODE and the adjoint ODE are stiff "in the same regions", i.e. the state derivative changes quickly when the state itself changes quickly. Hence, when unrolling the forward solve, the steps are already small in the regions where the adjoint ODE is also stiff. Therefore it typically suffices to use the less complicated discretize-then-optimize, which has a less complex computational graph and therefore comes with lower compilation times. The above explanation of course only holds when the forward ODE and the adjoint ODE "match"; when modifying the adjoint ODE (as will be discussed in the following section) it may be beneficial to use optimize-then-discretize.

C.3 Theoretical results on adaptive ODE integration

There exists a rich history of results on analyzing discrete adjoint sensitivities, supporting the claims on gradient accuracy improvements from non-adaptive solvers. For instance, Sandu (2006); Walther

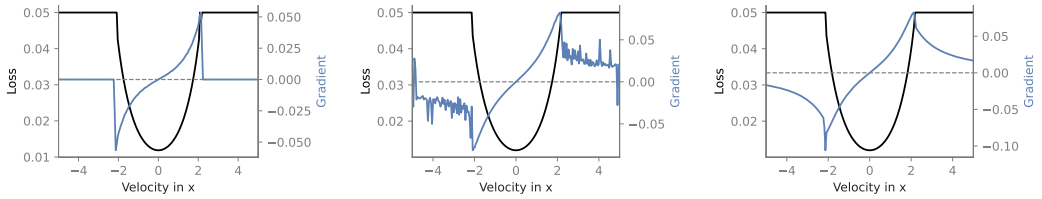


Figure 19: **Optimize-then-discretize vs discretize-then-optimize in the presence of CFD.** Results are for a reduced billiard example as in Fig. 8 without table-ball contacts and gravity. **Left:** DiffMJX adaptive integration results in well-behaved gradients, but gradient is zero if objects do not collide. **Middle:** Using CFD with *discretize-then-optimize* results in small gradient oscillations. The adaptive integrator selects large stepsizes as no contacts are happening (forward ODE is non-stiff), whereas in the gradient computation the CFD are added (adjoint ODE is stiff). In turn, unrolling the large stepsizes cause discretization errors. Note that this phenomenon persists no matter how low the tolerance of the forward ODE solver is set, as the stepsize controller is unaware of the CFD mechanism that is only used in the gradient computation. **Right:** Using *optimize-then-discretize* allows the solver for the adjoint ODE to select small stepsizes when the adjoint ODE is stiff due to CFD. This fixes the gradient oscillations.

(2007) establish that for one-step Runge–Kutta methods of order-p, the reverse-mode sensitivities obtained by differentiating the discrete solve are themselves order-p consistent and stable. This directly links reductions in forward global error to reductions in gradient error at the same order. Calvo et al. (2008) show that (under standard Lipschitz/stability conditions) the global state error scales like the (relative/absolute) tolerance enforced by the controller; the adjoint/sensitivity error inherits this scaling with constants depending on the assumed Lipschitz constant. As our setting is an application of adaptive integration to the ODE given by the simulator, these results apply out of the box and establish a theoretical basis for the observed improvements in gradient accuracy from the use of adaptive solvers. We hope that these changes will strengthen the presentation.

The advantages and disadvantages of discretize-then-optimize (DTO) vs optimize-then-discretize (OTD) have also been researched for many years. This includes Hinze et al. (2008) comparing DTO and OTD, giving conditions for equivalence, and providing error bounds in the discretized optimality system. Hager (2001) proves accuracy of discrete optimality systems under Runge-Kutta time discretization and establishes conditions ensuring correct gradients. Moreover, Bonnans & Laurent-Varin (2006) provide order conditions for (partitioned) Runge–Kutta schemes in optimal control and implications for obtaining matching accuracy in the adjoint/gradient after discretization.

C.4 CONTACTS FROM DISTANCE WITH THE STRAIGHT-THROUGH-TRICK

In discretize-then-optimize, automatic differentiation automaticaly uses the vector-jacobian-products derived from $\tilde{F}_\theta$ instead of F_θ. In optimize-then-discretize, with the straight-through-trick applied to F, we replace the vector-jacobian-products in the adjoint equations Eq. (6) as follows

$$\tilde{a}_x(T) = \frac{dL}{dx(T)} \qquad \dot{\tilde{a}}_x(t) = -\tilde{a}_x(t)^\top \frac{\partial \tilde{F}_\theta}{\partial x}(t, x(t)) \tag{8}$$

$$\tilde{a}_\theta(T) = 0 \qquad \dot{\tilde{a}}_\theta(t) = -\tilde{a}_x(t)^\top \frac{\partial \tilde{F}_\theta}{d\theta}(t, x(t)). \tag{9}$$

Solving these altered adjoint equations gives replacements for the respective gradients, which in our case incorporate gradient information on contacts from distance. Again, this approach does not require any additional implementation effort, as the straight-through-trick allows to just use the existing Diffrax implementation of optimize-then-discretize.

Optimize-then-discretize vs Discretize-then-optimize in the presence of CFD. As discussed in the previous section, adding contacts from distance may change the preferred choice of gradient computation, as the forward and adjoint ODE may have different stiffness values in different regions.

To illustrate this, we again consider a version of the billiard-toy example from Fig. 8. This time, we turn gravity off and disable the table-ball contacts to isolate the ball-ball contact, which means that the ODE is linear before and after the collision. We again compute the loss and gradient over a range of initial parameters, the results are reported in Fig. 19. As before, we observe that with the DiffMJX adaptive integrator, we get well-behaved gradients, but the gradient becomes zero when the two balls do not collide.

Now, we activate the CFD mechanism and repeat the experiment. In this case, the gradient show oscillations due to the different stiffness settings of the forward and adjoint ODE: The solver for the forward ODE takes very large steps when no ball-ball collision happens, but the adjoint ODE incorporates the collision signal and hence is stiff. Simply unrolling the forward integration therefore makes a discretization error, resulting in the oscillations. Note, that we did not observe this phenomenon in Fig. 8, as this experiment includes ball-table collisions which cause the adaptive solver of the forward ODE to take small steps even when no ball-ball collision is happening. The solution is to use optimize-then-discretize. Here, the adaptive solver for the adjoint ODE can select stepsizes according to the stiffness of the adjoint ODE. This is also confirmed by the results in Fig. 19 (right).

This experiments highlights a specific corner case, and we did not observe issues in our other experiments when using discretize-then-optimize with CFD, as we typically do not have such extreme variations of the stiffness in the forward ODE. However, we believe it is beneficial to be aware of this potential caveat and how one can resolve it with the tools available in DiffMJX.

D DIFFMJX & CFD: THE SHARP BITS

CFD adds additional contacts to the solver which can slow down computation. The documentation of MuJoCo XLA[1] advises to use meshes with 200 vertices or fewer due to computational limitations on the current implementation of collision detection. If MJX is used with CFD, then every contact is added to the contact solver for which $r < w_c$. In turn, for dense meshes and large w_c, computation time notably increases. The parameter w_c can be made arbitrarily small such that, in its limit for $w_c = 0$, we retain vanilla MJX with our improvements on the differentiability of the collision detector. That said, for in-hand manipulation tasks and robot locomotion tasks, the number of objects colliding at one instance is small enough to render CFD a useful tool for easing optimization with penalty-based simulators.

Adaptive integration slows down training if tolerances are set too small. For the very large initial cube side length initial value of 140 mm, DiffMJX saw a significant drop in computation speed forcing us to abort these runs. This is not surprising as penetrations amounting to 40% of the

[1] https://mujoco.readthedocs.io/en/stable/mjx.html#mjx-the-sharp-bits

total cube's width cause huge contact forces that significantly stiffen the ODE. In Appendix E.2, we reduce the integration error tolerance which allowed optimization to converge even for large initial parameters. Alternatively, one could speed up training for these too large geometry parameter initializations by first using soft impedance settings that during training becomes annealed to become increasingly stiff. The same principle applies to *scenes with multiple objects*: overly tight integration error tolerances can substantially slow down the simulation. As integration error scales with relative contact velocity, scenarios involving large state changes along contact normals trigger smaller step sizes. Notably, while the ball in the billiard shot example (Fig. 8) was rolling over the table, the integration stepsize did only marginally decrease. Whereas during the collision between balls, the integrator adjusted the stepsize notably.

Finite differences can be a viable alternative for computing low-dimensional gradients. In general, gradient computation with MJX via automatic differentiation is notably slower than running only the forward simulation. This makes zeroth-order methods such as predictive sampling favorable whenever the task can be solved with them. This is usually the case for low-dimensional systems or computing gradients of only a few parameters, e.g. compting gradients via finite differences for less then ten parameters seems to work well in practice. As sampling suffers from the curse of dimensionality, automatic differentiation for computing gradients scales to extremely high-dimensional control problems which enabled us to do gradient-based MPC on the musculoskeletal systems.

CFD should be combined with sampling to solve nonconvex optimizations. For MPC, our CFD mechanism tends to be effective when the object needs to be pushed in a specific direction by the hand, but it performs poorly in grasping tasks. This limitation arises because, at a distance, the cumulative force arising from the hand's CFDs conglomerate into a force that pushes objects away. However, the gradient does not encode the possibility of grasping, as this signal only emerges when the object is inside the hand. Fundamentally, this issue reflects the non-convex nature of the optimization landscape. However, we do not believe that this is a problem that should be solved using a CFD mechanism; rather, in such tasks, the best approach would be to combine CFD with sampling to overcome the non-convexity while maintaining the favorable scaling of gradients with dimensionality.

Parallelizing DiffMJX simulations. Locomotion policies trained with reinforcement learning typically rely on domain randomization with many environments executed in parallel. DiffMJX supports such batched execution via JAX's vmap, analogous to MJX. However, if a single policy induces large contact forces, then this may trigger very small integration steps, which results in the other simulations to wait for a slow simulation to finish. This slowdown is mitigated by increasing the integration error tolerances or by early-terminating the respective simulation once a predefined solver-step budget is exceeded. However, it is important to note that such a slowdown is only observed when many contacts are happening in a single environment. On the other hand, when few contacts are happening in each parallelized environment, there is no additional slowdown due to vmapping. This is because the selected adaptive stepsize in Diffrax is not global across the vmapped environments, it is fully vmapped and can take different values across environments. WE experimentally verify this in Fig. 20, see the caption for details.

E Evaluation details and further experiments

E.1 Gradient analysis of MJX

In Fig. 4, the time discretization errors arising in penalty-based simulators are first illustrated on a minimal toy example. The code to generate this toy example is shown in Fig. 16. After illustrating on this toy example that time discretization errors can be mitigated by reducing the stepsize, it is shown in Fig. 3 that these errors also occur in MuJoCo. Fig. 21 is an extended version of Fig. 3 containing all of MuJoCo's basic shape primitives.

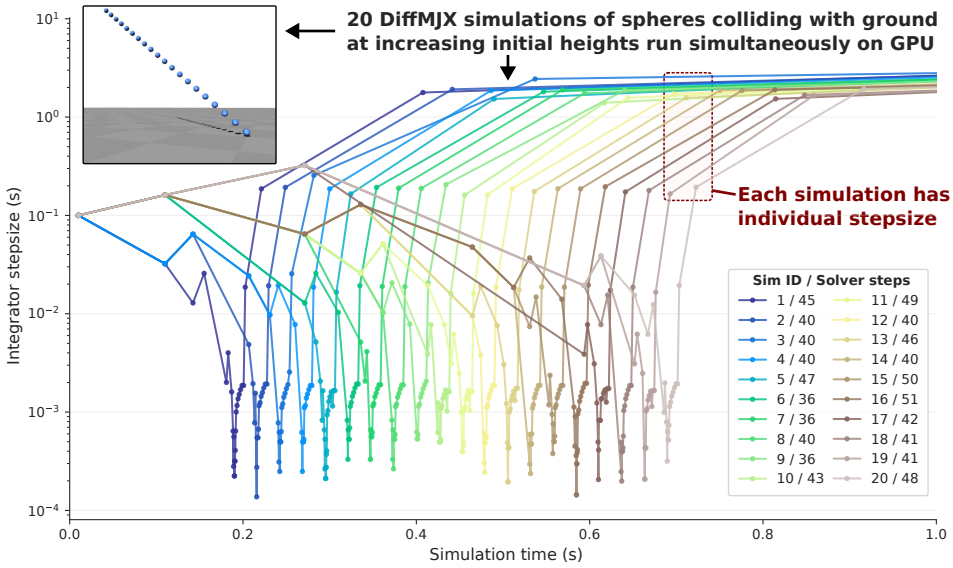


Figure 20: We vmap the DiffMJX simulation of a ball drop across 20 different environments. In each environment, a single ball is dropped (with an initial downwards velocity and no gravity) onto a plane, from which it bounces back up. The 20 environments start with the ball at different heights, such that the contact happens at a different simulation time for each environment. We then plot the selected adaptive stepsize of the solver over the simulation time for each environment, and observe that the stepsize is indeed independent for each run, with a reduction in stepsize only occurring at the single contact in a specific environment. Therefore, when vmapping across environments, contacts in one simulation do not slow down other simulations.

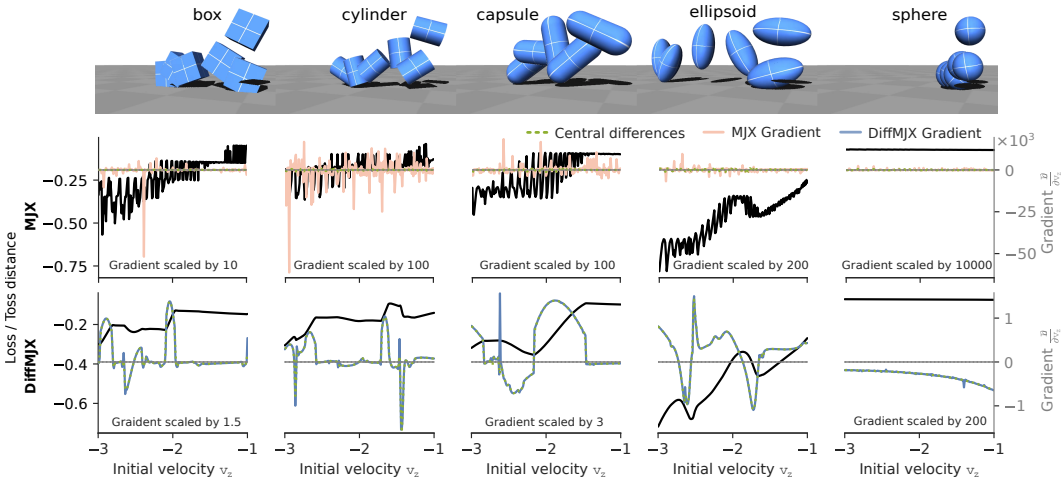


Figure 21: Simulation of geometric primitives thrown onto a surface using MJX or DiffMJX. Contacts at default stepsize 0.002 s, `solref=[0.005 1.0]`, and `solimp=[0.0 0.95 0.001 0.5 2]` cause MJX's gradients of the toss distance to deviate significantly from central difference gradients, while DiffMJX maintains close agreement.

E.2 System identification experiments

Contactnet dataset The contactnet dataset[2] consists of 550 trajectories of an acrylic cube that has been repeatedly tossed onto a wooden table. As reported in Pfrommer et al. (2021); Acosta et al.

[2]https://github.com/DAIRLab/contact-nets

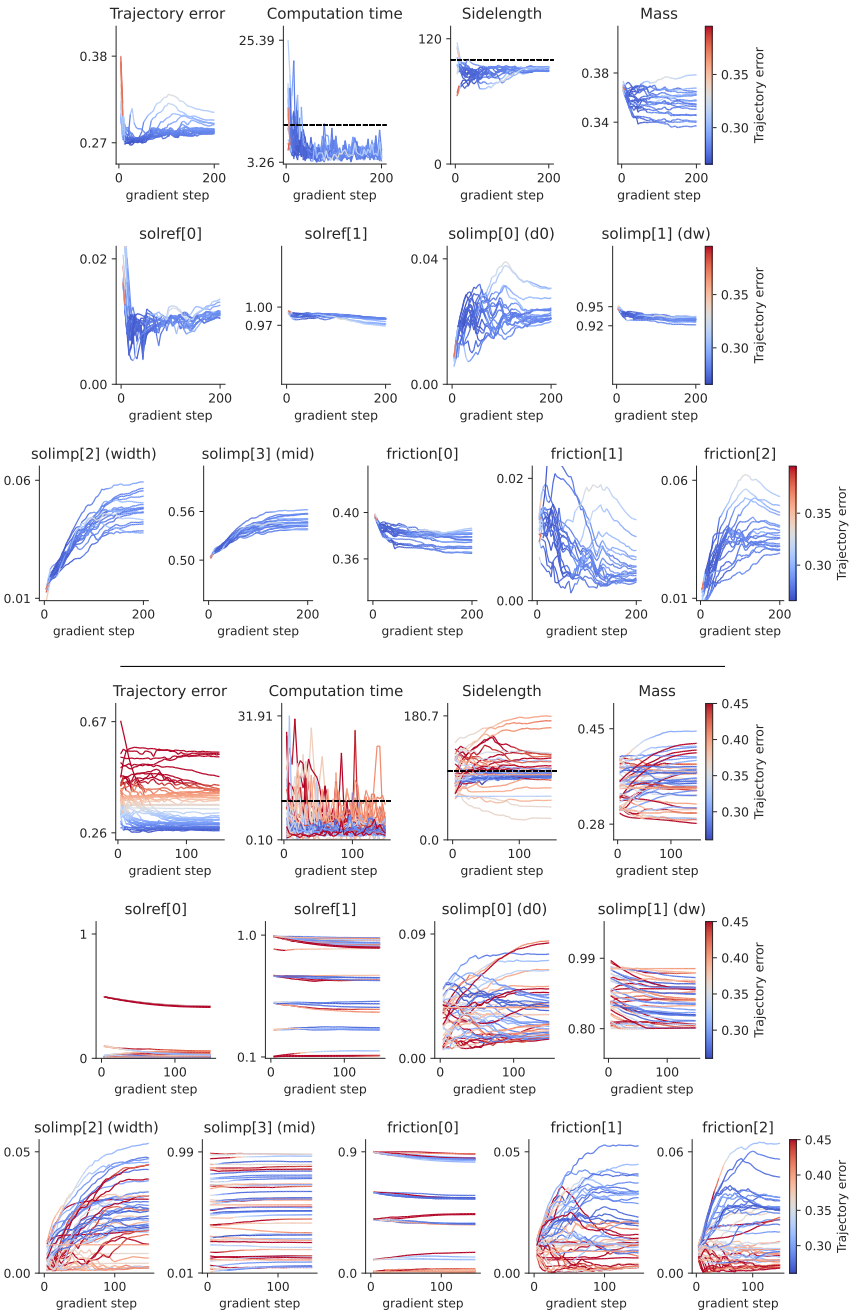


Figure 22: **Top:** Identification of MuJoCo parameters with DiffMJX on Contactnets cube toss dataset starting from pre-tuned inital conditions. **Bottom:** Identification starting from random initial parameters. The cube's geometry has a significant effect on the time of impact such that the optimization converges to a sidelength close to 100 mm to reduce the train loss. The effect of other model parameters on the system dynamics is more inter-twined. For example, a too large mass resulting in larger penetrations during impact can be compensated by increasing the constraint stiffness via `solref[0]` or `solimp[0]`.

(2022), data has been collected at 1480 Hz and the cube's physical parameters amount to a side length of 10 cm, mass of 0.37 kg, inertia of 0.0081 kg m^2, a friction coefficient of 0.18, and restitution of 0.125.

Training setup The trajectories are split into segments of length five. In the supervised train loop, given a trajectory segment's initial state, each simulator is unrolled for four steps. Subsequently, an mean square error (MSE) loss between the segment's states and predicted states is computed. Before being fed to the loss function, each state's quaternion is converted to a rotation matrix to avoid representation singularities negatively affecting training Geist et al. (2024); Brégier (2021). For adaptive integration, DiffMJX is set to use "RecursiveCheckpointing" and a 5th order explicit Runge–Kutta method ("Tsit5" in Diffrax) with PID error tolerances of 1e-5. We use the Adam Kingma & Ba (2015) implementation from the Optax library DeepMind et al. (2020) for gradient-based optimization with parameters `[b1 = 0.5, b2 = 0.9, eps = 1e-6]`. MuJoCo's parameter are set to a timestep of 0.006767 s (the data collection sampling time), `iterations=4` (contact solver iterations), `ls_iterations=10` (solver steps), `tolerance=1e-8`, `impratio=1.0`, `solref=[0.02 1]`, `solimp=[0.05 0.95 0.01 0.5 2]`, and friction `solimp=[0.4 0.01 0.1]`. The cube's mass is set to the reported groundtruth value and its inertia is computed by MuJoCo's default equal density approximation. DiffMJX uses the same MuJoCo settings with the exception of `iterations=2` (adaptive integration does not require as many solver iterations). The CFD impedance function is set `[0.0, 0.01, 0.01, 1.0, 4.0]` that is $d_c = 0.01$, $d_0 = 0$ with $w_c = 1\,\mathrm{m}$.

E.2.1 Identification of additional MuJoCo parameters

In this section, we extend the experiment from Section 5.1 and use DiffMJX to estimate MuJoCo's simulation parameters alongside the cube's side length. In these experiments, the same setup is used for DiffMJX as detailed in the previous section. In comparison to the experiment in Section 5.1, the integration error tolerance is increased to `1e-4` to reduce computation time. Moreover, the parameters are constrained using either a softplus function or a softclip function where the softening hyper-parameter has been carefully tuned. While the models are trained on an MSE between trajectory segments, we evaluate the model on the "trajectory error" being the mean absolute error between the first forty trajectories in the dataset and DiffMJX's trajectory prediction. In the these experiments, DiffMJX estimates the following parameters simultaneously:

- sidelength as also estimated in Section 5.1,
- mass from which MuJoCo automatically computes the cube's inertia,
- solref parameters (time constant t_c, damping ratio ϕ_d) determining the constraint stiffness,
- solimp parameters determining the constraint's ability to generate force,
- friction parameters determining the extend of the contact friction cone.

In total, we conduct two additional experiments with the Contactnets dataset:

Starting from pre-tuned initial conditions: The optimization is started from the same pre-tuned initial conditions as used in Section 5.1. The prediction horizon is set to $N = 10$. The training results are shown in Fig. 22 (Center). Each model requires around 10 seconds of computation time per gradient step.

Starting from random initial conditions: The optimization is started with random initial parameters. The prediction horizon is set to $N = 4$ such that the majority of runs require around 5 seconds for a gradient step. The training results are shown in Fig. 22 (Bottom).

In both experiments, the trajectory error remains above 0.25. As noted by Parmar et al. (2021), the dynamics of a hard cube exhibit a degree of chaotic behavior. Consequently, small variations in initial conditions – potentially arising from sensor noise or external disturbances such as wind – can substantially alter the cube's trajectory after one of its corners hits the table. Nevertheless, the reduction in trajectory error from approximately 0.32 to 0.26 results in the simulated cube notably more closely matching the real-world data, as shown in Fig. 22.

E.3 MPC experiment

Experimental setup. The MyoHand and MyoArm collision parameters are set to `solref=[0.02 1.0]`, and `solimp=[0.0 0.95 0.001 0.5 2]`, the racket collision parameters are set to `solref=[-100000 0]` for elastic collision. When using CFD, we set `solimp=[0.1 0.95 0.001 0.5 2]` and use `solimp-CFD=[0.0 0.1 1.0 1.0 4]`.

The timestep of the simulation is set to 0.0025, the constraint solver is the Newton solver with 4 iterations and 16 linesearch iterations. All models use our refinements for improving differentiability of MuJoCo's collision detector.

All MPC experiments are performed on a NVIDIA GeForce RTX 3060. In the in-hand manipulation task the runtimes for the gradient-based MPC are 2.9h ($+2.2$h JIT) without CFD and 6.4h ($+2.1$h JIT) with CFD. The sampling with 1024 samples takes 35min ($+3$min JIT). In the bionic tennis task, the runtimes for the gradient-based MPC are 9.7h ($+1.2$h JIT) with CFD. The sampling based MPC runs for 1.4h with 2048 samples.

Note, that we did not optimize any of the methods for speed in this experiment, i.e. in baoding with gradients the task was solved after less than half the total timesteps. The main point here is that we can solve the challenging task using a purely gradient-based method that scales to high-dimensional systems. The large runtime of the gradient-based approach is mainly due to inefficient automatic differentiation of the MJX simulation, which remains the strongest limitation on gradient-based approaches at this point. One potential improvement could be to implement implicit differentiation of the constraint solver in MJX, which could allow for much more efficient gradient computation.