Learning Deployable Locomotion Control via Differentiable Simulation

Clemens Schwarke[1,2], Victor Klemm[1], Joshua Bagajo[1],
Jean-Pierre Sleiman[1,3], Ignat Georgiev[4], Jesus Tordesillas[1,5], Marco Hutter[1]
[1]ETH Zurich, [2]NVIDIA, [3]RAI Institute, [4]Georgia Institute of Technology,
[5]Comillas Pontifical University

Abstract: Differentiable simulators promise to improve sample efficiency in robot learning by providing analytic gradients of the system dynamics. Yet, their application to contact-rich tasks like locomotion is complicated by the inherently non-smooth nature of contact, impeding effective gradient-based optimization. Existing works thus often rely on soft contact models that provide smooth gradients but lack physical accuracy, constraining results to simulation. To address this limitation, we propose a differentiable contact model designed to provide informative gradients while maintaining high physical fidelity. We demonstrate the efficacy of our approach by training a quadrupedal locomotion policy within our differentiable simulator leveraging analytic gradients and successfully transferring the learned policy zero-shot to the real world. To the best of our knowledge, this represents the first successful sim-to-real transfer of a legged locomotion policy learned entirely within a differentiable simulator, establishing the feasibility of using differentiable simulation for real-world locomotion control.

Keywords: Differentiable Simulation, Contact Modeling, Quadruped Locomotion

Figure 1: A walking quadrupedal robot trained in a differentiable simulation. The supplementary video is available at: https://youtu.be/UC4U4xn1e3w

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful framework for optimizing control policies that can solve a variety of robotic tasks, including legged locomotion [1, 2]. Many RL algorithms rely on sampling the task objective to estimate its gradient with respect to the policy parameters, often referred to as Zeroth-order Gradient (ZoG) estimation. This approach allows optimization even when the underlying system dynamics, such as those in conventional physics simulators, are non-differentiable. However, recent advancements in differentiable simulation frameworks enable the efficient computation of analytic First-order Gradients (FoGs) of the system dynamics [3, 4, 5, 6, 7].

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

These analytic gradients promise significantly improved sample efficiency and potentially better asymptotic performance compared to ZoG estimates, due to their lower variance [8, 9, 10].

Despite their potential, leveraging FoGs effectively remains challenging, particularly for contact-rich robotic tasks such as legged locomotion [3, 4, 11]. The core difficulty lies in the non-smooth and discontinuous nature of contact modeling, complicating the optimization landscape and hindering effective FoG-based optimization. To mitigate the issue, many works employ soft contact models [10, 12, 13, 14]. While these models provide smooth, continuous gradients, they often sacrifice physical accuracy, confining results to simulation. Conversely, hard contact models offer high physical fidelity but yield discontinuous and less informative gradient signals, making FoG-based optimization difficult. Consequently, learning robust locomotion policies purely within a differentiable simulator and successfully transferring them to the real world has remained an open challenge [4, 10, 11].

To bridge this gap, we propose a differentiable contact model specifically designed to provide informative gradients for optimization while maintaining high physical fidelity suitable for real-world deployment. Inspired by the effects of stochastic smoothing observed in current RL frameworks [15], our model is derived by analytically smoothing a hard contact formulation, akin to the approach in [16] for quasi-dynamic systems. To validate the benefits of our approach, we compare it against a soft contact model often used in recent works [6, 10, 12, 13] and a hard contact formulation [17, 18].

Applying our model, we train a quadrupedal locomotion policy entirely within our differentiable simulator using the Short-Horizon Actor-Critic (SHAC) algorithm [12], utilizing analytic gradients derived from the proposed contact model. Crucially, we show that the learned policy transfers directly to a real quadrupedal robot. To the best of our knowledge, this represents the first successful sim-to-real transfer of a legged locomotion policy learned entirely within a differentiable simulator leveraging its analytic gradients. This result establishes the feasibility of using differentiable simulation for learning real-world locomotion control, paving the way for more sample-efficient learning paradigms in contact-rich robotic domains.

In summary, our main contributions are:

- A differentiable contact model that combines gradient informativeness and physical fidelity for contact-rich dynamics.
- A demonstration of learning legged locomotion with high sample efficiency, using the analytic gradients of our differentiable simulation.
- Successful zero-shot sim-to-real transfer of a learned locomotion behavior, validating the efficacy of the proposed contact formulation.

2 Related Work

The benefits of optimizing with FoGs have been reported for various applications, such as soft body manipulation [19], system identification from video [20, 21], or grasp synthesis [22]. However, despite promising results, FoG-based methods often encounter challenges related to complex and non-smooth optimization landscapes, which can result in diverging gradients and unstable learning [8].

Several approaches have been proposed to mitigate these difficulties. To improve gradient quality in the presence of discontinuities, prior works have explored techniques such as smoothing collision geometries [22], employing "leaky gradients" to provide information in the absence of contact [22], and utilizing randomized smoothing or analytically smoothed dynamics [15, 16]. The concept of smoothing contact dynamics has also been explored in other areas, such as the field of model-predictive control [23, 24]. Some works address the issue of gradient divergence when differentiating long trajectories via Backpropagation Through Time (BPTT) by training concurrent controllers [9] or optimizing actions at each time step [25]. Furthermore, hybrid methods combining FoGs and ZoGs have been investigated, using FoGs for sample generation [26] or adaptively inter-

polating between gradient types [8]. Notably, the SHAC algorithm [12] successfully integrates FoGs into an actor-critic framework, demonstrating competitive performance on tasks like MuJoCo's ant locomotion [27].

Despite these advancements, applying FoG-based learning to achieve physically realistic locomotion remains challenging [4, 11]. While simulation results using differentiable simulators seem promising [10, 12, 13], their applicability to real-world robotics remains an open question due to potential discrepancies between simulated and real dynamics, especially concerning contact. Ultimately, the goal of learning robotic control is deployment on physical systems, making sim-to-real transfer a critical benchmark. To date, sim-to-real transfer of learned locomotion policies leveraging gradient-based optimization has been extremely limited. One notable success was recently achieved in [28], however, they employed a non-differentiable simulator for forward simulation and a simplified single rigid-body dynamics model for gradient computation. This cleverly avoids the direct trade-off between gradient quality and physical realism within a single simulator but introduces the complexity of managing and aligning two separate models. Our work, in contrast, focuses on utilizing a single, fully differentiable simulation framework.

A key component enabling FoG-based learning is the differentiable simulator itself. Numerous frameworks for differentiable rigid-body dynamics have been developed, differing in their differentiation methods, such as automatic differentiation [4, 6, 11], symbolic differentiation [5], or implicit differentiation [7], as well as their contact modeling approaches. For an introduction to contact modeling, we refer the reader to [29, 30]. Generally, simulators employ either soft contact models [4, 6, 14] which provide smooth gradients amenable to optimization but may lack physical accuracy, or hard contact models, often based on a Linear Complementarity Problem (LCP) [5] or a Nonlinear Complementarity Problem (NCP) [7], which generally offer higher fidelity but yield less informative gradients. For legged locomotion, this trade-off has so far either prevented successful optimization [4, 11] or successful transfer to hardware. Our work addresses this gap by proposing a contact model designed to balance gradient informativeness with the physical fidelity required for sim-to-real transfer.

3 Method

Non-differentiable rigid-body simulators mainly employ hard contact models because they allow for large simulation time steps, do not require parameter tuning, and model physics with high fidelity. However, hard contact models introduce discontinuities into the dynamics and consequently into the optimization objective.

3.1 The Challenge with Discontinuities

To understand the effect of discontinuities on the optimization process, consider a generic optimization problem

$$\min_{\theta} \mathcal{L}(\theta)\,, \tag{1}$$

where $\mathcal{L}$ is the objective or loss function and θ is the optimization variable. ZoG-based algorithms sample $\mathcal{L}$ stochastically to construct a gradient estimate, effectively optimizing the expected value

$$\mathbb{E}_w[\mathcal{L}(\theta + w)] = \int_w p(w)\mathcal{L}(\theta + w)\, dw\,, \tag{2}$$

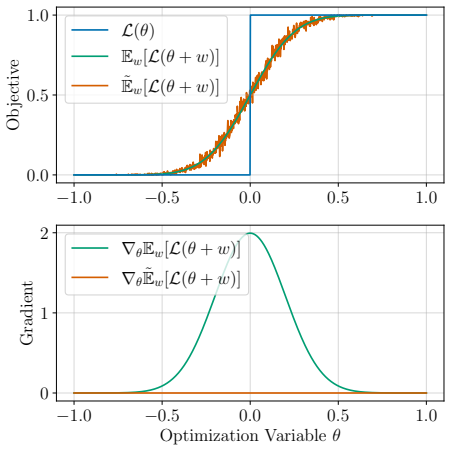

Figure 2: An objective function $\mathcal{L}(\theta)$ and its expected value under stochastic noise. While the approximation $\tilde{\mathbb{E}}$ is an unbiased estimator of the true expected value, the gradient approximation of the expected value $\nabla_\theta \tilde{\mathbb{E}}$ is biased if $\mathcal{L}$ is discontinuous. A comprehensive analysis is given in [8].

where w is stochastic noise and $p(w)$ is its probability distribution. Under a finite number of samples, the expectation is approximated by

$$\tilde{\mathbb{E}}_w[\mathcal{L}(\theta + w)] = \frac{1}{N} \sum_{n=0}^{N} \mathcal{L}(\theta + w_n) \tag{3}$$

with $w_n \sim p(w)$. Figure 2 illustrates how introducing stochasticity effectively smooths discontinuities in the objective landscape, explaining the success of RL methods even in such scenarios. Applying the same idea in the gradient domain yields

$$\nabla_\theta \tilde{\mathbb{E}}_w[\mathcal{L}(\theta + w)] = \frac{1}{N} \sum_{n=0}^{N} \nabla_\theta \mathcal{L}(\theta + w_n)\,, \tag{4}$$

which does not accurately approximate the gradient of the stochastic objective in the presence of discontinuities. Instead, the sampled gradient estimate is biased, often yielding zero or misleading directions near discontinuities, as depicted in Fig. 2. The benefit of stochastic smoothing for ZoG estimation does thus not directly transfer to FoGs. Even without discontinuities, stiff dynamics may lead to what [8] refers to as empirical bias, apparent under a small number of samples N. In essence, the goal is to compute informative FoGs, without the need for sampling inherent to ZoG estimation.

Optimizing with hard contact models faces an additional challenge stemming from discrete-time contact resolution. Most simulators detect contacts only at the beginning of each time step, allowing interpenetration between detection points. This discretization effect can alter or even invert the gradient direction compared to the underlying continuous-time dynamics, further complicating optimization [3].

3.2 Differentiable Rigid-Body Simulation

To address these challenges, we implement a rigid-body simulation with a smooth, differentiable contact model within NVIDIA Warp [6]. Warp is a Python-based kernel programming framework that facilitates high-performance computation through GPU parallelization and source code transformation to CUDA. It supports Automatic Differentiation (AD) by automatically generating adjoint kernels, rendering the implemented dynamics differentiable. Our simulation advances the system dynamics in generalized coordinates and is based on Moreau's time stepping scheme [31], similar to [17, 18], which exhibits improved stability properties over the commonly used semi-implicit Euler integration [32]. A Gauss-Seidel algorithm resolves the NCP of hard contact, which we adapt to achieve smooth dynamics. A detailed description of the simulation algorithm can be found in Appendix A.1.

3.3 Analytic Smoothing of Contact Dynamics

While stochasticity smoothes the discontinuities associated with hard contact, it does not provide meaningful FoGs, as previously presented in Sec. 3.1. Therefore, we propose to smooth the hard contact model analytically by substituting the discontinuous function of the contact force with respect to the penetration depth with a sigmoid function[1]. Fundamentally, we achieve this by scaling the contact forces with

$$\text{sigmoid}(d, \kappa) = \frac{1}{1 + e^{-d\kappa}}\,, \tag{5}$$

where d is the penetration depth and κ determines the stiffness of the function.

Algorithm 1 outlines the modified Gauss-Seidel scheme incorporating analytic smoothing. Inputs to the contact solver are the Delassus Matrix $\boldsymbol{G}$, which expresses the system's inverse inertia in contact coordinates, $\boldsymbol{c}$, which contains dynamic quantities that need to be counteracted by the contact

[1] While smoothing the function with Gaussian noise, often employed to introduce stochasticity in RL methods, would yield the error function [8], this work relies on the sigmoid function for its lower computational cost. The sigmoid function is equivalent to the step function smoothed with logistic noise [16].

impulses, and $\boldsymbol{p}$, an initial guess for the impulses. The modified version additionally requires $\boldsymbol{d}$, which contains the penetration depth for all active contacts, and the stiffness parameter κ. The iteration count N is fixed to allow the solver loop to be unrolled during backpropagation [18]. The contact set C contains all potential contacts whose contact distance is below a certain threshold[2]. Further details about the general algorithm, such as the choice of the r-factor or the $\mathrm{prox}(\cdot)$ operator, are provided in Appendix A.1.

Algorithm 1 Modified Gauss-Seidel Iteration

Input: $\boldsymbol{G}, \boldsymbol{c}, \boldsymbol{p}, \boldsymbol{d}, \kappa$
Output: $\boldsymbol{p}$
  for N solver iterations **do**
    for contact $j \in C$ **do**
      $\boldsymbol{s} \leftarrow \boldsymbol{0}$
      $r \leftarrow 0$
      for contact $k \in C$ **do**
        if $j = k$ **then**
          $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{G}_{jk}\boldsymbol{p}_k$
        else
          $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{G}_{jk}\boldsymbol{p}_k \cdot \mathrm{sigmoid}(d_k, \kappa)$
          $r \leftarrow r + \det(\boldsymbol{G}_{jk})$
      $r \leftarrow \frac{1}{1+r}$
      $\boldsymbol{p}_j \leftarrow \mathrm{prox}(\boldsymbol{p}_j - r(\boldsymbol{s} + \boldsymbol{c}_j))$
  $\boldsymbol{p} \leftarrow \boldsymbol{p} \cdot \mathrm{sigmoid}(\boldsymbol{d}, \kappa)$

Analytic smoothing is incorporated via two distinct sigmoid scaling steps. Within each solver iteration, scaling the impulse of other contacts k by $\mathrm{sigmoid}(d_k, \kappa)$ ensures that the calculation for contact j accounts for the influence of contact k based on the impulse magnitude it is expected to apply, given its penetration depth d_k. Once the solver iterations are complete, the final impulses $\boldsymbol{p}$ are scaled by $\mathrm{sigmoid}(\boldsymbol{d}, \kappa)$ to effectively apply the smoothing. This introduces contact forces at a distance, enabling gradient-based optimization to actively seek contact when desirable.

The level of smoothing may be incrementally reduced during the training process to shift towards a more precise model by adjusting κ. The model can become arbitrarily stiff without destabilizing the simulation, as it converges to the hard contact case with growing κ. However, we did not find such scheduling to be necessary for the studied task, as training and transfer were not sensitive to the level of smoothing.

Figure 3 demonstrates the effects of the proposed contact model on a toy example: an inelastic mass that is falling from different heights for a fixed time, potentially colliding with the ground. Hard contact exhibits two discontinuities. The first is visible as a sawtooth pattern in the mass's position, resulting from discrete time stepping and contact detection. The second discontinuity, evident in the mass's velocity, originates from the discontinuity of the normal contact force, which propagates

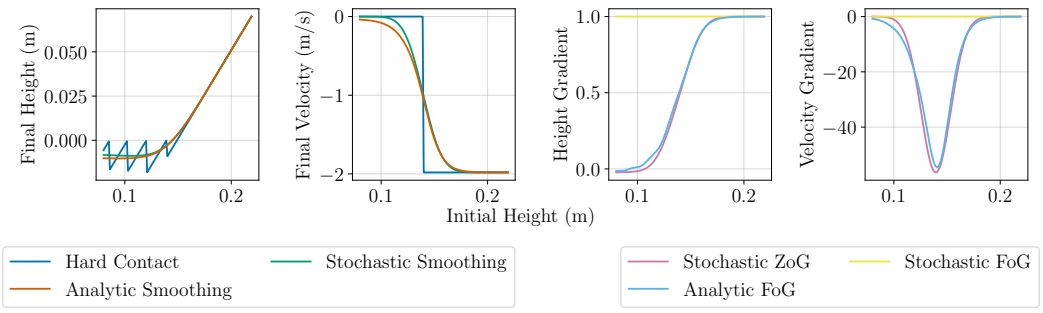

Figure 3: The final height, final velocity, and their gradients with respect to the initial height of a falling mass under gravity. The mass collides with the ground if the initial height is small enough (left half of each graph). Hard contact exhibits discontinuities in the position and velocity domains. Stochasticity smooths these discontinuities but yields a biased and uninformative FoG gradient. The analytically smoothed contact model induces similar effects on the dynamics as stochasticity, with the advantage of informative and unbiased FoGs.

[2]The threshold should be chosen to limit the number of active contacts and thus computational cost, while allowing for forces at a distance for informative gradients. We set the threshold to ∞ and only consider foot contacts.

through the dynamics and becomes apparent in the velocity domain. Analytic smoothing effectively mitigates these issues, closely replicating the smoothing effect of stochasticity, while providing informative and unbiased gradient signals for optimization.

3.4 Considerations for Successful Learning and Sim-to-Real Transfer

Successfully learning legged locomotion and transferring it to physical hardware necessitates further consideration, particularly when training within a differentiable simulator. Key aspects include the choice of optimization algorithm, the fidelity of physical modeling, and strategies to enhance policy robustness.

First, directly optimizing over long task horizons using FoGs via BPTT can lead to diverging and noisy gradients, especially in contact-rich scenarios [28, 25]. To mitigate this, we employ the SHAC algorithm [12]. As a mixed-order method, SHAC utilizes FoGs over a limited horizon and approximates the long-term return using a value function trained with ZoGs. This approach circumvents the challenges of BPTT by restricting backpropagation to shorter, more manageable trajectory segments.

Second, the reward formulation needs to be fully differentiable. The formulation in [2] provides a suitable starting point for learning the quadrupedal velocity-tracking task but needs adaptation. For instance, we replace the non-differentiable feet air time reward, encouraging the robot to take large steps, with a differentiable reward that encourages the feet to follow a certain height trajectory, as detailed in Appendix A.3.

Third, achieving sim-to-real transfer requires accurate modeling of the actuator dynamics. A widely adopted approach is to model the dynamics with a learned actuator network [1]. However, integrating such networks would significantly increase the complexity of the differentiation graph required for FoG computation, due to their recurrence to capture temporal dependencies. Therefore, we fit a simple Proportional-Derivative (PD) law to real-world actuator data [33]. This provides a reasonable approximation of the actuator dynamics while maintaining computational tractability for differentiation.

Finally, we incorporate domain randomization to account for unmodeled effects, such as sensor noise or force disturbances, to robustify the learned policy against the sim-to-real gap. Domain randomization further smoothes the optimization objective stochastically. A complete description of the environment setup, including the randomization and reward formulation, is provided in Appendix A.3.

4 Experimental Results

We first train a quadrupedal robot to walk using a simple environment formulation, detailed in Appendix A.2, similar to the MuJoCo ant environment studied in [12]. With this, we compare our method against the soft contact model used in [10, 12, 13] and the hard contact model obtained by not applying the proposed analytic smoothing. We also compare training with SHAC against training with the purely ZoG-based, state-of-the-art RL algorithm Proximal Policy Optimization (PPO) [34] to confirm the improvements in sample-efficiency presented in [10, 12, 13, 28]. We then progress to training a velocity-tracking task with the quadrupedal robot ANYmal [35].

4.1 Comparison between Contact Models

In the following, we examine whether different contact models enable learning locomotion and sim-to-sim transfer to an accurate hard contact model. Table 1 presents the final performance metrics for the locomotion task trained using SHAC, comparing different contact models. The results were obtained by averaging the performance of 10 training runs with different random seeds (0 to 9) after 1000 training iterations. Each evaluation involved simulating 100 parallel environments for 100 s, with a maximum episode length of 10 s. All three contact models enable the agent to successfully

Table 1: The mean episode performance and standard deviation for different contact models.

	Evaluation with Training Model		Evaluation with Hard Contact	
Training Model	Return	Episode Length (s)	Return	Episode Length (s)
Soft Contact	2231 ± 192	9.69 ± 0.28	325 ± 250	1.58 ± 1.15
Hard Contact	2059 ± 125	9.57 ± 0.15	2059 ± 125	9.57 ± 0.15
Smoothed Contact	$\mathbf{2311 \pm 62}$	$\mathbf{9.88 \pm 0.09}$	$\mathbf{2255 \pm 99}$	$\mathbf{9.73 \pm 0.25}$

learn the task, achieving high returns and episode lengths when evaluated within their respective training environments.

However, the models exhibit significant differences in their characteristics and transferability. The soft contact model suffers from two main issues. Firstly, smaller simulation steps are required to retain stability, reducing simulation speed by a factor of 2 compared to the hard or smoothed contact simulations. Secondly, and more critically, policies trained with the soft contact model fail to transfer to the more physically realistic hard contact setting. This confirms that while the soft contact model facilitates efficient optimization [10, 12, 13], its lack of physical accuracy prevents transfer to the real world. Increasing the contact stiffness to achieve higher realism destabilizes the gradient and prevents learning.

Conversely, the hard contact model provides physically accurate interactions. Interestingly, optimization with SHAC is successful despite the inherent discontinuities in the contact dynamics. This suggests that the implicit smoothing provided by the value function of SHAC helps navigate the challenging optimization landscape. Nevertheless, policies trained directly with hard contact exhibit unnatural walking behavior, involving suboptimal footholds and erratic movements, as can be seen in the supplementary video and the experiments in Appendix A.4. This may be attributed to misleading gradients arising from the contact discontinuities encountered during training.

The analytically smoothed contact model yields the highest performance when evaluated in its training environment, as well as when transferred to hard contact. As analytic smoothing is able to closely replicate the characteristics of stochastic smoothing, it might retain the underlying hard contact dynamics within its domain, explaining the successful transfer. While the approximately $10\,\%$ increase in return compared to training with hard contact might appear modest, the qualitative difference in behavior is significant. The resulting locomotion behaviors are considerably smoother and show more natural and efficient gaits.

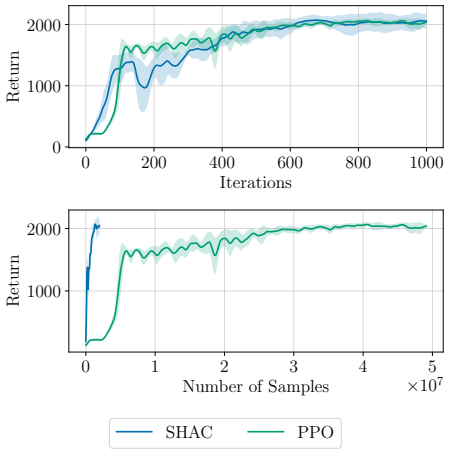

Figure 4: The episode return for the algorithms SHAC and PPO throughout training in terms of iterations and number of samples. SHAC is trained with 64 and PPO with 2048 parallel environments. The reward is averaged over five training runs with different random seeds (0 to 4).

4.2 Comparison between RL Algorithms

Next, we compare SHAC to the publicly available PPO implementation of RSL RL [2]. Both algorithms exhibit analogous convergence properties and achieve nearly identical final rewards, as shown in Fig. 4. Nevertheless, SHAC outperforms PPO in terms of sample efficiency by over an order of magnitude, owing to the reduced variance of ZoGs. However, PPO demonstrates superior performance in terms of computational time with $0.18\,\mathrm{s}$ per iteration compared to $0.33\,\mathrm{s}$ with SHAC, as detailed further in Appendix A.4. While SHAC benefits from faster rollouts due to using fewer parallel environments, its learning phase is slower because it requires backpropagating through the entire rollout.

This observation aligns with the findings in [12] and suggests that the advantages of FoG-based optimization become more apparent for higher-dimensional problems, such as humanoid control or learning from vision [36].

4.3 Training a Velocity-Tracking Policy for Real-World Deployment

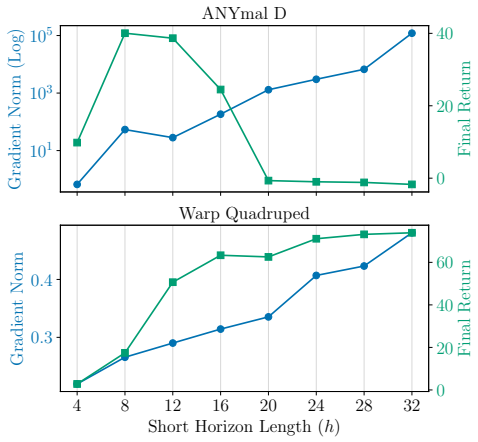

Figure 5: The mean gradient norm during training of the velocity-tracking task and the final return after 1000 iterations across different horizon lengths for both robots.

Moving from the quadruped provided with Warp to the ANYmal robot proves challenging, due to its mass distribution and inertia. While the previously used quadruped features a light base and high leg inertia, ANYmal has a heavy base and light shanks with low inertia, resulting in significantly less smooth dynamics. Furthermore, ANYmal's actuators deliver four times the maximum torque. The implications on the learning process can be seen in Fig. 5. While the mean gradient norm during training with Warp's quadruped remains well below 1.0 for any horizon length, the gradient norm with ANYmal increases exponentially and prohibits successful learning when backpropagating FoGs over more than 16 steps. This highlights another major mismatch between simulation benchmarks, such as the ant environment [27], and real physical systems, further compared in Appendix A.4. We use a short horizon length of 12 to facilitate stable optimization.

After training, the learned locomotion skill transfers to the real robot zero-shot. The robot is able to closely follow the velocity commands given by the operator as shown in Fig. 6. Notably, the real velocity trajectory closely matches the one recorded in our differentiable simulation, confirming its sim-to-real capabilities. The learned control policy demonstrates high levels of robustness, withstanding force perturbations, as shown in the accompanying video. In summary, our experiments confirm the successful learning, zero-shot transfer, and robust real-world execution of locomotion policies trained using the proposed method.

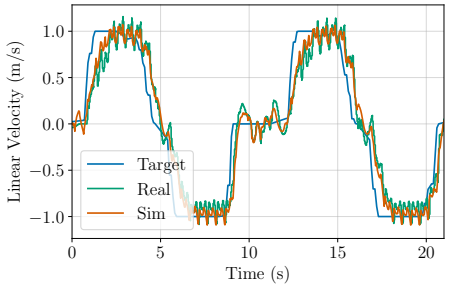

Figure 6: The target velocity for walking forward/backward and the measured velocity on the real robot and in simulation.

5 Conclusion and Future Work

This work addressed the challenge of leveraging analytic gradients for the contact-rich task of legged locomotion by introducing an analytically smoothed contact model that balances informative gradients with the physical fidelity needed for hardware deployment. We confirmed the advantages of analytic smoothing by comparing our model to two alternative contact modeling approaches. We then successfully transferred a locomotion skill, learned entirely within our differentiable simulator, zero-shot to a real quadrupedal robot. This validates the feasibility of FoG-based optimization even for complex, contact-rich tasks, offering a path towards more sample-efficient robot learning.

In future work, it would be interesting to scale this approach to systems with high-dimensional action spaces, like humanoids, or high-dimensional observation spaces, such as images, to leverage the improved sample efficiency. Lastly, incorporating differentiable rough terrain would make this approach more competitive with current RL methods.

6 Limitations

While our approach enables successful sim-to-real transfer, several limitations remain. FoG-based optimization with SHAC, although sample-efficient, incurs higher wall-clock time per iteration than optimization with PPO due to the added need for differentiation. This computational overhead might negate the gained benefits of higher sample efficiency in low-dimensional optimization problems. Furthermore, to maintain computational tractability, we used a simple PD law for actuator modeling rather than a more complex learned network. This simplification might reduce simulation fidelity, potentially affecting transfer for tasks demanding high actuator precision. Finally, the need for full differentiability imposes a constraint not present in ZoG methods, requiring more effort in aspects such as simulation design or reward engineering.

Acknowledgments

This project was supported by the Swiss National Science Foundation through the National Centre of Competence in Automation (NCCR automation) and received funding from the European Union's Horizon Europe Framework Programme under grant agreement No. 101070596.

References

[1] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[2] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[3] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand. Difftaichi: Differentiable programming for physical simulation. In *ICLR*, 2019.

[4] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax-a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[5] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu. Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact. *Robotics: Science and Systems*, 2021. ISSN 2330765X. doi:10.15607/RSS.2021.XVII.034.

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

[7] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 9, 2022.

[8] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[9] N. Wiedemann, V. Wüest, A. Loquercio, M. Müller, D. Floreano, and D. Scaramuzza. Training efficient controllers via analytic policy gradient. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1349–1356. IEEE, 2023.

[10] I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. *arXiv preprint arXiv:2405.17784*, 2024.

[11] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13(March):1–9, 2019. ISSN 16625218. doi: 10.3389/fnbot.2019.00006.

[12] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.

[13] E. Xing, V. Luk, and J. Oh. Stabilizing reinforcement learning in differentiable multiphysics simulation. *arXiv preprint arXiv:2412.12089*, 2024.

[14] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros. ADD: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics*, 39(6), 2020. ISSN 15577368. doi:10.1145/3414685.3417766.

[15] H. J. T. Suh, T. Pang, and R. Tedrake. Bundled Gradients Through Contact Via Randomized Smoothing. *IEEE Robotics and Automation Letters*, 7(2):4000–4007, 2022. ISSN 23773766. doi:10.1109/LRA.2022.3146931.

[16] T. Pang, H. J. Suh, L. Yang, and R. Tedrake. Global Planning for Contact-Rich Manipulation via Local Smoothing of Quasi-Dynamic Contact Models. *IEEE Transactions on Robotics*, pages 1–20, 2023. ISSN 19410468. doi:10.1109/TRO.2023.3300230.

[17] C. Gehring, R. Diethelm, R. Siegwart, G. Nützi, and R. Leine. An evaluation of moreau's time-stepping scheme for the simulation of a legged robot. In *IDETC/CIE 2014*, number DETC2014-34374, 2014.

[18] J. Carius, R. Ranftl, V. Koltun, and M. Hutter. Trajectory optimization with implicit hard contacts. *IEEE Robotics and Automation Letters*, 3(4):3316–3323, 2018. ISSN 23773766. doi:10.1109/LRA.2018.2852785.

[19] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan. Plasticinelab: a Soft-Body Manipulation Benchmark With Differentiable Physics. *ICLR 2021 - 9th International Conference on Learning Representations*, pages 1–18, 2021.

[20] S. Le Cleac'h, H. X. Yu, M. Guo, T. Howell, R. Gao, J. Wu, Z. Manchester, and M. Schwager. Differentiable Physics Simulation of Dynamics-Augmented Neural Objects. *IEEE Robotics and Automation Letters*, 8(5):2780–2787, 2023. ISSN 23773766. doi:10.1109/LRA.2023.3257707.

[21] J. K. Murthy, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben, et al. gradsim: Differentiable simulation for system identification and visuomotor control. In *ICLR*, 2020.

[22] D. Turpin, L. Wang, E. Heiden, Y.-C. Chen, M. Macklin, S. Tsogkas, S. Dickinson, and A. Garg. Grasp'd: Differentiable contact-rich grasp synthesis for multi-fingered hands. In *European Conference on Computer Vision*, pages 201–221. Springer, 2022.

[23] S. Le Cleac'h, T. A. Howell, S. Yang, C.-Y. Lee, J. Zhang, A. Bishop, M. Schwager, and Z. Manchester. Fast contact-implicit model predictive control. *IEEE Transactions on Robotics*, 40:1617–1629, 2024.

[24] G. Kim, D. Kang, J.-H. Kim, S. Hong, and H.-W. Park. Contact-implicit mpc: Controlling diverse quadruped motions without pre-planned contact modes or trajectories. *arXiv preprint arXiv:2312.08961*, pages 29–84, 2023.

[25] M. A. Z. Mora, M. Peychev, S. Ha, M. Vechev, S. Coros, M. Zamora, M. Peychev, S. Ha, M. Vechev, and S. Coros. PODS: Policy Optimization via Differentiable Simulation. *38th International Conference on Machine Learning*, pages 7805–7817, 2021. ISSN 2640-3498.

[26] Y. L. Qiao, J. Liang, V. Koltun, and M. C. Lin. Efficient Differentiable Simulation of Articulated Bodies. *Proceedings of Machine Learning Research*, 139:8661–8671, 2021. ISSN 26403498.

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

[28] Y. Song, S. Kim, and D. Scaramuzza. Learning quadruped locomotion using differentiable simulation. *arXiv preprint arXiv:2403.14864*, 2024.

[29] P. C. Horak and J. C. Trinkle. On the similarities and differences among contact models in robot simulation. *IEEE Robotics and Automation Letters*, 4(2):493–499, 2019.

[30] Q. Le Lidec, W. Jallet, L. Montaut, I. Laptev, C. Schmid, and J. Carpentier. Contact models in robotics: a comparative analysis. *IEEE Transactions on Robotics*, 2024.

[31] J. J. Moreau. Unilateral contact and dry friction in finite freedom dynamics. In *Nonsmooth mechanics and Applications*, pages 1–82. Springer, 1988.

[32] J. Bender, K. Erleben, and J. Trinkle. Interactive simulation of rigid body dynamics in computer graphics. *Computer Graphics Forum*, 33(1):246–270, 2014. ISSN 14678659. doi: 10.1111/cgf.12272.

[33] F. Bjelonic, F. Tischhauser, and M. Hutter. Towards bridging the gap: Scalable sim-to-real transfer for legged locomotion. In preperation, 2025.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv e-prints*, 2017.

[35] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 38–44. IEEE, 2016.

[36] J. Y. Luo, Y. Song, V. Klemm, F. Shi, D. Scaramuzza, and M. Hutter. Residual policy learning for perceptive quadruped control using differentiable simulation. *arXiv preprint arXiv:2410.03076*, 2024.

[37] R. Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[38] S. Andrews, K. Erleben, and Z. Ferguson. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses*, pages 1–172. 2022.

[39] N. Parikh, S. Boyd, et al. Proximal algorithms. *Foundations and trends in Optimization*, 1(3): 127–239, 2014.

[40] C. Studer. *Numerics of unilateral contacts and friction: modeling and numerical time integration in non-smooth dynamics*, volume 47. Springer Science & Business Media, 2009.

[41] F. d. A. Belbute-Peres, K. R. Allen, K. A. Smith, J. B. Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018. ISSN 10495258.

Appendix

In the following, we provide additional details referenced in the main text, covering the simulation framework, experimental setups, and supporting results.

A.1 Simulation Implementation

This section describes the implemented simulation routine to provide deeper intuition into the contact resolution algorithm and the simulation in general. We also detail the soft contact formulation used in [6, 10, 12, 13].

The fundamental principle of simulating a system of rigid bodies forward in time is the integration of the system's Equation of Motion (EoM) to find future velocities and positions. The EoM can be stated in a general form as

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} + \boldsymbol{J}_c^\mathsf{T} \boldsymbol{f}_c \,, \tag{6}$$

where $\boldsymbol{q}$ describes the position, $\dot{\boldsymbol{q}}$ the velocity[3], and $\ddot{\boldsymbol{q}}$ the acceleration of the system in generalized coordinates. $\boldsymbol{H}(\boldsymbol{q})$ denotes the inertia matrix, $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ includes centrifugal, Coriolis, and gravitational forces, $\boldsymbol{\tau}$ is the actuation of the system, and $\boldsymbol{f}_c$ includes all contact forces and is projected onto the generalized coordinate space by the contact Jacobian $\boldsymbol{J}_c$. Note that the EoM may also be formulated in maximal coordinates. However, many robotics simulators rely on generalized coordinates to reduce the number of necessary system constraints.

Since analytic integration is generally intractable, numerical integration schemes approximate the continuous-time behavior of the system. First-order integrators, such as Euler's method, are often chosen for their simplicity. We implement Moreau's Time Stepping scheme [31], a more accurate integration method also discussed in [17, 18]. As detailed in Alg. 2, this scheme begins with an explicit Euler half-step to find the system's position at the midpoint of the simulation step i with length h.

Subsequently, the simulator evaluates the individual EoM terms to calculate the acceleration required for integration. First, $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is found by assuming all accelerations to be zero and then solving the inverse dynamics problem, i.e., finding the forces that lead to a given motion. The Recursive Newton-Euler Algorithm (RNEA) efficiently computes the inverse dynamics for kinematic trees [37]. Second, $\boldsymbol{H}$ can be constructed using the Composite-Rigid-Body Algorithm (CRBA) [37]. Arguably, the most challenging task is finding the appropriate contact forces $\boldsymbol{f}_c$.

Instead of computing the forces and integrating them over the time step, the contact solver directly determines the required impulses $\boldsymbol{p}$ to satisfy all contact constraints at the end of the simulation step. The first part of the procedure, summarized in Alg. 2 as TOCONTACT$(\cdot)$, involves projecting parts of the EoM onto the contact domain using the contact Jacobian $\boldsymbol{J}_c$. This Jacobian relates variations in the generalized coordinates $\boldsymbol{q}$ with variations in the local contact coordinates. Note that $\boldsymbol{J}_c$ includes all active contacts of the system, i.e.,

$$\boldsymbol{J}_c = \begin{bmatrix} \boldsymbol{J}_1 \\ \boldsymbol{J}_2 \\ \vdots \\ \boldsymbol{J}_{n_c} \end{bmatrix} \,, \tag{7}$$

where n_c is the number of active contacts. Given the spatial Jacobian $\boldsymbol{J}_s$, a byproduct of the CRBA algorithm, the contact Jacobians for each active contact j can be readily computed by

$$\boldsymbol{J}_j = \boldsymbol{J}_{s_k,P} - [\boldsymbol{r}]_\times \boldsymbol{J}_{s_k,R} \,, \tag{8}$$

where index k selects the part of $\boldsymbol{J}_s$ that corresponds to the rigid body at which the contact occurs, indices P and R denote the positional and rotational part of the Jacobian, and $\boldsymbol{r}$ is a vector from the rigid body's origin to the contact point.

[3]The velocity is generally not equal to the time derivative of the generalized coordinate vector $\boldsymbol{q}$. However, to align with the literature, the velocity term is still referred to as $\dot{\boldsymbol{q}}$, despite the slight abuse of notation.

Algorithm 2 Moreau's Time Stepping Scheme

Input: $\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \boldsymbol{\tau}_i$
Output: $\boldsymbol{q}_{i+1}, \dot{\boldsymbol{q}}_{i+1}$

```
q_mid ← q_i + (h/2) q̇_i                                  ▷ Explicit Euler half-step
h ← RNEA(q_mid, q̇_i, τ_i)                                ▷ RNEA merges control inputs into h
H ← CRBA(q_mid)
J_c, G, c, p ← TOCONTACT(H, h, q_mid, q̇_i)
for N solver iterations do                               ▷ Gauss-Seidel iteration
    for j = 1, ..., n_c do                               ▷ Iterate over n_c active contacts
        s ← 0
        r ← 0
        for k = 1, ..., n_c do
            s ← s + G_jk p_k
            r ← r + det(G_jk)
        r ← 1/(1+r)
        p_j ← prox(p_j − r(s + c_j))
q̇_{i+1} ← q̇_i + H^{-1}(J_c^T p − hh)                     ▷ Midpoint step
q_{i+1} ← q_i + (h/2)(q̇_i + q̇_{i+1})
```

The first result of the projection using $\boldsymbol{J_c}$ is the Delassus Matrix $\boldsymbol{G}$, which expresses the system's inverse inertia in the contact coordinates. It is found using

$$\boldsymbol{G} = \boldsymbol{J}_c \boldsymbol{H}^{-1} \boldsymbol{J}_c^{\mathsf{T}} \,. \tag{9}$$

However, computing the inverse inertia $\boldsymbol{H}^{-1}$ explicitly is generally avoided due to its computational cost and potential numerical instability. Therefore, $\boldsymbol{G}$ is computed more efficiently by utilizing the factors of $\boldsymbol{H}$, e.g., from reordered Cholesky factorization. The next term required is the vector $\boldsymbol{c}$, computed with

$$\boldsymbol{c} = \boldsymbol{J}_c(\dot{\boldsymbol{q}}_i + h\boldsymbol{H}^{-1}\boldsymbol{h}) \,, \tag{10}$$

which includes dynamic quantities that need to be counteracted by the contact impulses to achieve $\boldsymbol{v}_c = 0$ at the time step's end.

All contact impulses are initialized to

$$\boldsymbol{p}_j = -\boldsymbol{G}_{jj}^{-1} \boldsymbol{c}_j \,, \tag{11}$$

where $\boldsymbol{p}_j$ and $\boldsymbol{c}_j$ represent the j-th elements of $\boldsymbol{p}$ and $\boldsymbol{c}$, corresponding to the j-th active contact. In contrast to [18], tangential impulses are not set to zero, which yields more accurate results for a limited number of iterations in this work.

Initializing $\boldsymbol{p}$ according to Eq. 11 does not yet account for the interactions between contacts and the constraints of the friction cone. Thus, the second part of the contact handling procedure involves iterating over all contacts in a Gauss-Seidel fashion to converge to appropriate impulses considering interactions between contacts. In this approach, the impulses for each contact are updated sequentially using the most recently computed values of all other impulses. Every impulse update aims to reduce the current constraint violation at the corresponding contact. The update for $\boldsymbol{p}_j$, before enforcing the friction cone constraint, can be expressed as

$$\boldsymbol{p}_{j,\text{update}} = \boldsymbol{p}_j - r_j \,(\sum_{k=1}^{n_c} \boldsymbol{G}_{jk}\boldsymbol{p}_k + \boldsymbol{c}_j) \,. \tag{12}$$

The convergence properties of the iteration scheme are influenced by the r-factor, and several strategies for choosing an appropriate value have been proposed [38]. In this work, r is chosen to be

$$r_j = \frac{1}{1 + \sum_{k=1}^{n_c} |\boldsymbol{G}_{jk}|} \,, \tag{13}$$

a minor variation from [18], to ensure that r is always bounded. Lastly, a proximal projection [39, 40] with the proximal operator

$$\text{prox}(\boldsymbol{p}_j) = \begin{cases} \mathbf{0}, & \text{if } p_{n_j} \leq 0 \\ \boldsymbol{p}_j, & \text{if } p_{n_j} > 0 \wedge \|\boldsymbol{p}_{t_j}\| \leq \mu p_{n_j} \\ [p_{n_j}, \mu p_{n_j} \frac{\boldsymbol{p}_{t_j}^\mathsf{T}}{\|\boldsymbol{p}_{t_j}\|}]^\mathsf{T}, & \text{if } p_{n_j} > 0 \wedge \|\boldsymbol{p}_{t_j}\| > \mu p_{n_j} \end{cases} \tag{14}$$

at each update step ensures that the contact forces always remain within the bounds of the friction cone.

Once the iterative solver has determined the final contact impulses $\boldsymbol{p}$ after N iterations, the generalized velocity $\dot{\boldsymbol{q}}_{i+1}$ can be computed using

$$\dot{\boldsymbol{q}}_{i+1} = \dot{\boldsymbol{q}}_i + \boldsymbol{H}^{-1}(\boldsymbol{J}_c^\mathsf{T}\boldsymbol{p} - h\boldsymbol{h})\,. \tag{15}$$

As with the computation of $\boldsymbol{G}$ and $\boldsymbol{c}$, multiplication by the inverse inertia is performed efficiently using the factorization of $\boldsymbol{H}$ rather than explicit inversion. Finally, the generalized positions are updated using a trapezoidal rule, averaging the velocities at the beginning and end of the step with

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i + \frac{h}{2}(\dot{\boldsymbol{q}}_i + \dot{\boldsymbol{q}}_{i+1})\,. \tag{16}$$

This completes simulation step i, yielding the state $(\boldsymbol{q}_{i+1}, \dot{\boldsymbol{q}}_{i+1})$ needed for the next iteration.

Implementing this routine in Warp [6] allows us to obtain gradients via AD. While Warp handles differentiation of the core dynamics, the iterative nature of the Gauss-Seidel contact solver requires special consideration. By fixing the number of iterations, we make the solver amenable to AD, as the loop can be unrolled during backpropagation. This method offers considerable implementation simplicity compared to analytical alternatives based on the implicit function theorem [7, 23, 41]. Furthermore, it bypasses the need to derive and implement complex analytical sensitivities, e.g., of the contact Jacobian or inverse inertia. Consequently, for small iteration counts, $N = 10$ in this work, differentiating the unrolled solver within Warp presents a practical and potentially more efficient strategy for obtaining gradients through the contact dynamics in our framework.

The soft contact model used as a baseline in this work does not rely on an iterative solver. Instead, normal contact forces can be directly computed using

$$f_n = \begin{cases} k_p d - k_d \min(v_n, 0), & \text{if } d \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

and tangential friction forces follow

$$\boldsymbol{f}_t = -\frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|_2} \min(k_f \|\boldsymbol{v}_t\|_2, \mu f_n)\,, \tag{18}$$

where $\boldsymbol{v}$ is the relative velocity in the contact frame, and μ is the friction coefficient. The parameters k_p, k_d, and k_f have to be tuned and trade off physical accuracy against simulation stability and gradient smoothness.

A.2 Environment Setup for Quadrupedal Locomotion

The experiments in Sec. 4.1 and Sec. 4.2 were conducted using a simple environment, closely related to the one studied in [12] for the ant robot, for better comparability and reproducibility. However, we train the quadruped provided with Warp [6] to walk forward with a constant velocity of $1.0\,\mathrm{m/s}$ to better align with the goal of achieving deployable locomotion control. Table 4 specifies the used rewards. The observations received by the control policy are identical to [12], but listed in Tab. 5 for completeness. The policy outputs position commands for the joints that are then tracked with a PD law. The maximum torque applied by the actuators is $20\,\mathrm{Nm}$. Environments are terminated if the center of the quadruped's base falls below $0.25\,\mathrm{m}$ with respect to the ground or if the maximum episode length of $10\,\mathrm{s}$ is reached. All training parameters are listed in Tab. 6.

A.3 Environment Setup for Velocity-Tracking with ANYmal

The environment formulation for the velocity-tracking task differs from the previously discussed setup in its reward and observation formulations, detailed in Tab. 7 and Tab. 8, respectively. As mentioned in the main text, we modify the reward formulation of [2] to ensure differentiability. The original formulation encourages large steps via long foot swing durations, which aids transfer to real terrain but is inherently non-differentiable. We replace this reward with a differentiable reward for tracking a foot height reference. Although the task is learnable without the reward, it significantly enhances transferability to ground that is not perfectly flat. Note that this reward can be removed when training on rough terrain. The height reference is generated using a sinusoid, phase-shifted by π for alternating feet, and can be expressed as

$$z^*_{foot} = 0.1 \cdot \max(\sin(4\pi t), 0)\,, \tag{19}$$

where t is the current time in seconds.

Additionally, domain randomization is introduced to facilitate sim-to-real transfer. This includes adding noise to observations, varying the robot mass and friction coefficient across environments, and changing the robot's base velocity at randomly sampled time steps. Table 9 specifies the randomization ranges from which values are uniformly sampled. Furthermore, each environment receives a sampled velocity command that the robot is tasked with tracking. Lastly, to better approximate the real-world actuator dynamics in simulation, we fit the parameters of the PD law for joint position target tracking and the joint armature using real-world data [33].

A.4 Additional Experimental Results

This section supplements the main experimental results with additional data on computational performance during training, robot characteristics relevant for well behaved gradients, and examplary joint trajectories during locomotion.

Table 2 provides a breakdown of the computational time per training iteration for the algorithms SHAC and PPO. All experiments in this work were conducted on a single NVIDIA RTX 2080 Ti graphics card and the reported times are averaged over 5 training runs. While SHAC requires significantly fewer samples per iteration compared to PPO, its learning phase takes longer due to the need for backpropagation through the simulation rollout. This results in a longer total time per iteration for SHAC compared to PPO, despite faster rollouts. Thus, the benefit of FoG-based methods is expected to become relevant in higher-dimensional problem settings, where sample generation is a bottleneck.

Table 2: The computational time for one training iteration.

Algorithm	Rollout Time (s)	Learning Time (s)	Total Time (s)	Samples
SHAC	0.086 ± 0.009	0.244 ± 0.013	0.330 ± 0.019	2048
PPO	0.130 ± 0.009	0.053 ± 0.005	0.183 ± 0.012	49152

Table 3 compares physical properties of the ANYmal D robot used in our final experiments with the Warp Quadruped and the commonly used MuJoCo Ant benchmark. While all three robots are of comparable size, they exhibit substantial differences in total weight and in the ratio of base to leg mass. ANYmal's base to shank mass ratio is over an order of magnitude higher compared to the other two robots. Combined with its higher maximum actuator torque, this leads to substantially less smooth dynamics, as discussed in Sec. 4.3 and reflected in the gradient norm analysis in Fig. 5. These differences underscore the importance of evaluating new methods on physically realistic models and, ultimately, on real robots.

Figure 7 highlights the difference in motion quality between policies trained with smoothed and hard contact models. The policy trained with smoothed contact exhibits smooth, periodic joint trajectories. In contrast, the policy trained with hard contact produces a less regular gait. Furthermore, the

Table 3: Comparison of Robot Properties.

Robot	Height (m)	Torso Mass (kg)	Shank Mass (kg)	Torque (N m)
MuJoCo Ant [27]	0.75	0.33	0.10	1
Warp Quadruped [6]	0.58	6.20	2.00	20
ANYmal D	0.70	25.00	0.76	80

smoothed contact policy utilizes a significantly smaller range of joint motion, resulting in a more efficient motion that contributes to the higher return presented in Tab. 1.

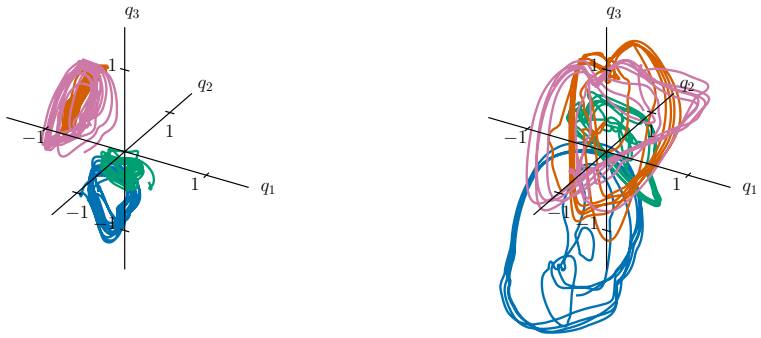

Figure 7: The joint position trajectories of the four legs (shown in different colors) of the quadruped robot, comparing a policy trained with smoothed contact (left) to a policy trained with hard contact (right). Both policies are evaluated using hard contact.

Figure 8 shows joint position trajectories for the ANYmal robot during locomotion, comparing real-world execution with simulations using both smoothed and hard contact. The trajectories from the simulation closely match the real-world data, validating the successful sim-to-real transfer. Furthermore, the trajectories from the smoothed contact simulation are highly similar to those from the hard contact simulation. This strong agreement suggests that our smoothed contact model captures the essential dynamics of hard contact and that the remaining sim-to-real gap mainly originates from factors beyond contact modeling, such as unmodeled actuator dynamics or system parameters. The smooth and periodic nature of all trajectories reflects the stable gait learned by the policy.

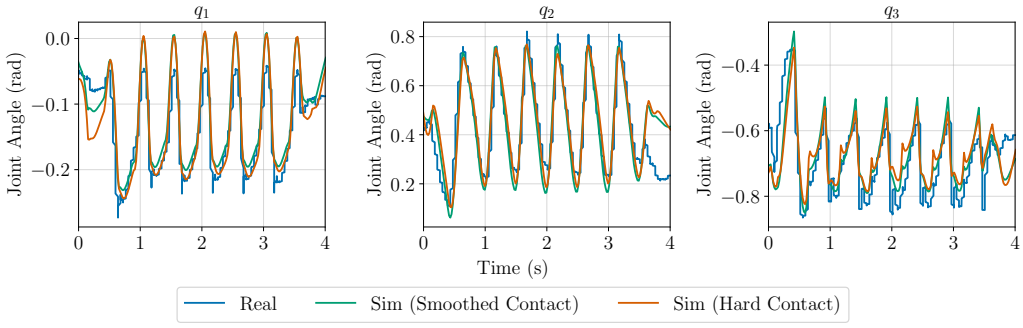

Figure 8: The joint position trajectories of ANYmal's left front leg, comparing real-world execution with simulation using smoothed and hard contact.

Table 4: The rewards for the quadruped locomotion task.

Name	Formula	Weight
Base velocity	$\exp(-\lvert v_x - 1.0\rvert)$	1.0
Base height	$\exp(-\lvert z - 0.45\rvert)$	0.5
Base alignment	$\boldsymbol{e}_z^{\text{world}} \cdot \boldsymbol{e}_z^{\text{base}}$	0.5
Action magnitude	$\sum_i \exp(-\lvert a_i\rvert)$	0.01
Joint velocity	$-\lVert\dot{\boldsymbol{q}}\rVert^2$	0.001

Table 5: The observations for the quadruped locomotion task.

Name	Symbol
Base height	$z \in \mathbb{R}^1$
Base orientation	$\boldsymbol{\xi} \in \mathbb{R}^4$
Linear base velocity	${}_{\mathcal{B}}\boldsymbol{v} \in \mathbb{R}^3$
Angular base velocity	${}_{\mathcal{B}}\boldsymbol{\omega} \in \mathbb{R}^3$
Joint position	$\boldsymbol{q} \in \mathbb{R}^{12}$
Joint velocity	$\dot{\boldsymbol{q}} \in \mathbb{R}^{12}$
Base alignment	$\boldsymbol{e}_z^{\text{world}} \cdot \boldsymbol{e}_z^{\text{base}} \in \mathbb{R}^1$
Base heading alignment	$\boldsymbol{e}_x^{\text{world}} \cdot \boldsymbol{e}_x^{\text{base}} \in \mathbb{R}^1$
Previous action	$\boldsymbol{a}_{\text{prev}} \in \mathbb{R}^{12}$

Table 6: The parameters for the quadruped locomotion task.

Name	Value
Environment	
Episode length	$10.0\,\mathrm{s}$
Environment time step length	$0.01\,\mathrm{s}$
Simulation	
Simulation time step length	$0.01\,\mathrm{s}$
Soft simulation time step length	$0.0005\,\mathrm{s}$
Gauss-Seidel iterations	10
Smoothing parameter κ	300
Friction coefficient	0.8
Joint stiffness	20.0
Joint damping	1.0
Soft contact stiffness	$1.2 \cdot 10^4$
Soft contact damping	$3.0 \cdot 10^1$
Soft contact friction	$9.0 \cdot 10^2$
SHAC	
Training iterations	1000
Parallel environments	64
Short horizon length	32
Actor learning rate	0.002
Critic learning rate	0.002
Actor learning rate decay	0.995
Critic learning rate decay	0.997
Discount factor	0.99
Value estimation	0.95
Target value network	0.2
Critic iterations	16
Critic mini batches	4
Maximum actor gradient norm	1.0
Maximum critic gradient norm	10.0
PPO	
Training iterations	1000
Parallel environments	2048
Rollout length	24
Learning rate	0.001
Discount factor	0.99
Value estimation	0.95
Value loss coefficient	1.0
Entropy coefficient	0.0
Learning epochs	5
Mini batches	4
Desired KL divergence	0.01
Maximum gradient norm	1.0
Network architecture	
Actor MLP dimensions	$(128, 64, 32)$
Critic MLP dimensions	$(64, 64)$
Activation function	ELU

Table 7: The rewards for the velocity-tracking task.

Name	Formula	Weight
Linear velocity tracking	$\exp(-\frac{1}{0.25}\|{}_{\mathcal{B}}\boldsymbol{v}_{xy} - {}_{\mathcal{B}}\boldsymbol{v}^*_{xy}\|^2)$	1.0
Angular velocity tracking	$\exp(-\frac{1}{0.25}({}_{\mathcal{B}}\omega_z - {}_{\mathcal{B}}\omega^*_z)^2)$	0.5
Foot height tracking	$\sum_j \frac{z^*_{foot,j}}{0.1} \cdot \exp(-\frac{1}{0.05}(z_{foot,j} - z^*_{foot,j})^2)$	3.0
Linear velocity error	$-{}_{\mathcal{B}}v_z^2$	2.0
Angular velocity error	$-\|{}_{\mathcal{B}}\boldsymbol{\omega}_{xy}\|^2$	0.05
Base height	$\exp(-\frac{1}{0.1}(z - 0.45)^2)$	1.0
Base orientation	$-\|{}_{\mathcal{B}}\boldsymbol{g}_{xy}\|^2$	0.5
Action magnitude	$-\sum_i \|a_i\|$	0.05
Action rate	$-\|\boldsymbol{a} - \boldsymbol{a}_{\text{prev}}\|^2$	0.01
Joint acceleration	$-\|\ddot{\boldsymbol{q}}\|^2$	$2.5 \cdot 10^{-7}$
Joint torque	$-\|\boldsymbol{\tau}\|^2$	$2.5 \cdot 10^{-5}$

Table 8: The observations for the velocity-tracking task.

Name	Symbol
Linear base velocity	${}_{\mathcal{B}}\boldsymbol{v} \in \mathbb{R}^3$
Angular base velocity	${}_{\mathcal{B}}\boldsymbol{\omega} \in \mathbb{R}^3$
Projected gravity	${}_{\mathcal{B}}\boldsymbol{g} \in \mathbb{R}^3$
Velocity command	$({}_{\mathcal{B}}\boldsymbol{v}^*_{xy}, {}_{\mathcal{B}}\omega^*_z)^\top \in \mathbb{R}^3$
Joint position	$\boldsymbol{q} \in \mathbb{R}^{12}$
Joint velocity	$\dot{\boldsymbol{q}} \in \mathbb{R}^{12}$
Previous action	$\boldsymbol{a}_{\text{prev}} \in \mathbb{R}^{12}$
Phase	$\sin(4\pi t) \in \mathbb{R}^1$

Table 9: The parameters for the velocity-tracking task.

Name	Value
Environment	
Episode length	$20.0\,\mathrm{s}$
Environment time step length	$0.02\,\mathrm{s}$
Simulation	
Simulation time step length	$0.005\,\mathrm{s}$
Gauss-Seidel iterations	10
Smoothing parameter κ	300
Joint stiffness	85.0
Joint damping	0.6
Joint armature	0.1
SHAC	
Training iterations	20000
Parallel environments	128
Short horizon length	12
Actor start learning rate	0.005
Critic start learning rate	0.002
Actor final learning rate	$1.0 \cdot 10^{-5}$
Critic final learning rate	$1.0 \cdot 10^{-5}$
Discount factor	0.99
Value estimation	0.95
Critic iterations	16
Critic mini batches	4
Maximum actor gradient norm	1.0
Maximum critic gradient norm	10.0
Randomization	
Friction coefficient range	$[0.5, 1.25]$
Added base mass range	$[-5.0\,\mathrm{kg}, 5.0\,\mathrm{kg}]$
Random base velocity range	$[-0.5\,\mathrm{m/s}, 0.5\,\mathrm{m/s}]$
Random base velocity interval range	$[10.0\,\mathrm{s}, 15.0\,\mathrm{s}]$
Velocity command range for ${}_{\mathcal{B}}\boldsymbol{v}^*_{xy}$	$[-1.0\,\mathrm{m/s}, 1.0\,\mathrm{m/s}]$
Velocity command range for ${}_{\mathcal{B}}\omega^*_z$	$[-1.0\,\mathrm{rad/s}, 1.0\,\mathrm{rad/s}]$
Velocity command resampling range	$[10.0\,\mathrm{s}, 15.0\,\mathrm{s}]$
Network architecture	
Actor MLP dimensions	$(128, 64, 32)$
Critic MLP dimensions	$(64, 64)$
Activation function	ELU