Towards Effective Sim-to-Real Policy Transfer via Sim-to-Sim Transfer on Quadruped Robots

***Abstract*—Reinforcement learning (RL) has enabled quadruped robots to acquire agile locomotion behaviors in simulation, but transferring these controllers across domains remains challenging due to discrepancies in dynamics, contact modeling, and actuation. While most prior work focuses on simulation-to-real (sim-to-real) transfer, simulation-to-simulation (sim-to-sim) transfer provides a controlled setting for isolating engine-level differences without the confounding factors of sensor noise or unmodeled real-world effects. In this work, we study the transfer of a PPO-trained locomotion policy for the ANYmal-C quadruped from IsaacLab (PhysX) to MuJoCo. We characterize how differences in constraint solvers, contact formulations, and actuator dynamics manifest during locomotion by analyzing discrepancies in base motion, joint trajectories, joint velocities, and contact forces over 25-second rollouts. Our results show that the controller transfers stably and produces qualitatively similar gaits in both simulators, yet exhibits systematic joint-space deviations and substantial differences in contact force distribution—particularly in hip flexion/extension joints and symmetric limb pairs. We open source our implementation and evaluation pipeline for future research in sim-to-sim and sim-to-real transfer.**

***Index Terms*—sim-to-real robotics, sim-to-sim robotics, transfer learning, reinforcement learning, quadrupeds**

I. Introduction

Reinforcement learning (RL) has enabled impressive locomotion behaviors in simulation. However, policies often fail to generalize from simulation to the real-world. A central challenge in robotics is the simulation-to-reality (sim-to-real) gap: the performance of a robot in simulation often fails to match its behavior in the physical world. This is due to a number of factors including inaccuracies in modeling contact dynamics, sensing, actuation, and environmental disturbances. Control policies learned in simulation are particularly sensitive to these discrepancies, since even a small difference in dynamics or timing can accumulate into large deviations in locomotion capability.

While sim-to-real transfer has received considerable study, a related and important problem is simulation-to-simulation (sim-to-sim) transfer, where a policy trained in one simulator is executed in another. Differences in physics engines responsible for the contact solvers, actuator models, integration schemes, and numerical tolerances, can lead to divergence in the policy behavior even when the underlying robot model is identical. Studying sim-to-sim transfer offers a controlled setting for diagnosing these differences and failure modes without the complexity and cost of hardware experiments.

In this paper, we study the transfer of a learned locomotion policy for the ANYmal-C quadruped, shown in Figure 1, from IsaacLab (based on the NVIDIA PhysX engine) to MuJoCo. This setting isolates the effects of engine-level physics discrepancies without the confounding factors of sensing noise or unmodeled real-world dynamics, providing a clean and reproducible benchmark for cross-simulator policy robustness. We first quantify the degradation in performance that occurs under direct transfer and then perform a systematic analysis over contact and actuator parameters.

Fig. 1. The AnyMal-C quadruped robot asset from Nvidia's IsaacSim/IsaacLab.

We make two primary contributions: (1) a detailed characterization of sim-to-sim transfer of a learned locomotion policy between IsaacLab and MuJoCo for the ANYmal-C quadruped, analyzing discrepancies in base motion, joint behavior, and contact forces; and (2) an open-source, end-to-end pipeline for training, transferring, and evaluating locomotion policies across the two physics engines.

II. Related Literature

Across both sim-to-sim and sim-to-real settings, prior work falls broadly into two families: (i) robust training, which seeks to make policies insensitive to model variations through randomization or improved simulation fidelity, and (ii) adaptation, which adjusts policies or models at deployment to compensate for residual mismatches. These approaches frame the landscape in which sim-to-sim transfer of quadruped locomotion policies can be analyzed and improved.

Robust training via randomization and improved simulation: Domain and dynamics randomization aim to make policies invariant (or at least robust) to a wide range of possible environment parameters by randomizing physical and perceptual properties during training. Dynamics randomization has been shown to produce policies that tolerate significant

mismatch between training and deployment dynamics and was demonstrated on robotic manipulation tasks and locomotion. (1) provide an extensive study of dynamics randomization for transferring control policies, and (2) demonstrate how system identification combined with randomization can enable agile quadruped locomotion to transfer from sim to hardware.

Tobin et al. (3) show that aggressive domain randomization on the visual channel, randomizing textures, lighting, distractors, and camera pose in a low-fidelity simulator, can yield object detectors that transfer directly to real RGB images with centimeter-level accuracy, enabling closed-loop grasping with no real-image pretraining. Their results support the broader view that training over a deliberately over-dispersed family of simulated environments can compensate for large appearance and modeling gaps, a principle we exploit in our use of domain randomization for training the low-level quadruped locomotion policy.

System identification and adaptive simulation tuning: Rather than relying solely on broad randomization, a complementary strategy is to estimate simulator parameters (sometimes called system identification or sysID) and adapt the simulation so that it better matches the target dynamics. Several works formulate closed-loop pipelines that use real (or target simulator) rollouts to adapt simulator parameter distributions or to select simulation instances that reduce the gap to the target domain. Such real-to-sim-to-real and adaptive randomization schemes frequently outperform naive randomization when a modest amount of target data is available (2; 4).

Residual and action-space adaptation methods: A large line of work studies augmenting classical controllers or pretrained policies with learned residuals that correct model mismatch. Residual methods are attractive for sim-to-sim or sim-to-real transfer because they are sample-efficient: a low-capacity residual model can be fit using a small calibration dataset in the target domain to correct systematic errors in forces/torques or action semantics (5). Representative robotics studies formalize residual reinforcement learning and demonstrate improved transfer in contact-rich tasks (5). In sim-to-sim transfer settings, where the policy's structure remains valid but joint-level biases and contact-phase discrepancies emerge, residual action learning may offer a lightweight mechanism for correcting systematic errors without retraining the main controller policy.

Meta-learning and fast adaptation: Model-Agnostic Meta-Learning (MAML) introduces a general and gradient-based approach for learning policies that can rapidly adapt to new tasks using only a small number of gradient updates and limited data (6). Rather than training a single robust policy, MAML trains an initialization of policy parameters that is explicitly optimized so that subsequent fine-tuning is sample-efficient across variations in task dynamics or objectives. In the context of simulation-to-simulation transfer, meta-learning offers a mechanism to adapt a locomotion controller to discrepancies introduced by varying actuator models, contact dynamics, or integrators, while requiring far fewer target-domain rollouts than training from scratch. Meta-learned policies substantially outperform conventional pretraining when adapting across related RL tasks, underscoring the relevance of meta-initialization approaches for cross-sim transfer of quadruped locomotion policies (6).

Positioning of sim-to-sim work: Although many of the techniques above were developed for sim-to-real transfer, the same mechanisms are relevant to sim-to-sim transfer (e.g., IsaacLab to MuJoCo) where the underlying robot model is the same but engine-level details (contact solvers, integrators, actuator models, numerical tolerances) may differ. While training with one simulator, evaluating in another, and then deploying the policy in the real world has been shown to work with humanoid robots (7), this approach still needs validation for quadrupeds and other mobile robots.

Studying sim-to-sim transfer isolates engine-level failure modes and enables controlled diagnostics and ablations (timestep, contact model, actuator differences) that are difficult to perform on hardware. Prior quadruped and locomotion works show the value of combined sysID + robust training pipelines; residual/adaptation methods and representation learning provide the leading methodological families to compare against when proposing new cross-sim adaptation schemes.

Helpful documentation: A variety of resources informed this project's approach to sim-to-sim transfer for quadruped locomotion. The IsaacLab Sim-to-Sim documentation (8) provides the core methodology for reproducing policies across different physics engines. The Train-a-Robot tutorials (9) in the IsaacLab documentation detail the training pipeline, parallel environment setup, and policy deployment workflow. NVIDIA's technical report (10) on Sim-to-Real transfer with the Spot quadruped offers practical insights into policy structure, curriculum design, and robustness considerations that are also relevant to sim-to-sim transfer. Finally, the comparative analysis of contact models in robotics (11) helps with understanding discrepancies between PhysX and MuJoCo in terms of how friction, contact, and normal forces influence locomotion. Collectively, these references shape the experiments and comparisons presented in this work.

III. Simulator Differences

A. Constraint Solvers in Robotics Simulation

Constraint solvers play a critical role in robotic simulation by enforcing the kinematic and dynamic relationships that govern a robot's motion, such as joint couplings, actuator limits, and constraints at contacts. These solvers ensure that simulated robots behave in accordance with physical laws despite the discretizations and numerical approximations inherent in real-time simulation. Constraint solvers strongly affect the stability of control policies, and an inaccurate constraint enforcement during training of the control policy could lead to unrealistic robot behaviors that do not transfer from simulation to the real-world.

Modern robotics simulators, including NVIDIA PhysX (used in Isaac Lab) and MuJoCo, adopt different constraint-solving paradigms that reflect a trade-off between compu-

tational efficiency and physical accuracy. PhysX employs iterative solvers such as the Projected or Temporal Gauss–Seidel methods (12; 13), which resolve constraints sequentially and are well-suited for massively parallel, GPU-accelerated simulation, albeit with limited precision in contact dynamics.

In contrast, MuJoCo formulates contact and joint constraints as a mixed linear complementarity problem (mLCP) (14; 15), solved using Newton-type methods that enforce all constraints simultaneously, resulting in smoother and more physically consistent behaviors at greater computational cost. Consequently, PhysX is typically preferred for large-scale reinforcement learning experiments requiring thousands of parallel environments, while MuJoCo is often chosen for high-fidelity analysis and control policy validation where accurate contact modeling is paramount (16; 17).

One of the most significant differences between IsaacLab (which utilizes NVIDIA's PhysX engine) and MuJoCo is their formulation of multi-body dynamcis. PhysX operates on maximal coordinates, where each link in a robot is treated as an independent rigid body with six degrees of freedom (DOF). Joints are enforced as algebraic constraints that remove DOFs between bodies, typically solved via Projected Gauss-Seidel (PGS) or Temporal Gauss-Seidel (TGS) iterative solvers (18). This approach allows for scalability in complex environments but introduces the potential for the robot's limbs to separate slightly at the joints, if constraints are momentarily violated under high-impulse conditions.

In contrast, MuJoCo utilizes generalized coordinates (also known as reduced coordinates), modeling the robot as a branching kinematic tree where the number of degrees of freedom exactly matches the number of generalized velocities (19). In this formulation, joint constraints are implicit in the equations of motion, rendering joint separation mathematically impossible. For a Reinforcement Learning (RL) policy transferring from IsaacLab to MuJoCo, this discrepancy manifests as a change in the effective stiffness of the system; a policy may exploit the subtle compliance and damping provided by PhysX's constraint relaxation, failing when applied to the theoretically rigid kinematic chains of MuJoCo (20).

B. Contact Modeling

The different handling of contact dynamics presents a potential reality gap in sim-to-sim transfer. IsaacLab's PhysX backend predominantly employs a hard contact model based on the Linear Complementarity Problem (LCP). This formulation enforces strict non-penetration constraints, causing contact forces to act discontinuously. This means instantly preventing overlap the moment surfaces touch (18).

Conversely, MuJoCo employs a soft contact model formulated as a convex optimization problem, where contacts are treated as constraints with regularized stiffness (21). This allows for physically plausible deformation at the contact surfaces, generating restoring forces proportional to the penetration depth. While this convexity ensures a unique inverse dynamics solution and improves solver stability during the random exploration phases of RL, it fundamentally alters the impulse response of the system. A policy trained in IsaacLab may learn to rely on the instantaneous, rigid reaction forces of the LCP solver for locomotion stability. When transferred to MuJoCo, the same control actions may induce unexpected oscillations or "sinking" behaviors due to the compliant nature of the soft contact constraints.

C. The Best of Both

To get the best of both worlds, we can approach the problem of training a control policy for the ANYmal-C by first using a massively parallelized GPU-enhanced Reinforcement Learning approach using IsaacLab. Then the policy can be deployed onto the robot in MuJoCo to analyze the physical stability of the policy. Iteration between IsaacLab policy training and MuJoCo deployment and evaluation provides fast training with physically accurate evaluations. This gives us the opportunity to test and refine the policy before a real-world deployment, minimizing the potential for damage to the real hardware or the surrounding environment.

IV. Problem Formulation

We formulate quadruped locomotion as a partially observable Markov decision process (POMDP). The full simulator state includes base pose and velocity, joint configurations, contact forces, frictional interactions, ground geometry, and physics engine parameters; however, the policy observes only a subset of this information. Following standard practice in legged locomotion RL, we train a policy using the observed state as if it were Markov, relying on the policy's internal representation to manage unobserved contact dynamics and other physics engine specific effects. In the sim-to-sim setting, these unobserved parameters differ across physics engines (e.g., contact solvers, actuator models, numerical integration tolerances) and are a primary source of policy transfer degradation.

Therefore, We approximate the solution as an Markov Decision Process (MDP) defined by the tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mathcal{D}),$$

where:

- **State (observation) vector** $\mathcal{S} \subseteq \mathbb{R}^{48}$ (for the flat environment) represents the observations of the robot at each timestep, including:
  - Base linear and angular velocities in the robot frame, $\mathbf{v}_{\text{lin}}, \mathbf{v}_{\text{ang}} \in \mathbb{R}^3$,
  - Projected gravity vector in the base frame,
  - Joint positions and velocities relative to default positions,
  - Command vector $\mathbf{c} \in \mathbb{R}^3$ (X/Y linear velocity and yaw rate),
  - Previous actions $\mathbf{a}_{t-1}$
- **Action vector** $\mathcal{A} \subseteq \mathbb{R}^{12}$ corresponds to the joint position targets relative to the default joint positions:

$$\mathbf{a}_t \in [-1, 1]^{12} \quad \text{(scaled by the action scale)}.$$

- **Transition dynamics** $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ are defined by the simulator in which the policy executes (IsaacLab during training; MuJoCo during evaluation), including rigid-body dynamics, contact, and friction, with randomized parameters such as base mass. In the sim-to-sim case, $\mathcal{P}$ differs across simulators.
- **Reward function** $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is composed of multiple terms:

$$r_t = \sum_i w_i \, r_i(\mathbf{s}_t, \mathbf{a}_t),$$

  where the terms include:
  - Linear velocity tracking in the XY plane (exponential),
  - Yaw rate tracking (exponential),
  - Z-velocity and angular velocity penalties,
  - Joint torque and acceleration penalties,
  - Action rate penalty,
  - Feet air-time reward (to encourage stepping/trotting rather than shuffling/skidding),
  - Undesired contact penalty,
  - Flat orientation penalty.
- **Termination conditions** $\mathcal{D}$ are defined by:
  - *Time-out:* episode length reaches the maximum allowed steps T_{max}.
  - *Fall / crash:* any contact force on the base link exceeds a threshold $F_{\text{base}} > 1.0$.
- **Discount factor** $\gamma \in [0, 1]$ is used for cumulative return calculation over the episode. This is set based on the parameters described in Section VI-A.

V. System Overview

A. Simulation Hardware and Software Setup

All training, simulation, and evaluation, is performed on a dedicated host machine with an RTX 4090 GPU with 32 GB of RAM. IsaacLab version 2.3.0 is used with IsaacSim version 5.1.0 for initial policy training. MuJoCo version 3.3.6 is used for policy evaluation in an alternative simulation environment. Data is collected from both simulation environments for comparison during policy evaluations.

B. Simulated Quadruped Model

The ANYmal-C quadruped robot, shown in Figure 1, is used for this sim-to-sim experiment and performance characterization. For IsaacLab, the robot is used in the standard configuration provided from Nvidia's website (22). For deployment in MuJoCo, the configuration used comes from the MuJoCo Managerie repository (23). These models are intended to match the real hardware specifications. Table I shows the joint configurations for the quadruped. The same joint naming convention is used for both simulators, though the order of the joints is different so we re-order those in our scripts to align them. MuJoCo groups the joints by leg (e.g. LF leg, RF leg) and IsaacSim orders them by type (e.g. all HAA, all HFE).

TABLE I
ANYmal-C Joint Names and Functions

Joint Name	Body / Leg	Function / Motion
LF_HAA	Left Front Hip	Hip Abduction / Adduction
LF_HFE	Left Front Thigh	Hip Flexion / Extension
LF_KFE	Left Front Shank	Knee Flexion / Extension
RF_HAA	Right Front Hip	Hip Abduction / Adduction
RF_HFE	Right Front Thigh	Hip Flexion / Extension
RF_KFE	Right Front Shank	Knee Flexion / Extension
LH_HAA	Left Hind Hip	Hip Abduction / Adduction
LH_HFE	Left Hind Thigh	Hip Flexion / Extension
LH_KFE	Left Hind Shank	Knee Flexion / Extension
RH_HAA	Right Hind Hip	Hip Abduction / Adduction
RH_HFE	Right Hind Thigh	Hip Flexion / Extension
RH_KFE	Right Hind Shank	Knee Flexion / Extension

VI. Solution Approach

To leverage the complementary strengths of modern physics simulators, our approach combines IsaacLab's massively parallel, GPU-accelerated reinforcement learning framework with MuJoCo's high-fidelity physics simulation. First, we train a control policy for the ANYmal C quadruped in IsaacLab, exploiting its ability to simulate thousands of environments concurrently to achieve rapid policy convergence. The trained policy is then deployed in MuJoCo to evaluate its physical realism, stability, and contact dynamics under a more accurate constraint formulation. By iteratively alternating between policy training in IsaacLab and validation in MuJoCo, the controller can be refined *before* real-world deployment, reducing the sim-to-real gap and improving overall policy robustness.

A. Training Configuration and Reinforcement Learning Setup

The locomotion policy for the ANYmal-C quadruped was trained using the `RSL-RL` (24) framework within IsaacLab. Training was based on the Proximal Policy Optimization (PPO) algorithm (25), which optimizes a clipped surrogate objective to ensure stable on-policy learning:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, and $\hat{A}_t$ is the generalized advantage estimate (GAE) computed using $\lambda = 0.95$ and $\gamma = 0.99$.

B. Training Environment and Parameters

Training is performed in IsaacLab using GPU-parallelized simulation with 4096 environments running concurrently. We used the flat terrain policy, `AnymalCFlatPPORunnerCfg`, which is optimized for smoother, planar surfaces with smaller networks ([128, 128, 128]) and reduced training iterations for faster convergence. See the full config in the Appendix 10.

Our configuration uses the following PPO hyperparameters:

- Number of environment steps per update: 24
- Number of learning epochs per iteration: 5
- Mini-batches per epoch: 4
- Learning rate: 1×10^{-3} (adaptive schedule)
- Entropy coefficient: 0.005

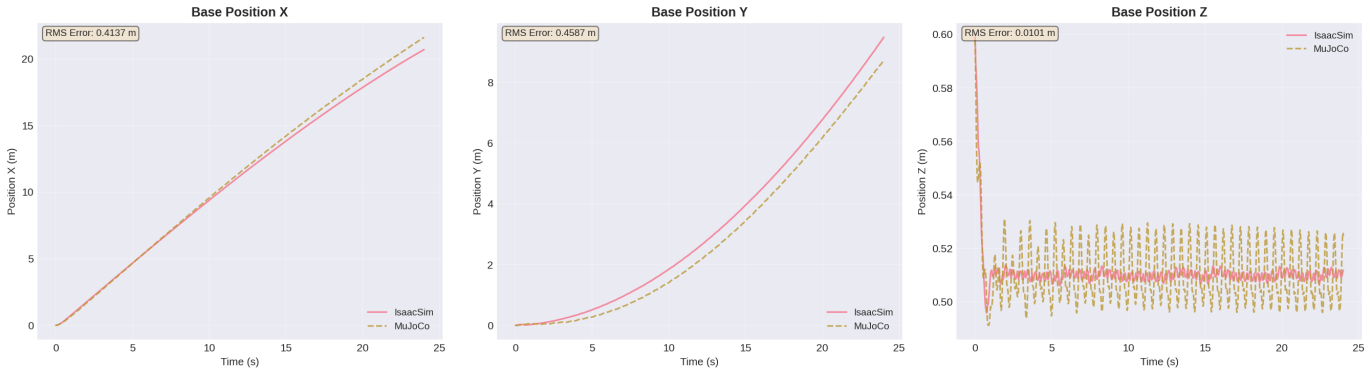

Fig. 2. Base position results for the simulation over 25 seconds. MuJoCo's results are dashed lines and IsaacLab's are straight lines. The drift between x-y directions is fairly comparable between simulators but the drift grows over time resulting in RMS errors of $\approx 0.41m$ for X and $\approx 0.46m$ for Y. The oscillation in the Z direction shows similar phase between the two simulators, with a small RMS error. However the amplitude is larger in MuJoCo than IsaacLab, perhaps due to less effective damping.

- Value loss coefficient: 1.0
- Clipping parameter: $\epsilon = 0.2$
- Maximum gradient norm: 1.0

Training was run for a maximum of 300 iterations for flat-ground tasks, saving checkpoints every 50 iterations. This took approximately 11 minutes. The actor and critic networks used the Exponential Linear Unit (ELU) activation and separate hidden-layer architectures. The model size is 40,844 parameters.

C. Policy Transfer Between Simulators

Our solution follows a sim-to-sim transfer pipeline between IsaacLab and MuJoCo for the ANYmal-C quadruped robot. To ensure cross-simulator compatibility, we apply domain adaptation steps including matching simulation frequencies (200 Hz physics, 50 Hz control), consistent gravity and integrator settings, and joint order remapping between the two models. The trained policy outputs joint position offsets relative to a nominal stance; these are reordered and directly applied to MuJoCo's position actuators. This setup enables us to evaluate the learned controller's physical stability and generalization under MuJoCo's distinct constraint solver and contact model, allowing for iteration between training in IsaacLab and testing in MuJoCo until satisfactory transfer performance is achieved.

The disparity between the constraint solver in PhysX and the actuator dynamics in MuJoCo required a modified control strategy. While the training environment used a nominal stiffness of $K_p = 40.0$, the explicit MuJoCo simulation required a higher stiffness ($K_p = 100.0$) to prevent catastrophic instability. To ensure stable deployment of the IsaacLab-trained policy in the MuJoCo environment, a first-order low-pass filter (exponential moving average) was applied to the policy's action outputs. While the policy was trained in IsaacLab to output raw joint position targets at 50 Hz, direct application of these high-frequency commands in MuJoCo resulted in significant instability. To mitigate this instability, the action commands were smoothed using a coefficient $\alpha = 0.2$ ($a_t = \alpha \cdot \pi(s_t) + (1 - \alpha) \cdot a_{t-1}$) which was determined empirically. This filtering served as a necessary domain adaptation step to bridge the gap between the training simulation from IsaacLab and the target deployment environment in MuJoCo.

VII. Results

We evaluate sim-to-sim policy transfer performance by comparing joint trajectories produced by the same reinforcement learning controller when executed in IsaacLab (PhysX) and MuJoCo. The analysis focuses on joint positions, velocities, and the corresponding tracking errors for all twelve actuated joints on the ANYmal-C quadruped: hip abduction/adduction (HAA), hip flexion/extension (HFE), and knee flexion/extension (KFE) for each leg (see Table I).

A. Base Position

Figure 2 shows the base position results over a 25 second simulation playback. The policy produces similar forward and lateral motion in the two simulators. The x and y positions track similarly wth a small drift beginning to accumulate over time resulting in RMS error of $\approx 0.41m$ in X and $\approx 0.46m$ in Y. In the Z-direction, the RMS error is minimal, $\approx 0.01m$. The oscillations from the gait are nearly in phase and similar in amplitude, though we see higher amplitudes in MuJoCo than IsaacSim/IsaacLab.

B. Contact Forces

Figure 3 shows the contact forces in the Z-direction for both simulators. MuJoCo reports contact forces in the -Z direction. To make for easier visual comparison, we take the absolute value of the MuJoCo contact forces so they can be plotted on the positive scale. Here the policy seems to transfer the timing of the contacts (e.g. stance and swing) quite well, but the load distribution appears to differ quite significantly. Given that the contact models are different between the two simulators, this is not surprising because differences in stiffness, damping, and friction, can translate to differences in the way the load shifts from one leg to another.

What does come as a surprise is the difference in RMS error between the symmetrical feet of the robot. The left

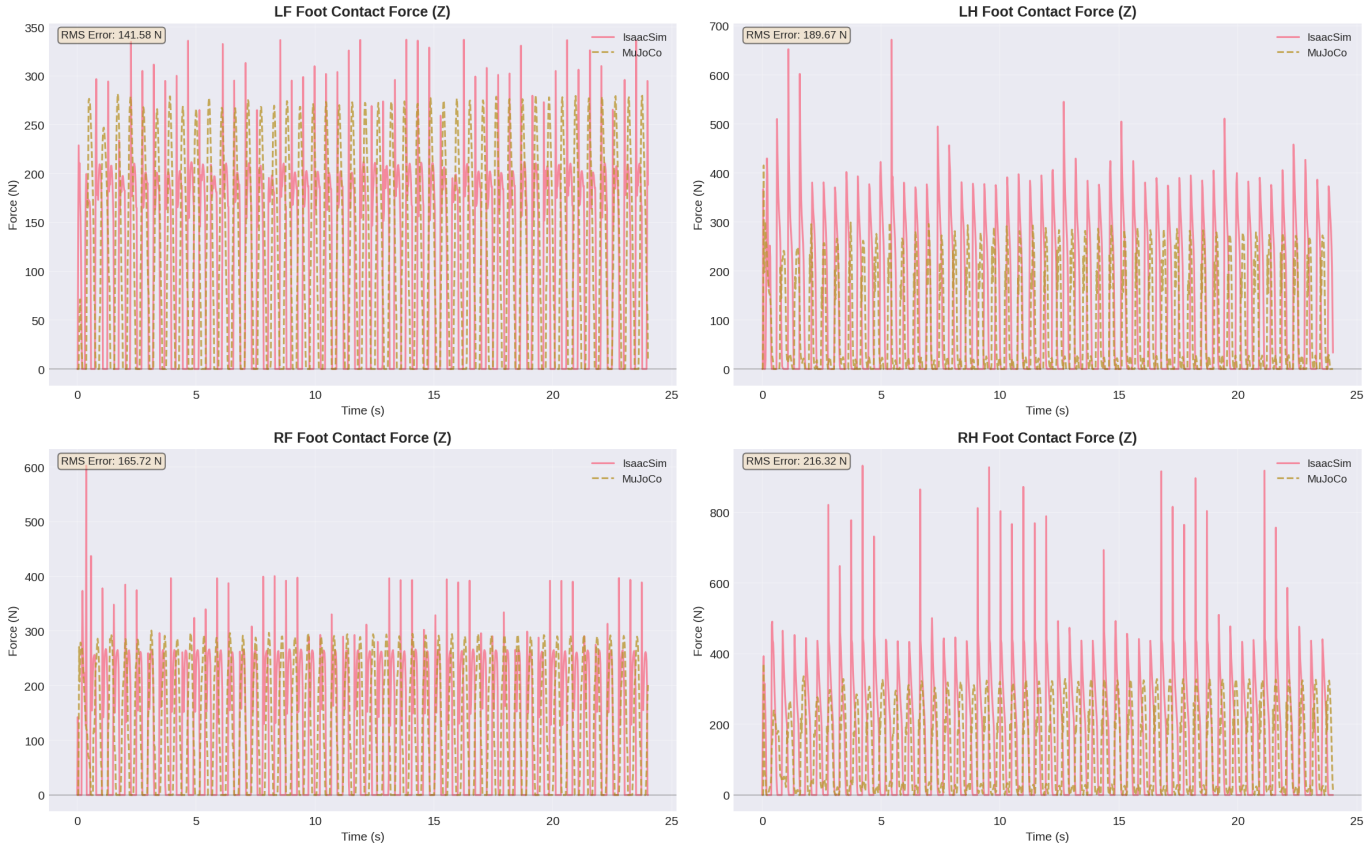

Fig. 3. Contact forces for all 4 feet in the Z-direction. +Z is up in both simulators, but the contact forces are computed differently. MuJoCo's contact forces are reported in the -Z direction and IsaacSim/IsaacLab's are reported in the +Z direction. We flip the sign of MuJoCo's contact forces to make this comparison visually easier to understand. The max forces possible on all feet are higher in IsaacSim/IsaacLab, sometimes by 100s of Newtons. Also of note is the asymmetry in contact force load between right and left front legs and right and left hind legs, indicating model discrepancies.

foot and right foot have differing RMS errors, and differeing peak contact force amplitudes even within the same simulator. For example, the IsaacSim/IsaacLab data shows periodic peak contact forces of 400N for the right front foot and only 340N peak periodic contact forces for the left front foot. Mujoco data shows a similar discrapancy between periodic peak contact forces between the left and right front legs, 300N to 275N respectively. We also see especially high error on the right hind leg (RH) which shows an RMS error $\approx 26N$ higher than the left hind leg (LH) indicating a potential asymmetry somewhere in the model. It is not clear whether this results from a robot model asymmetry, a contact force, or dynamics model difference.

C. Joint Positions

The left side of Figure 4 shows the aggregate joint position error distribution. Most of the joint position errors are within the range of -0.25 to 0.25 radians. The mean error is $\approx -0.011\ rad$, the standard deviation is $\approx 0.239\ rad (\approx 13.7\,\text{deg})$, and the median is $\approx -0.004 rad$. The error is mostly centered near zero but with long and heavy tails. On average, the policy reaches very similar configurations in both simulators but with non-negligible variability, occasionally producing large errors likely corresponding to specific joints or phases where the model mismatch is most influential.

The right side of Figure 4 shows the aggregate joint position errors by joint. Here we see that hip flexion and extension joints are the primary sources of configuration mismatch, while the hip abduction/adduction joints are comparatively well aligned. The knee joints fall somewhere in the middle of those two in terms of their comparative alignment. With the hip flexion and extension (HFE) joints, we see biases where the medians are significantly below zero and there are long negative tails. These joints strongly control the leg swing and stance angle which are impacted by ground contact and base pitch so this may be another impact of cross simulator discrepancies in contact force, friction, or dynamics models.

In Figure 5, the top plot shows the RMS joint position error over time and the bottom plot shows the maximum absolute joint position error over time. The RMS error over all joints stays in the $0.15 - 0.40$ rad range, with peaks at the start due to the initialization of the robot's position. The max absolute error at each time step ranges from about $0.2 - 1.3$ rad early on, but stabilizes after start-up below 1.0 rad. Both errors exhibit strong periodic structure correlated to the gait cycle of the quadruped. The stable periodic nature of the error indicates

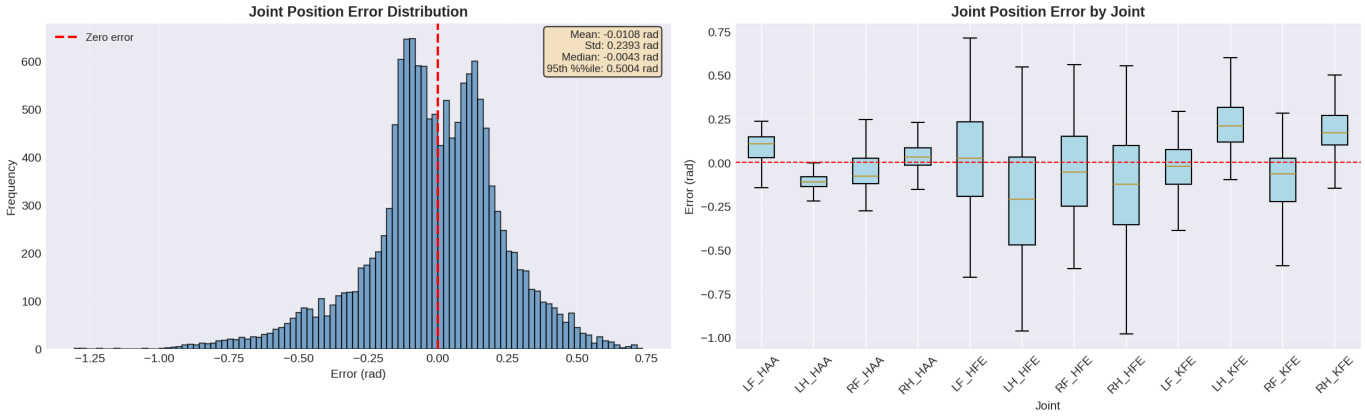

Fig. 4. Joint position error distribution (left) and the per joint position error (right). The largest variance in joint errors occurs for the hip flexion and extension joints. The knee flexion joints see less variance than the hip flexion joints, but still have higher errors and variance overall than the hip adduction/abduction joints.

that the cross simulation error is neither drifting, nor growing unbounded, and is likely tied just to the gait cycle of the quadruped.

The quantitative joint tracking errors are summarized in Figure 6. Across all twelve joints, the root-mean-square (RMS) position error ranges from 0.07 rad (best, RH_HAA) to 0.40 rad (worst, RH_HFE). Errors are smallest for the abduction/adduction joints, and largest for the flexion/extension joints, likely because they are heavily influenced by contact forces. These results suggest that policy transfer is highly sensitive to discrepancies in contact modeling.

Figure 7 shows joint position trajectories over a 25-second roll out in both simulators. The IsaacLab traces (solid lines) and MuJoCo traces (dashed lines) generally follow consistent periodic patterns, indicating that the control policy learned in IsaacSim/IsaacLab produces stable locomotion when transferred to MuJoCo without fine-tuning. However, systematic offsets are visible for several joints, particularly the hip and knee flexion joints, which suggests differences potentially in contact or dynamics models, or actuator behavior, between the two physics engines.

D. Joint Velocities

Joint velocity comparisons, shown in Figure 8, highlight the same periodic structure but with increased amplitude in MuJoCo, with the exception of the RH_KFE joint where IsaacLab/IsaacLab shows higher amplitudes. The velocity profiles track each well with peaks and zero-crossings lining up as well as similar amplitudes. Larger differences exist at startup likely due to initialization differences, but during steady gait the velocity profiles match quite well. The only exception is the LF and RF hip abduction/adduction joints which see significantly higher velocities in MuJoCo than in IsaacSim/IsaacLab. The RF_HFE and RF_KFE joints also see higher velocities in MuJoCo than IsaacLab/IsaacSim, suggesting the policy has asymmetrical performance when transferred from IsaacSim/IsaacLab to MuJoCo.

E. Results Summary

Overall, the transferred policy exhibits qualitatively correct gait cycles and stable locomotion in MuJoCo, despite moderate degradation in tracking precision. The joint-level deviations remain bounded and periodic, confirming that the control structure learned in IsaacSim/IsaacLab generalizes to a different physics solver without catastrophic failure. However, the low-level dynamics are not particularly well-matched as the contact forces differ substantially and larger joint space errors occur in the hip flexion and extension joints. These discrepancies are tolerated well enough for the robot to walk in both simulators, but how the robot walks in each can differ on a per joint basis.

VIII. Future Work

This work demonstrated a basic characterization of the locomotion policy transfer of a quadruped on flat ground. Characterizing the policy transfer on rough or uneven terrain could provide additional insights.

IX. Conclusion

This work demonstrates that sim-to-sim transfer of a PPO-trained ANYmal-C locomotion policy from IsaacLab to MuJoCo is feasible, but reveals meaningful discrepancies arising from differences in physics engine assumptions. Although the transferred controller maintains stable gait cycles and achieves qualitatively consistent locomotion, systematic variations occur in joint positions, joint velocities, and especially contact forces, with notable asymmetries across symmetric limbs. These findings highlight that even when robot models are matched, engine-level differences in contact solvers, actuator dynamics, and integration schemes significantly influence low-level behavior. Sim-to-sim transfer thus serves as a valuable diagnostic tool for understanding—and ultimately narrowing—the sim-to-real gap. Future work should investigate residual action correction, system identification for cross-sim alignment, domain adaptation strategies, and RL architectures explicitly designed to generalize across physics engines, with

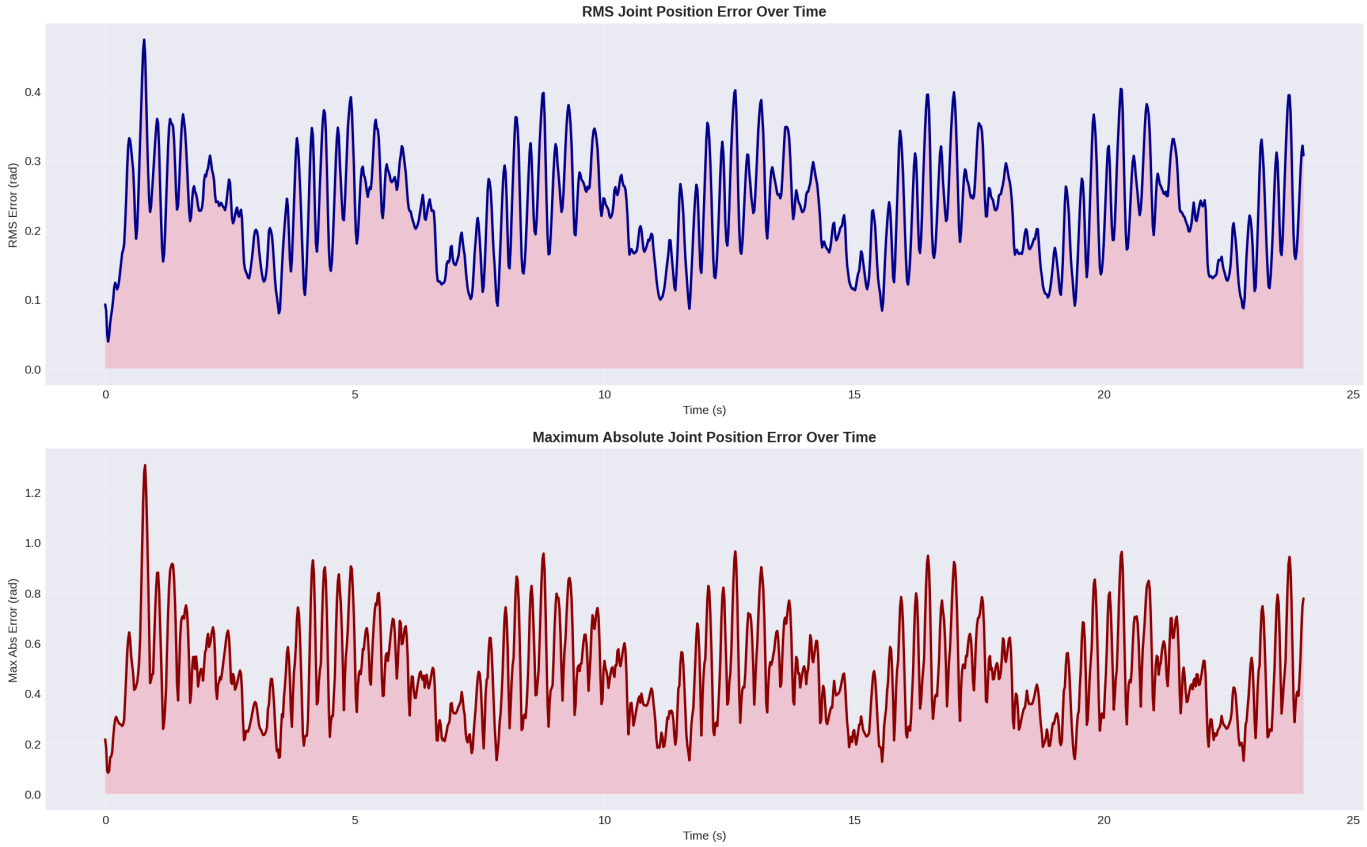

Fig. 5. RMS (top) and Max Absolute (bottom) errors for joint positions over 25 seconds of simulation. The RMS errors represents overall joint position disagreement between the two simulators at each moment in time. The periodic pattern tracks with the gait cycle. It is likely that the peaks are occurring when the leg swing occurs because there's less contact, and thus less constraint. The valleys, where the error is minimized, are likely due to the stanced phase where the robots feet are planted and thus more constrained. For the bottom plot of maximum absolute errors, we see the worst case error at different moments in time. Both forms of joint position error appear stable and do not appear to be growing over time.

the longer-term goal of improving the robustness of real-world quadruped locomotion.

Contributions and Release

The authors grant permission for this report to be posted publicly.

Acknowledgment

Thank you to the Collaborative AI and Robotics Lab at the University of Colorado Boulder for the generous donation of GPU resources for this project.

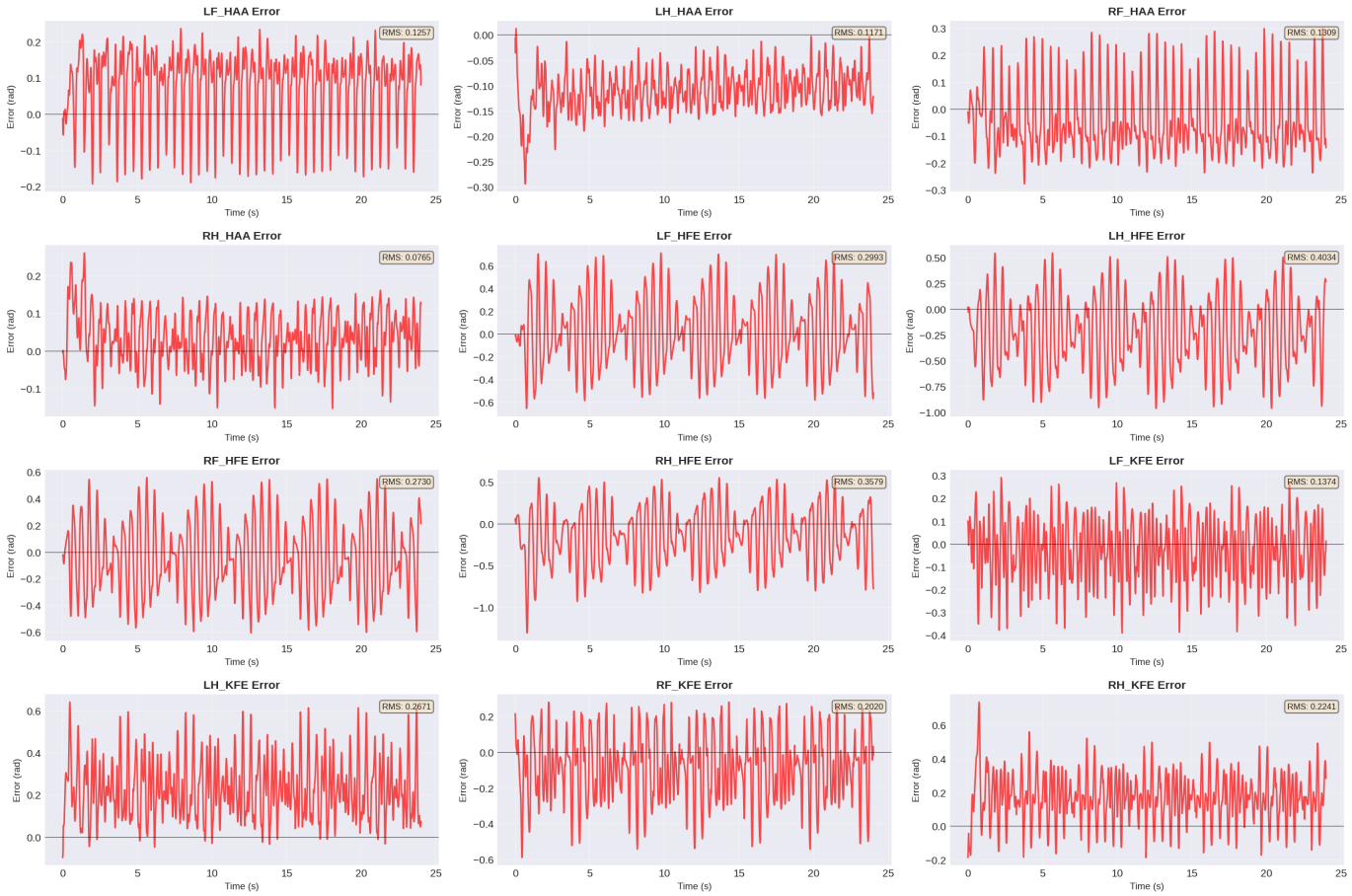

Fig. 6. RMS errors for all 12 joints over 25 seconds of simulation replay. IsaacSim/IsaacLab results are shown in pink (solid lines) and MuJoCo results are shown in brown (dashed lines).

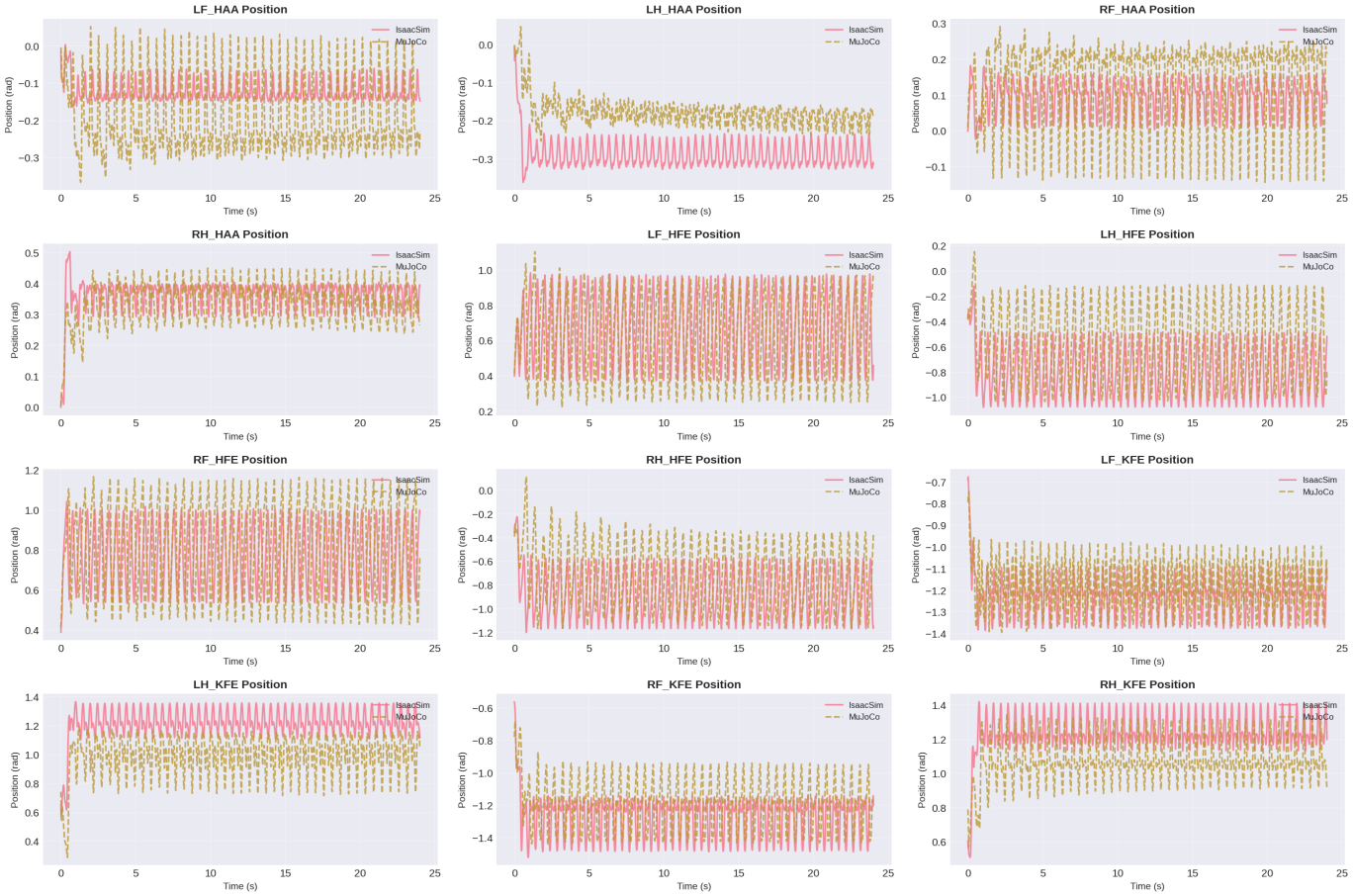

Fig. 7. Position for all 12 joints over 25 seconds of simulation replay. IsaacSim/IsaacLab results are shown in pink (solid lines) and MuJoCo results are shown in brown (dashed lines).

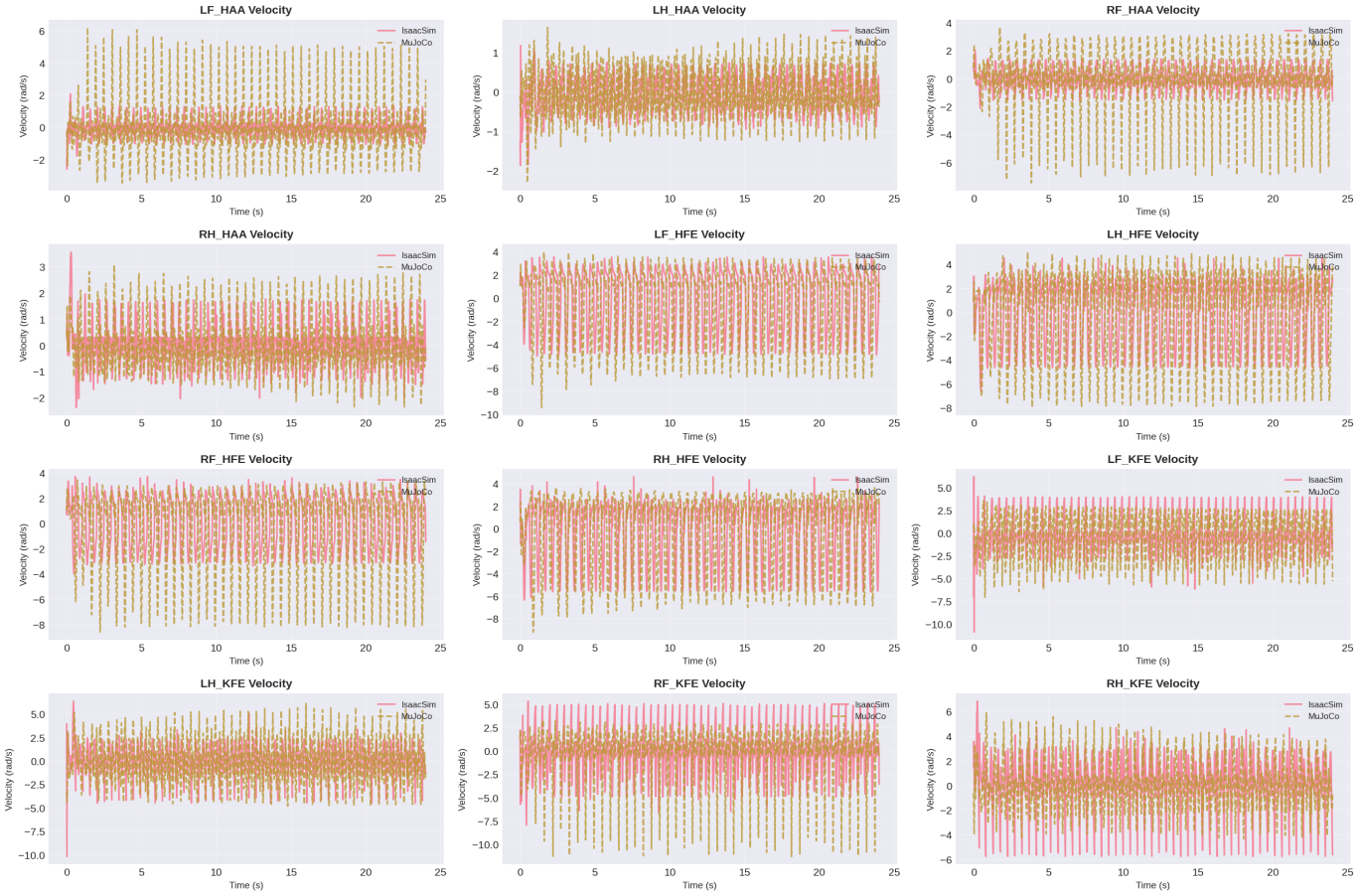

Fig. 8. Velocities for all 12 joints over 25 seconds of simulation replay. IsaacSim/IsaacLab results are shown in pink (solid lines) and MuJoCo results are shown in brown (dashed lines).

APPENDIX

```
@configclass
class AnymalCRoughPPORunnerCfg(RslRlOnPolicyRunnerCfg):
    num_steps_per_env = 24
    max_iterations = 1500
    save_interval = 50
    experiment_name = "anymal_c_rough"
    policy = RslRlPpoActorCriticCfg(
        init_noise_std=1.0,
        actor_obs_normalization=False,
        critic_obs_normalization=False,
        actor_hidden_dims=[512, 256, 128],
        critic_hidden_dims=[512, 256, 128],
        activation="elu",
    )
    algorithm = RslRlPpoAlgorithmCfg(
        value_loss_coef=1.0,
        use_clipped_value_loss=True,
        clip_param=0.2,
        entropy_coef=0.005,
        num_learning_epochs=5,
        num_mini_batches=4,
        learning_rate=1.0e-3,
        schedule="adaptive",
        gamma=0.99,
        lam=0.95,
        desired_kl=0.01,
        max_grad_norm=1.0,
    )

@configclass
class AnymalCFlatPPORunnerCfg(AnymalCRoughPPORunnerCfg):
    def __post_init__(self):
        super().__post_init__()

        self.max_iterations = 300
        self.experiment_name = "anymal_c_flat"
        self.policy.actor_hidden_dims = [128, 128, 128]
        self.policy.critic_hidden_dims = [128, 128, 128]

...

@configclass
class AnymalCFlatEnvCfg(AnymalCRoughEnvCfg):
    def __post_init__(self):
        # post init of parent
        super().__post_init__()

        # override rewards
        self.rewards.flat_orientation_l2.weight = -5.0
        self.rewards.dof_torques_l2.weight = -2.5e-5
        self.rewards.feet_air_time.weight = 0.5
        # change terrain to flat
        self.scene.terrain.terrain_type = "plane"
        self.scene.terrain.terrain_generator = None
        # no height scan
        self.scene.height_scanner = None
        self.observations.policy.height_scan = None
        # no terrain curriculum
        self.curriculum.terrain_levels = None
```

Fig. 9. Configuration for the ANYmal-C PPO policy.

```
# Copyright (c) 2021-2025, ETH Zurich and NVIDIA CORPORATION
# All rights reserved.
#
# SPDX-License-Identifier: BSD-3-Clause

from __future__ import annotations

import torch
import torch.nn as nn
from torch.distributions import Normal

from rsl_rl.networks import MLP, EmpiricalNormalization

class ActorCritic(nn.Module):
    is_recurrent = False

    def __init__(
        self,
        obs,
        obs_groups,
        num_actions,
        actor_obs_normalization=False,
        critic_obs_normalization=False,
        actor_hidden_dims=[256, 256, 256],
        critic_hidden_dims=[256, 256, 256],
        activation="elu",
        init_noise_std=1.0,
        noise_std_type: str = "scalar",
        **kwargs,
    ):
        if kwargs:
            print(
                "ActorCritic.__init__ got unexpected arguments, which will be ignored: "
                + str([key for key in kwargs.keys()])
            )
        super().__init__()

        # get the observation dimensions
        self.obs_groups = obs_groups
        num_actor_obs = 0
        for obs_group in obs_groups["policy"]:
            assert len(obs[obs_group].shape) == 2, "The ActorCritic module only supports 1D observations."
            num_actor_obs += obs[obs_group].shape[-1]
        num_critic_obs = 0
        for obs_group in obs_groups["critic"]:
            assert len(obs[obs_group].shape) == 2, "The ActorCritic module only supports 1D observations."
            num_critic_obs += obs[obs_group].shape[-1]

        # actor
        self.actor = MLP(num_actor_obs, num_actions, actor_hidden_dims, activation)
        # actor observation normalization
        self.actor_obs_normalization = actor_obs_normalization
        if actor_obs_normalization:
            self.actor_obs_normalizer = EmpiricalNormalization(num_actor_obs)
        else:
            self.actor_obs_normalizer = torch.nn.Identity()
        print(f"Actor MLP: {self.actor}")

        # critic
        self.critic = MLP(num_critic_obs, 1, critic_hidden_dims, activation)
        # critic observation normalization
        self.critic_obs_normalization = critic_obs_normalization
        if critic_obs_normalization:
            self.critic_obs_normalizer = EmpiricalNormalization(num_critic_obs)
        else:
            self.critic_obs_normalizer = torch.nn.Identity()
        print(f"Critic MLP: {self.critic}")

        # Action noise
        self.noise_std_type = noise_std_type
        if self.noise_std_type == "scalar":
            self.std = nn.Parameter(init_noise_std * torch.ones(num_actions))
        elif self.noise_std_type == "log":
            self.log_std = nn.Parameter(torch.log(init_noise_std * torch.ones(num_actions)))
        else:
            raise ValueError(f"Unknown standard deviation type: {self.noise_std_type}. Should be 'scalar' or 'log'")

        # Action distribution (populated in update_distribution)
        self.distribution = None
        # disable args validation for speedup
        Normal.set_default_validate_args(False)
...// code continues.
```

Fig. 10. Actor Critic definition for the ANYmal-C PPO policy.

References

[1] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[2] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems (RSS) XIV*, 2018.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[4] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[5] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6023–6029.

[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017, pp. 1126–1135.

[7] X. Gu, Y.-J. Wang, and J. Chen, "Humanoid-gym: Reinforcement learning for humanoid robot with zero-shot sim2real transfer," *arXiv preprint arXiv:2404.05695*, 2024.

[8] N. I. Team, "Sim2sim transfer in isaaclab," https://isaac-sim.github.io/IsaacLab/main/source/experimental-features/newton-physics-integration/sim-to-sim.html, 2024, accessed: 2025-01-29.

[9] ——, "Train a robot in isaaclab," https://isaac-sim.github.io/IsaacLab/main/source/setup/installation/pip_installation.html#train-a-robot, 2024, accessed: 2025-01-29.

[10] NVIDIA. (2024) Closing the sim-to-real gap: Training spot quadruped locomotion with nvidia isaac lab. Accessed: 2025-01-29. [Online]. Available: https://developer.nvidia.com/blog/closing-the-sim-to-real-gap-training-spot-quadruped-locomotion-with-nvidia-isaac-lab/

[11] G. Amador, S. Veer, O. S.-L. Ramos, and N. Rojas, "Contact models in robotics: A comparative analysis," *arXiv preprint arXiv:2304.06372*, 2023. [Online]. Available: https://arxiv.org/abs/2304.06372

[12] D. Baraff, "Fast contact force computation for nonpenetrating rigid bodies," in *SIGGRAPH '94*, 1994, pp. 23–34.

[13] M. Silcowitz, S. Niebe, and K. Erleben, "Projected gauss–seidel subspace minimization method for interactive rigid-body dynamics," in *GRAPP/VISIGRAPP 2010*, 2010.

[14] M. Anitescu and F. A. Potra, "Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems," *Nonlinear Dynamics*, vol. 14, no. 3, pp. 231–250, 1997.

[15] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction," *International Journal for Numerical Methods in Engineering*, vol. 39, pp. 2673–2691, 1996.

[16] NVIDIA Corporation, *NVIDIA Isaac Sim Physics and PhysX Documentation*, 2025, https://docs.isaacsim.omniverse.nvidia.com/latest/physics/.

[17] DeepMind Technologies, *MuJoCo Physics Engine Documentation*, 2024, https://mujoco.readthedocs.io/en/stable/.

[18] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

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

[20] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4397–4404.

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

[22] NVIDIA. (2025) Robot assets — isaac sim documentation. Accessed: Nov. 5, 2025. [Online]. Available: https://docs.isaacsim.omniverse.nvidia.com/5.1.0/assets/usd_assets_robots.html

[23] G. DeepMind. (2022) Mujoco menagerie: A collection of high-quality simulation models for mujoco. Accessed: Nov. 5, 2025. [Online]. Available: https://github.com/google-deepmind/mujoco_menagerie

[24] C. Schwarke, M. Mittal, N. Rudin, D. Hoeller, and M. Hutter, "Rsl-rl: A learning library for robotics research," *arXiv preprint arXiv:2509.10771*, 2025.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.