2026 IEEE International Conference on Robotics and Automation (ICRA)
June 1-5, 2026, Vienna, Austria
IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

A Lightweight Physics-Informed Neural Network for sim-to-real of biped robot

Yan Liu[1], Xizhe Zang[1], Xuehe Zhang[1], Chao Song[1], Boyang Chen[1], and Jie Zhao[1]

***Abstract*— n this paper, we present a low-cost, easy-to-implement sim-to-real framework for biped locomotion that narrows the reality gap using only simulation data, without motion-capture or additional real-world measurements.n this paper, we present a low-cost, easy-to-implement sim-to-real framework for biped locomotion that narrows the reality gap using only simulation data, without motion-capture or additional real-world measurements.I First, a walking policy for the BRUCE robot is trained in Isaac Gym via reinforcement learning. Next, we develop a compact, physics-informed neural network (PINN) grounded in Euler-Lagrange structure and augmented with an LSTM to predict simulator forward dynamics. Trained solely on simulation trajectories, the PINN forecasts next-step joint angles and velocities of the simulated robot given the physical robot's current state and control inputs. During hardware deployment, and consistent with a whole-body control architecture, these predicted states serve as reference joint states while the policy outputs provide feedforward torque commands; a feedforward-plus-feedback torque controller then computes the executed joint torques, thereby reducing the sim-to-real gap. Experiments on BRUCE demonstrate that our method better reproduces simulated behavior and attains higher tracking accuracy than direct policy transfer. Furthermore, the dynamics predictor runs at 1 kHz on embedded hardware, showing superior real-time performance relative to existing learning-based models.**

I. INTRODUCTION

Robust biped locomotion in human-centric environments holds great promise but remains challenging owing to the robot's complex dynamics and high number of degrees of freedom. Recently, deep reinforcement learning (DRL) has enabled the training of sophisticated walking policies in simulation, producing robust gaits across diverse terrains.

However, due to discrepancies between the CAD-exported robot description files (e.g., URDF) and the actual manufactured robot, along with the inability of simulators to precisely replicate real-world environments and robot dynamics, the mathematical model governing the behavior of the simulated robot (referred to as 'simulator robot dynamics') may differ in parameters from the model governing the physical robot (referred to as 'real robot dynamics'). This mismatch between simulated and real-world dynamics is a primary source of the sim-to-real gap, which can significantly hinder the transfer of policies to hardware.

This work was supported in part by the National Natural Science Foundation of China Integration Project under Grant 92048301 and in part by the Special Project on Humanoid Robots of the National Key Laboratory of Robot Technology and Systems.*(Corresponding authors: Yan Liu; Xizhe Zang; Xuehe Zhang.)*

[1]The authors are with the State Key Laboratory of Robotics and System, School of Mechatronics Engineering, Harbin Institute of Technology (HIT), Harbin 150001, China (e-mail: liuyan98@stu.hit.edu.cn; zangxizhe@hit.edu.cn; zhangxuehe@hit.edu.cn).

Contemporary methods [1], including domain randomization and domain adaptation, broaden the parameter distribution of the simulator robot dynamics by randomizing key quantities such as joint damping and surface friction, with the goal of covering the variability present in real robot dynamics and thereby improving policy generalization. While this approach increases robustness, it often yields conservative behaviors and raises training difficulty, which can lead to suboptimal policies. Alternative strategies, such as Sim-to-Real System Identification [2], attempt to explicitly align simulated and real dynamics, but they require accurate force/torque sensing and large amounts of real-world data, a requirement that is tractable for manipulators yet challenging for floating-base biped robots.

An alternative perspective is to enhance the control pipeline rather than the simulator itself. In this view, the controller learned by reinforcement learning is treated as a nominal controller, meaning it produces the desired behavior under the simulator robot dynamics but not necessarily under the real robot dynamics. From a control perspective, the behavior induced by the simulator and nominal controller is taken as a nominal trajectory for the physical system, and an outer-loop tracking controller is designed to make the real robot follow this nominal trajectory. By compensating for mismatches between simulator and reality in the control loop, this architecture can reduce the sim-to-real gap, but it requires a simulator robot dynamics capable of generating nominal trajectories.

Since the simulator robot dynamics map the current state and the nominal controller output to a nominal trajectory for the physical robot, a natural question is whether a simulator such as Isaac Gym can be used to compute this trajectory online. There are two practical obstacles: First, simulator cannot run at high frequency on embedded hardware due to computational constraints; second, injecting noisy real-world state measurements into a simulator can produce invalid contact configurations (for example, limb penetration into the ground or unintended floating contacts), which lead to erroneous state predictions. These limitations motivate learning the simulator robot dynamics via physics-informed or data-driven modeling for real-time reference generation.

Physics-informed neural network (PINNs) have made significant progress in robot dynamics modeling. Prior work has incorporated Hamiltonian and Lagrangian mechanics into neural network architectures [3], [4], producing models that generalize better than pure black-box networks and that respect fundamental physical structure. Building on this idea, [5] applied structured physics-informed models to control

of the inverted pendulum. This method provide a strong foundation for learning the simulator robot dynamics in this work.

In this paper, we address the sim-to-real gap for biped locomotion by developing a lightweight, real-time capable predictor of simulator robot dynamics and integrating it into a feedforward-plus-feedback control architecture. Our contributions are threefold:

1) We introduce a WBC-inspired transfer framework that integrates the simulator dynamics predictor and the learned RL policy within a single feedforward-plus-feedback control loop. This low-cost, easy-to-implement scheme operates solely on simulation data, requiring no motion-capture [6] or additional real-world measurements [7], and it demonstrably reduces the sim-to-real gap in dynamic locomotion.

2) We develop a compact neural network grounded in Euler–Lagrange structure and augmented with an LSTM, which serves as a partial whole-body dynamics model. Trained on simulation trajectories, the network predicts next-step joint angles and velocities directly, without explicit contact-force or floating-base state estimation, enabling fast inference while capturing the robot's core joint behavior.

3) We validate the proposed approach on the BRUCE biped robot and demonstrate that it reduces the sim-to-real gap in dynamic locomotion tasks. lightweight

II. RELATED WORK

A. Sim-to-Real

Several approaches have been proposed to bridge the sim-to-real gap. Common methods, e.g., domain randomization [8] and domain adaptation [9],

inject variability into simulator parameters so that learned policies generalize across a wider range of conditions; however, this strategy often yields overly conservative behaviors. More recently, [6] propose a "real-to-sim" network that trains the simulator to imitate motions produced by policies deployed on hardware. This reduces the gap without extensively modifying the simulator but relies on high-precision motion capture and requires three distinct training stages. Rather than altering the simulator, other works focus on real-world control: for example, [1] model the sim-to-real discrepancy as Gaussian noise and apply an online Kalman filter to mitigate it. Several promising approaches [7], [2] aim to improve real-world control or align hardware behavior with simulation; however, these methods tend to be complex, are often slow to run in real time, and achieving accurate synchronization is particularly challenging for bipedal robots.

B. Physics-Informed Neural Network

lightweight

Classical system identification [10] derives analytical models from limited data. However, as robotic systems become increasingly complex, obtaining accurate analytical models grows significantly more difficult. Purely data-driven methods treat the system as a black box [11] and demand large datasets, yet they often generalize poorly. Physics-informed neural networks (PINNs) bridge these approaches by embedding known physical laws into the network, thereby reducing data requirements and improving generalization while capturing complex dynamics. Subsequent work has extended this idea to mechanics and robotic systems [12]; variants such as energy-based and Lagrangian networks have been demonstrated on inverted pendulums [13]. Although these methods produce accurate, physically consistent predictors, their architectures tend to be large and rely on on-the-fly automatic differentiation, resulting in high computational cost that hinders real-time deployment on embedded hardware.

III. METHOD

In this section, we present our sim-to-real method, summarized in Fig. 1. First, a locomotion policy is trained in Isaac Gym via deep reinforcement learning. The trained policy is then executed in simulation to collect time-series trajectories of joint positions, joint velocities, and applied joint torques. Using these trajectories together with the Euler–Lagrange equations, we train a compact physics-informed neural network (PINN) that predicts the simulator's next-step joint states (positions and velocities) from the current state and control inputs.

During deployment, the trained PINN runs in parallel with the learned policy on the physical robot. At each control step the PINN supplies predicted next-step joint positions and velocities as reference targets for the joint-level PD controllers, while the policy provides feedforward joint torques. The PD corrective torques are combined with the policy's feedforward torques and commanded to the hardware. This simulator-informed feedback, together with learned feedforward control, helps the physical robot track its simulated trajectory more closely and thereby reduces the sim-to-real gap.

A. PINN-Based Robot Forward Dynamics

Policies trained in simulation assume that commanded torques will instantaneously produce the next-step joint angles and velocities predicted by the simulator. In reality, unmodeled motor dynamics and latency cause actual state transitions to deviate from their simulated counterparts, thereby widening the sim-to-real gap. If the real robot could track the simulator's next-step joint state—rather than merely applying the policy's torques—it would follow the simulated trajectory much more closely.

To enable this capability, we learn a forward-dynamics model of the simulator that maps current joint states and torques to next-step joint angles and velocities. Specifically, we design a compact physics-informed neural network (PINN) grounded in the robot's Euler–Lagrange equations and augmented with an LSTM module to capture temporal dependencies. The PINN is trained entirely on simulated torque–state trajectories collected from Isaac Gym, and runs in real time on embedded hardware. Once deployed, it predicts the simulator's next-step joint states, which serve as reference inputs to the joint PD controllers; the PD corrective torques are combined with the policy's feedforward torques

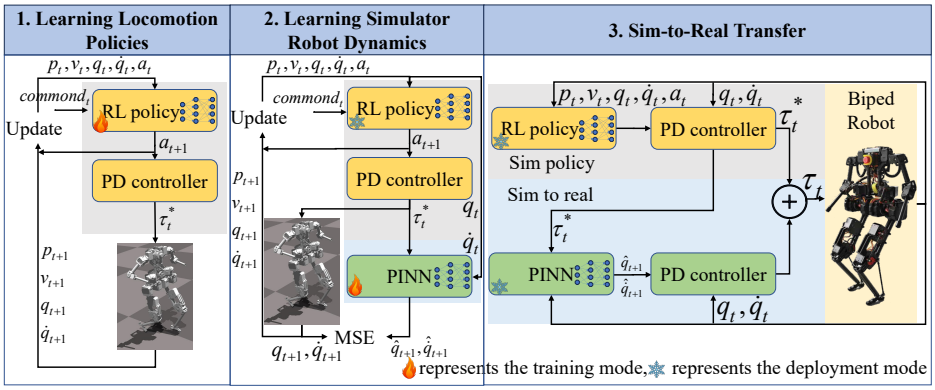


Fig. 1: Overview of the training and sim-to-real pipeline. First, a reinforcement-learning (RL) policy is trained in simulation and executed in Isaac Gym to collect state trajectories, including joint angles, joint velocities, applied joint torques, and resulting next-step states. These recorded trajectories are used to train the proposed physics-informed neural network (PINN) model of the simulator's forward dynamics. Finally, the learned gait policy is deployed on the real robot and the PINN-based model is used for reference states generation.

and commanded to the robot, thereby closing the sim-to-real loop at the control level.

To introduce our proposed dynamics predictor, we first briefly review the robot's dynamics model:

$$M_g\ddot{q}_g + C_g\dot{q}_g + G_g = S\tau + \sum_{i=1}^{m} J_i^T F_{ext,i}, \tag{1}$$

where $M_g \in \mathbb{R}^{(n+6)\times(n+6)}$ and $C_g \in \mathbb{R}^{(n+6)\times(n+6)}$ are the full-body inertia and Coriolis matrices, respectively, and $G_g \in \mathbb{R}^{n+6}$ is the gravity vector. The actuated-part selection matrix is $S = [0_{6\times n}; I_n] \in \mathbb{R}^{(n+6)\times n}$. The generalized coordinates are $q_g = [p^\top, q^\top]^\top$, where $p \in \mathbb{R}^7$ denotes the floating-base position and orientation (orientation represented as a quaternion) and $q \in \mathbb{R}^n$ collects the joint angles, hence $q_g \in \mathbb{R}^{n+7}$. The generalized velocity is $\dot{q}_g = [v^\top, \dot{q}^\top]^\top \in \mathbb{R}^{n+6}$, with $v \in \mathbb{R}^6$ the floating-base linear and angular velocity and $\dot{q} \in \mathbb{R}^n$ the joint velocities; $\ddot{q}_g$ denotes the generalized acceleration. Here $\tau \in \mathbb{R}^n$ denotes the actuation torques. For the i-th leg, $J_i \in \mathbb{R}^{6\times(n+6)}$ is the contact Jacobian and $F_{ext,i} \in \mathbb{R}^6$ is the corresponding ground reaction wrench, which comprises a three-dimensional force and a three-dimensional moment.

Since obtaining the floating-base pose, velocity, and contact forces is challenging in real-world settings, we restrict attention to the reduced joint-space dynamics:

$$M\ddot{q} + C\dot{q} + G = \tau + d, \tag{2}$$

where $q \in \mathbb{R}^n$ denotes the joint angles, and $M \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{n\times n}$, and $G \in \mathbb{R}^n$ are the joint-space inertia, Coriolis/centrifugal, and gravity terms. The vector $d \in \mathbb{R}^n$ captures disturbance terms in the joint accelerations that arise from unmodeled floating-base dynamics and contact interactions.

Note that Eq. 2 is not simply obtained by multiplying submatrices of Eq. 1; taking M as an example:

$$M_g\ddot{q}_g = \begin{bmatrix} M_b & M_c \\ M_c^T & M_j \end{bmatrix} \begin{bmatrix} \ddot{p} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} M_b\ddot{p} + M_c\ddot{q} \\ M_c^T\ddot{p} + M_j\ddot{q} \end{bmatrix} = \begin{bmatrix} * \\ M\ddot{q} \end{bmatrix}, \tag{3}$$

Hence M contains coupling terms associated with the floating-base acceleration. Accounting explicitly for the floating-base state requires modeling contact forces, which substantially complicates the overall modeling task. For humanoid robots, joint torques produce changes in joint states that in turn affect the floating-base state; therefore we posit the existence of an implicit mapping from joint states to the floating-base state, and we learn this mapping with a neural network in later sections. From this perspective, Eq. 2 may be regarded as a subsystem of a data-corrected version of the full dynamics.

The forward dynamics model maps the current joint states and applied torques to the joint states at the next timestep. By rearranging Eq. 2, the joint accelerations can be written as:

$$\ddot{q} = M^{-1}(\tau + f_n), \tag{4}$$

where $f_n = d - C\dot{q} - G$ is a lumped term that captures unmodeled dynamics and disturbances arising from the floating base, contact interactions, Coriolis/centrifugal effects, gravity modeling errors, and other unmodeled effects. In [14], the Coriolis-like terms and the potential-energy gradient are computed via automatic differentiation in PyTorch; while accurate, this incurs substantial computational overhead and can be prohibitive for real-time control. To mitigate this cost, we approximate f_n with a long short-term memory (LSTM) network.

Simultaneously, to avoid the costly inversion of the full inertia matrix M, we learn an explicit approximation of M^{-1} via a lightweight neural network. To ensure that the learned inverse is symmetric and positive-definite, we parameterize it following [12]:

$$M^{-1} = (L + b)^T(L + b), \tag{5}$$

where L is a learned lower-triangular matrix and b is a small diagonal bias that enforces positive definiteness [12]. Once $\ddot{q}$ is obtained from Eq. (4), the next joint positions and

velocities are computed using a residual Euler integration scheme:

$$\begin{aligned} q_{t+1} &= q_t + \dot{q}_t\,\Delta t + \delta q, \\ \dot{q}_{t+1} &= \dot{q}_t + \ddot{q}_t\,\Delta t. \end{aligned} \tag{6}$$

where Δt is the control timestep and δq is a learned correction term. Standard forward Euler is only first-order accurate and does not fully capture the true relationship between velocities and positions. Higher-order integrators such as RK4 [15] can reduce this error, but they require four evaluations of the dynamics model per timestep (each evaluation runs the neural network once), making them too costly for real-time control. To preserve a lightweight design, we use part of the LSTM's output as a corrective term for the Euler integrator, thereby achieving RK4-level accuracy with a single network evaluation per timestep.

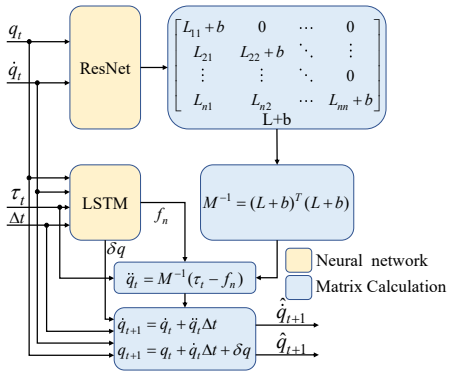


Fig. 2: The structure of PINN-Based robot forward dynamics

Fig. 2 depicts the detailed architecture of our forward-dynamics network. The model consists of two primary components: a ResNet module that learns an approximation of the inverse inertia matrix M^{-1}, and an LSTM module that models the nonlinear residual term f_n. Importantly, the network predicts only joint angles and velocities—omitting accelerations, which are difficult to measure accurately in both simulation and hardware— because our primary objective is to forecast the simulator's next-step states.

After specifying the architecture, we roll out the trained locomotion policy in Isaac Gym to collect time-series trajectories of joint positions, velocities, and torques. These trajectories form the training set. The network parameters are learned by minimizing the following loss:

$$L_o = \frac{1}{N}\sum_{k=1}^{N} \lambda_1 \left\| q_{t+1,k} - \hat{q}_{t+1,k} \right\|^2 + \lambda_2 \left\| \dot{q}_{t+1,k} - \hat{\dot{q}}_{t+1,k} \right\|^2 \tag{7}$$

where N is the number of training samples, $q_{t+1,k}$ and $\dot{q}_{t+1,k}$ are the k-th joint position and velocity sampled from the simulator at time $t+1$, and $\hat{q}_{t+1,k}$, $\hat{\dot{q}}_{t+1,k}$ are the corresponding PINN predictions. We optimize the network parameters using the AdamW optimizer.

B. Sim-to-Real Transfer

After training the dynamics predictor in simulation, we deploy it together with the learned walking policy on the real robot to reduce the sim-to-real gap. The biped walking policy—trained via reinforcement learning (Section IV)—runs at 100 Hz on the embedded controller and outputs next-step joint targets a_t. A high-frequency PD controller (1 kHz) then converts these targets into desired joint torques:

$$\tau_t^* = K_p(a_t - q_t) - K_d\dot{q}_t, \tag{8}$$

In simulation, τ_t^* is applied directly to each joint with zero delay, yielding perfect tracking of the learned gait. On hardware, however, actuator dynamics and timing latency prevent instantaneous torque realization, causing the robot's actual joint angles and velocities to deviate from their simulated counterparts.

To correct this mismatch, we run the dynamics predictor at 1 kHz on the same hardware to predict the simulator's next-step joint state $\hat{q}_{t+1}, \hat{\dot{q}}_{t+1}$. These predictions serve as reference inputs to the PD controller, yielding the augmented control law:

$$\tau_t = \tau_t^* + K_p(\hat{q}_{t+1} - q_t) + K_d(\hat{\dot{q}}_{t+1} - \dot{q}_t), \tag{9}$$

This formulation mirrors the structure of whole-body control (WBC) [16]; however, unlike WBC—which obtains $\hat{q}_{t+1}, \hat{\dot{q}}_{t+1}$ via inverse kinematics and inverse dynamics—our method produces these targets directly from the learned forward-dynamics model. By combining the policy-derived torques with simulator-informed feedback, the real robot is able to track its simulated trajectory more closely, thereby reducing the sim-to-real gap.

IV. EXPERIMENTS

We evaluate our approach in three stages. First, we train a walking policy for the BRUCE biped in Isaac Gym using deep reinforcement learning. Second, we run the trained policy in simulation to collect time-series joint positions, velocities, and applied torques; these trajectories form a dataset that we use to train and validate the PINN-based forward-dynamics model. Third, we deploy both the learned policy and the dynamics predictor on the physical BRUCE platform and conduct hardware experiments to evaluate the effectiveness of our sim-to-real transfer.

A. Biped Robot Walking Policy Training

In this subsection, we begin by training a walking policy in Isaac Gym for BRUCE [17] — a 0.7 m-tall, 4.5 kg biped with 16 degrees of freedom (10 in the legs, 5 per leg; 6 in the arms, 3 per arm). For this study we lock the arm joints and focus exclusively on legged locomotion.

The control problem is formulated as a partially observable Markov decision process [18] $\mathcal{M} = \langle S, A, T, O, R, \gamma \rangle$, where S denotes the full system state (e.g., joint angles, joint velocities, base pose, and commanded velocities), O the agent's observations (a subset of S available at runtime), A the action space (ten desired leg joint positions), T the simulator dynamics, R the reward function, and γ the discount factor. To exploit richer information during training, the critic network receives the full state S, while the actor operates on observations O, thereby matching the partial

observability that the real robot will encounter [19]. The components of the observation and state spaces are listed in Table I.

TABLE I: Overview of observation space and state space

Components	Dims	Observation	State
Clock Input $(\sin(t), \cos(t))$	2	✓	✓
Commands $(\dot{P}_{x,y,\gamma})$	3	✓	✓
Joint Position (θ)	10	✓	✓
Joint Velocity $(\dot{\theta})$	10	✓	✓
Linear Velocity $(\dot{P}^b_{xyz})$	3	✓	✓
Angular Velocity $(\dot{P}^b_{\alpha\beta\gamma})$	3	✓	✓
Euler Angle $(P^b_{\alpha\beta\gamma})$	3	✓	✓
Last Actions (a_{t-1})	10	✓	✓
Joint Torques	10		✓
Base Height $(\dot{P}^b_z)$	1		✓
Frictions	1		✓
Body Mass	1		✓
Push Force	2		✓
Push Torques	3		✓

In Table I, "Clock Input" denotes the periodic signal $[\sin(2\pi t/C_T), \cos(2\pi t/C_T)]$, where C_T is the gait period [20]. The robot's base pose is represented as $P^b = [x, y, z, \alpha, \beta, \gamma]$, with (x, y, z) the Cartesian coordinates of the base and (α, β, γ) its Euler angles. Finally, $\dot{P}_{x,y,\gamma}$ denotes the user-commanded x- and y-axis velocities and yaw rate.

Because we focus solely on leg motion, the action space is a 10-dimensional vector of desired leg joint angles. During training, the actor network outputs the means of ten Gaussian distributions (with variances adapted via a KL-divergence constraint [20]), and actions are sampled at 100 Hz. At deployment, the actor's output means are used directly. These desired angles are converted into feed-forward torques by the PD law in Eq. (8) and executed at 1 kHz on the embedded controller.

A critical component of our DRL setup is the reward function. To ensure that BRUCE learns a stable walking gait, we designed a composite reward composed of terms encouraging velocity tracking, upright posture maintenance, and energy efficiency, as detailed in Table II. Portions of the reward function in Table II are adapted from Table 4 of [20]. In particular, we define $\Phi(w, e) := \exp\big(-w\|e\|^2\big)$, and we track a desired swing-foot height $P^{\text{des}}_{f,z}$ computed as in Eq. (10) of [21]. The indices I_c and I_d denote the current and desired stance foot (left or right), respectively; I_d is determined according to Section III-B of [20]. Additional terms appearing in the reward are the stance-foot velocity v_{st}, the foot position P_f, the desired inter-foot distance D^{des}_f, the desired inter-knee distance D^{des}_k, and the yaw and roll joint angles $q_{\text{yaw},*}$ and $q_{\text{roll},*}$ for each leg.

Robot simulation must model both the robot dynamics and the environment. While we reduce the sim-to-real gap in the dynamics domain by learning a PINN, environmental variability is handled via domain randomization so the policy encounters a broader range of conditions. In our experiments we randomize surface friction coefficients, inject simulated sensor noise, and perturb neural-network inference delays to emulate perception/control latency. Specific parameter ranges and implementation details are provided in [20].

TABLE II: Reward definitions

Reward	**Definition**(r_i)	**weight**(μ_i)
Alive	if not reset 1 else 0	0.15
Swing feet height tracking	$\Phi(1000, P_{f,z} - P^{des}_{f,z})$	1
Feet contact patten	if $I_c == I_d$ 1 else -0.5	1
Feet slip	$\|v_{st}\|^2$	-0.05
Feet distance	$\Phi\big(10, \|P_{f,l} - P_{f,r}\| - D^{des}_f\big)$	1
Knee distance	$\Phi\big(10, \|P_{k,l} - P_{k,r}\| - D^{des}_k\big)$	1
Keeping yaw and roll joint position as default	$\big(\Phi(4, q_{yaw,l}) + \Phi(5, q_{roll,l}) + \Phi(4, q_{yaw,r}) + \Phi(5, q_{roll,r})\big)/4$	0.5
Linear velocity tracking	$\Phi(5, \dot{P}_{xy} - \dot{P}^{des}_{xy})$	2
Angular velocity tracking	$\Phi(7, \dot{P}_{\gamma} - \dot{P}^{des}_{\gamma})$	1
Velocity mismatch	$\big(\Phi(25, \dot{P}_z) + \Phi(2, \|\dot{P}_{\alpha\beta}\|)\big)/2$	0.5
Orientation tracking	$\Phi(5, P_{\alpha\beta})$	1
Base height tracking	$\Phi(5, P_z - 0.445)$	0.5
Base acceleration smoothness	$\Phi(1, \dot{P}_{xyz\alpha\beta\gamma} - \dot{P}^{last}_{xyz\alpha\beta\gamma})$	0.05
Action smoothness	$\|a_t - 2a_{t-1} + a_{t-2}\|_2$	-0.002
Torque,velocity smoothness	$\|\tau\|_2 + 10\|\dot{q}\|_2$	-5×10^{-6}
Joint acceleration smoothness	$\|(\dot{q} - \dot{q}^{last})/\Delta t\|_2$	-5×10^{-7}

The robot's velocity commands $\dot{P}_{x,y,\gamma}$ (forward x, lateral y, and yaw rate γ) were sampled uniformly from $(-0.3, 0.8) \times (-0.5, 0.5) \times (-0.5, 0.5)$. After 16 hours of training on an NVIDIA GeForce RTX 4060 GPU, the learned policy was deployed back into Isaac Gym. Fig. 3 plots the commanded versus actual forward velocity and yaw rate in simulation. Although the policy was trained only on forward velocity in $[-0.3, 0.8]$, it successfully tracks speeds from -0.5 to 1.0 m/s and yaw rates in $[-0.5, 0.5]$ rad/s, demonstrating clear generalization beyond its training range. Please see the supplementary video for detailed experimental records.[1].

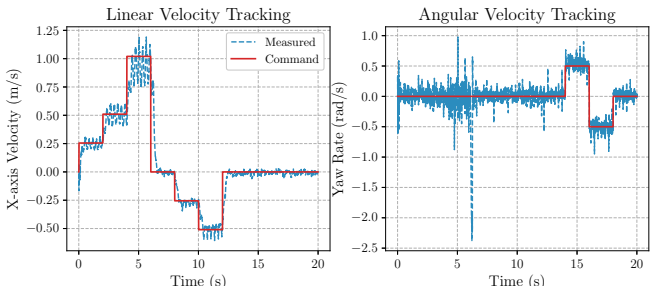


Fig. 3: Forward velocity and yaw rate trajectories in simulation.

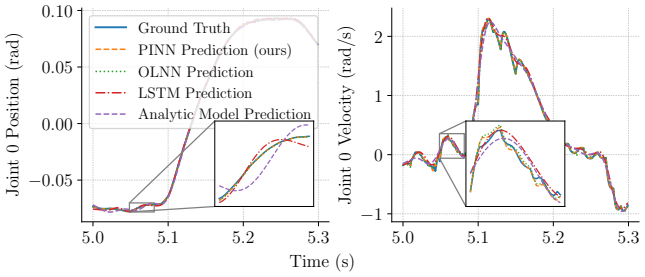
B. PINN-Based Simulation Dynamics

To train the forward-dynamics predictor, we first collect a dataset in Isaac Gym by sampling velocity commands $\dot{P}_{x,y,\gamma}$ uniformly from $(-0.5, 1.0) \times (-0.5, 0.5) \times (-0.5, 0.5)$ and running the learned walking policy for 90 s. Joint angles q, velocities $\dot{q}$, and torques τ are recorded at 1 kHz to form

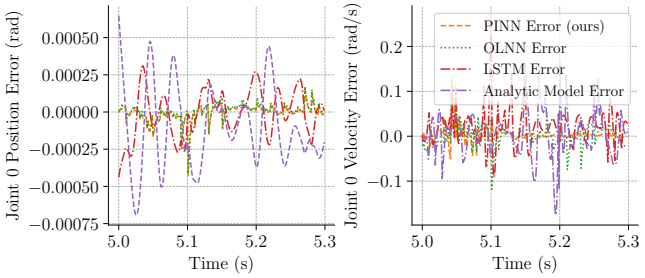
[1] https://youtu.be/q5G57TyjBPQ

the training set $\{(q_t, \dot{q}_t, \tau_t)\}_{t=0}^{T}$. We use 80% of this data for training and reserve 20% for validation.

The PINN architecture (Fig. 2) comprises a ResNet with layer widths $[20, 32, 48, 55]$ to approximate the inverse inertia matrix M^{-1}, followed by a three-layer LSTM (72 units per layer) to model the residual nonlinear term f_n. Each module is flanked by normalization and denormalization layers to improve training stability. We train for 1000 epochs using the AdamW optimizer (learning rate 10^{-2}, weight decay 10^{-2}), decaying the learning rate by a factor of 0.4 every 150 epochs. The bias term b in Eq.,(5) is fixed at 0.3 to ensure positive definiteness of the learned inverse inertia.



(a)



(b)

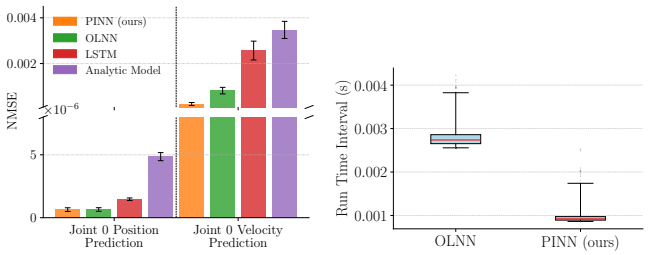
Fig. 4: (a) Position/velocity trajectories of Joint 0 and predicted values from relevant algorithms. (b) Prediction errors of joint position/velocity for the proposed algorithm versus baseline methods.

We randomly selected trajectories from the test set to validate our method and compared it to three baselines: OLNN [14], a purely data-driven LSTM dynamics model, and an analytic Euler-Lagrange model obtained via classical system identification [22]. The LSTM baseline denotes a forward-dynamics model implemented solely with an LSTM; the analytic model denotes a parameterized Euler-Lagrange model identified from data. The overall prediction results are summarized in Fig. 4.

Figs. 4a and 4b present, respectively, the predicted position and velocity trajectories of joint 0 produced by each method and the corresponding error curves relative to ground truth. The analytic model and the pure-LSTM baseline exhibit substantially larger prediction errors than the other methods. Two factors explain this outcome. First, the PINN injects Lagrangian structure as an inductive bias, which improves generalization from limited simulation data and reduces test error compared with an unconstrained LSTM. Second, while the analytic model exploits Lagrangian form, it represents quantities such as the inertia matrix with fixed symbolic parameters; by contrast, the PINN parameterizes these terms with neural networks, allowing data-driven corrections to the analytic form and greater expressive power to capture subtle effects. These two aspects together account for the PINN's lower prediction error relative to the LSTM and analytic baselines.

To quantify predictive performance we compute the normalized mean squared error (NMSE) [23] between predicted and ground-truth joint positions and velocities. Fig. 5a reports NMSE for joint 0 position and velocity; notably, the position NMSE of the PINN and OLNN are essentially equal. We emphasize that OLNN obtains next-step positions using an RK4 integrator, which requires four network evaluations per timestep, while our method reaches comparable predictive accuracy with a single network evaluation.

OLNN tightly integrates Lagrangian structure by differentiating a network-parameterized inertia matrix to obtain nonlinear dynamic terms, and this differentiation substantially increases per-step computation. To assess real-time suitability, we deployed both methods on the onboard computer and recorded single-step runtimes; the results are plotted in Fig. 5b. Our method completes a forward pass in under 1 ms on average, whereas OLNN requires approximately 2.8 ms per step. As discussed in Sec. III-B, the sim-to-real control pipeline requires the simulator dynamics surrogate to run at 1000 Hz; only the presented method meets this real-time requirement.



(a) (b)

Fig. 5: (a) NMSE of position/velocity predictions for Joint 0: proposed versus baseline algorithms. (b) Computational time per execution on an onboard computer: proposed method versus OLNN (red line indicates mean computational time).

C. Sim-to-Real Transfer

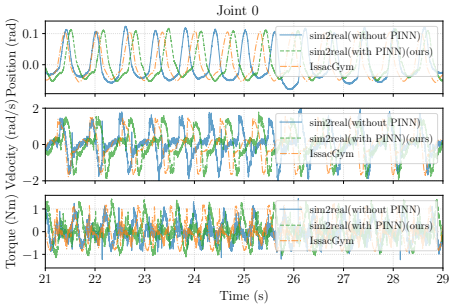
In this subsection we deploy both the learned walking policy and the trained simulation-dynamics predictor on the real robot to evaluate the sim-to-real transfer scheme introduced in Sec. III-B. To preserve consistency with the simulation, the joint-level PD gains are kept identical between simulation and hardware. This is done to avoid widening the sim-to-real gap.

Concretely, in Eq. 8 the ankle joints use $K_p = 2.5,\ K_d = 0.35$, while all other joints use $K_p = 20,\ K_d = 0.7$. In Eq.

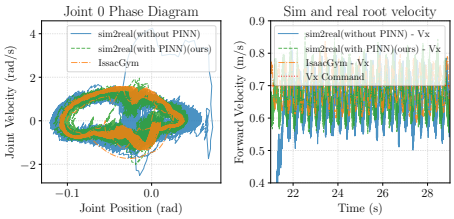
9, the ankle joints use $K_p = 10,\ K_d = 1$, and every other joint is controlled with $K_p = 60,\ K_d = 2$.

The selection of the lower-gain PD parameters in Eq. 8 follows the observation in [24] that low-gain controllers are more suitable for reinforcement learning training. By contrast, the higher gains used in Eq. 9 are chosen because the simulation-dynamics predictor operates at high frequency; larger gains enable rapid convergence to the predictor's desired states and improve high-frequency tracking performance.

1) Joint Trajectory Comparison: To quantify transfer fidelity, the robot was commanded to move forward at 0.4 m/s. We compared joint 0's angle, velocity, and torque trajectories under three conditions: pure simulation, naive sim-to-real transfer without a PINN, and our transfer method that employs a PINN; the results are shown in Fig. 6a. The plots demonstrate that our method yields joint trajectories substantially closer to the simulated reference than does naive transfer. Consistently, the phase-space trajectories in Fig. 6b (left) indicate that our scheme better preserves the simulated limit cycle, and the velocity profiles in Fig. 6b (right) show that, while both transfer methods reach the target speed of 0.4 m/s, our approach more closely matches the simulated speed profile.



(a) Joint 0's angle, angular velocity, and torque



(b) Phase trajectory of joint 0 and forward velocity of BRUCE

Fig. 6: (a) Joint 0's angle, angular velocity, and torque under simulation, direct sim-to-real, and PINN-based sim-to-real methods. (b) Phase trajectory of joint 0 and forward velocity of BRUCE under simulation, direct sim-to-real, and PINN-based sim-to-real methods.

2) Walking Trajectory Comparison: To assess sim-to-real transfer performance on hardware, BRUCE was commanded to walk with a forward speed of 0.4 m/s and a turning rate of 0.4 rad/s, which ideally produces a circular trajectory. Fig. 7 compares the actual paths produced by the two sim-to-real methods. Although disturbances and sensor noise cause deviations from the ideal circle, the trajectory produced by our method remains substantially closer to the desired circular path than that resulting from direct sim-to-real transfer.

Fig. 7: Tracking performance of BRUCE following a circular reference trajectory under two different sim-to-real transfer methods.

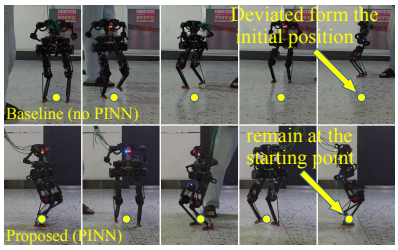


Fig. 8: In-place turning performance of the robot under two different sim-to-real methods.

3) Maximum-Speed Tracking Comparison: To further evaluate the sim-to-real transfer schemes, BRUCE was tested at its maximum reverse speed, -0.5 m/s, and at its maximum in-place turning rate, 0.5 rad/s. Fig. 8 and 9 illustrate BRUCE's motion under the two transfer schemes at these extreme command values. Fig. 10 presents a more detailed comparison for the in-place turning task: the left panel shows the robot trajectories, and the right panel shows the measured backward speed traces for the two deployment methods.

From these results we observe a clear difference in robustness. Under direct deployment, the robot steadily drifts away from the origin during in-place turning and is unable to maintain stable backward locomotion at -0.5 m/s. In contrast, the proposed deployment method keeps the robot closer to the origin during turning and sustains the commanded reverse speed without failure, demonstrating superior performance in these extreme conditions.

V. CONCLUSIONS

In this work, we propose a sim-to-real transfer method for the BRUCE robot. First, a robust walking policy is trained in simulation via reinforcement learning. To reduce the sim-to-real gap, we introduce a lightweight physics-informed neural network (PINN) that learns the simulator's forward dynamics from simulated trajectories. The PINN's predicted simulator trajectories are used as reference targets, and the learned policy provides nominal feed-forward control. An auxiliary feedback controller is then designed to make the physical robot track these PINN-based references. This control-centric workflow requires no real-world data collection and offers a

Fig. 9: Stability comparison of the robot at maximum reverse speed under two sim-to-real transfer methods.

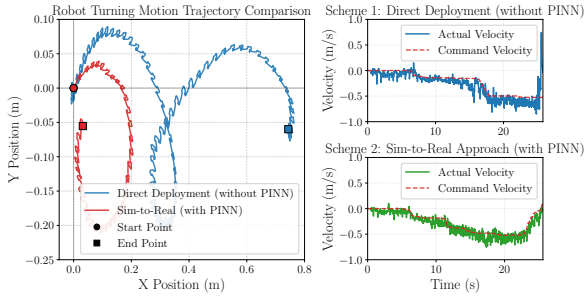


Fig. 10: (Left) Stationary turning trajectories of the robot under both the direct deployment and our proposed sim-to-real schemes. (Top Right) Maximum backward velocity test results under the direct deployment scheme. (Bottom Right) Maximum backward velocity test results under our proposed sim-to-real scheme.

practical alternative to dynamics randomization for improving sim-to-real transfer.

The PINN is computationally efficient and lightweight, satisfying onboard real-time control requirements. Our fully simulation-based pipeline shows that combining structured, data-driven dynamics modeling with policy learning can enable rapid, low-cost sim-to-real transfer for legged robots. We did not evaluate multi-terrain scenarios because flat-ground experiments suffice to validate that our method reduces the dynamics gap between simulation and reality. Nevertheless, a separate environmental gap—such as surface friction, ground compliance—remains and can further degrade transfer performance. In future work we will develop control and learning strategies that explicitly model or adapt to these environment-induced discrepancies.

ACKNOWLEDGMENT

REFERENCES

[1] Q. Dong, P. Zeng, G. Wan, Y. He, and X. Dong, "Kalman filter-based one-shot sim-to-real transfer learning," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 311–318, 2023.

[2] X. Zhang, S. Liu, P. Huang, W. J. Han, Y. Lyu, M. Xu, and D. Zhao, "Dynamics as prompts: In-context learning for sim-to-real system identifications," *IEEE Robotics and Automation Letters*, 2025.

[3] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[4] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," *arXiv preprint arXiv:2003.04630*, 2020.

[5] J. K. Gupta, K. Menda, Z. Manchester, and M. Kochenderfer, "Structured mechanical models for robot learning and control," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 328–337.

[6] T. He, J. Gao, W. Xiao, Y. Zhang, Z. Wang, J. Wang, Z. Luo, G. He, N. Sobanbab, C. Pan, *et al.*, "Asap: Aligning simulation and real-world physics for learning agile humanoid whole-body skills," *arXiv preprint arXiv:2502.01143*, 2025.

[7] A. Wagenmaker, K. Huang, L. Ke, K. Jamieson, and A. Gupta, "Overcoming the sim-to-real gap: Leveraging simulation to learn to explore for real-world rl," *Advances in Neural Information Processing Systems*, vol. 37, pp. 78 715–78 765, 2024.

[8] 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)*. IEEE, 2018, pp. 3803–3810.

[9] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

[10] L. Ljung, "System identification," in *Signal analysis and prediction*. Springer, 1998, pp. 163–173.

[11] G. Wang, Q.-S. Jia, J. Qiao, J. Bi, and M. Zhou, "Deep learning-based model predictive control for continuous stirred-tank reactor system," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3643–3652, 2020.

[12] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," *arXiv preprint arXiv:1907.04490*, 2019.

[13] M. Lutter, K. Listmann, and J. Peters, "Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7718–7725.

[14] J. K. Gupta, K. Menda, Z. Manchester, and M. J. Kochenderfer, "A general framework for structured learning of mechanical systems," *arXiv preprint arXiv:1902.08705*, 2019.

[15] M. Lutter and J. Peters, "Combining physics and deep learning to learn continuous-time dynamics models," *The International Journal of Robotics Research*, vol. 42, no. 3, pp. 83–107, 2023.

[16] D. Kim, S. J. Jorgensen, J. Lee, J. Ahn, J. Luo, and L. Sentis, "Dynamic locomotion for passive-ankle biped robots and humanoids using whole-body locomotion control," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 936–956, 2020.

[17] Y. Liu, J. Shen, J. Zhang, X. Zhang, T. Zhu, and D. Hong, "Design and control of a miniature bipedal robot with proprioceptive actuation for dynamic behaviors," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8547–8553.

[18] M. T. Spaan, "Partially observable markov decision processes," in *Reinforcement learning: State-of-the-art*. Springer, 2012, pp. 387–414.

[19] I. M. A. Nahrendra, B. Yu, and H. Myung, "Dreamwaq: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5078–5084.

[20] 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.

[21] Q. Wu, C. Zhang, and Y. Liu, "Custom sine waves are enough for imitation learning of bipedal gaits with different styles," in *2022 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 2022, pp. 499–505.

[22] C. Gaz, M. Cognetti, A. Oliva, P. R. Giordano, and A. De Luca, "Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4147–4154, 2019.

[23] R. Chaves, J. Ramírez, J. Górriz, M. López, D. Salas-Gonzalez, I. Alvarez, and F. Segovia, "Svm-based computer-aided diagnosis of the alzheimer's disease using t-test nmse feature selection with feature correlation weighting," *Neuroscience letters*, vol. 461, no. 3, pp. 293–297, 2009.

[24] D. Kim, G. Berseth, M. Schwartz, and J. Park, "Torque-based deep reinforcement learning for task-and-robot agnostic learning on bipedal robots using sim-to-real transfer," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6251–6258, 2023.