Learning to Walk: Using Reinforcement Learning for Humanoid Robot Mobility

zijie zhu* and xi zhang*,

***Abstract*—This project focuses on the development of bipedal walking capabilities for humanoid robots using reinforcement learning (RL). The primary objective is to enhance the autonomy and adaptability of humanoid robots for real-world applications such as assistance, disaster response, and autonomous service robots. A key challenge addressed in this work is the "sim-to-real" transfer, where learned behaviors in simulated environments are transferred to real-world scenarios. The project leverages RL techniques, for example, Proximal Policy Optimization (PPO), to teach the robot how to walk, turn, and avoid obstacles in diverse environments, including varying terrains and under external disturbances. The system is initially developed in simulation environments (Isaac Gym and gazebo) and then transferred to a physical robot, Dora2, for real-world testing. Through this research, the project aims to bridge the gap between simulation and real-world deployment, enabling humanoid robots to operate autonomously in complex, dynamic environments.**

I. Introduction

This project focuses on humanoid robots, particularly on the development of bipedal walking capabilities using reinforcement learning (RL) techniques. The primary motivation behind this research is to advance the field of robotics by enabling humanoid robots to walk autonomously in a human-like manner, which has significant implications for real-world applications such as assistance robots, disaster response, and autonomous service robots.

One of the key challenges in this domain is ensuring effective transfer of learned behaviors from simulated environments to the real world, a problem often referred to as "sim-to-real" transfer. In this project, we investigate the use of reinforcement learning to teach the robot bipedal locomotion, with an emphasis on improving the transferability of the learned policies from simulations to real-world scenarios. Achieving successful simulation-to-reality transfer will bring us closer to developing practical humanoid robots capable of operating autonomously in complex environments.

Another critical challenge is enabling the robot to navigate in various outdoor terrains, endure external disturbances, carry payloads, and adjust its gait based on different conditions. This requires the robot to not only learn how to walk but also how to adapt dynamically to changing environments and tasks, making it more versatile and resilient in real-world applications.

The motivation for this research stems from the need for humanoid robots that can operate autonomously in complex, dynamic environments. Current robots, while effective in controlled settings, often struggle to handle the variability and unpredictability of real-world conditions. This gap in robotic capabilities highlights the importance of developing adaptive, learning-based systems. By using reinforcement learning, this research aims to create robots that can continuously improve and optimize their behaviors, enabling them to better navigate diverse terrains, handle external disturbances, and adjust to changing environmental conditions. Ultimately, this work will help pave the way for more capable, autonomous humanoid robots that can perform a wide range of tasks in real-world scenarios.

Identify applicable funding agency here. If none, delete this.

II. Related works

Jonah Siekmann [9] introduces a novel probabilistic reward framework for reinforcement learning that enables a bipedal robot to learn a full spectrum of locomotion gaits.Following his research,Helei Duan [2] propose a fully-learned system that enables bipedal robots to react to local terrain while maintaining commanded travel speed and direction through sim-to-real learning for vision-based bipedal locomotion over challenging terrains.And Xinyang Gu [3] proposed an open source reinforcement learning framework designed to train the motor skills of humanoid robots, emphasizing zero-shot transfer from simulation to real environments.Jonah Siekmann [1] also demonstrates how Deep Reinforcement Learning (DRL) and Curriculum Learning (CL) can be employed to train a hexapod robot in simulation, improving its walking and obstacle avoidance performance when transferred to real-world environments.In a similar vein, T. Huang [5] explores the application of DRL for training a bipedal robot to develop stable walking patterns in simulations, with the aim of transferring these behaviors to real-world hardware.Furthermore, V. Gupta [4] presents a framework for robotic manipulation using DRL, focusing on asynchronous policy updates to help robots efficiently learn complex tasks from high-dimensional sensory inputs like images.

Ilija Radosavovic [7] propose a novel learning-based approach for controlling humanoid robots, focusing on real-world locomotion. The controller utilizes a causal transformer model that takes the history of proprioceptive observations and actions as input, predicting the next action. This model is trained through large-scale reinforcement learning in a simulated environment with randomized parameters and then deployed to a real-world humanoid robot zero-shot. The key hypothesis is that the observation-action history implicitly

encodes information about the environment, allowing the transformer to adapt its behavior in context without requiring weight updates. Experiments demonstrate the controller's ability to navigate diverse outdoor terrains, withstand external disturbances, carry payloads, and adapt its gait to varying conditions. Notably, the controller exhibits emergent behaviors like arm swinging and recovery from foot trapping, even though these were not explicitly programmed. The researchers highlight the controller's robustness and adaptability, comparing it favorably to a state-of-the-art model-based controller. They conduct extensive experiments in both simulated and real-world settings, including tests on slopes, rough terrain, and under external forces. Ablation studies demonstrate the advantages of the transformer architecture over other neural networks, the benefits of a larger context window for the transformer, and the effectiveness of joint training with imitation learning and reinforcement learning. Although acknowledging limitations such as slight movement asymmetry and imperfect velocity tracking, the authors suggest that this scalable learning-based approach holds promise for advancing real-world humanoid locomotion and encourage future research exploring the incorporation of additional input modalities.

Jonah Siekmann [10] addresses the simulation-to-real challenge, where simulation-trained policies often fail in real-world applications due to discrepancies between simulated and real-world environments. To overcome this, the authors propose combining deep-reinforcement learning (DRL) with curriculum learning (CL). The robot is trained in a simulated environment, with the task complexity gradually increasing, which allows the robot to first learn basic behaviors before progressing to more complex tasks. This approach significantly enhances the robot's real-world stability and performance in tasks like walking, turning, and obstacle avoidance. The study demonstrates that the learned policies can be transferred to the real robot, mitigating issues like sensor noise and unmodeled dynamics. The paper highlights how CL can prepare robots for real-world environments, providing a robust framework for real-world robotic deployment.

There are many ros packages available for our project.The tf package in ROS is a vital tool for managing coordinate transformations in robotic systems. It allows robots to understand and manage spatial relationships between different frames of reference, such as sensors, actuators, and the environment. This is essential for tasks like localization, sensor fusion, and motion planning. Key components of tf include the TransformListener, which listens for and queries transformation data, and the TransformBroadcaster, which broadcasts transformation data across the system. The tf package supports both 2D and 3D transformations and is crucial for integrating sensors like cameras, LIDAR, and IMUs, ensuring data from these sensors is aligned with the robot's frame of reference. The package's ability to handle real-time transformations is key for enabling accurate robot behavior in dynamic environments, supporting tasks such as simultaneous localization and mapping (SLAM) and multi-robot coordination.

And Gym-gazebo is an open source toolkit for developing and comparing reinforcement learning algorithms using ROS and Gazebo. It builds upon the original gym-gazebo, upgrading it to use ROS and providing a more convenient architecture tailored for industrial robotic applications. The toolkit consists of three main software blocks: gym-gazebo, ROS, and Gazebo. Create environments and register them in OpenAI's Gym, allowing interaction with simulated or real robots via ROS. The environments provide core functions like initialization, executing actions, and resetting, which are called by RL algorithms. The package currently offers four environments for the MARA modular robotic arm, with varying levels of complexity regarding target reaching, collision avoidance, and end-effector orientation. It provides command-line customization options and uses a training-optimized Gazebo plugin for maximum simulation speed. The toolkit has been successfully used to train policies using Proximal Policy Optimization (PPO), achieving accuracies in the millimeter scale for target reaching tasks. Gym-gazebo aims to advance reinforcement learning methods to be applicable in real industrial tasks, bridging the gap between research and production environments in robotics.

III. METHOD

A. Reinforcement Learning For Robot Control

Reinforcement Learning (RL) offers a structured approach to optimize robot control by learning policies that map observations to actions to maximize cumulative rewards. The control problem is modeled as a Markov Decision Process (MDP), represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$: State space, representing the robot's configurations and sensory inputs.
- $\mathcal{A}$: Action space, corresponding to control commands.
- $\mathcal{P}(s'|s,a)$: State transition probability, describing the likelihood of transitioning to state s' from state s after action a.
- $\mathcal{R}(s,a)$: Reward function, quantifying immediate feedback for action a in state s.
- γ: Discount factor, emphasizing long-term rewards.

The objective is to find an optimal policy $\pi^*(a|s)$ that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbf{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \right].$$

We employ Proximal Policy Optimization (PPO) to iteratively improve the policy. PPO optimizes a clipped surrogate objective to balance exploration and stability. The policy π is parameterized by a neural network π_θ, and the loss function for PPO is defined as:

$$\mathcal{L}(\theta) = \mathbf{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right],$$

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 updated and old policies.
- A_t: Advantage function, defined as $A_t = Q^\pi(s_t, a_t) - V^\pi(s_t)$.

- ϵ: Clipping parameter to restrict policy updates, ensuring stable learning.

The value function $V^{\pi}(s)$ is trained concurrently using a mean-squared error loss:

$$\mathcal{L}_V(\phi) = \mathbf{E}_t \left[\left(V_\phi(s_t) - G_t \right)^2 \right],$$

where G_t is the target return calculated from rewards.

Additionally, domain randomization is applied during training, varying physical parameters (e.g., friction, mass) to ensure the learned policy generalizes to diverse scenarios. The learned policy is first validated in simulation and then deployed on the physical robot, demonstrating its capability to handle real-world tasks.

B. System Description

The goal of this project is to enable the humanoid robot *Dora2* to achieve real-world walking by transitioning from virtual environments to physical deployment. The system design follows a structured pipeline, starting with simulation and progressing to physical validation.

1) Simulation in IsaacGym: The process begins by incorporating *Dora2*'s URDF file into *IsaacGym*, a high-performance simulation platform for reinforcement learning (RL). In this phase, the robot's walking behavior is trained by optimizing motor torque values, with the reinforcement learning algorithm *Proximal Policy Optimization (PPO)* used to stabilize policy updates. PPO minimizes large policy changes by using a "clipped objective function," ensuring smoother and more reliable updates.

2) State Representation and Control Framework: The robot's state is represented by a set of key variables:

- **Base Pose and Joint Positions:** The robot's base pose, denoted as $P_b = [x, y, z, \alpha, \beta, \gamma]$, includes the position coordinates x, y, z and orientation angles α, β, γ in Euler notation. The joint positions for each motor are represented by θ, and the corresponding joint velocities are denoted by $\dot{\theta}$.
- **Gait Cycle:** The walking behavior follows a gait cycle, which consists of two double support phases (DS) and two single support phases (SS). The cycle time, CT, defines the duration of one full gait cycle. Sinusoidal functions are used to generate reference motion, modeling the repetitive movements of the pitch, knee, and ankle joints.
- **Foot Contact Mask:** A periodic stance mask $I_p(t)$ indicates foot contact patterns in sync with the reference motion. For instance, when the left foot is lifted, the right foot enters the single support phase, represented as $[0, 1]$, while during double support phases, the mask is $[1, 1]$.
- **Proportional-Derivative (PD) Control:** The joint positions are controlled using a PD controller, with the target joint position chosen as the action. The policy network integrates proprioceptive sensor data, a periodic clock signal $[\sin(2\pi t/CT), \cos(2\pi t/CT)]$, and velocity commands $\dot{P}_{x,y,\gamma}$. Input frames and their dimensions are summarized in Table I.

3) Validation in gazebo: After training in IsaacGym, the model is transferred to *gazebo* for validation. gazebo, with its accurate physical dynamics, allows us to test the model across various simulation environments, ensuring the learned walking behavior generalizes beyond the training setup. This sim-to-sim validation phase confirms the model's adaptability to diverse conditions.

4) Modifications for ROS1 and Gazebo: Next, the official *ROS1* source code is modified to enable handheld control in *Gazebo*, a widely-used robot simulation platform. These modifications allow practical testing and fine-tuning of the system in a simulated environment, facilitating the transition to physical deployment.

5) Deployment to Physical Hardware: Finally, the trained and validated model is deployed to the physical *Dora2* robot for real-world testing and performance evaluation. This final step is crucial for assessing the robot's ability to walk in dynamic and unpredictable real-world environments.

TABLE I
SUMMARY OF OBSERVATION SPACE

Components	Dims	Observation	State
Clock Input $(\sin(t), \cos(t))$	2	✓	✓
Commands $\dot{P}_{x,y,\gamma}$	3	✓	✓
Joint Position θ	12	✓	✓
Joint Velocity $\dot{\theta}$	12	✓	✓
Angular Velocity $\dot{P}_b(\alpha, \beta, \gamma)$	3	✓	✓
Euler Angle $P_b(\alpha, \beta, \gamma)$	3	✓	✓
Last Actions a_{t-1}	12	✓	✓
Frictions	1	✓	
Body Mass	1	✓	
Base Linear Velocity	3	✓	
Push Force	2	✓	
Push Torques	3	✓	
Tracking Difference	12	✓	
Periodic Stance Mask	2	✓	
Feet Contact Detection	2	✓	

6) Control Framework: The control policy operates at a high frequency of 100Hz, providing enhanced granularity and precision compared to standard RL locomotion approaches. The internal PD controller operates at an even higher frequency of 1000Hz, ensuring precise control over joint movements. For training simulations, *IsaacGym* is used, while *gazebo*, known for its accurate physical dynamics, is used for sim-to-sim validation. This hybrid approach leverages the high-speed GPU-based parallel simulation of IsaacGym and the accurate but slower CPU-based simulation of gazebo.

C. Reward design for walking

The reward function is a critical component of reinforcement learning, guiding the agent towards desirable behaviors. In this work, we define the reward function based on several aspects of robot performance, including joint positions, foot distances, velocity tracking, and more. Each individual reward term is designed to address specific aspects of the robot's behavior and is multiplied by a scaling factor to modulate its impact on the overall reward.

Joint Position Reward: The reward based on joint positions is calculated as:

$$r_{\text{joint}} = \exp\left(-2 \cdot \|\mathbf{p}_{\text{joint}} - \mathbf{p}_{\text{target}}\|_2\right) - 0.2 \cdot \|\mathbf{p}_{\text{joint}} - \mathbf{p}_{\text{target}}\|_2, \tag{1}$$

where $\mathbf{p}_{\text{joint}}$ and $\mathbf{p}_{\text{target}}$ are the current and target joint positions, respectively. The term penalizes large deviations from the target positions.

Foot Distance Reward: The reward for maintaining an optimal foot distance is:

$$r_{\text{feet}} = \frac{1}{2}\left(\exp(-|d_{\text{min}}| \cdot 100) + \exp(-|d_{\text{max}}| \cdot 100)\right), \tag{2}$$

where $d_{\text{min}} = \max(0, \text{foot\_dist} - \text{fd})$ and $d_{\text{max}} = \max(0, \text{foot\_dist} - \text{max\_df})$ represent the minimum and maximum deviations of foot distance from the target range.

Knee Distance Reward: The reward based on knee distance is computed as:

$$r_{\text{knee}} = \frac{1}{2}\left(\exp(-|d_{\text{min}}| \cdot 100) + \exp(-|d_{\text{max}}| \cdot 100)\right), \tag{3}$$

where d_{min} and d_{max} are similar to those in the foot distance term, but applied to the knee joints.

Foot Slip Reward: To minimize foot slip, the following reward is calculated:

$$r_{\text{slip}} = \sum_i \sqrt{\|\mathbf{v}_i\|} \cdot \text{contact}_i, \tag{4}$$

where $\mathbf{v}_i$ is the velocity of the foot and contact_i is the contact condition for foot i.

Feet Air Time Reward: The reward based on foot air time is:

$$r_{\text{air}} = \sum_i \left(\text{air\_time}_i \cdot \mathbf{1}_{\text{first\_contact}_i}\right), \tag{5}$$

where air_time_i represents the time the foot is in the air, and $\mathbf{1}_{\text{first\_contact}_i}$ indicates the first contact with the ground.

Feet Contact Number Reward: This reward term is computed as:

$$r_{\text{contact}} = \frac{1}{N}\sum_i \mathbf{1}_{\text{contact}_i == \text{stance}_i}, \tag{6}$$

where N is the number of feet, and $\mathbf{1}_{\text{contact}_i == \text{stance}_i}$ penalizes mismatches between foot contacts and the expected gait phase.

Base Height Reward: The reward for maintaining the robot's base height is:

$$r_{\text{height}} = \exp\left(-|h_{\text{base}} - h_{\text{target}}| \cdot 100\right), \tag{7}$$

where h_{base} is the robot's base height and h_{target} is the target height.

Base Acceleration Reward: To promote smooth motion, we calculate the reward for base acceleration:

$$r_{\text{acc}} = \exp\left(-\|\mathbf{a}_{\text{base}}\| \cdot 3\right), \tag{8}$$

where $\mathbf{a}_{\text{base}}$ is the robot's base acceleration.

Velocity Mismatch Reward: The reward for velocity mismatch is:

$$\begin{aligned} r_{\text{vel}} = &\exp\left(-|v_{\text{lin}} - v_{\text{command}}| \cdot 10\right) \\ &+ \exp\left(-\|v_{\text{ang}} - v_{\text{command}}\| \cdot 5\right). \end{aligned} \tag{9}$$

Low Speed Reward: The low speed reward is defined as:

$$r_{\text{low\_speed}} = \begin{cases} -1 & \text{if } |v_{\text{linear}}| < 0.5 \cdot |v_{\text{command}}| \\ 0 & \text{if } |v_{\text{linear}}| > 1.2 \cdot |v_{\text{command}}| \\ 1.2 & \text{otherwise.} \end{cases} \tag{10}$$

Action Smoothness Reward: Finally, the action smoothness term is given by:

$$\begin{aligned} r_{\text{smooth}} = &\sum_i \left(\|\mathbf{a}_i - \mathbf{a}_{i-1}\|^2 + \|\mathbf{a}_i + \mathbf{a}_{i-1} - 2 \cdot \mathbf{a}_{i-2}\|^2\right) \\ &+ 0.05 \cdot \sum_i |\mathbf{a}_i|. \end{aligned} \tag{11}$$

Total Reward For Walking

The total reward r_{total} is a weighted sum of the individual reward components:

$$r_{\text{total}} = \sum_i \text{scale}_i \cdot r_i. \tag{12}$$

Each term r_i corresponds to one of the individual reward functions described above, and scale_i represents the scaling factor for that term.

D. Reward design for standing

The reward function is a fundamental aspect of reinforcement learning, driving the agent toward achieving stable and desirable standing behavior. In this work, we define the reward function based on several performance criteria, such as orientation stability, joint position tracking, and contact balance. Each individual reward term addresses specific aspects of standing performance and is modulated by a scaling factor to balance its influence on the overall reward.

Orientation Maintenance Reward: The reward for maintaining an upright orientation is defined as:

$$r_{\text{ori}} = \exp\left(-\sigma_{\text{ori}} \cdot \|\mathbf{o}_{\text{current}} - \mathbf{o}_{\text{target}}\|\right), \tag{13}$$

where $\mathbf{o}_{\text{current}}$ and $\mathbf{o}_{\text{target}}$ represent the current and target orientations, respectively, and σ_{ori} is a sensitivity parameter.

Joint Position Reward: To encourage the robot to maintain its default posture, the joint position reward is given by:

$$r_{\text{joint}} = \exp\left(-\sigma_{\text{joint}} \cdot \|\mathbf{p}_{\text{current}} - \mathbf{p}_{\text{default}}\|\right), \tag{14}$$

where $\mathbf{p}_{\text{current}}$ and $\mathbf{p}_{\text{default}}$ are the current and default joint positions.

Feet Contact Balance Reward: The reward for maintaining balanced foot contact is calculated as:

$$r_{\text{feet}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\text{contact}_i=\text{stable}}, \tag{15}$$

where N is the number of feet, and $\mathbf{1}_{\text{contact}_i=\text{stable}}$ indicates whether foot i maintains stable contact.

Collision Penalty: A negative reward is applied if any collision is detected:

$$r_{\text{collision}} = \begin{cases} -1 & \text{if collision detected,} \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Stability Maintenance Reward: To ensure the robot remains stationary and resists external perturbations, the following reward is defined:

$$r_{\text{stability}} = \exp\left(-\sigma_{\text{stability}} \cdot \|\mathbf{v}_{\text{base}}\|\right), \tag{17}$$

where $\mathbf{v}_{\text{base}}$ is the linear velocity of the robot's base.

Action Smoothness Reward: To discourage abrupt joint movements, the action smoothness reward is given by:

$$r_{\text{smooth}} = -\sigma_{\text{smooth}} \cdot \|\mathbf{a}_{\text{current}} - \mathbf{a}_{\text{previous}}\|^2, \tag{18}$$

where $\mathbf{a}_{\text{current}}$ and $\mathbf{a}_{\text{previous}}$ are the current and previous actions.

Total Reward For Standing

The overall reward is a weighted sum of the individual components:

$$r_{\text{total}} = \sum_i \text{scale}_i \cdot r_i, \tag{19}$$

where r_i represents each individual reward term, and scale_i is the corresponding scaling factor.

E. System Evaluation

To evaluate the system, several testing methods will be employed. In simulation vs reality comparison, we will first train the Dora2 robot to perform basic walking tasks (e.g., straight-line walking, turning, obstacle avoidance) in Isaac Gym [6], then compare the performance in simulation with real-world performance. This will involve measuring deviations between the simulated and real walking paths to evaluate the model's adaptability. Additionally, the trained model will be validated in gazebo under different environment settings to assess robustness across various conditions. In real-world testing, the trained control policy will be deployed onto the actual Dora2 robot and tested in varying environmental conditions (e.g., different terrains, obstacles) to assess the robot's ability to walk stably and adapt to external disturbances.

Evaluation will be based on task completion, stability metrics, performance comparison, and real-time response. A successful walking task is defined as completing a predefined path (e.g., walking 5 meters in a straight line, with at least 8 successful runs out of 10 trials) without falling. Stability metrics will focus on the number of falls and the deviation from the path, with acceptable tolerances set for both. Performance comparison will involve comparing motor torque data between simulation and real-world tasks, ensuring that the control strategy transferred from simulation to reality is effective. Finally, real-time response will measure the robot's response time and control system's performance, ensuring quick execution of commands.

IV. Experiments

A. Training Details

Reinforcement learning (RL) training was conducted in Isaac Gym using the Proximal Policy Optimization (PPO) algorithm [8]. Policies were trained with a batch size of 64 trajectories (400 timesteps per trajectory), a learning rate of 0.0003, and 5 epochs per update. Training was terminated after 250 million samples, which required approximately 40 hours on an NVIDIA RTX 4090 GPU.

The observation space consisted of joint positions, velocities, angular velocities, and command inputs (e.g., linear and angular velocities). To synchronize motion phases, a periodic clock signal was included in the observations. Domain randomization was applied to enhance the sim-to-real transferability by introducing variations in robot and environmental parameters.

The initial goal was to train a walking policy capable of forward, backward, and turning maneuvers. Once the walking policy achieved satisfactory performance, it was exported as a TorchScript JIT model for further validation.

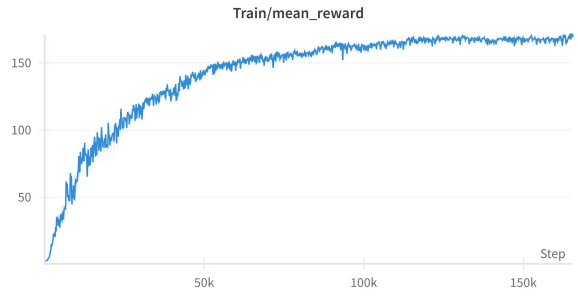

Fig. 1. Mean reward for walking policy during training in Isaac Gym.

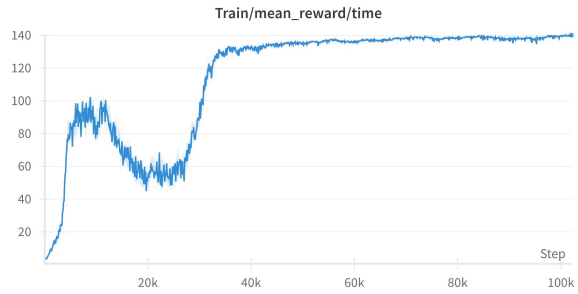

Fig. 2. Mean reward for standing policy during training in Isaac Gym.

B. Sim-to-Sim Validation

The trained walking policy was imported into gazebo for cross-environment validation. This step was crucial to ensure the policy's robustness and consistency when tested in a more accurate physics engine. Validation metrics included trajectory alignment, phase portraits, and foot-ground contact patterns.

Once validated in gazebo, the policy was converted to ONNX format for compatibility and deployment. Rigorous testing of the ONNX model's input-output integrity ensured that no critical bugs occurred during real-world implementation.

C. ROS and Gazebo Simulation

The ONNX policy was integrated into a ROS-based control framework and tested in Gazebo, where proprietary company code was adapted to support the new walking policy. Gazebo simulations replicated real-world conditions, including uneven terrain and external disturbances.

The walking policy demonstrated satisfactory performance, successfully navigating complex terrains and recovering from minor disturbances. With consistent results in Gazebo, the policy was deemed ready for real-world deployment.

D. Real-World Testing and Iterative Refinements

Initial real-world tests revealed significant discrepancies between simulation and physical performance, attributed to unmodeled dynamics and environmental factors. To address this, the environment configuration parameters were updated, and the policy was retrained with refined domain randomization settings. The revised policy underwent repeated validation in Isaac Gym and Gazebo before redeployment.

Certain aggressive maneuvers observed during real-world tests posed potential risks to the robot's hardware. A safety-focused policy was trained to prioritize stability and reduce impact forces, resulting in improved hardware safety and robustness in both simulated and real-world environments.

E. Standing Policy and Autonomous Control

Following the validation of the walking policy, a standing policy was trained using a similar approach. This policy aimed to enable the robot to maintain balance and resist external perturbations independently. The training process for the standing policy included domain randomization and safety considerations to ensure real-world applicability.

Control transitions between walking and standing policies were implemented through a handheld controller, enabling seamless switching during operation. The final system achieved fully autonomous control, allowing the robot to transition between walking and standing with minimal operator intervention. This demonstrated the system's versatility and robustness in real-world scenarios.

F. Discussion

The experiments confirm the effectiveness of our approach in training and deploying robust walking and standing policies. The sim-to-sim validation in Gazebo played a critical role in bridging the sim-to-real gap, while iterative refinements during real-world testing addressed discrepancies caused by unmodeled dynamics.

Future work will focus on further optimizing domain randomization techniques and safety measures to enhance deployment reliability. Additional efforts will be directed towards improving the policies' performance on diverse and challenging terrains to extend the robot's operational capabilities.

V. Conclusion

This work presents a comprehensive framework for training, validating, and deploying locomotion policies for humanoid robots in both simulated and real-world environments. By leveraging reinforcement learning in Isaac Gym, followed by cross-environment validation in Gazebo, we successfully minimized the sim-to-real gap, enabling zero-shot transfer to real hardware.

The iterative refinement process proved crucial for addressing discrepancies between simulation and real-world performance, particularly for dynamic walking and turning maneuvers. The introduction of a safety-focused policy significantly improved hardware protection during real-world deployment. Furthermore, the standing policy and autonomous control transitions between standing and walking demonstrate the versatility and robustness of the trained models.

While the proposed framework achieves stable and reliable locomotion, further improvements are needed to enhance adaptability and responsiveness to external disturbances. Future work will focus on integrating tactile sensors into the robot's feet, enabling precise feedback on ground contact forces. This advancement is expected to significantly improve gait stability, enhance adaptability to uneven terrains, and allow for more dynamic behaviors. Additionally, efforts will continue to optimize domain randomization techniques and hardware-specific adaptations to bridge the sim-to-real gap more effectively.

Appendix

This appendix provides a comprehensive summary of the parameters and scaling factors used in the reward functions for both walking and standing tasks.

A. Walking Task Reward Parameters

Table II lists the parameters and scaling factors for the walking reward terms.

B. Standing Task Reward Parameters

Table III summarizes the parameters and scaling factors for the standing reward terms.

References

[1] J. Albrecht, A. Sutton, B. Lee, and J. Hurst. Learning to walk via deep reinforcement learning. *IEEE Transactions on Robotics*, 35(4):872–884, 2019.

[2] Helei Duan, Bikram Pandit, Mohitvishnu S. Gadde, Bart Jaap van Marum, Jeremy Dao, Chanho Kim, and Alan Fern. Learning vision-based bipedal locomotion for challenging terrain. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 56–62, 2023.

TABLE II
REWARD PARAMETERS FOR WALKING

Term	Scale	Parameter Value
Joint Position	0.8	$\sigma_{\text{joint}} = 2.0$
Foot Distance	1.0	fd $= 0.15$, max_df $= 0.25$
Knee Distance	0.9	Same as Foot Distance Parameters
Foot Slip	1.2	-
Feet Air Time	0.6	-
Feet Contact Number	1.0	-
Base Height	1.5	$h_{\text{target}} = 0.8$
Base Acceleration	0.5	-
Velocity Mismatch	2.0	-
Low Speed	-1.0	Speed Thresholds: $0.5, 1.2$
Action Smoothness	0.05	$\sigma_{\text{smooth}} = 0.1$

TABLE III
REWARD PARAMETERS FOR STANDING

Term	Scaling Factor	Parameter Value
Orientation Maintenance Reward	1.5	$\sigma_{\text{ori}} = 5.0$
Joint Position Reward	0.7	$\sigma_{\text{joint}} = 3.0$
Feet Contact Balance Reward	1.5	-
Collision Penalty	-1.5	-
Stability Maintenance Reward	3.0	$\sigma_{\text{stability}} = 2.0$
Action Smoothness Reward	-0.002	$\sigma_{\text{smooth}} = 0.1$

[3] Xinyang Gu, Yen-Jen Wang, and Jianyu Chen. Humanoid-gym: Reinforcement learning for humanoid robot with zero-shot sim2real transfer. *ArXiv*, abs/2404.05695, 2024.

[4] V. Gupta, J. Lee, A. Shia, and J. Y. Chen. Deep reinforcement learning for robotic manipulation with asynchronous policy updates. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5314–5320, 2021.

[5] T. Huang, Y. Li, R. A. Knepper, and R. D. S. Mellinger. Learning to walk in 2d with deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):4258–4265, 2020.

[6] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[7] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*, 9, 2023.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[9] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan W. Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315, 2020.

[10] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan W. Hurst. Sim-to-real: Six-legged robot control with deep reinforcement learning and curriculum learning. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315, 2020.