Sim-to-Real Humanoid Walking: Minimal Input PPO with Domain Randomization

Matthew Lee [*1] **Mallika Parulekar** [*1]

Abstract

Locomotion is a complex problem in robotics, especially for bipedal robots. In this work, we develop a reinforcement learning-based walking policy for Z-Bot, a 1.6-foot, 8-pound open-source humanoid robot, using Proximal Policy Optimization (PPO). We train a walking controller in simulation and address the sim-to-real transfer gap through domain randomization and curriculum learning. Our experiments analyze minimal input policies, showing that joint velocity information is crucial for stability, while angular velocity and action history can be omitted with minimal performance loss. To improve real-world transferability, we compare linear and performance-based curriculum learning, finding that the latter is more effective, with mass variations significantly affecting training stability. Despite overcoming joint mapping and scaling issues, successful walking in KOS-Sim, the Mujoco-based simulator for deployment, remains an open challenge, highlighting the complexities of robust locomotion policy training.

Our code can be found at this link and a video of our trained walking policies can be seen here.

1. Introduction

Bipedal locomotion remains a challenging problem in robotics, requiring precise coordination and continuous balance correction in order to balance the weight of not only the body but also the legs. This challenge is particularly pronounced in low-cost, lightweight humanoid robots, where actuation delays and modeling inaccuracies can lead to difficulty in training. In this work, we aim to develop a robust walking policy for the Zeroth Bot (Z-Bot), an open-source, hobbyist-scale humanoid robot developed by K-Scale Labs. Standing approximately 1.6 feet tall and weighing 8 pounds, the Z-Bot is an accessible subject for testing reinforcement learning-driven locomotion.

Our goal is to train a minimal, yet effective, walking policy that enables stable locomotion using Proximal Policy Optimization (PPO). A major hurdle in this process is the sim-to-real gap, where policies trained in simulation often fail to transfer effectively to the real-world robot due to discrepancies in dynamics like friction, difference in mass, and unmodeled physical effects. To bridge this gap, we incorporate domain randomization and curriculum learning, gradually increasing the complexity of training conditions to improve real-world robustness.

2. Background / Related Work

2.1. Sim-to-Real Transfer

Transferring learned policies in simulation to the real world, known as sim-to-real transfer, is a key challenge in reinforcement learning (RL). Due to discrepancies between simulated and real-world environments, policies trained purely in simulation often fail when deployed in real settings. To combat this, researchers often apply the technique of domain randomization, where they introduce variability into the simulation environment during training in order to learn a robust policy that would work in the real world. One of the first demonstrations of domain randomization showed that training deep neural networks for object localization on simulated images with randomized rendering enabled them to generalize to real-world images without additional training (Josh Tobin, 2017). For locomotion, parameters that affect motion such as mass, friction, PD gains and inertia are randomized in order to account for discrepancies in the physical robot or its interactions with the surface (Nolan Fey & Agrawal, 2025), (Jie Tan & Vanhoucke, 2018).

2.2. Physics Simulators

Physics simulators play a crucial role in the study of learning locomotion, providing a controlled and efficient environment for training and evaluating movement policies. These simulators model the dynamics of agents, capturing forces, contacts, and constraints that influence locomotion. MuJoCo (Multi-Joint Dynamics with Contact)

[*]Equal contribution [1]Stanford University, California, USA. Correspondence to: Matthew Lee <leemat@stanford.edu>, Mallika Parulekar <mallika.parulekar@stanford.edu>.

(Emanuel Todorov, 2012) is known for its efficient and accurate rigid-body dynamics, making it popular in reinforcement learning (RL) and robotics research. Genesis is an emerging physics simulator designed to provide high-performance and customizable simulation environments, particularly for reinforcement learning and robotics applications. Its main benefit is a significant speedup (higher frames per second) than other simulators (Zhou Xian, 2024).

2.3. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a widely used RL algorithm that improves policy gradient methods by balancing performance and stability. Introduced by Schulman et al. (2017), PPO (John Schulman, 2017) is designed to optimize policies in an efficient and stable manner by using a clipped surrogate objective function to prevent excessively large policy updates, which can destabilize training. This approach strikes a balance between allowing sufficient exploration and preventing drastic changes to the policy. PPO is often used in locomotion, where it can refine and adjust the robot controller's actions to help the robot adapt to uneven or complex terrain while maintaining stability and smooth locomotion (Yilin Zhang & Hashimoto, 2024).

3. Approach

3.1. Training Setup

3.1.1. State Space and Action Space

The **state space** of our baseline policy consists of the following components:

- **Base angular velocity (3):** The angular velocity of the robot's base, represented as a three-dimensional vector $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$, which describes the rotational motion of the base in the world frame.
- **Projected gravity (3):** A three-dimensional vector representing the gravity direction projected onto the robot's local frame. This provides an implicit estimate of the robot's orientation relative to gravity.
- **Target commands (3):** A set of high-level control commands given to the robot, consisting of desired forward velocity, lateral velocity, and yaw rate, which guide the motion.
- **Degrees of freedom (DOF) velocities and positions (20):** The positions and velocities of the robot's 10 actuated joints.
- **Previous actions (10):** The most recent actions taken by the policy, which helps in modeling actuation dynamics and ensuring smooth transitions in movement.

The **action space** consisted of the joint angles for the ten actuated joints in the robot's legs. Each leg has one controllable joint in the ankle and knee, and three controllable joints (pitch, yaw, roll) in the hip, as seen in Figure 1. The proportional-derivative (PD) controllers take in the joint angles as input and compute torques at a high frequency in order to help the actuator achieve the positions.

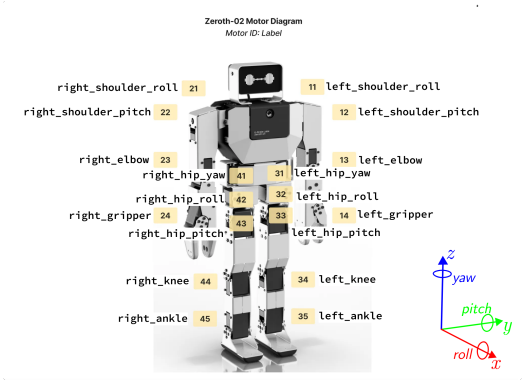

Figure 1. Z-Bot motor diagram showing controllable joints

3.1.2. Training and Model Architecture

We followed the K-Scale documentation in order to set up a Genesis playground to train the ZBot. The codebase we worked with had existing code to train the ZBot with PPO using the RSL RL library RSL RL library , a high-performance reinforcement learning framework optimized for robotics. The library includes an Actor and Critic model, where the Actor is responsible for producing the policy by predicting what actions to take next based on the state space and the Critic computes an estimate of the value of the state, which is used to compute the advantage in PPO. The architecture of these models is shown in Figure 2. Other parameters such as learning rate, gamma, and the clip parameter are written in a training configuration that is passed into the RSL RL class object. We train these models locally on MacBook Pros using Pytorch, which provides a backend to do computation using Apple's Metal Performance Shaders (MPS) on the local GPU.

3.1.3. Simulation and Environments

We trained locomotion policies on 1024 environments in parallel for 300 iterations, which took around 30 minutes on our laptops. The policy execution frequency was 50Hz, and every 4 seconds, a new target command in the state space was issued to the robot. Each training iteration consisted of 48 steps per environment, and each episode (which could span multiple iterations) was a maximum of 1000 steps.

While the documentation was fairly straightforward, we had to make a change to the code in order to view the

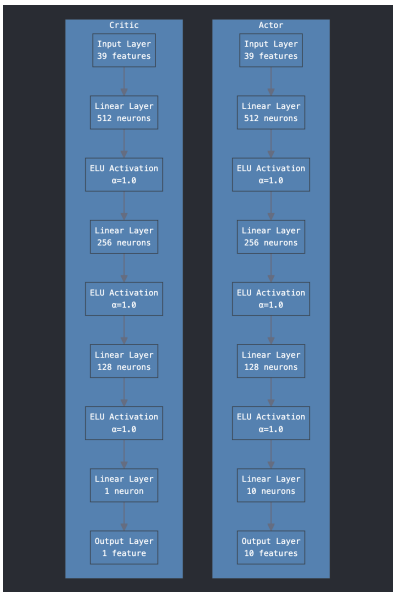

Figure 2. Architecture of Actor and Critic models used in RSL RL library

trained walking policy. Specifically, we had to explicitly start the Genesis viewer in the main thread while running the simulation in another thread, in order to view it successfully.

3.2. Reward Function

The reward function in our environment is defined in `reward_cfg`, specifying individual components and their weightings, tuned heuristically. Key components include **Tracking Linear Velocity** and **Tracking Angular Velocity**, which minimizes error between desired and actual velocity; **Base Height**, penalizing deviations from target height; **Feet Air Time**, rewarding consistent foot lift timing; **Action Rate**, discouraging abrupt changes; **Joint Deviation**, keeping joint positions close to default values; and **Vertical Linear Velocity**, penalizing excessive jumping.

Initially, training used the original URDF, defining robot dynamics and constraints. A later update introduced **5× lower maximum torque**, requiring PD gain adjustments and affecting locomotion stability. The updated policy exhibited unnatural side-to-side motion, prompting two new reward terms: **Lateral Stability** (penalizing lateral shifts) and **Step Phase Coherence** (rewarding proper left-right leg alternation). Their impact on stability is evaluated in the results section.

3.3. Porting over to Stable Baselines

We were also interested in trying other RL algorithms in addition to PPO, to see if that made a difference in the quality of the trained locomotion policy. Since RSL RL only has an implementation of PPO, we decided to port over to using Stable-Baselines-3 (SB3), which provides reliable implementations of RL algorithms such as PPO, DDPG, SAC and TD3. As a first step, we wanted to reproduce the RSL RL results for PPO using SB3. We set up a wrapper class called ZBotGymEnv to ensure compatibility of the ZBotEnv with SB3, since it requires environments to follow the Gymnasium API. We also set up another wrapper class called ZBotVecEnv in order to support vectorized environments for our parallel training. We specified a custom policy so that the architecture of the actor and critic in SB3 matched that of RSL RL and checked to ensure that parameters matched.

While we were able to eventually run the code without error, we would get a mean reward of 0.01 that was very unstable (fluctuated and sometimes becomes negative during training) and a mean episode length of less than 1. One cause for this may have been the adaptive learning implemented in RSL RL that was not present in SB3. After the milestone, we eventually decided to stick to the RSL RL setup and focus on developing the minimal input policies and implementing domain randomization.

3.4. Minimal Policy Setup

Learning to walk in simulation has become increasingly popular in recent months, with policies trained successfully across various simulators (Isaac Sim, Genesis, Mujoco) and humanoid platforms. These policies typically rely on a rich set of proprioceptive inputs, including base state information, joint positions and velocities, and IMU signals. An interesting challenge is determining the minimal set of inputs and network complexity required to reliably train a humanoid to walk. By methodically simplifying our reinforcement learning (RL) setup, we aim to analyze what information is essential and what can be removed without significantly degrading performance.

Our baseline policy operates with a **39-dimensional state space**, which includes **base angular velocity (3)**, **projected gravity (3)**, **target commands (3)**, **DOF positions (10)**, **DOF velocities (10)**, and **previous actions (10)**. To systematically investigate minimal policy setups, we will compare multiple ablated versions against this baseline:

- **No Angular Velocity**: Removing angular velocity from the IMU input will help evaluate whether projected gravity can provide sufficient orientation information for stable locomotion.
- **No Action History**: A standard Markov Decision Process (MDP) formulation typically does not include past actions. By eliminating this history, we analyze whether the agent can still learn effective policies.
- **No DOF Velocities**: Removing joint velocity informa-

tion will help evaluate whether joint positions can effectively capture the information that velocities would have provided.

By training and evaluating these minimal setups, we aim to gain insight into the role of each state component in humanoid locomotion. The results of these experiments will be discussed in the results section.

3.5. Domain Randomization and Curriculum Learning

Domain randomization helps policies generalize by introducing variability in **robot link masses**, **robot-to-ground friction**, and **torque controller PD gains**, preventing overfitting. We apply **multiplicative randomization**, setting a **start range** for easy conditions and an **end range** for harder ones. For instance, a **5.0 kg** link may have mass variations from $[0.9, 1.1]$ initially, expanding to $[0.8, 1.2]$, increasing from $\pm 10\%$ to $\pm 20\%$.

Curriculum learning typically follows a **linear interpolation strategy**, where difficulty increases at a constant rate: given $T_{\max}$ training steps, difficulty is scaled as $t/T_{\max}$. However, this rigid schedule does not adapt to the agent's performance.

To address this, we introduce a **performance-based curriculum**, where difficulty progresses only as the policy improves. The agent remains in easier settings until it reaches a reward threshold, then difficulty increases smoothly using a **quadratic function** to prevent abrupt jumps. This approach allows stable, adaptive learning. The full algorithm is shown in Algorithm 1.

Algorithm 1 Performance-Based Curriculum Learning

if `avg_reward` $<$ `min_reward_threshold` **then**
  return `Uniform(curriculum.start[0], curriculum.start[1])`
end if
`progress` $\leftarrow \min\left(1.0, \frac{\texttt{avg\_reward} - \texttt{min\_reward\_threshold}}{\texttt{target\_reward} - \texttt{min\_reward\_threshold}}\right)$
`smoothed_progress` $\leftarrow$ `progress`^{2}
`min_value` $\leftarrow$ `curriculum.start[0]`$+$
  `(curriculum.end[0]` $-$ `curriculum.start[0])`
  $\times$`smoothed_progress`
`max_value` $\leftarrow$ `curriculum.start[1]`$+$
  `(curriculum.end[1]` $-$ `curriculum.start[1])`
  $\times$`smoothed_progress`
return `Uniform(min_value, max_value)`

Unlike fixed-rate difficulty increases, this algorithm adjusts difficulty based on the agent's **performance progress**. Initially, parameters (e.g., mass, friction, PD gains) vary within a narrow range. When the agent's **average reward exceeds a threshold**, difficulty increases smoothly using a **quadratic scaling function** to prevent sudden jumps.

If performance declines, difficulty decreases accordingly rather than continuing to increase. This ensures adaptation occurs only when the policy improves. Training results for both curriculum strategies are discussed in the results section.

3.6. Sim2Sim and Sim2Real

In order to try out trained policies out on the robot, we also attempted Sim2Sim. To run our policies on a physical Z-bot, we would need to have our trained policy successfully execute on a Z-Bot in KOS-Sim, the simulation backend for the K-Scale Operating System (KOS), which uses the same gRPCs (remote-procedure calls) as the physical Z-Bot. Thus, Sim2Sim transfer was a crucial step towards Sim2Real in our setting.

We took multiple steps towards bridging the Sim2Sim gap. KOS-Sim and the code in Genesis had different joint mappings, so we came up with simple functions in order to map to and from each kind of joint mapping. This way the trained policy would get inputs (of DOF velocities / positions and previous actions) in the original Genesis joint space, while the simulator would be able to output joint angles in the KOS-Sim joint space. We also made sure that the projected gravity was aligned in both simulators, that our radian outputs were converted to degrees, and that we applied the same clipping and scaling to outputted actions in KOS-Sim as we did in Genesis.

4. Experimental Results

4.1. Original URDF: Minimal Policy Experiments

The **original URDF** refers to the initial ZBot model provided by K-Scale Labs, which was used for most of our experiments. It remained our primary test model throughout development, as the updated URDF only became available **two days before the final poster presentation**. Consequently, all minimal policy experiments were conducted on the original model.

Our **minimal policy setup** experiments aimed to systematically reduce state information and analyze its impact on learning. We tested four configurations:

1. **Baseline Policy** (full 39-state input)
2. **No Angular Velocity** (36-state input: removes IMU angular velocity)
3. **No Action History** (29-state input: removes previous action history)
4. **No DOF Velocities** (29-state input: removes joint velocity inputs)

To track performance, we logged key metrics using **Weights**

& Biases (WandB):

- **train/mean_reward**: The average reward over the most recent 100 completed episodes across all environments.
- **train/mean_episode_length**: The average episode duration before termination (falling or exceeding stability limits). The **maximum episode length** is 1000 steps, equivalent to **20 seconds of simulated time** at 50 Hz. Policies reaching 1000 steps indicate stable locomotion.

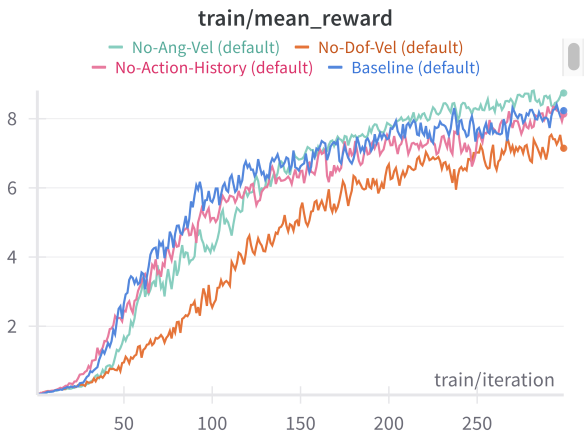

Figure 3. Original URDF: Mean Reward vs Train Iteration

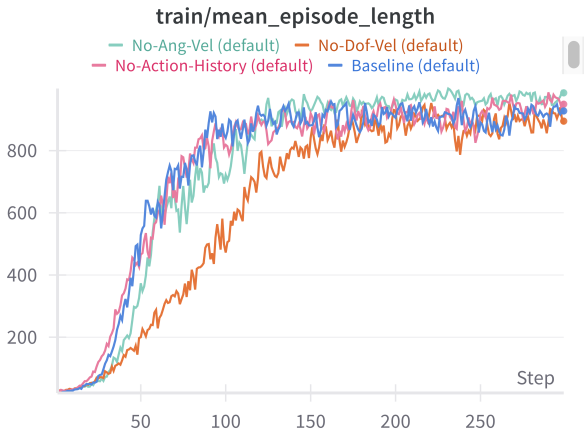

Figure 4. Original URDF: Mean Episode Length vs Train Iteration

Overall, **mean reward and episode length exhibit strong correlation**, as expected—**higher rewards indicate better stability, leading to longer episodes**. Across all four setups, both reward and episode length generally increase over training and **converge near the 1000-step limit**, signifying stable locomotion.

All four setups successfully achieved walking behavior in **Genesis Sim**.

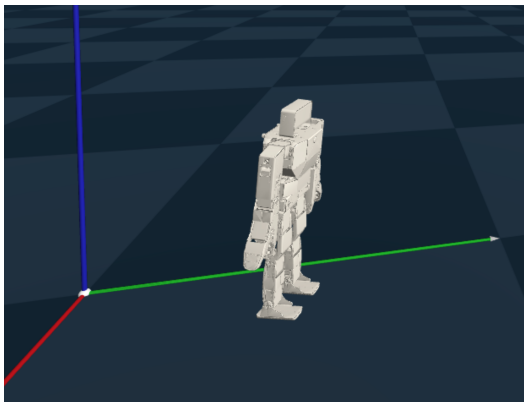

Figure 5. Freeze-frame of successful walking behavior in simulation.

Among the four, the **No DOF Velocities** setup performed the worst, with **lower mean reward and shorter episode lengths**, suggesting that joint velocity information is crucial for precise control and stable walking. Meanwhile, the **No Angular Velocity** and **No Action History** setups performed similarly to the baseline, indicating that IMU angular velocity and action history may be less critical for walking in this setup.

4.2. Original URDF: Domain Randomization Experiments

As discussed earlier, **domain randomization** introduces variability in physical parameters to improve policy robustness. We randomized **robot link mass multipliers, foot-to-ground friction multipliers, and PD controller gains**, comparing two curriculum learning strategies: **linear curriculum**, which increases difficulty at a constant rate, and **performance-based curriculum**, which adjusts difficulty based on the policy's reward performance.

For these experiments, the **default domain randomization multipliers** were set as follows: mass multipliers remained at $\pm 0\%$ initially (no change), friction multipliers had a fixed start and end range of $\pm 10\%$, and PD gain multipliers had a fixed start and end range of $\pm 25\%$. The four experiments conducted were:

1. **Baseline (Default Linear Curriculum)** – Uses the default multipliers.
2. **Linear Curriculum with Increased Variation** – Friction variation increases from $\pm 10\%$ to $\pm 20\%$, and mass variation from $\pm 0\%$ to $\pm 10\%$.

3. **Performance-Based Curriculum with Increased Variation** – Same as (2), but difficulty adapts dynamically based on policy performance.

4. **Performance-Based Curriculum with Reduced Mass Variation** – Same as (3), but mass variation is capped at $\pm 3\%$ instead of $\pm 10\%$ to improve stability.

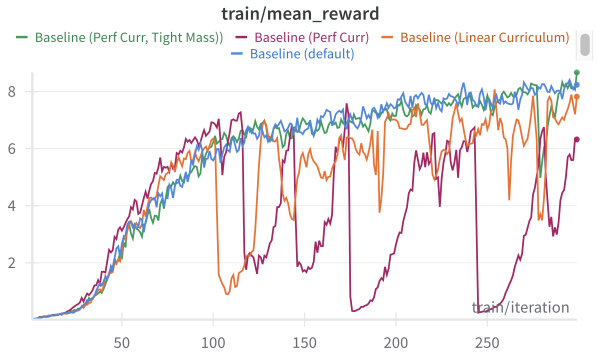

Figure 6. Curriculum Learning Experiments: Mean Reward vs Train Iteration

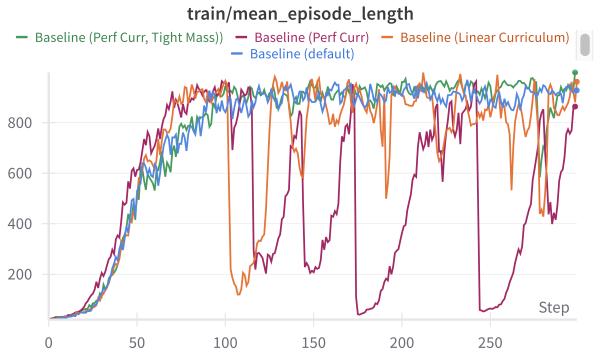

Figure 7. Curriculum Learning Experiments: Mean Episode Length vs Train Iteration

Across all four experiments, evaluation in simulation with default mass and friction values showed successful walking behaviors. However, **Experiments 2 and 3** resulted in noticeably more unstable walking compared to **Experiments 1 and 4**, suggesting that larger mass variations during training negatively impacted stability.

The **baseline (Experiment 1)** performed well, with mean reward and episode length converging smoothly, consistent with the minimal policy experiments. In **Experiment 2**, where mass and friction variations were increased with a linear curriculum, training was **highly unstable**, with **wildly fluctuating reward and episode lengths**. Training **collapsed multiple times** before partially recovering around the **300th iteration**, indicating that more iterations may be required for full convergence.

A similar instability occurred in **Experiment 3** with performance-based curriculum learning. Initially, training was stable, but when the **mean reward exceeded 6** (the threshold where difficulty increased), training **rapidly destabilized**, crashing to zero reward multiple times before recovering near the end. This suggests that difficulty **increased too aggressively**, making it hard for the agent to adapt.

In **Experiment 4**, reducing mass variation to $\pm 3\%$ instead of $\pm 10\%$ significantly **improved stability**, while maintaining the same performance-based curriculum. Training curves closely matched the baseline without any crashes. This indicates that **mass randomization has a disproportionate impact on stability**, while friction variation is more forgiving.

Overall, these results suggest that **mass variation has a much stronger effect on training stability than friction variation**. Large mass variations disrupted convergence, leading to instability in both training curves and observed walking behavior. Further investigation is needed to understand why mass multipliers cause such instability—possible factors include changes in foot-ground contact dynamics, altered torque requirements for stable stepping, or unexpected interactions with the PD controller.

4.3. Updated URDF: Experiments and Reward Engineering

Near the end of our project, we received an **updated URDF** from K-Scale Labs, incorporating new system identification data and refined physical properties. The most significant change was a **5× reduction in maximum torque output**, significantly weakening actuation capabilities. This directly affected the robot's ability to maintain stability and control. To understand the impact of reduced torque and explore possible improvements, we conducted new experiments.

We tested four setups using the updated URDF:

1. **New URDF Baseline** – Full state space, default curriculum learning, and default reward function.

2. **New URDF with Reward Engineering (Lateral Stability + Phase Incentive)** – Added a **lateral stability penalty** to reduce side-to-side motion and a **phase coherence incentive** to encourage a more traditional walking gait.

3. **New URDF with Reward Engineering (Only Phase Incentive)** – Kept only the phase coherence incentive to assess its impact independently.

4. **New URDF Without Angular Velocity** – Removed IMU angular velocity input while keeping the default reward function to analyze its effect on training.

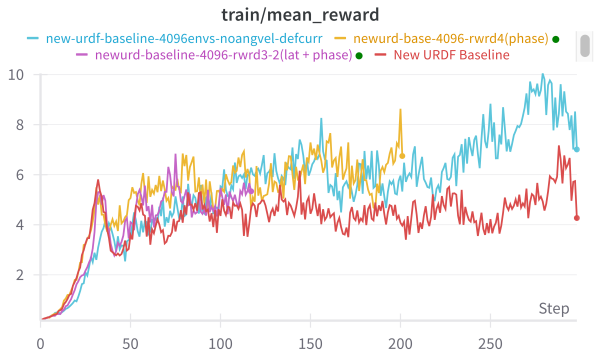

Figure 8. New URDF: Mean Reward vs Train Iteration

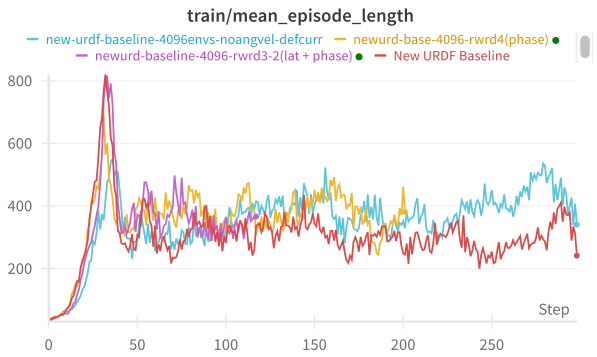

Figure 9. New URDF: Mean Episode Length vs Train Iteration

Training with the **new URDF baseline** led to **unexpected behaviors**. Unlike the original URDF, which reached **1000 steps per episode**, the new URDF, other than the initial spike, **never exceeded 400–500 steps** and worsened over time. The simulation revealed that the **robot adopted a different walking strategy**, shifting side to side instead of stepping one foot in front of the other.

To address this, we introduced **reward engineering** with two additional terms:

- **Lateral Stability Penalty** – Penalizes excessive lateral movement to discourage side-to-side shifting.
- **Phase Coherence Incentive** – Rewards alternating leg movements when joint angles are in **opposite phase** during stepping.

However, in **Experiment 2**, where we added both components, training remained unstable. The robot exhibited

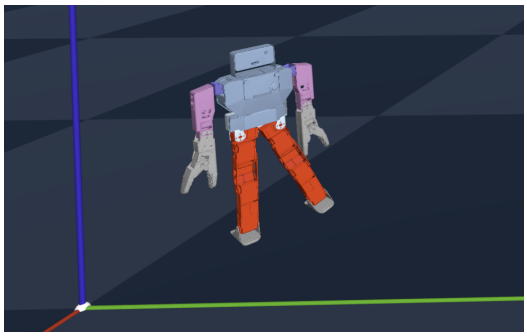

Figure 10. Freeze-frame of side-to-side walking behavior.

bizarre behaviors, such as walking on one foot and twisting instead of stepping properly. It **failed to reach 500 steps**, confirming that the issue extended beyond lateral movement and was likely linked to actuation constraints.

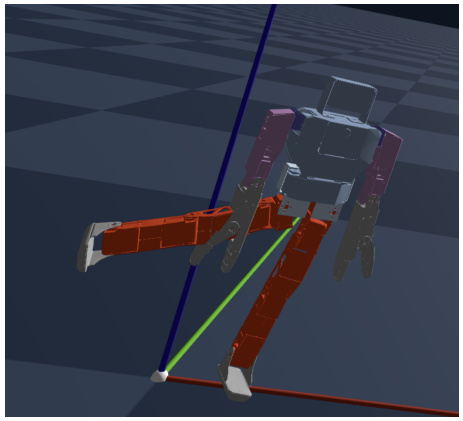

Figure 11. Freeze-frame of robot walking on one foot and shuffling in the reward-engineered experiment.

In **Experiment 3**, we **removed the phase coherence incentive**, keeping only the **lateral stability penalty**. This change **did not improve performance**—training still failed to reach 500 steps, and unstable walking persisted. This indicates that **penalizing lateral movement alone was insufficient** for achieving a proper gait.

Finally, in **Experiment 4**, we tested a **no angular velocity** setup while maintaining the default reward function. Initially, training progressed **normally for 100–200 iterations**, but by the **300th iteration**, the robot **stopped walking entirely**. Instead, it **fell into the splits position and remained stuck, vibrating in place**.

The **5× reduction in torque output significantly impacted learning**, forcing the robot to develop alternative walking strategies. With weaker actuation, it struggled to maintain stability using previous control strategies. This suggests that **reward modifications alone were insufficient**—further adjustments to curriculum learning or action scaling may be

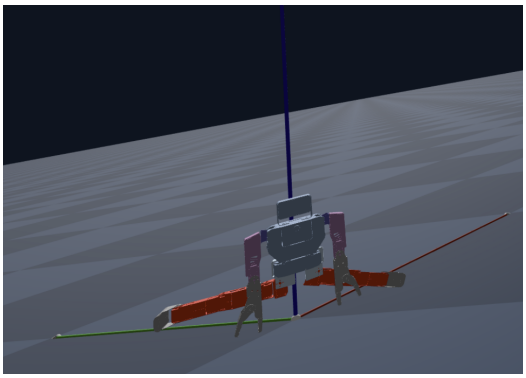

Figure 12. Freeze-frame of robot falling into splits and vibrating in place in the no angular velocity experiment.

necessary.

Despite our attempts at **reward engineering**, neither the **lateral stability penalty** nor the **phase coherence incentive** successfully guided the robot to a traditional gait. The **robot's behavior remained highly unstable**, indicating that additional constraints or refinements are needed.

K-Scale Labs is continuing to refine the URDF, and further modifications may provide a more accurate representation of the robot's real-world capabilities.

4.4. Sim2Sim Results

Unfortunately we were not able to get a successful walking policy in KOS-Sim using the latest URDF, despite our attempts. Eventually we were able to get a policy that was able to stand stably for a short while but not make any forward progress in walking. We hypothesize that the root cause of the issue may be that torque is calculated and applied differently in each simulator given the joint positions. Other groups working with K-Scale doing Sim2Sim from Genesis also faced similar issues.

5. Conclusion

In this project, we investigated reinforcement learning-based humanoid locomotion for ZBot, an open-source bipedal robot, using Proximal Policy Optimization (PPO) as our primary training algorithm. To work toward sim-to-real transfer, we explored minimal policy setups, domain randomization, and curriculum learning to develop a policy that relies on the least necessary input while maintaining stable walking. Our experiments demonstrated that reducing input complexity can still produce effective walking policies, though joint velocity information improves control performance. Domain randomization experiments revealed that mass variations have a significant impact on training stability, while friction variations are more forgiving. The updated URDF, with 5× lower torque, resulted in unstable gaits, highlighting the need for reward engineering and further refinements in control strategies.

Beyond training in Genesis, we began the process of sim-to-sim transfer to KOS-Sim, the Mujoco-based backend of the K-Scale Operating System, as a prerequisite to sim-to-real deployment. This required multiple changes including remapping of joints and recalculation of projected gravity. While we were not able to replicate successful walking in KOS-Sim, we were able to achieve a stable standing pose, which shows promise for future success with a walking policy.

Future Work

Future work includes testing alternative RL algorithms such as TDMPC or BRO for improved sample efficiency and fine-tuning hyperparameters. Bridging the sim-to-real gap will require both better domain randomization (adding more noise during training) and improved hardware modeling (more accurate actuator characterization). Ultimately, real-world deployment will be difficult due to actuation discrepancies and simple friction models, but ongoing system identification at K-Scale Labs aims to refine these. The immediate next step is getting a Genesis policy to successfully transfer to KOS-Sim before attempting real-world testing.

Acknowledgments

Thank you Keenon Werling for being our project mentor and giving us helpful feedback during office hours and the poster session. Thank you to K-Scale Labs and Paweł Budzianowski for providing the project idea and ZBot hardware.

Contributions

Both members contributed to all aspects of the project, including problem definition, coding, experiment design, training, analysis, and sim-to-sim transfer efforts.

References

Emanuel Todorov, Tom Erez, Y. T. Mujoco: A physics engine for model-based control. *International Conference on Intelligent Robots and Systems*, 2012. URL https://ieeexplore.ieee.org/document/6386109.

Jie Tan, Tingnan Zhang, E. C. A. I. Y. B. D. H. S. B. and Vanhoucke, V. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018. URL https://arxiv.org/pdf/1804.10332.

John Schulman, Filip Wolski, P. D. A. R. O. K. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Josh Tobin, Rachel Fong, A. R. J. S. W. Z. P. A. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017. URL https://arxiv.org/abs/1703.06907.

Nolan Fey, Gabriel B. Margolis, M. P. and Agrawal, P. Bridging the sim-to-real gap for athletic loco-manipulation. *arXiv preprint arXiv:2502.10894*, 2025. URL https://arxiv.org/abs/2502.10894.

Yilin Zhang, Jiayu Zeng, H. S. H. S. and Hashimoto, K. Dual-layer reinforcement learning for quadruped robot locomotion and speed control in complex environments. *Applied Sciences*, 2024. URL https://www.mdpi.com/2076-3417/14/19/8697.

Zhou Xian, e. a. Genesis: A universal and generative physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.