# Minimal Walking Policy for Real World Humanoid

**Manolo Alvarez** [*1] **Martina Del Gaudio** [*1] **Luciano Gonzalez** [*1]
**Mentor: Pawel Budzianowski** [2]

## Abstract

In this work we investigate the minimal RL setup required for stable humanoid walking by systematically reducing observation space while maintaining locomotion performance. Using PPO, we establish a baseline walking policy for the Zbot humanoid platform and conduct a feature importance analysis to identify essential proprioceptive inputs. We also explore TDMPC2 and VMP as alternative approaches to improve sample efficiency and generalization but ultimately fail to produce policies that match the performance of our baseline locomotion policy trained using PPO. Our results highlight the challenges of reducing input dimensionality and the trade-offs in model-based and prior-driven methods. Future work focuses on real-world deployment and sim-to-real transfer.

## 1. Introduction

Humanoid locomotion through reinforcement learning (RL) has seen significant advancements, with increasingly robust policies trained in simulation environments such as Mujoco and Genesis successfully deployed on real-world robotic platforms. Despite this progress, most existing approaches rely on high-dimensional proprioceptive inputs, extensive training data, and complex policy architectures which introduce inefficiencies in both learning and deployment, limiting the scalability of these methods for real-world applications. Furthermore, these approaches are reliant on expensive actuators and high-accuracy sensors. This raises the challenge of developing more efficient, generalizable, and data-efficient locomotion policies that are robust to less sensor readings and actuator control, without compromising stability.

A fundamental aspect of policy efficiency is the **dimensionality of the observation space**. Many RL-based locomotion policies incorporate redundant or weakly relevant sensory inputs, uselessly increasing model complexity and training requirements. When transferring policies to hardware, larger observation spaces mean more sensors are required, increasing the cost of hardware needed to run a given policy. Additionally, **data efficiency** plays a crucial role in making RL-based locomotion viable for real-world deployment, where collecting large datasets through physical interactions is often impractical. Finally, **policy structure**, including the choice of learning framework and optimization strategies, can significantly impact generalization and robustness.

In this work, we survey a number of approaches to the optimization of humanoid locomotion policies along these dimensions. We establish a baseline walking policy trained with Proximal Policy Optimization (**PPO**) (Schulman et al., 2017) and perform feature importance analysis to determine the relative importance of observation dimensions to final policy outputs. Additionally, we explore Temporal Difference Model Predictive Control (**TDMPC2**) (Hansen et al., 2024) and Versatile Motion Priors (**VMP**) (Serifi et al., 2024) as alternative approaches whose more explicit modeling of the environment could make up for the reductions in observation data. While we were unable to outperform PPO in our experiements with these alternative approaches, they provide valuable insights into the trade-offs between model complexity, learning efficiency, and policy robustness.

## 2. Background

For our baseline humanoid locomotion policy, PPO was an obvious choice given its established record of robust performance across diverse reinforcement learning tasks and its data efficiency (Schulman et al., 2017). Its ability to leverage learning from off-policy data is particularly valuable when no large preexisting datasets are available, and instead data must be collected via computationally expensive policy rollouts in simulation.

In addition to PPO, we experimented with Versatile Motion Priors (VMP) (Serifi et al., 2024) to train a minimal locomotion policy. VMP leverages unstructured motion data by training a variational autoencoder to extract a latent kinematic motion space that captures the essential dynamics of natural movement. We hypothesized that explicitly learning these latent kinematic representations could mitigate the challenges of reduced input data in a minimal policy setting, thereby providing a structured prior that guides the policy toward more natural and efficient locomotion.

We also evaluated Temporal Difference Model Predictive

Control (TDMPC) (Hansen et al., 2024) as a model-based alternative to PPO. TDMPC uses a task-oriented latent dynamics model for short-horizon trajectory optimization and a terminal value function for long-term return estimation, both learned jointly via temporal difference learning. By focusing on reward-centric latent representations rather than modeling irrelevant environmental details, the hypothesis was that TDMPC could yield strong sample efficiency and adaptability to locomotion. However, its computational overhead during planning and reliance on accurate reward signals posed challenges in training the agent to walk.

We intended to integrate VMP's self-supervised motion encoding (stage 1) with TDMPC-2's reinforcement learning framework (stage 2) [1]. Our goal was to combine VMP's robust motion encoding with TDMPC-2's efficient trajectory optimization and long-term return estimation. While this hybrid approach appeared promising for balancing generalization and minimal proprioceptive dependency, we did not proceed with its implementation due to time constraints and technical challenges with each framework.

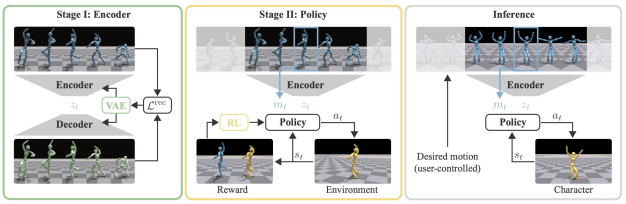


*Figure 1.* VMP Framework

# 3. Approach

## 3.1. Simulation and Hardware Considerations

For our project, we trained policies targeting the Zbot from Kscale labs. For its relative simplicity, we chose to train policies in the Genesis simulator, leveraging Kscale's Genesis training repo. After training in Genesis, the next steps were to test our policies in a sim-to-sim transfer on kos-sim, a Mujoco-based simulator that more accurately modeled the dynamics and actuator control methods of the actual Zbot. As we will discuss later, we faced struggles training a policy that could succeed in the sim-to-sim transfer, so we were unable to pursue our planned final step of deploying our policy to actual harware.

## 3.2. Zbot Overview

Zbot is a lightweight humanoid robot designed by Kscale Labs for research in RL-driven locomotion. It features a (relativey) low-cost yet functional actuation system, making it an ideal platform for testing minimal humanoid locomotion policies. Compared to larger humanoids, Zbot's design prioritizes efficiency, modularity, and robustness, enabling rapid iteration and evaluation of locomotion strategies.

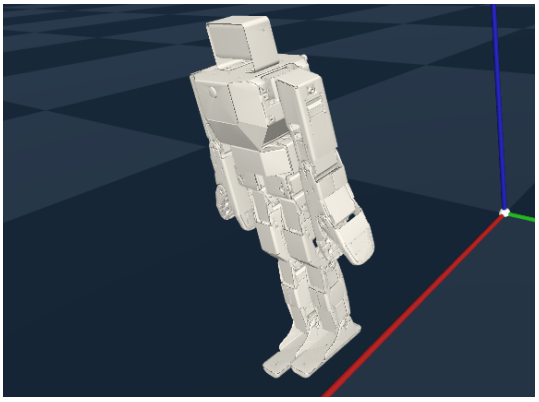

*Figure 2.* Example of Zbot in Genesis

## 3.3. Baseline Training with PPO

To establish a reference point for performance, we trained a baseline walking policy using PPO in the Genesis simulator.

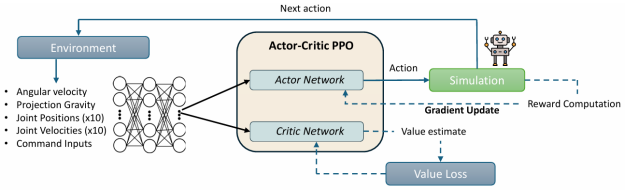


*Figure 3.* PPO Diagram

This baseline serves as a benchmark for evaluating subsequent modifications aimed at reducing proprioceptive inputs while maintaining locomotion stability. Training was conducted with default parameters reflecting the hardware constraints and sensor configurations of real-world deployment. The policy was trained for 500 iterations in a high-parallelization setup with 4096 environments, leveraging **adaptive learning rate scheduling and entropy regularization** to promote stable learning dynamics. The neural network architecture consisted of three hidden layers (512, 256, 128 neurons each) with ELU activations, optimizing both actor and critic networks jointly. Our reward function contains components for **linear and angular velocity tracking**, base height stability, and action regularization to encourage smooth, energy-efficient motion.

To monitor learning progression, we analyzed **total reward trajectories, episode lengths, and individual component rewards** over training iterations. As shown in Figure [13], the policy demonstrated steady improvement, converging to a stable locomotion pattern. Interestingly, after a prolonged plateau, the mean total reward exhibits a late-stage increase, which is coming from **feet air time**. Upon qualitative analysis of the policy rollout renderings, we have determined that

the policy is learning to reward hack this reward component by standing on one foot. This reward hacking happened reliably towards the end of 500-iteration training runs, so we chose to run evaluations on the 300-iteration checkpoints for all learned policies. For our baseline, the learned policy at iteration 300 is a stable walking policy, which tracks the desired commands as intended. A rendering of our baseline policy is included in our supplementary materials as **baseline_policy.mp4**.

These baseline results provide a foundation for subsequent experiments reducing observation space and optimizing model efficiency while preserving robust locomotion behavior.

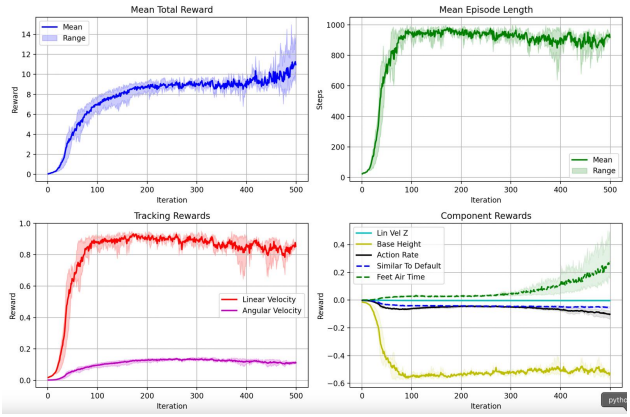


*Figure 4.* Baseline Analysis for Zbot with PPO

### 3.4. Versatile Motion Priors

We then implemented the VMP algorithm, which introduces a two-stage processing pipeline to control physics-based characters. The first stage involves extracting a kinematic latent space using a Variational Autoencoder (VAE), while the second stage trains a dynamics-aware policy using Reinforcement Learning.

Our VAE implementation learns a compressed representation of short motion windows, capturing the temporal and spatial structure of the motion distribution. Our model encodes motion windows $M_t$ of length $2W + 1$, where we used $W = 7$ to create 15-frame windows. The latent dimension was set to 64 with a $\beta$-VAE formulation using $\beta = 0.05$ to balance reconstruction quality with latent space regularization.

Specifically, the encoder of the VAE, $e_\psi(z_t|M_t)$, maps the motion window $M_t$ to a distribution over latent codes $z_t$. We model this as a multivariate Gaussian distribution:

$$q_\psi(z_t|M_t) = \mathcal{N}(\mu_\psi(M_t), \sigma^2_\psi(M_t)) \tag{1}$$

where the mean $\mu_\psi$ and variance $\sigma^2_\psi$ are functions of the encoder's parameters $\psi$. The decoder reconstructs the motion window from the latent code, i.e $M'_t = d_\phi(z_t)$.

The loss function for training the VAE consists of two terms: a reconstruction loss $L_{\text{rec}}$ and a KL-divergence regularization term $L_{\text{KL}}$:

$$L_{\text{rec}}(M_t, M'_t) = \frac{1}{2W+1} \sum_{i=t-W}^{t+W} l_{\text{rec}}(m_i, m'_i) \tag{2}$$

where the per-frame reconstruction error is given by:

$$l_{\text{rec}}(m_i, m'_i) = \|v_i - v'_i\|^2 + \|s_i - s'_i\|^2 \tag{3}$$

where $s_t$ are the joint positions and $v_t$ the joint velocities. In the original function from the paper, there were additional parameters that we had to remove due to the structure of provided motion data. The KL-divergence loss enforces a standard normal prior on the latent space:

$$L_{\text{KL}}(q_\psi(z_t|M_t) \parallel p(z_t)) = D_{\text{KL}}(\mathcal{N}(\mu_\psi, \sigma^2_\psi) \parallel \mathcal{N}(0, I)) \tag{4}$$

The total loss function for the VAE is then:

$$L_{\text{VAE}} = L_{\text{rec}} + \beta L_{\text{KL}} \tag{5}$$

Figure 5 shows the training progression of our best performing VAE model. The model achieves convergence after approximately 300 epochs, with both training and validation losses stabilizing, indicating that the model has learned to effectively encode motion windows.

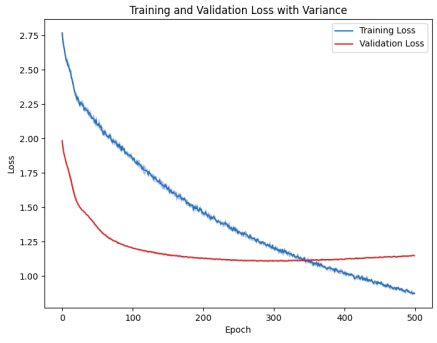


*Figure 5.* Training and validation loss for the VAE model over training epochs.

The reconstruction errors through different training stages can be found in Figures 6 and 7. As shown, the reconstruction quality significantly improved between epoch 50 and epoch 500.

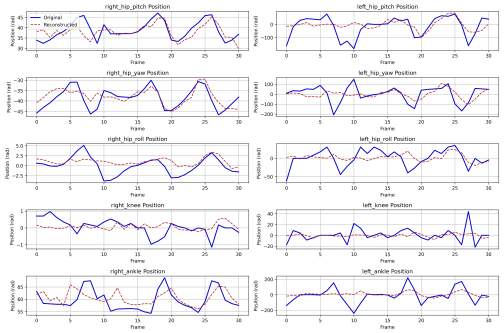


*Figure 6.* Reconstruction error in epoch 50

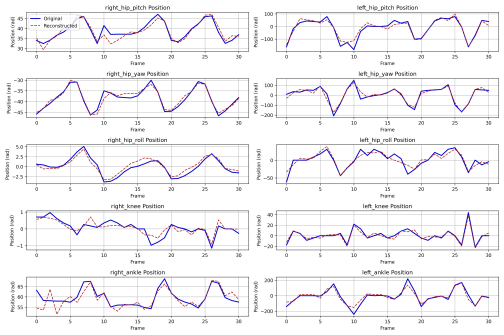


*Figure 7.* Reconstruction error in epoch 500

In the second stage our policy was trained using an extended version of base PPO to handle conditional inputs from our VAE's latent space. The reinforcement learning objective for training the policy was formulated as:

$$\begin{aligned} \text{maximize}_\theta \quad & \mathbb{E}_{s_t\sim\rho_\pi, a_t\sim\pi_\theta(\cdot|s_t,z_t)}[R(s_t,a_t,s_{t+1},z_t)] \\ \text{subject to} \quad & s_{t+1} = f(s_t,a_t) \\ & z_t \in \mathcal{Z} \quad \text{(latent space)} \end{aligned}$$

Where $\pi_\theta$ is the policy parameterized by $\theta$, $\mathcal{Z}$ is the VAE's latent space, and $R$ is the reward function. The policy takes actions based on both the current state $s_t$ and the latent code $z_t$ representing the motion window. We experimented with different reward formulations. Our initial implementation used a reward function inspired by (Serifi et al., 2024):

$$\begin{aligned} R(s_t,a_t,s_{t+1},z_t) =& c^{track}\cdot r_{track} + c^{alive}\cdot r_{alive} + \\ &+ c^{smooth}\cdot r_{smooth} \end{aligned}$$

Where $r_{track}$ measures the fidelity of tracking the reference motion, $r_{alive}$ incentivizes the character to maintain balance, and $r_{smooth}$ penalizes high-frequency motions and excessive torques. However, despite achieving reasonable reconstruction quality in the VAE stage, models trained with this reward function didn't show any improvement in training.

We then decided to test with a simplified reward function to prioritize essential stabilization objectives:

$$R(s_t,a_t,s_{t+1},z_t) = w_1\cdot r_{height} + w_2\cdot r_{progress}$$

The modified reward formulation was used in subsequent experiments, as shown in Figure 18.

While the new policy exhibited some improvement in early training phases, achieving increased rewards, it remained unstable during evaluation in simulations, frequently failing to sustain balance. This suggests persistent challenges in leveraging the VAE's latent space to ensure long-term motion coherence.

### 3.5. Temporal Difference Model-Predictive Control

In parallel to VMP, we implemented TDMPC, which combines model-based and model-free reinforcement learning through a framework leveraging a learned latent dynamics model and a terminal value function. The approach uses short-horizon trajectory optimization to plan locally optimal actions while estimating long-term returns via temporal difference learning. This enables efficient policy learning without modeling irrelevant environmental details.

During training, it minimizes a temporally weighted objective:

$$\mathcal{J}(\theta;\Gamma) = \sum_{i=t}^{t+H} \lambda^{i-t}\mathcal{L}(\theta;\Gamma_i)$$

Where $\Gamma \sim \beta$ is a trajectory $(s_t, \boldsymbol{a}_t, r_t, s_{t+1})_{t:t+H}$ sampled from a replay buffer $\beta$, and $\lambda$ is a constant that weights near-term predictions higher, and the single-step loss:

$$\mathcal{L}(\theta;\Gamma_i) = c_1 \underbrace{||R_\theta(\boldsymbol{z}_i,\boldsymbol{a}_i) - r_i||_2^2}_{\text{reward}} \tag{6}$$

$$+ c_2 \underbrace{||Q_\theta(\boldsymbol{z}_i,\boldsymbol{a}_i) - (r_i + \gamma Q_{\theta^-}(\boldsymbol{z}_{i+1}, \pi_\theta(\boldsymbol{z}_{i+1})))||_2^2}_{\text{value}} \tag{7}$$

$$+ c_3 \underbrace{||d_\theta(\boldsymbol{z}_i,\boldsymbol{a}_i) - h_{\theta^-}(s_{i+1})||_2^2}_{\text{latent state consistency}} \tag{8}$$

The loss jointly optimizes for reward prediction, value prediction, and a latent state consistency loss that regularizes the learned representation. $c_{1:3}$ are constant coefficients balancing the three loses. Predictions are recurrent and made entirely in latent space.

For trajectory optimization, we used Model Predictive Path Integral (MPPI) control to iteratively refine action sequences over a finite horizon. At each decision step, actions were sampled from both the learned policy $\pi_\theta$ and the MPPI distribution, with returns estimated based on the learned model and terminal value function. The optimization process was guided by the top-performing trajectories, balanc-

ing exploration and exploitation through adaptive variance constraints.

While TDMPC showed promise in theory, we encountered challenges in its application to locomotion. Specifically, the robot frequently exploited the reward function by performing non-generalizable behaviors like raising its foot without meaningful locomotion progress, highlighting potential reward hacking issues. These observations suggest difficulties in aligning the learned latent dynamics with robust control objectives over extended sequences. Also, given the time limitations, it is possible that we did not train the latent encoder long enough to learn a useful embedding. Hence, our original plan to combine it with the learned encoder from VMP.

### 3.6. Feature Importance Analysis

Since VMP and TDMPC didn't show promising results, we continued our analysis on PPO. To quantify the contribution of different input features to policy decision-making, we conducted a **sensitivity-based feature importance analysis** to provide insights into potential observation space reductions.

To improve the reliability of sensitivity estimates, we implemented several refinements. Instead of using random perturbations, we adopted a **fixed central difference approach**, ensuring numerical stability and eliminating variance due to stochastic sampling. Additionally, we standardized base observations by using the **mean observation state** as the default reference point, supplemented with a multi-point analysis selecting representative states across training runs. Explicit random seed control was enforced to enhance reproducibility. The aggregated feature ranking, presented in Figure 8, highlights the dominance of **joint linear velocities, base angular velocities, and projected gravity components**. While features related to **most hip yaw states show lower sensitivity**.

However, variability in feature importance across observations indicated by min-max error bars, as show in Figure 15, suggests that some features exhibit fluctuating influence due to policy exploration dynamics. A critical improvement in this study was the **phase-aware analysis of feature sensitivity**. Because locomotion dynamics vary across the gait cycle (Figure 16) , aggregating importance scores across entire trajectories may obscure meaningful variations. To address this, we segmented training data into distinct gait phases—including stance, swing, and transitions—and computed feature importance separately for each phase. This decomposition, shown in Figures 9 and 14, reveals how different sensory inputs fluctuate in influence depending on the robot's locomotor state. These findings will directly inform the design of minimal observation policies.

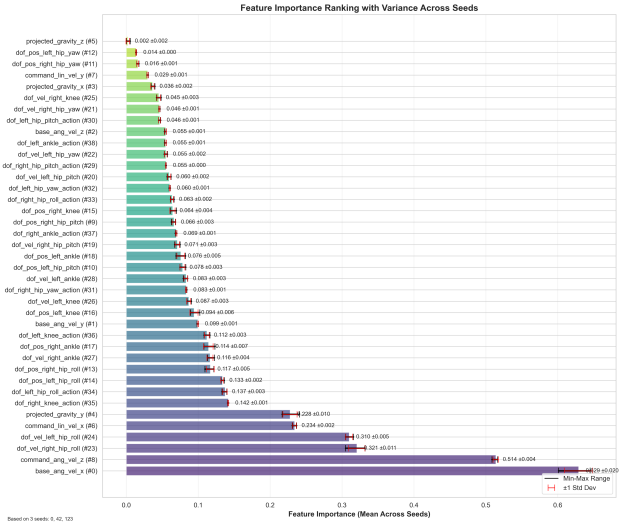


*Figure 8.* Feature importance for Zbot with PPO

### 3.7. Minimal Policy Exploration

To investigate the minimal set of proprioceptive inputs required for stable locomotion, we conducted a reduced observation space experiment. This involved training a policy using with a subset of sensory inputs, selectively removing features identified as less influential in our learned policy analysis of the baseline model.

This involved training policies with the the top 34 and top 30 features of the baseline policy. When we trained model with reduced observations, the observations were not masked, they were fully excluded from the observation vector, reducing its dimensionality. The critic model was always trained with full-size observation vectors, regardless of the reductions done to the actor's observations.

### 3.8. Sim to Sim Policy Transfer

With the ultimate goal of deploying a policy onto the Zbot hardware, we also made attempts to deploy our policy trained in Genesis onto Mujoco-powered kos-sim platform. KOS is the Kscale operating system for controlling the Zbot, and kos-sim is a simulator which mimics the HW dynamics and control as closely to the real world as possible. In order to perform the sim-to-sim transfer, we first had to train policies using a different URDF than the one we had trained our baseline and first minimal policy attempts with. This new URDF more acurrately modeled the actual Zbot hardware, introducing more realistic actuator torque limits.

As we will discuss later in the Results section, the transition to a new URDF halted our progress on minimal policy exploration, instead forcing us to shift focus to reward function

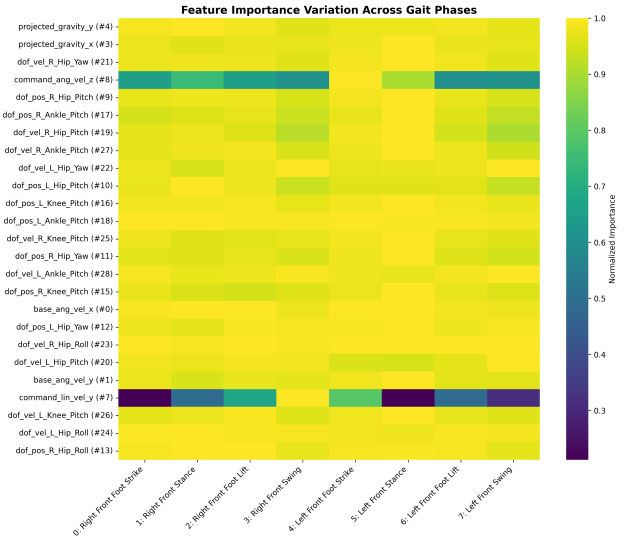


*Figure 9.* Heatmap of feature importance across different gait phases

experimentation. By the time we had a good policy trained on the new URDF, we had run out of time to try sim-to-sim or sim-to-real.

## 4. Experiment Results

### 4.1. Minimal Policy Exploration

Our initial (pre-milestone) exploration of minimal policies yielded unimpressive policies, when compared to our baseline. As shown in figure 10, training metrics seemed to indicate that the minimal policies perform on-par with the baseline policy (they also begin to reward hack **feet air time** towards later iterations). However, the evaluations suggest otherwise. Qualitatively, when running the three policies with a simple fixed command instructing the robot to walk forward at 50% speed, the **top_34** policy delivers a coherent walking policy, but with a noticeable instability not present in the steady gait of the baseline policy.

The **top_30** policy is even worse. This policy has an unstable gait, and struggles to follow the command to walk forward. While the **top_34** policy seems alright when asked to walk forward, both minimal polices substantially underperform the baseline when run through our quantitative evaluation, in table 1. For this evaluation, we ran 30 rollouts for each trained policy, with a randomly sampled command for each episode.

After switching to the new URDF, we initially struggled to train any successful walking policies, even with the same training setup used for the initial baseline. This sparked the reward function exploration discussed in the next section.

*Table 1.* Evaluation Results

| Metric | Baseline | Top_34 | Top_30 |
|---|---|---|---|
| Mean Reward | $11.16 \pm 5.34$ | $3.03 \pm 5.18$ | $2.21 \pm 3.80$ |
| Min Reward | 0.59 | $-0.01$ | 0.10 |
| Max Reward | 25.89 | 18.55 | 17.35 |
| Median Reward | 9.90 | 0.39 | 0.60 |
| Mean Ep. Len. | $939.9 \pm 228.7$ | $258.0 \pm 377.7$ | $151.4 \pm 243.0$ |
| Min Ep. Len. | 81 | 20 | 15 |
| Max Ep. Len. | 1001 | 1001 | 1001 |
| Median Ep. Len. | 1001.0 | 49.5 | 44.5 |

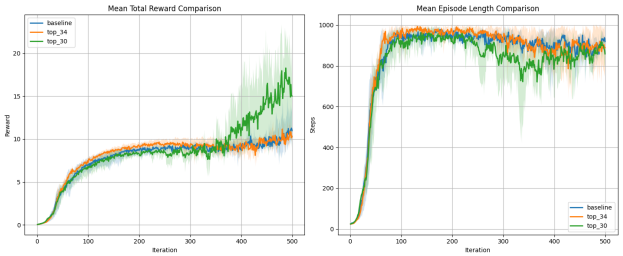


*Figure 10.* Mean total reward per episode and mean episode length during training across different observation sizes

This exploration was fruitless, and eventually proven unnecessary as we eventually, managed to successfully train a new baseline policy using the same reward function as the original baseline policy.

After establishing a baseline with the new URDF, and informed by our feature rankings in figure 8 and feature heatmap across gait in figure 9, we removed the least important positional and velocity joint vectors from our input space to train, successfully, our the top 34 most relevant features (**top_34_new**). The simulated walking behavior was indistinguishable from our baseline. We then took this a step further and trained a policy with the top 29 features (**top_29_new**), but interestingly this time, the policy learned to hack our reward function by holding its legs out and up from the ground with its arms. Thus, failing to achieve the desired walking behavior

### 4.2. Reward Function Exploration

The change in URDF presented a roadblock to minimal policy exploration and sim-to-sim transfer, as we initially stuggled to train even a full-observation policy using the new URDF. While the reward function experiments were ultimately fruitless, and the whole exploration rendered pointless after we managed to train a new baseline policy and minimal policy using our initial reward function for-

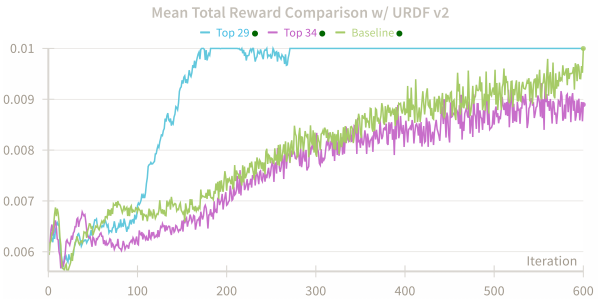


*Figure 11.* Total reward per iteration during training across different observation sizes with the new URDF

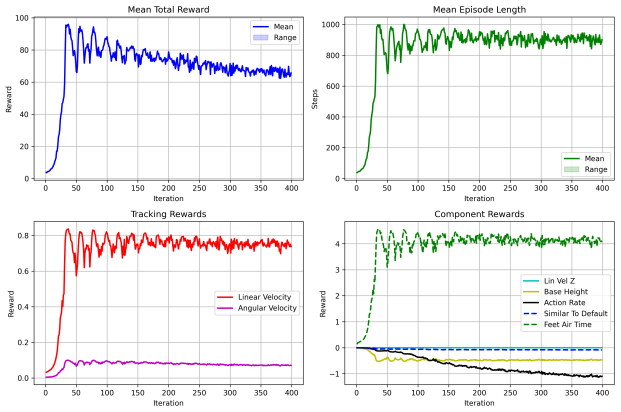


*Figure 12.* Training performance during our initial attempt to train a policy with the new URDF

mulation, considerable time was spent exploring alternative reward functions.

The components in our baseline reward function were:

- $R_{lin\_vel\_tracking}$ - Linear velocity tracking
- $R_{ang\_vel\_tracking}$ - Angular velocity tracking
- $R_{z\_vel}$ - Term penalizing any z-axis velocity
- $R_{action\_rate}$ - Term penalizing changes in actions across time steps (to encourage smoother actions)
- $R_{similar\_to\_default}$ - Term penalizing joint positions far from the default
- $R_{base\_height}$ - Term penalizing deviations from base height target
- $R_{feet\_air\_time}$ - Term rewarding feet air time

Initial training with the new URDF produced policies which reported good episodic returns on paper, even outperforming the rewards earned by our baseline, but which failed to perform the desired task.

As seen in the chart above, the policy quickly learns a high-reward policy and actually decays in performance over the course of training. The produced policy simply step in place, hacking the feet air time without actually making any forward progress. A rendering of this policy can be found in our supplementary material as **new_urdf_rew_hacking_policy.mp4**.

Many approaches were taken to modify the reward function such that it couldn't be hacked, with no luck. Initially, we reweighed components so that **linear velocity tracking** was weighed more than **feet air time** in the reward function.

After removing the **feet air time** reward component altogether, we noticed the zbot was still accumulating **linear velocity tracking** without making forward progress. To address this, the linear tracking reward was modified to compare a moving average linear velocity against the target, rather than instantaneous velocities as we had been doing. The moving averages were calculated via an exponential moving average, as maintaining velocity measurements and recomputing actual averages each timestep proved too computationally expensive. The moving average formula used was:

$$\text{EMA}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$

We experimented with alpha values of $\alpha = 0.182$ (approximating a 10 timestep window, equivalent to 0.2s) and $\alpha = 0.0392$ (approximating a 50 timestep window, equivalent to 1s), but these policies still reward hacked. We modified the linear tracking to compute reward based on error relative to the command magnitude, and added penalty for motion in the opposite direction to the target motion. We tried a number of other less noteworthy reward function modifications to resolve the hacking, still with no luck.

After a discussion with Emma during the poster session, where we discussed how to get more exploration during training given that our policies were converging to reward hacking approach extremely quickly, we introduced randomization to the actuator starting positions and decreased the learning rate. This finally led to training which seemed promising:

Unfortunately, this policy seems to be hacking the linear velocity tracking by earning partial rewards spinning in a circle, as seen in **random_starts_policy.mp4** in our supplemental materials.

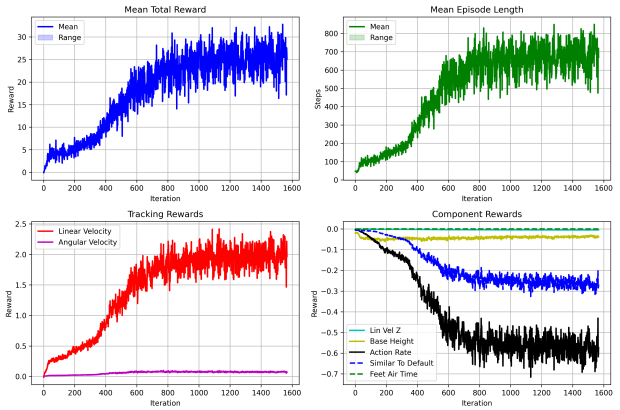


*Figure 13.* Training performance with randomized start positions and decreased learning rate

## 5. Conclusion

Many of our training attempts across PPO, VMP, and TDMPC produced policies which learned to hack our reward functions rather than produce the desired behavior. Given the complexity of VMP and TDMPC, we believe that validating the algorithms on simpler tasks would have been useful for confirming their correctness before deploying them onto the more complex task of humanoid locomotion. Even though the trained VAE demonstrated the ability to learn a structured latent representation of robotic motion sequences, the RL stage encountered substantial difficulties in leveraging this representation for robust control. Given the limited input space, it is possible that the latent encoding did not provide additional meaningful information and instead acted as a source of noise, impairing the policy's learning process. In our approach, it was unclear whether there were issues in the formulation of our reward functions, or in the implementation of the algorithms themselves. Validating correctness on a simpler problem would have at least helped us control for one of those variables.

Despite the lack of results from reward experimentation, we were eventually able to train a viable baseline and minimal policy using the new URDF (on a different computer than the rest of the attempts to train on the new URDF). It's unclear why this training worked on one machine and not the other, given we shared training environments, but it is a further piece of evidence pointing to the fickle nature of training RL policies in high-dimensional state and action spaces with complex dynamics. Tiny changes in the dynamics of the environment can lead to collapse in performance of previously successful training approaches.

These positive outcomes demonstrate that, with a judicious reduction of the observation space informed by our sensitivity analysis, it is possible to maintain critical sensor inputs while streamlining the policy's decision-making process. Ultimately, these findings underscore the promise of adaptive training strategies that not only mitigate the sensitivity of RL methods to slight environmental variations but also pave a clearer path for robust propioceptive analysis in humanoid robotics.

We encourage future work to evaluate the sim-2-real transfer with the minimal policies and experiment with reward functions, hyperparameters, and scaling coefficients of VMP and TDMPC.

## 6. Code Access

The code we used for this project can be accessed here: https://github.com/manolo-alvarez/Minimal-Locomotion

Policy renderings can be found in our Supplementary Materials drive.

## 7. Contributions

- **Luciano Gonzalez (lucigon)**
  - PPO baseline training, minimal policy training, policy training for reward exploration
  - Initial sim-to-sim attempts
  - Code for analyzing and visualizing PPO training data
  - Better data collection and analysis for PPO Eval
  - Modifications to ZbotEnv for reward exploration and observation masking
- **Martina Del Gaudio (mdgaudio)**
  - Implemented Variational Autoencoder (VAE) training, validation, and visualization pipelines.
  - Developed ZbotEnv, zbot_train and zbot_eval modules for VMP.
  - Extended the PPO architecture to incorporate conditioning on latent motion windows, introducing a ConditionalActorCritic class for policy training.
  - Integrated the VAE and Zbot reinforcement learning setup to enable full VMP training.
  - Code for visualizing VMP training data.
- **Manolo Alvarez (manoloac)**
  - Implemented TDMPC training, validation, and visualization pipelines
  - Modified zbot env, training, and eval, files for TDMPC
  - Feature analysis code
  - Trained successful PPO baseline and minimal policy on new URDF
  - Final sim-to-sim attempts.

## References

Hansen, N., Su, H., and Wang, X. TD-MPC2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Oxh5CstDJU.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Serifi, A., Grandia, R., Knoop, E., Gross, M., and Bächer, M. Vmp: Versatile motion priors for robustly tracking motion on physical characters. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '24, pp. 1–11, Goslar, DEU, 2024. Eurographics Association. doi: 10.1111/cgf.15175. URL https://doi.org/10.1111/cgf.15175.

# A. Additional Figures

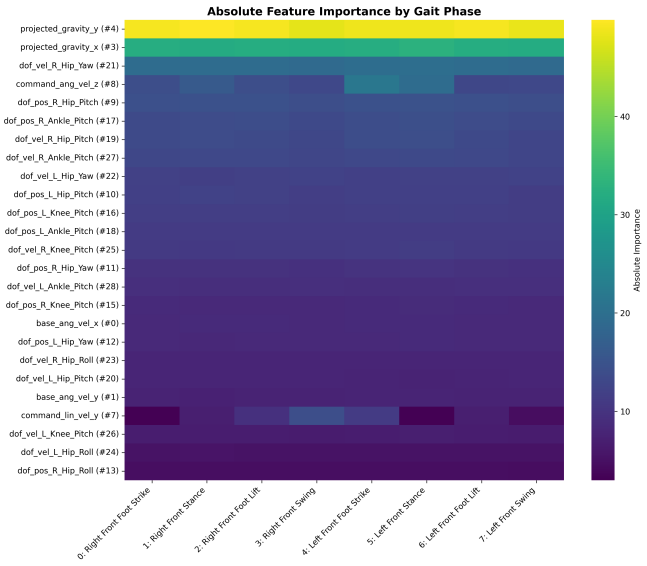


*Figure 14.* Heatmap of absolute feature importance across different gait phases

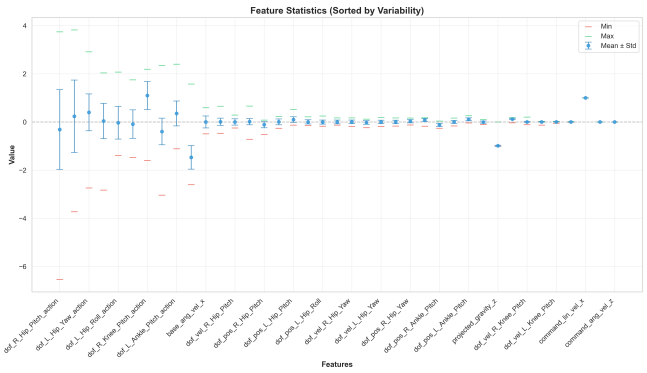


*Figure 15.* Feature statistics across runs

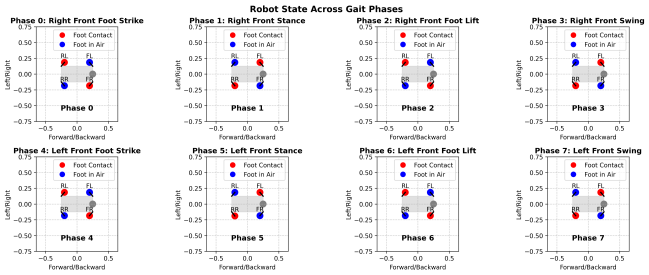


*Figure 16.* Gait phase visualization

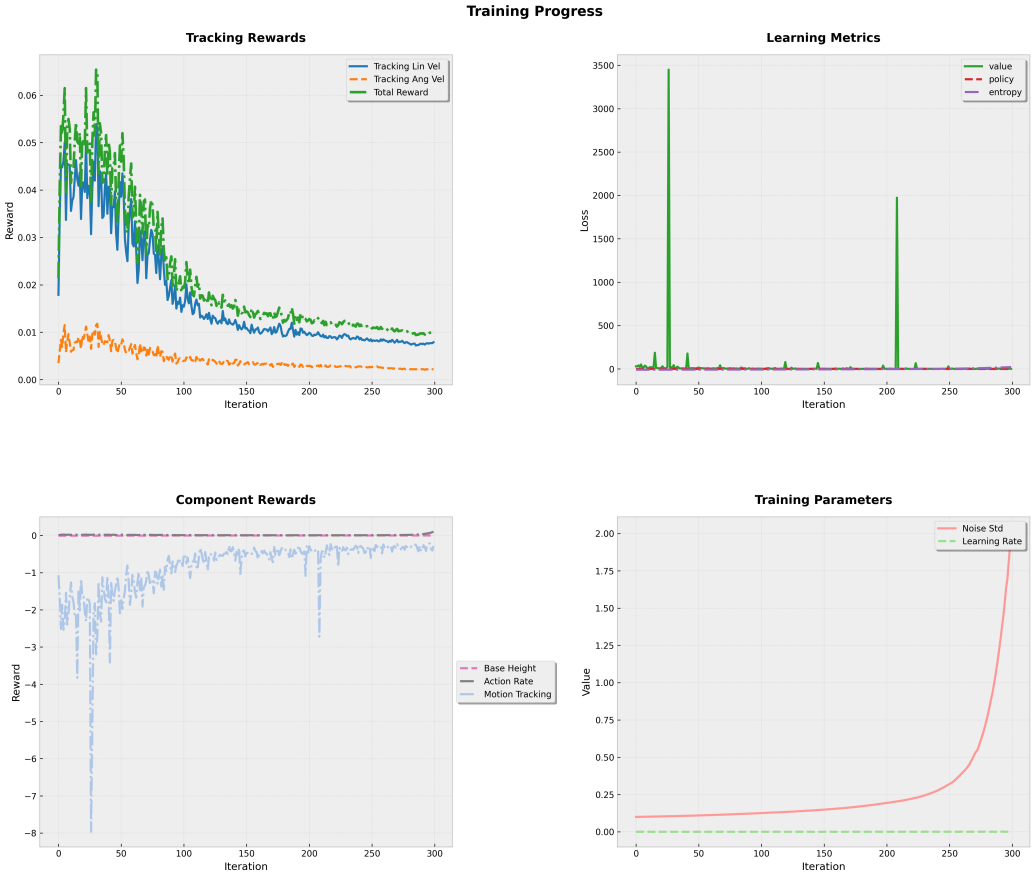


*Figure 17.* Training rewards In VMP approach for the complex reward function

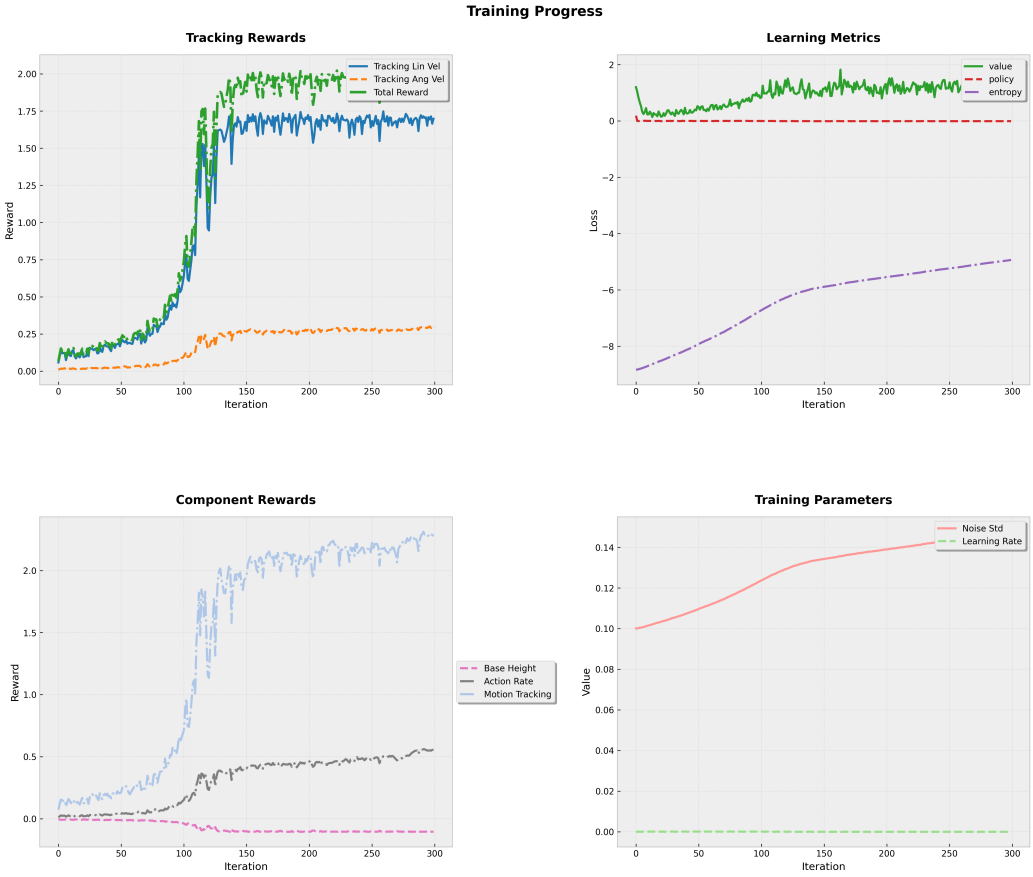


*Figure 18.* Training rewards in VMP approach for the simplified reward function