Learning Harpy Locomotion with Reinforcement Learning

Jorge Ortega, Patricia Meza

Northeastern University
CS 5180

April 19, 2025

1 Introduction

1.1 Motivation

Reinforcement learning (RL) offers a promising alternative to traditional control methods for robotic locomotion, as it learns directly from interaction rather than relying on precise modeling and simplifications. In this project, we explore whether RL can generate stable walking gaits for Harpy, a custom-built bipedal robot from the Silicone Synapse Lab. Harpy features an unusual design with a heavy torso and virtually massless legs, which poses unique challenges for control. While existing controllers rely heavily on mathematical simplifications such as perfect stiffness and idealized mass distribution, we use a physics-based MuJoCo simulation to investigate whether walking is feasible for this robot configuration and torque constraints using reinforcement leaning techniques. Our goal is to examine how reward shaping, mass distribution, and torque constraints influence gait quality, and what characterizes a 'good' walking strategy in this context.

1.2 Technical Problem Statement

We aim to develop an RL-based controller that enables stable and efficient forward walking for a robot with Harpy-like properties. Our hypothesis is that reinforcement learning can discover compensatory control strategies for morphologies that are difficult to model analytically. We investigate how the following factors influence learned gait quality:

- Reward structure (e.g., forward velocity vs. control effort)
- Mass distribution and overall morphology
- Torque limitations on joints
- Training time and number of parallel environments

2 Environment and Simulator

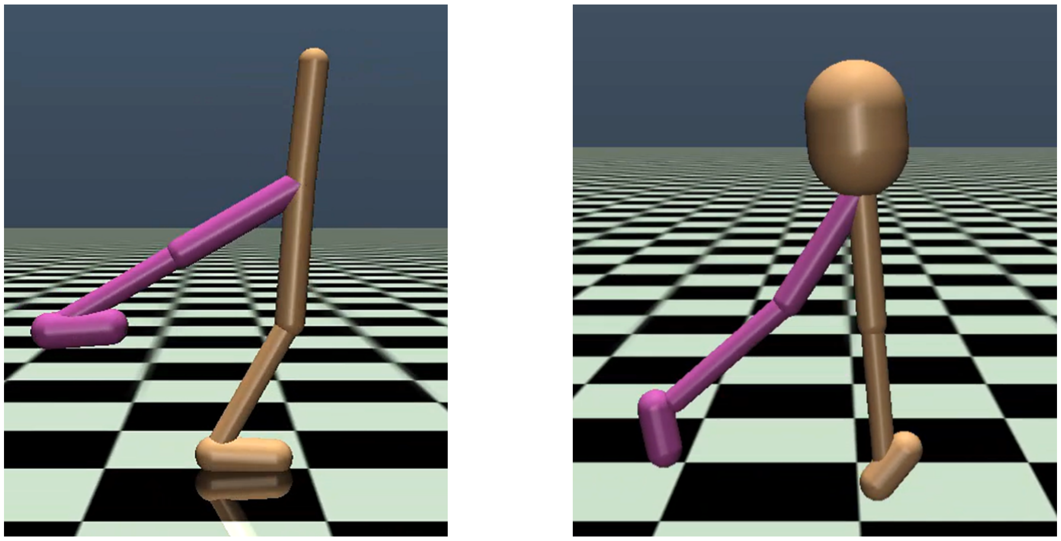
We used the `Walker2d-v5` environment from the Gymnasium MuJoCo suite as a base. This environment models a planar two-legged walker with six actuated joints and includes gravity, joint limits, and contact physics. Each leg consists of a thigh, shin, and foot, and the walker is rewarded for moving forward while staying upright [1].

Default Configuration

The default walker has a total mass of approximately 23.7 kg, with the weight evenly distributed over the torso, thighs, shins, and feet. All actuators are configured with a gear ratio of **100**, resulting in torque limits of ± 100 Nm.

Harpy Configuration

To better simulate the physical characteristics of the Harpy robot, we applied the following changes: The total body mass was reduced to 6 kg, with the torso set to 4.0 kg to reflect Harpy's compact design. The remaining segments were adjusted as follows: Thighs – 0.65 kg, Legs – 0.3 kg, Feet – 0.1 kg. Gear ratios were reduced to 20 for most joints and 10 for the ankle joints, resulting in torque limits of ± 20 Nm and ± 10 Nm, respectively. The feet were shrunk and repositioned slightly backward and downward to improve balance and mimic Harpy's morphology. The torso geometry was shortened (for aesthetic purposes).



Original Configuration for Walker2d-V5 Modified Configuration to resemble Harpy

Figure 1: Screenshot from Mujoco environment of both versions of the walker used in experiments.

2.1 State and Action Spaces

Observation (State) Space:

- 17-dimensional continuous vector containing the positions and velocities of all body parts
- `Box(min=-Inf, max=Inf, shape=(17,), dtype=float64)`

Action Space:

- 6-dimensional continuous vector representing torques applied at each hinge joint
- `Box(min=-1, max=1, shape=(6,), dtype=float32)`

2.2 Reward Function

The environment uses a shaped reward function that encourages forward movement and penalizes excessive control effort. We later modified these weights to study their effects on learning behavior.

$$\textbf{Reward} = \textbf{healthy\_reward} + \textbf{forward\_reward} - \textbf{ctrl\_cost}$$

- Healthy Reward: +1 per timestep if the agent remains upright and within joint limits
- Forward Reward: +1 $\times$ forward velocity
- Control Cost: $-10^{-3} \cdot \sum u^2$, where u are the action values (torques)

3 Methods

We experimented with two reinforcement learning algorithms: Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradient (TD3). PPO served as a baseline, while TD3 was explored in more depth with multiple environment and reward modifications.

3.1 Proximal Policy Optimization (PPO)

PPO is a policy-gradient algorithm designed to optimize a stochastic policy by maximizing a surrogate objective function, while ensuring stable updates. PPO introduces clipped objective to prevent the new policy from staying too far from the old one, which stabilizes training:

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

where $r_t(\theta)$ is the probability ratio between the new and old policy, and $\hat{A}_t$ is the estimated advantage.

We implemented our PPO agent using the Stable-Baselines3 (SB3) library, a PyTorch-based framework that offers reliable implementations of modern RL algorithms [2]. The training code was developed to run four parallel instances of the Walker2d-v5 environment using `SubprocVecEnv` from SB3, which improves training stability and efficiency. The policy network used is a multilayer perceptron (MlpPolicy) with two hidden layers, each containing 64 to 256 units (variated in different experiments) and ReLU activations. It includes Generalized Advantage Estimation (GAE), clipped policy updates, and advantage normalization to stabilize learning. Training metrics — including episode rewards, policy loss, value loss, and KL divergence — were logged and visualized using `TensorBoard`, providing real-time feedback on the learning process. All training was executed on a CPU-based system, utilizing SB3's efficient multiprocessing support.

3.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is an actor-critic method designed to reduce overestimation bias by using two critics and delaying policy updates:

$$Q_{\text{target}} = r + \gamma \cdot \min(Q'_1, Q'_2) \text{ with target policy smoothing}$$

The TD3 algorithm was implemented following a similar structure to the homework. We used PyTorch for the neural networks, where the actor and critic networks were constructed as three-layer fully connected feedforward architectures, with two hidden layers of 256 units each and ReLU activations. The actor outputs actions through a final Tanh layer, scaled by the environment's action limit. Two critic networks estimate Q-values independently to enable clipped double Q-learning, and their corresponding target networks are softly updated to stabilize learning. Optimization was performed using the AdamW optimizer. Target policy smoothing was implemented by adding clipped Gaussian noise to target actions. For exploration, we used an Ornstein–Uhlenbeck noise process. Training was accelerated using GPU (RTX 3060) computation via CUDA when available.

4 Experiments and Results

See PPO forwards walking experiments here (49 sec): https://youtube.com/shorts/89PpLuFsB4k
See TD3 forwards walking experiments here (1 min): https://youtu.be/DlBghiJG6jE
See TD3 backwards walking experiments here (20 sec): https://youtu.be/yGeIv_q2mSc

4.1 Hyperparameter Tuning (PPO)

The Stable-Baselines3 library provides a stable initial configuration of hyperparameters for continuous control tasks. However, several adjustments were made based on empirical observations of reward curves, value loss trends, and policy stability. Table 1 summarizes a selection of training configurations explored during experimentation. In addition to modifying hyperparameters, we also varied aspects such as reward shaping weights and torque settings. These experiments were designed to deepen our understanding of how the policy interacts with the environment. Throughout training, we used "TensorBoard" to monitor key metrics and detect signs of instability—such as reward plateaus, divergence, or high value loss—which guided further tuning and refinements.

Configuration	γ	Learning rate	Batch size	GAE λ	vf_coef	clip_range_vf	target_kl
Baseline	0.99	0.0003	64	0.95	0.5	None	None
FastRun_5M_2	0.99	0.0001	256	0.95	0.1	0.1	0.01
ModTor_5M_1	0.99	0.0005	128	0.95	0.2	0.1	0.01

Table 1: Comparison of hyperparameter configurations tested during PPO training.

Figure 2 shows the reward progression of three PPO training runs on the Walker2D-v5 environment, each with different hyperparameter configurations stated in the previous table. The Baseline represents the default SB3 configuration, while FastRun_5M_2 corresponds to a run with modified reward shaping aimed at encouraging faster forward movement. ModTor_5M_1 includes adjustments to torque gear to produce a more realistic walking.

Each training trial was run for 5 million timesteps, which took approximately 2 hours to complete using a CPU setup. The Baseline run demonstrated the most stable learning behaviour, with fewer fluctuations and more consistent improvement compared to the other configurations. Therefore, it served as the foundation for extended training runs, which will be shown later on.

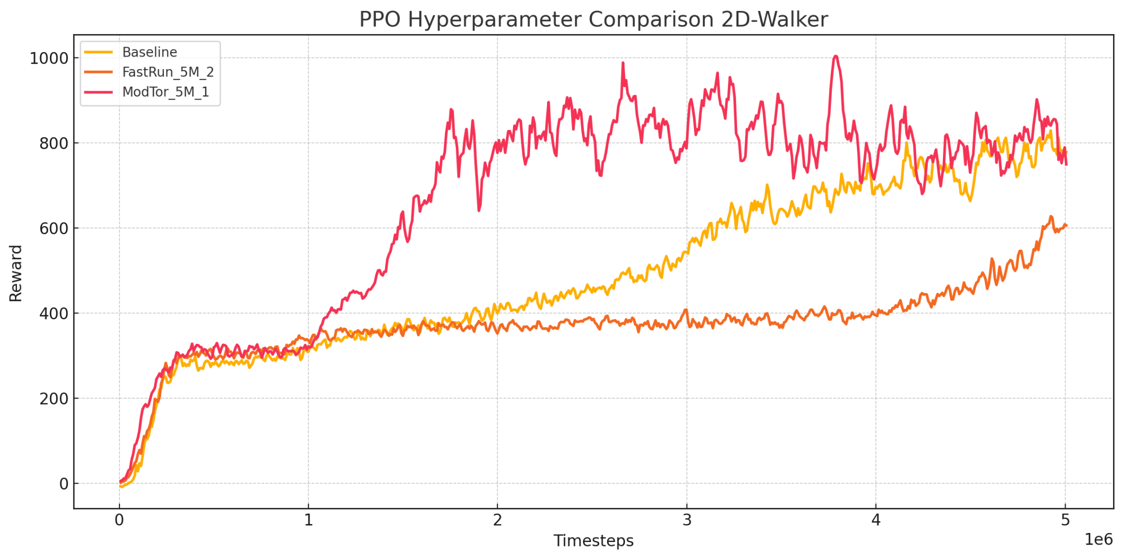
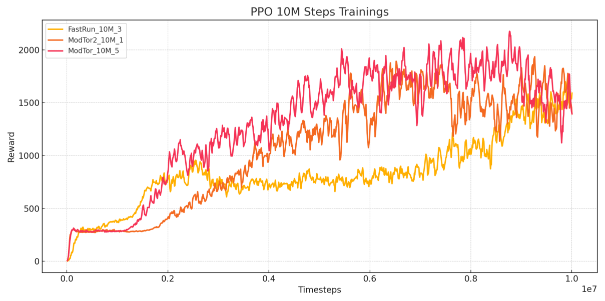


Figure 2: Reward progression across training for three PPO configurations in Walker2D-v5.

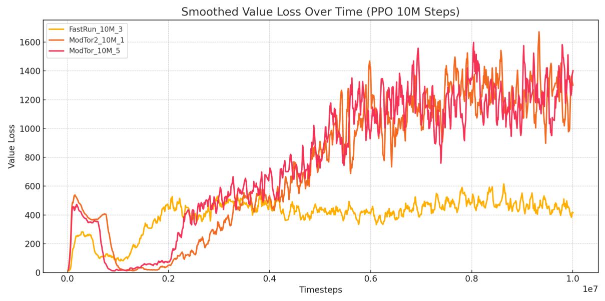
4.2 Full Experiments (PPO)

Each configuration was trained for 10 million timesteps which corresponds to an average of 4.3 hours. Although all agents achieved stable walking, the training dynamics revealed some inconsistencies, especially in the critic's behavior. The value loss plots showed substantial noise and lacked a clear convergence trend, particularly in the Modified-Torques runs. This was corroborated by the explained variance plot, which measures how well the value function predicts returns. While most runs maintained values above 0.94, Modified-Torques runs dropped below 0.90, suggesting reduced critic accuracy. An important factor contributing to this instability may be the rapid drop in entropy loss. A steep decline indicates that the policy became overly deterministic too quickly, likely converging to a narrow set of actions. When this happens, the agent might not explore alternative behaviors sufficiently, leading to limited data diversity to train the value function.

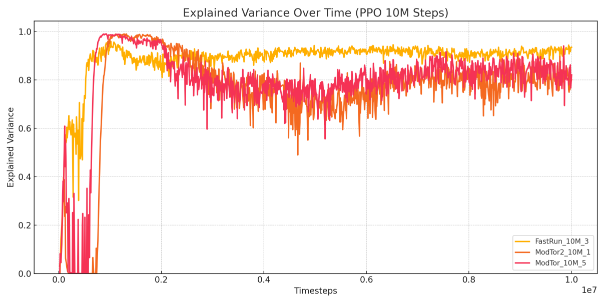
These results suggest that while the policy network successfully learned stable locomotion, the value function may require more regularization or architecture tuning for improved learning stability.



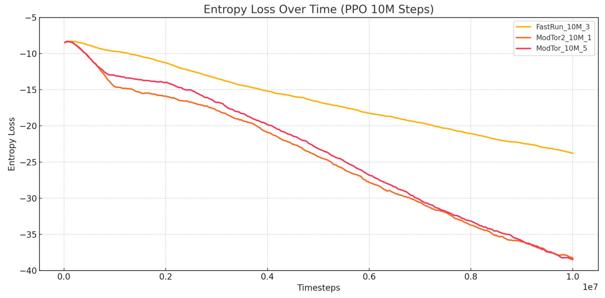
(a) Reward over time



(b) Value loss over time



(c) Explained variance during training



(d) Entropy loss over time

Figure 3: Training performance metrics for PPO on Walker2D-v5 (10M timesteps)

Due to compatibility issues with parallel environment training in Stable-Baselines3, it was not possible to directly modify the body geometry or mass properties of the MuJoCo Walker2D model during PPO experiments. As a result, adapting the agent to resemble the physical configuration of the Harpy robot required an indirect approach.

To approximate the Harpy's dynamics, the actuator torque limits were scaled proportionally based on mass. The real Harpy weighs slightly over 6 kilograms, while the default Walker2D model in MuJoCo weighs approximately 24 kilograms—about four times heavier. Therefore, in the final PPO training run labeled ModTor2_10M_1, the torque limits were decreased to reflect this difference. Specifically, the hip and knee joints were assigned a torque of 85, and the ankle joints were set to 20. These values are roughly four times greater than those used in the TD3 experiments, which were conducted with a Walker2D model modified to match Harpy's actual mass.

This torque scaling strategy allowed for more realistic motion dynamics within the PPO framework, despite constraints on modifying the robot's physical structure.

Table 2 shows the configuration used for the final PPO-trained policy, ModTor2_10M_1, which yielded the best gait performance among all tested runs. This configuration achieved the highest evaluation reward (exceeding 3400 per episode evaluated) and produced the most stable and natural-looking walking behavior during qualitative analysis.

Configuration	γ	**Learning rate**	**Batch size**	**GAE λ**	**vf_coef**	**clip_range_vf**	**target_kl**
ModTor2_10M_1	0.99	0.0003	64	0.95	0.5	0.2	0.05

Table 2: PPO Hyperparameters for the ModTor2_10M_1 Run

4.3 Hyperparameter Tuning (TD3)

Initial testing of the TD3 implementation began with a set of baseline hyperparameters. In general, these were the same values we used frequently in class examples, homework assignments, or that seemed the most intuitive. After confirming that this configuration produced stable learning, additional experiments were conducted with alternative values to encourage more exploration, increase training speed, or introduce more noise into the policy. All experiments were run using the same environment (Default 2D Walker-v5) and network architecture.

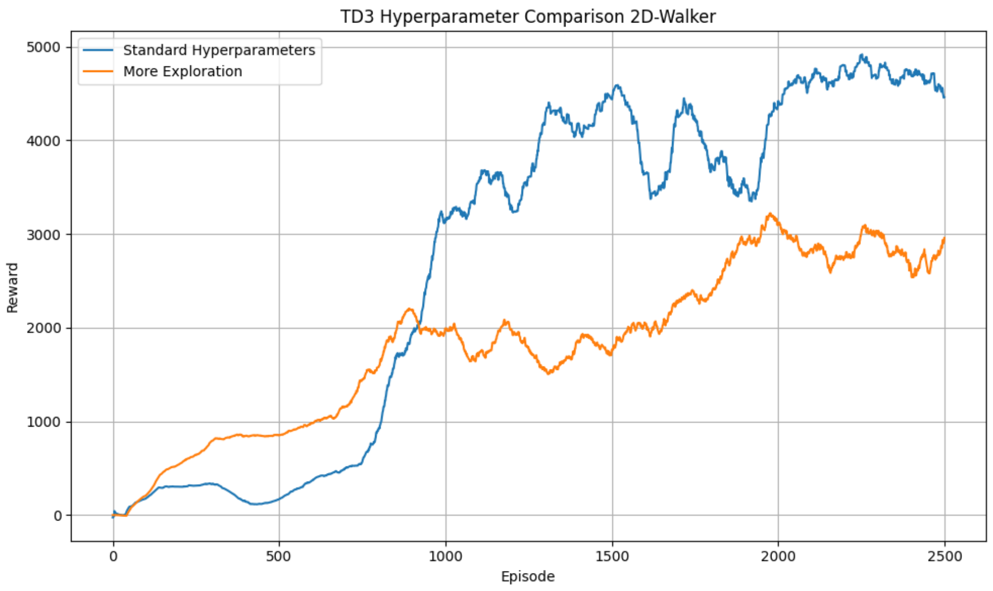


Figure 4: Moving average rewards vs episode for baseline and exploratory hyperparameters.

Each trial took approximately 2.5 hours to train. For the walker to achieve stable walking (which often resembled running) or a performance plateau, the agent typically needed to train for around 2000 episodes. Interestingly, although the exploratory configuration led to faster initial learning, the standard baseline parameters ultimately outperformed it in long-term stability and reward accumulation. The table below summarizes the hyperparameters used in each run.

Configuration	γ	τ	Noise σ	Policy Delay	Buffer Size	Batch Size	Max Episodes
Baseline	0.99	0.002	0.05	2	500,000	256	2500
Exploratory	0.98	0.005	0.20	2	500,000	256	2500

Table 3: Hyperparameters used for TD3 baseline and exploratory runs.

4.4 Harpy Configuration Experiments (TD3)

After tuning the hyperparameters and demonstrating that walking was feasible with the baseline configuration, the next step was to modify the agent's morphology to resemble that of the Harpy. The figure below compares several different Harpy-inspired configurations against the baseline from the previous section.

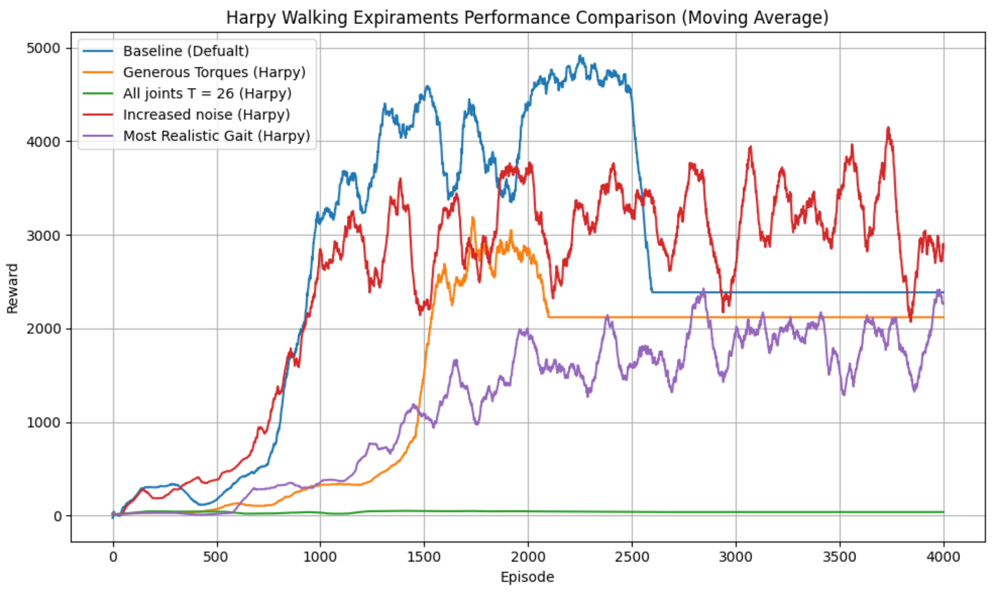


Figure 5: Moving average reward vs. episode for various Harpy-inspired morphology trials.

The average training time for each of these experiments was approximately 4.5 hours. The default Walker2D morphology used in the baseline trial has an unrealistically high force-to-weight ratio, which enables large leaps and exaggerated motion. As the Harpy model resembles the real hardware, a decrease in reward was expected due to more limited actuation.

In the first Harpy modification (orange curve), all joints were generously allowed up to 30 Nm of torque. This resulted in erratic jumping behavior without any sign of structured walking.

The second trial (red curve) increased exploration by adjusting the hyperparameters: τ was raised to 0.003 and the noise standard deviation to 0.06. This change was motivated by the observation that the previous configuration was slow to learn and heavily relied on ankle actuation while neglecting the knees. With these tweaks, the agent showed much better performance, converging around a reward of 3000—not far behind the baseline.

In the third trial (green curve), the torque was capped at 26 Nm for all joints. It continued to over-rely on its ankles to propel itself upward or just collaped, This trial failed to show learning.

In the final experiment (purple curve), the reward function was adjusted to encourage a more natural gait. The control cost was increased from 0.001 to 0.0015, promoting energy efficiency, and the forward velocity reward was reduced from 1 to 0.9 to discourage excessive running. Although the rewards were lower overall, the learning curve was smoother and more stable. This trial used the realistic torques: 20 Nm at all joints except for the ankles, which were limited to 10 Nm. The resulting gait appeared slower and more natural, aligning with the design goals for a Harpy-inspired walker.

4.5 Backwards Walking Experiment (TD3)

The final TD3 experiment explored whether the agent could learn to walk backwards by simply modifying the reward function. Specifically, we inverted the velocity reward weight from $+1$ to -1, encouraging negative forward velocity. Our prediction was correct—stable backward locomotion was successfully achieved in both the standard Walker2D configuration and the Harpy morphology.

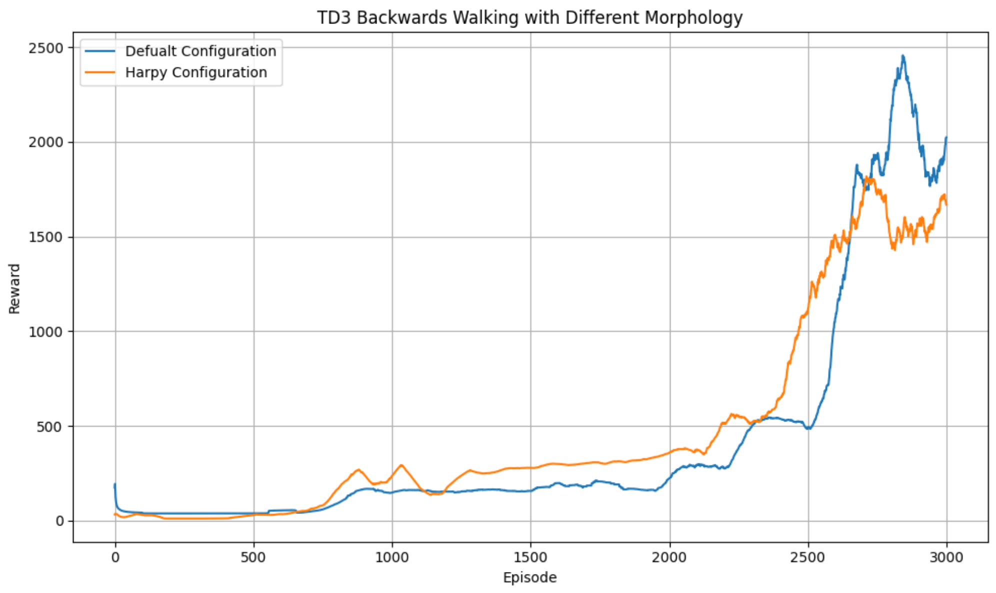


Figure 6: Moving average reward for backward walking trials with both morphologies.

The average training time for these trials was around 2 hours, primarily because most episodes were quite short. With more time, we expect the learning curves would have plateaued further. Interestingly, the Harpy morphology developed a more natural backward gait than the original Walker2D, despite slightly lower reward values. The default Walker configuration learned a somewhat unusual strategy, drifting backward using only ankle torque without taking any real steps. While this behavior was mechanically stable, it did not resemble walking. In contrast, the Harpy model engaged its knees significantly more than in the forward walking trials, resulting in one of the most natural-looking gaits observed throughout our experiments. This may be due to the reduced reliance on ankle torque when walking backward, which suits the Harpy's more biologically realistic actuation limits.

5 Analysis

Both Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradient (TD3) proved effective for learning stable locomotion. PPO, implemented with four parallel environments and guided by the Baseline PPO configuration served as a robust foundation, and tuning the reward structure and gear torque limits allowed us to control gait behavior effectively.

However, the best performing runs overall were achieved using TD3, with average rewards exceeding 3000 in some configurations. TD3's continuous action formulation and use of double critics allowed for more precise Q-value estimation, while its delayed actor updates and target policy smoothing provided greater stability during training. Once hyperparameters were tuned and exploration noise properly adjusted, TD3 was able to discover smoother and more natural walking gaits—particularly in the Harpy-inspired configurations.

Despite achieving stable locomotion through the PPO implementation, some configurations exhibited highly unstable value losses. These curves were much noisier than expected and lacked clear convergence, suggesting that the critic struggled to fit the returns. Additionally, while reward tuning succeeded in shaping gait quality, some changes (e.g., torque reductions) made the learning process more brittle. In TD3 experiments, certain torque setups resulted in erratic or unrealistic jumping behavior, revealing sensitivity to physical constraints and the need for tighter coupling between policy structure and morphology.

6 Discussion and Future Work

6.1 Challenges

Aside from the value function instability previously described, the main challenge encountered in both algorithms was modifying the morphology of the Walker2D agent, such as changing link lengths, body mass and body geometry to more closely resemble the Harpy robot. In the PPO implementation, attempts to adjust the shape of the walker often resulted in persistent errors when using SubprocVecEnv, the parallel environment wrapper in Stable-Baselines3. These issues seemed to stem from limitations in serializing custom MuJoCo models across subprocesses, which made it difficult to run modified environments in parallel. As a result, the agent customization was deferred to the TD3 setup and only the gear ratios were changed in the PPO approach.

6.2 Future Directions

This project was a solid first step towards exploring Harpy's RL implementation, but there is still plenty of room for improvement. Changing torques and masses helped approximate the physical limits of the real robot, but more detailed adjustments such as tuning joint speeds, friction, and range of motion would make the behavior more closely resemble the actual hardware. Incorporating a URDF model or at least adding Harpy's linkage joint would also enhance realism. As a next step, moving to a three-dimensional environment and integrating thruster-based control along with a more advanced reward structure, where a desired velocity or position is specified, could enable more dynamic behaviors such as larger jumps or thruster-assisted leaps. This would bring the simulation even closer to the robot's intended design.

References

[1] F. Foundation, "Walker2d," https://gymnasium.farama.org/environments/mujoco/walker2d/, 2025, accessed: March, 2025.

[2] Stable-Baselines3, "Ppo — proximal policy optimization," https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html, 2025, accessed: March, 2025.