Physics-Based 3D Humanoid Locomotion Using Proximal Policy Optimisation

Irene Maria Thomas[1], Shrimant Kamalakar Madbhavi[1], Vyshakh Vijayan Nambiar[1]

[1]*Department of Artificial Intelligence and Data Science, ISBM College of Engineering, Savitribai Phule Pune University, Pune, India.*

Keywords: Reinforcement Learning; Proximal Policy Optimization; Humanoid Locomotion; Unity ML-Agents.

Abstract: This paper presents a physics-based 3D humanoid locomotion system trained entirely via reinforcement learning using Proximal Policy Optimization (PPO) within the Unity ML-Agents framework. A multi-joint humanoid agent learns stable, goal-directed bipedal walking using proprioceptive observations including joint angles, angular velocities, body orientation, and binary foot-ground contact signals, without any hand-crafted controllers or motion capture data. The continuous action space drives 35 joint torques through PD control with joint limit constraints. A multi-term reward function promotes forward progress, upright posture, and contact discipline while penalizing falls and degenerate behaviors. Training employed 25 parallel agents over 2.8 million timesteps, completing in 1.23 hours. The final policy (ragdoll_smooth_v2) achieved a cumulative reward of +845.38, compared to approximately −12,000 at initialization, demonstrating significant policy improvement. The trained policy is exported as an ONNX model enabling real-time inference deployment. TensorBoard metrics including cumulative reward, episode length, policy loss, value loss, and curiosity loss confirm stable convergence and consistent gait emergence throughout training.

1. INTRODUCTION

Training articulated agents to achieve stable, goal directed locomotion in three dimensions remains a challenging problem in reinforcement learning. High dimensional continuous control, contact dynamics, and underactuated mechanics make bipedal locomotion particularly complex. In addition, poorly designed reward functions can lead to undesired behaviors such as crawling or backward motion instead of natural walking.

Traditional approaches such as zero moment point controllers and central pattern generators can produce stable gaits but lack adaptability across varying terrains and tasks, and require extensive manual tuning. Motion capture-based methods generate realistic movements but depend on expensive infrastructure and do not generalize well to new environments or disturbances. This work presents a learning-based approach in which a humanoid agent learns walking through interaction with a physics-based simulation using Proximal Policy Optimization. The proposed system removes the need for hand crafted controllers or motion data while enabling stable and goal directed locomotion. It includes a complete PPO based pipeline, an effective reward design.

2. RELATED WORK

Schulman et al. [2] introduced Proximal Policy Optimization as a stable and sample efficient on policy algorithm for continuous control. Its clipped objective constrains policy updates and prevents instability, making it well suited for high dimensional locomotion tasks. Kuindersma et al. [3] demonstrated real world humanoid locomotion using reinforcement learning controllers on Boston Dynamics platforms, highlighting the challenges of maintaining stability in physical environments and motivating simulation-based training.

NVIDIA Research [4] demonstrated large scale parallel training for quadruped locomotion with domain randomization, enabling efficient sim to real transfer. Their approach inspired the use of multiple parallel agents in this work. Similarly, Rudin et al. [5] showed that massively parallel deep reinforcement learning can learn locomotion policies rapidly, validating the effectiveness of parallel experience collection.

The Unity ML Agents Toolkit [1] provides an accessible and physics based framework with built in PPO support and ONNX export capabilities. This makes it suitable for reproducible and scalable humanoid locomotion research on hardware.

3. SYSTEM ARCHITECTURE

The system comprises four functional layers as described in this section.

3.1 Simulation Layer

The simulation is developed in Unity 2020.3.49f1 using the PhysX physics engine. A RobotKyle humanoid with 35 degrees of freedom is used, and the training scene includes 25 parallel agents operating in isolated environments while sharing a common PPO policy. The environment manager introduces controlled randomization in spawn positions, targets, and surface properties.

3.2 Agent Layer

The agent, implemented through WalkerAgent.cs, manages observation collection, action execution, reward calculation, and episode termination. Each episode begins with a randomized state and ends when stability conditions are violated or the maximum step limit is reached.

3.3 Learning Layer

The PPO trainer processes data from all parallel agents, applying Generalized Advantage Estimation and clipped updates. Parallel training significantly improves efficiency, allowing 2.8 million steps to be completed in 1.23 hours.

3.4 Inference Layer

The trained model is exported in ONNX format and deployed using Unity Barracuda for real time inference, enabling stable policy execution without requiring the training environment.

4. METHODOLOGY

4.1 Observation Space

The agent receives a proprioceptive observation vector including joint angles, angular velocities, torso orientation, center of mass velocity, foot ground contact signals, and target direction. All features are normalized for stable training, resulting in an observation space of approximately 225 dimensions.

4.2 Action Space

The policy outputs 35 continuous actions in the range [-1, 1], mapped to joint torques using a controller. Parameters such as gains, damping, and torque limits are tuned to ensure stable and realistic motion, with constraints applied at each step.

4.3 Reward Function

The reward combines multiple components to encourage forward motion, upright posture, and stable contact while penalizing non foot contacts and excessive actions. A terminal bonus is given upon reaching the target. The refined reward design significantly improved performance, achieving faster convergence and higher final rewards compared to the baseline.

4.4 PPO Algorithm

Proximal Policy Optimization is used with a clipped objective to ensure stable updates. Advantage estimates are normalized, and the value function is optimized alongside the policy with an entropy term to encourage exploration.

4.5 Parallel Training Strategy

Training is performed using 25 parallel agents collecting experiences simultaneously. This shared learning setup improves efficiency, enabling 2.8 million steps to be completed in 1.23 hours while maintaining consistent policy updates.

5. EXPERIMENTAL RESULTS

5.1 Training Run Comparison:

Two configurations were evaluated: ragdoll_run_1 and ragdoll_smooth_v2. The improved setup significantly outperformed the baseline, achieving higher rewards and faster convergence. Notably, ragdoll_smooth_v2 reached a positive final reward with approximately 87 percent less training time.

Table 1. Training Run Comparison

Metric	ragdoll_run_1	ragdoll_smooth_v2
Total Steps	1,600,000	2,800,000
Wall-clock Time	9.733 hours	1.23 hours
Parallel Agents	25 (Unity scene instances)	25
Smoothed Reward	–292.29	+536.81

with the final model achieving the most stable and natural walking performance.

Final Raw Reward	−38.98	+845.38
Convergence	~1.4M steps	~2.5M steps

5.2 Reward Progression

Training shows four phases: initial instability, balance learning, gait formation, and stable walking. The reward improved from approximately -12,000 to +845.38, indicating successful policy learning.

5.3 Loss Metrics

Training losses decreased steadily, with forward and inverse losses improving significantly. Policy loss remained stable, while value loss reduced considerably, confirming effective learning and stable updates.

5.4 Episode Length

Episode length reached the maximum of 1,000 steps after about 1.4 million steps, showing that the agent learned stable walking without early termination.

5.5 Reward Distribution

The reward distribution shifted from large negative values to consistently positive values over training, indicating overall performance improvement across agents.

6. ONNX POLICY EXPORT AND INFERENCE

After training, the final model was exported as *Walker-2800000.onnx* in ONNX format, enabling platform-independent inference without requiring Unity Editor or Python dependencies. The model takes a 225-dimensional observation vector as input and outputs 35 continuous joint torque values. During inference, Unity's Barracuda runtime executes the model at each physics step, producing deterministic actions. Validation in the Unity Editor (Inference Only mode) confirmed stable walking behavior consistent with training results.Three checkpoints—*Walker-999000.onnx*, *Walker-1999000.onnx*, *Walker-2800000.onnx*—were evaluated, showing progressive improvement in gait,

7. DISCUSSION

The *ragdoll_smooth_v2* configuration significantly outperformed the baseline, achieving faster convergence and positive rewards. This highlights the importance of well-designed reward shaping, where smoothed rewards and contact penalties helped stabilize learning and prevent unrealistic locomotion behaviors. Additionally, parallel training with 25 agents greatly improved data collection efficiency without affecting deployment, as the final ONNX model represents a single shared policy.

Despite these strengths, the policy remains limited in handling external disturbances, as training occurs in a controlled simulation. Future work should incorporate domain randomization to improve robustness and generalization in more dynamic environments.

8. CONCLUSION

This work demonstrated a PPO-based 3D humanoid locomotion system in Unity ML-Agents, where stable walking was learned purely through reinforcement learning. Training with 25 parallel agents over 2.8 million steps achieved strong performance, improving rewards from approximately −12,000 to +845.38, with consistent learning confirmed through TensorBoard metrics.

The exported ONNX model enables efficient real-time inference, making the system practical and reproducible. This approach provides a solid baseline for future research, including more complex environments, robustness improvements, and enhanced policy architectures.

ACKNOWLEDGEMENT

The authors thank Dr. D. D. Sarpate (Guide) and Prof. Anil Walke (HOD, AI & DS) for their valuable guidance and support throughout this project. Thanks also to Dr. P. K. Srivastava, Principal, ISBM College of Engineering, for his continuous encouragement.

REFERENCES

[1] J. Schulman, F. Wolski, P. Dhariwal,
A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv:1707.06347, 2017.

[2] S. Kuindersma et al., "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," Autonomous Robots, vol. 40, pp. 429–455,
2016.

[3] NVIDIA Research Team, "Training quadruped locomotion policies in Isaac Sim," NVIDIA Technical Blog, 2024.

[4] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in Proc. Conference on Robot Learning (CoRL), 2022.

[5] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in Proc. Robotics: Science and Systems, 2018.

[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in Proc. ICLR, 2016.

APPENDIX

A. Hyperparameter Configuration

The PPO model used key parameters including learning rate (3e−4), discount factor ($\gamma = 0.99$), GAE ($\lambda = 0.95$), clipping parameter ($\varepsilon = 0.2$), and batch size of 2048. These were tuned to ensure stable and efficient training.

B. Observation and Action Spaces

The agent operates on a ~225-dimensional observation space capturing joint states, velocities, and goal direction, and outputs 35 continuous actions corresponding to joint torques. All inputs were normalized for consistency.

C. Training Setup

Training was conducted with 25 parallel agents in Unity ML-Agents, sharing a common policy. This setup enabled faster experience collection and reduced overall training time while maintaining stable learning.