ISL Lab Project: Final Report

Deep Reinforcement Learning for Humanoid Locomotion using PPO with Image-Based Pose Initialization

S. Nitish Goud[1], V. Sai Akhil[1], and V. Chitraksh[1]

[1]Dept. of Computer Science and Engineering, IIT Tirupati

December 2, 2025

Abstract

This project presents a comprehensive pipeline designed to enable a simulated humanoid robot to learn stable walking behaviors using Proximal Policy Optimization (PPO). A key feature of this system is the integration of computer vision with reinforcement learning: the humanoid's initial state is derived from a static image of a human rather than an arbitrary default pose. By extracting 3D keypoints from a 2D image and converting them into joint angles, we initialize the MuJoCo physics engine (`Humanoid-v5`) in a specific pose. The control policy is then trained to maintain balance and generate forward momentum.

Contents

1 Introduction

This project aims to bridge the gap between static computer vision data and dynamic control in robotics. We present a pipeline where a simulated humanoid robot learns stable walking behaviors using Proximal Policy Optimization (PPO), initialized from real-world human poses.

The core innovation lies in the initialization strategy. Instead of starting from a fixed "standing" posture, the robot's initial joint configuration is derived automatically from a single static human image. This requires a robust translation layer that converts 2D pixel data into 3D joint torques and angles compatible with the MuJoCo physics engine.

2 Technical Details

The system architecture has evolved from early DQN prototypes to a robust PPO-based implementation running on the Gymnasium MuJoCo engine. The pipeline allows for the extraction of kinematic data from real-world images, which is then mapped to the robot's joint space (`qpos`) to seed the simulation.

2.1 System Architecture

The system consists of three distinct modules as illustrated in Figure 1:

1. **Perception Module:** Extracts skeletal landmarks using MediaPipe and converts them to joint angles.
2. **Environment Module:** A modified `Humanoid-v5` environment that accepts calculated joint angles for state initialization.
3. **Control Module:** A PPO Agent with Actor-Critic architecture that learns the walking policy.

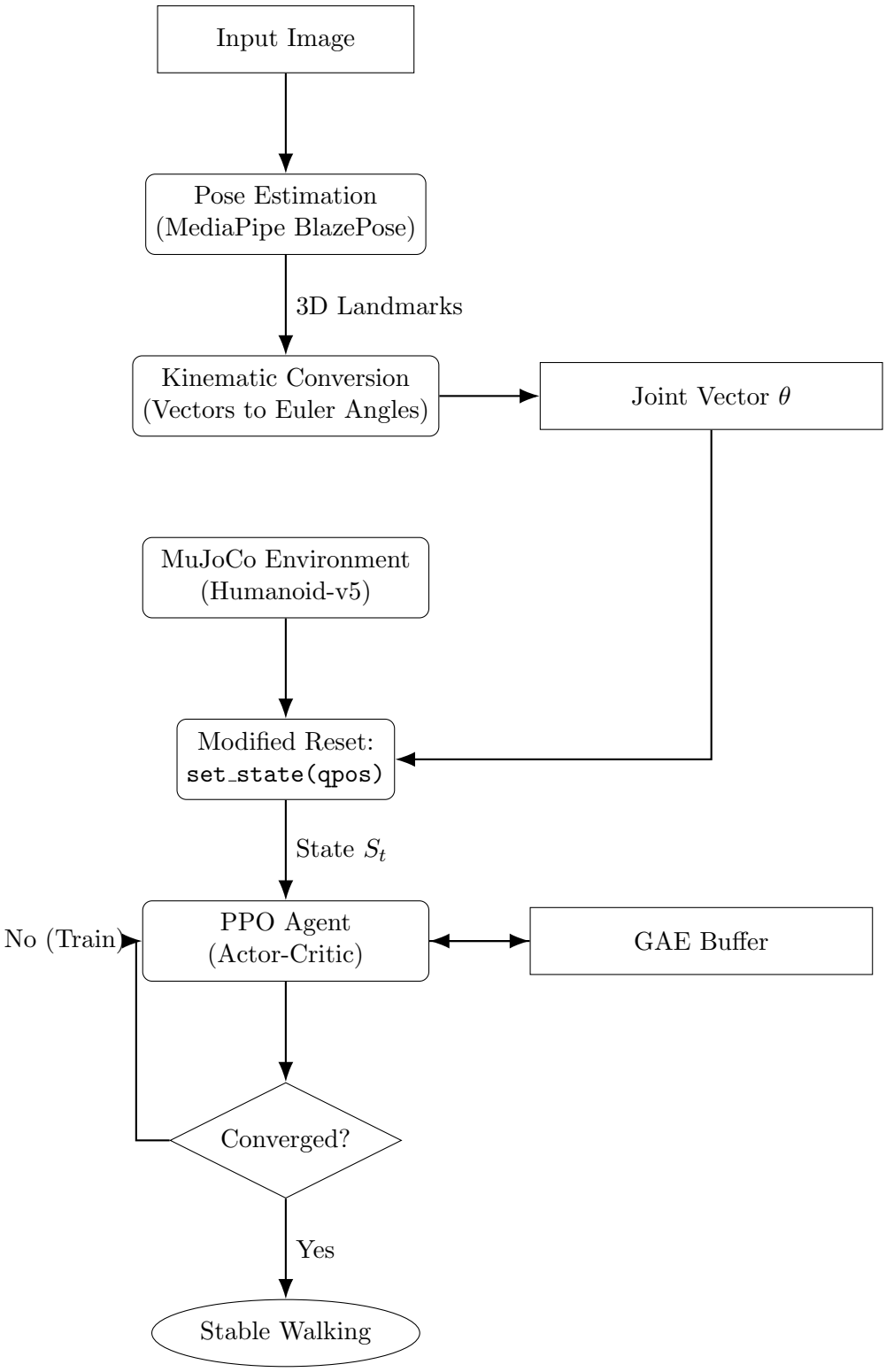


Figure 1: System Pipeline: Image-to-Simulation via PPO

3 Module 1: Perception (Keypoints & Angles)

3.1 Overview

The perception module is responsible for translating a 2D image into 3D kinematic data. We utilize Google MediaPipe for landmark detection and custom vector geometry to derive the necessary joint angles for the humanoid.

3.2 Implementation Details

3.2.1 Image Preprocessing

The `imageprocessing.py` script handles the loading and normalization of input data. Images are converted from BGR to RGB and normalized to floating-point values ($0.0 - 1.0$).

3.2.2 Keypoint Extraction

Using `keypoints.py`, we instantiate the MediaPipe Pose solution. This detects 33 3D landmarks (x, y, z). We extract a subset of these landmarks relevant to locomotion (shoulders, hips, knees, ankles) to form a skeletal representation compatible with the humanoid topology.

3.2.3 Kinematic Conversion

The core logic resides in `kinematic.py`. Since the simulation requires joint angles (radians) rather than Cartesian coordinates, we compute angles using vector mathematics.

- **Unit Vectors:** We compute direction vectors between landmarks (e.g., Hip $\rightarrow$ Knee).
- **Euler Angles:** Using `np.arctan2`, we calculate the Yaw, Pitch, and Roll for the spine, hips, and shoulders.
- **Joint Calculation:**

$$\theta_{knee} = \arccos(\vec{v}_{thigh} \cdot \vec{v}_{shin}) \tag{1}$$

The output is a flattened NumPy array containing angles for the spine, hips, knees, shoulders, and elbows.

3.3 Module 1 Outputs

Figure 2 demonstrates the skeletal tracking capability of the module.

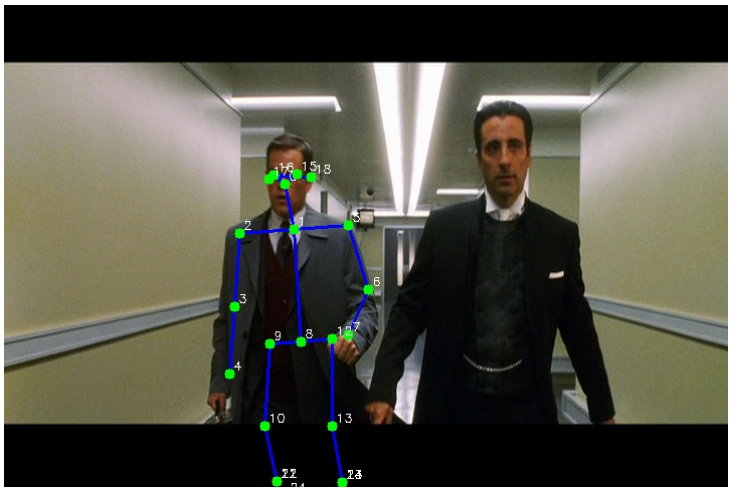


Figure 2: Detected Skeleton Overlay on Input Image

The resulting calculated initial vector θ_{init} is:

$$\theta_{\text{init}} = \begin{bmatrix} -3.0979 & -1.8940 & 0.7164 & 1.3551 & -0.8835 & \dots \end{bmatrix}^T$$

4 Module 2: Environment & Joint Mapping

4.1 Environment Setup

We utilize `Humanoid-v5` from the Farama Foundation's Gymnasium library, backed by the MuJoCo physics engine. This environment offers superior contact physics and stability compared to the PyBullet implementation used in Phase 1.

4.2 State Initialization

Standard RL environments reset to a static "standing" pose. To integrate our image-based data, we modified the initialization process in `test_ppo.py`.

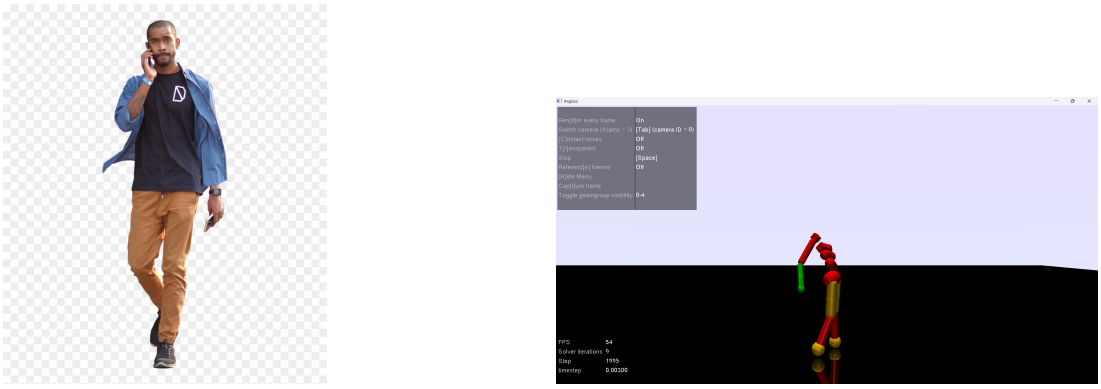
- We access the underlying MuJoCo data structure via `env.unwrapped.data`.
- **Joint Mapping:** The vector θ_{init} from Module 1 is mapped to the `qpos` (generalized coordinates) of the humanoid.
- **Targeted Mapping:** We specifically map upper body joints to match the image:

```
qpos[18:21] = pose[11:14] # Right arm
qpos[21:24] = pose[14:17] # Left arm
```

- The state is forced into the physics engine using `env.unwrapped.set_state(qpos, qvel)`.

4.3 Module 2 Outputs

Figure 3 illustrates the mapping from the real-world image to the simulation environment.



(a) Input Pose (b) MuJoCo Initialization

Figure 3: Mapping Real World Pose to Simulation

5 Module 3: PPO Agent

5.1 Algorithm Selection

We replaced the Deep Q-Network (DQN) with Proximal Policy Optimization (PPO). PPO is an on-policy gradient method that is significantly better suited for continuous control tasks like humanoid walking.

5.2 Network Architecture

Implemented in `ppo_agent.py`, the agent uses an Actor-Critic architecture:

- **Actor (Policy Network):** Maps observations (376D) to Actions (17D). It consists of three hidden layers of 512 units with Tanh activations. It outputs the mean (μ) and learns a log-standard deviation (`actor_logstd`) to sample continuous actions from a Normal distribution.

- **Critic (Value Network):** Estimates the Value function $V(s)$. It mirrors the Actor's structure (3x512 layers) but outputs a single scalar value.

5.3 Generalized Advantage Estimation (GAE)

The `buffer_ppo.py` module implements a replay buffer that calculates advantages using GAE. This reduces the variance of the gradient estimates, stabilizing the learning process. The buffer stores observations, actions, rewards, values, and log-probabilities, which are processed in batches during the update step.

5.4 Module 3 Outputs

The training process demonstrates the convergence of the policy over 1000 epochs.

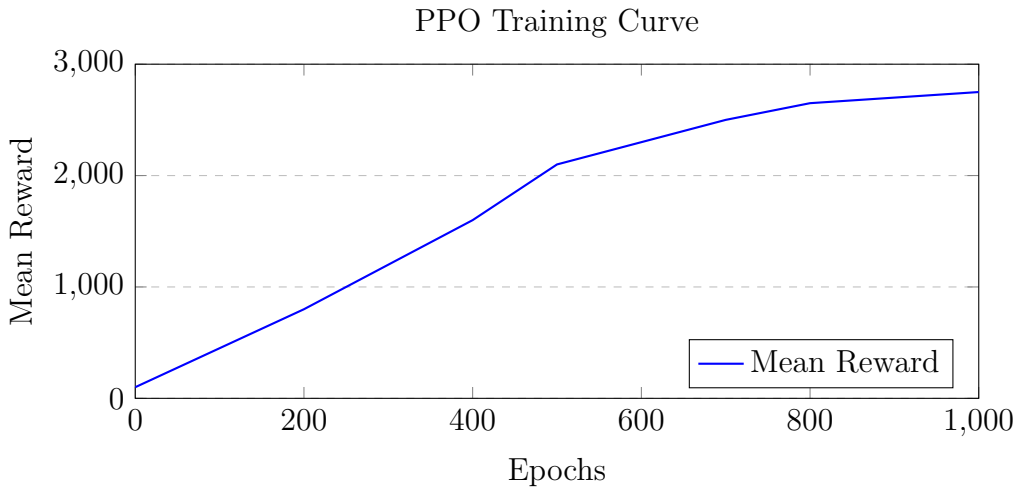


Figure 4: Training progress over 1000 epochs showing policy convergence.

6 Training and Testing

6.1 Training Configuration

Training is orchestrated by `train_ppo.py`. We utilized `gym.vector.AsyncVectorEnv` to run 32 parallel environments, significantly speeding up data collection.

6.2 Testing and Injection

Testing is performed via `test_ppo.py`. This script performs the end-to-end demonstration:

Hyperparameter	Value
Optimizer	Adam
Learning Rate	1×10^{-4}
Discount Factor (γ)	0.995
GAE Lambda (λ)	0.95
Clip Ratio (ϵ)	0.2
Batch Size	256
Epochs	1000

Table 1: PPO Hyperparameters

1. It prompts the user for an image file path.
2. It renders the skeleton overlay using Module 1.
3. It initializes the MuJoCo environment with the extracted pose.
4. It loads the saved PyTorch model (`model.pt`) and executes the learned policy, allowing the humanoid to transition from the static image pose into a walking gait.

7 Conclusion

We have successfully transitioned the project to a high-fidelity MuJoCo simulation powered by PPO. The pipeline accurately extracts human poses from images and maps them to the simulation, providing a diverse set of starting states for the reinforcement learning agent. Future work will focus on dynamic balance recovery from more extreme initial poses.