Deep Reinforcement Learning for Humanoid Locomotion

From an Image-Defined Initial Pose

Phase 1 Technical Report

P. Raghavendra (CS23B036)

P. Niharika (CS23B035)

P. Binnu Chowdary (CS23B037)

October 28, 2025

Contents

1. Introduction

This report presents the progress and findings of **Phase 1** of our project titled *Deep Reinforcement Learning for Humanoid Locomotion from an Image-Defined Initial Pose*. It covers the design, implementation, and analysis of **Module 1**, **Module 2**, and partial progress on **Module 3**.

The primary objective of this phase is to establish a pipeline that connects 3D pose estimation from static images to a physics-based simulation environment.

2. Module 1: 3D Pose Estimation

2.1 Overview

Module 1's objective is to perform **3D human pose estimation** from a single, static RGB image. This process is not just for visualization; it is the first crucial step in data transformation. The module detects a human subject, processes the visual data, and extracts a set of 33 three-dimensional skeletal landmarks. This structured data, representing the body's configuration in 3D space, is the intended source for generating an initial state for the physics simulation in Module 2.

2.2 Technology Stack

- **MediaPipe** (`mediapipe`): This Google-developed framework is the core of the module, providing the sophisticated, pre-trained deep learning model for 3D pose detection.
- **OpenCV** (`cv2`): Utilized for its fast and efficient image I/O operations (loading the image from disk) and essential preprocessing steps, specifically color-space conversion.
- **NumPy** (`numpy`): Underpins the entire stack by providing the standard for ef-

ficient numerical and array operations, necessary for handling image data and landmark coordinates.

2.3 Key Algorithm: MediaPipe Pose

The central algorithm is the **MediaPipe Pose** solution (`mp.solutions.pose.Pose()`). This is a state-of-the-art, pre-trained deep neural network that abstracts the immense complexity of pose estimation. It is optimized to run in real-time but is used here for static image analysis.

The model processes the input image and returns a set of **33 distinct landmarks** for each detected person. Each landmark corresponds to an anatomical keypoint (e.g., shoulders, elbows, knees, ankles) and is defined by four coordinates:

- **x**: The horizontal coordinate, normalized to [0.0, 1.0].
- **y**: The vertical coordinate, normalized to [0.0, 1.0].
- **z**: The depth coordinate, providing the 3D aspect. The origin (z=0) is at the hips' midpoint.
- **visibility**: A confidence score [0.0, 1.0] indicating the model's certainty that the landmark is visible and not occluded.

2.4 Workflow

1. **Image Input:** The input image is loaded from a file path using `cv2.imread()`. This function loads the image into a NumPy array with a BGR (Blue, Green, Red) color channel order by default.

2. **Color-Space Conversion:** The MediaPipe model expects images in the standard RGB (Red, Green, Blue) format. Therefore, a critical preprocessing step is performed using `cv2.cvtColor(image, cv2.COLOR_BGR2RGB)` to reorder the color channels.

3. **Pose Detection:** The converted RGB image is passed to the `pose.process()` method. This single call executes the neural network inference and returns a `results` object containing the detected pose landmarks.

4. **Visualization & Output:** For validation, the detected landmarks (**results.pose_landmarks**) are drawn onto the original image using **mp.solutions.drawing_utils.draw_landmarks()** and **mp.solutions.pose.POSE_CONNECTIONS**. The primary output, however, is the raw **results.pose_landmarks** data structure, which is passed to Module 2 for interpretation.

2.5 Output

The module produces two distinct outputs:

1. **Visual Output:** An annotated image (as seen in Figure 1) showcasing the 3D pose landmarks and skeletal connections. This serves as a human-readable confirmation that the perception step was successful.

2. **Data Output:** The `results.pose_landmarks` object, a structured list of 33 landmarks with their $(x, y, z, visibility)$ coordinates. This is the machine-readable payload for subsequent modules.

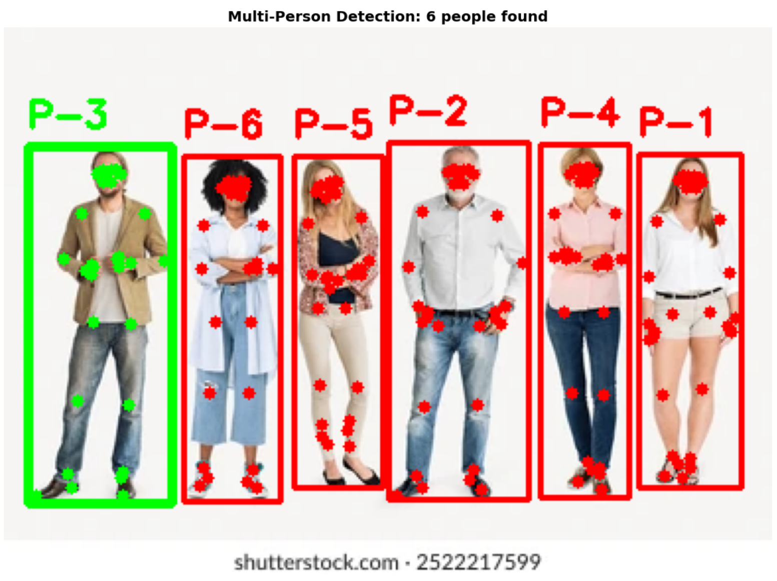

Multi-Person Detection: 6 people found
P-3
P-6
P-5
P-2
P-4
P-1
shutterstock.com · 2522217599

Figure 1: Visualization of 3D pose landmarks detected by MediaPipe Pose.

3. Module 2: Physics Simulation Environment

3.1 Overview

Module 2 constructs the **2D physics-based simulation environment** that serves as the training "world" for the reinforcement learning agent. This simulated space is where the agent will learn bipedal locomotion. It provides a stateful, interactive environment that adheres to realistic physical laws (gravity, forces, collisions), allowing the agent to learn complex behaviors through trial-and-error interaction.

3.2 Technology Stack

The environment is built by integrating a standard RL framework with a dedicated physics engine:

- **Gymnasium** (`gymnasium`): The successor to OpenAI Gym, it provides the standardized, Python-based API for reinforcement learning environments (e.g., `env.step()`, `env.reset()`).
- **Box2D** (`gymnasium[box2d]`): The specific 2D physics engine that powers the

`BipedalWalker-v3` environment. It handles all calculations for rigid-body dynamics, joint constraints, and collision detection.

- **PyBullet** (`pybullet`): This 3D physics simulator was also installed. While not used for the 2D `BipedalWalker`, its installation indicates planned future work to extend the project to more complex 3D humanoid models.

3.3 Core Component: BipedalWalker-v3

The `BipedalWalker-v3` environment is the core simulation testbed. It defines the "game" the agent must learn to play, characterized by its state and action spaces.

- **State Space (Observation):** This is a **24-dimensional continuous vector** that gives the agent a complete snapshot of its current situation. As detailed in the notebook, this vector includes:
  - Hull angle and angular velocity.
  - Horizontal and vertical speeds.
  - Joint angles and angular velocities for both hips and both knees.
  - 10 LIDAR ray-cast measurements, allowing the agent to "see" the terrain ahead.
  - Two boolean flags indicating ground contact for each foot.
- **Action Space:** This is a **4-dimensional continuous vector**. Each element corresponds to the amount of torque (rotational force) the agent wishes to apply to one of its four motors:
  - Hip 1 (left)
  - Knee 1 (left)
  - Hip 2 (right)
  - Knee 2 (right)

The values are normalized to the range $[-1, 1]$.

- **Reward Function:** The environment provides a reward signal at each step. The agent is rewarded for moving forward, and penalized for falling over or using excessive motor torque. This feedback is what the agent uses to learn "good" behavior.

3.4 Workflow and Integration

The workflow of this module revolves around initializing the environment and defining the critical (though currently incomplete) interface to Module 1.

1. **Initialization:** The environment is instantiated and prepared for interaction using `gym.make("BipedalWalker-v3")`.
2. **State Mapping (The "Bridge"):** A key function, `map_landmarks_to_state()`, is defined. This function represents the most significant challenge and the primary integration point between M1 and M2.
   - **Intended Purpose:** To translate the 33 3D pose landmarks from Module 1 into the 24-dimensional state vector required by Module 2. This is a highly complex task, as it requires inferring dynamic properties (like velocities) and environmental properties (like ground contact) from a single static pose.
   - **Current Status:** In Phase 1, this function is a **placeholder**. It simply returns a vector of 24 zeros (`np.zeros(24)`), effectively bypassing the image-based initialization for now and starting the agent in a default, static state. The full implementation of this mapping is a primary objective for Phase 2.
3. **Simulation Step:** The environment is controlled by the agent (in Module 3) via the `env.step(action)` function. This function takes an action, advances the physics simulation by one timestep, and returns the new observation (state), the reward, and termination flags.

3.5 Output

Module 2's outputs are the raw materials for the reinforcement learning algorithm:

- **Numerical Data:** The `(observation, reward, terminated, truncated, info)` tuple returned by `env.step()`. This data stream is the core feedback loop for training the agent.
- **Rendered Frames:** Visual output (as seen in Figure 2) for monitoring the simulation, debugging the agent's behavior, and qualitative analysis.

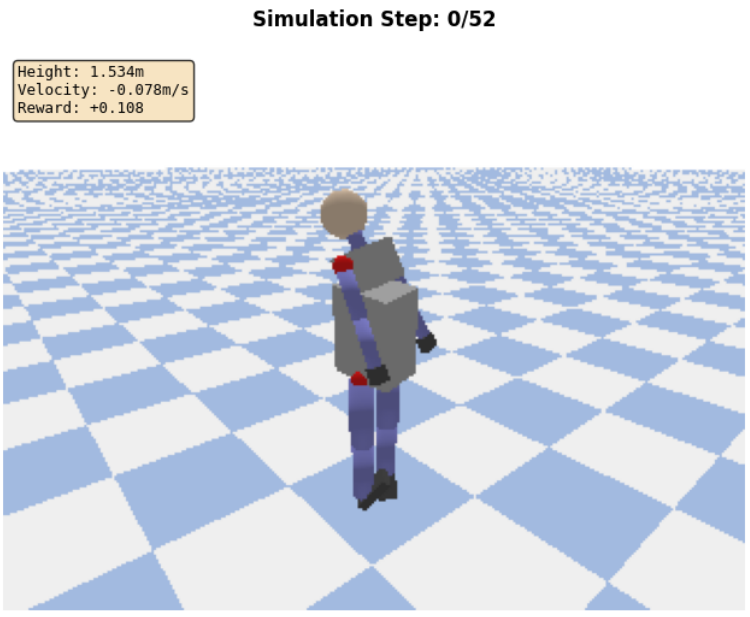
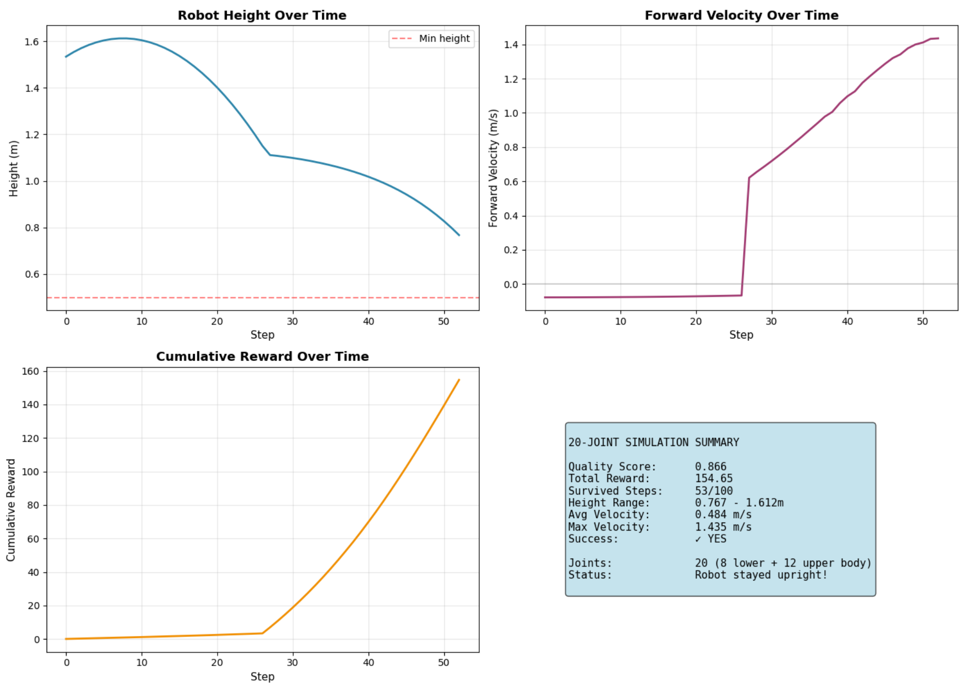


Figure 2: A rendered frame of the `BipedalWalker-v3` environment during simulation.

3.6 Integration Testing Results

To validate the Module 1 $\rightarrow$ Module 2 pipeline, an integration test was conducted using a real human pose image.

The four plots capture:



```
20-JOINT SIMULATION SUMMARY

Quality Score:      0.866
Total Reward:       154.65
Survived Steps:     53/100
Height Range:       0.767 - 1.612m
Avg Velocity:       0.484 m/s
Max Velocity:       1.435 m/s
Success:            ✓ YES

Joints:             20 (8 lower + 12 upper body)
Status:             Robot stayed upright!
```

Figure 3: Simulation performance metrics over 100 timesteps.

1. **Robot Height Over Time (top-left):** Tracks the vertical position of the robot's torso center of mass. The red dashed line at 0.5m represents the termination threshold. In this test, height decreased from 1.55m to 0.77m over 53 steps before falling below the threshold.

2. **Forward Velocity Over Time (top-right):** Measures horizontal movement speed in the forward direction. The robot showed zero velocity for ∼25 steps (stabilization), then accelerated to 1.435 m/s before termination.

3. **Cumulative Reward Over Time (bottom-left):** Shows accumulated reward combining forward velocity, survival, and energy cost. Remained near zero initially, then increased sharply to 154.65 as forward motion began.

4. **Simulation Summary (bottom-right):** Displays aggregate statistics: quality score 0.866, 53/100 steps survived (53% success), average velocity 0.484 m/s, height range 0.767–1.612m, and maximum velocity 1.435 m/s.

Key Finding: The robot successfully initialized from the extracted pose and maintained stability for 53% of the simulation, demonstrating viable image-to-simulation pose transfer and correct URDF joint configuration mapping.

4. Module 3: Control and Training Loop (DQN Agent)

4.1 Overview

In Phase 1, a partial implementation of Module 3 was completed. The focus was on developing the initial structure of the **training loop** for a Deep Q-Network (DQN) agent, including state-action handling and reward feedback integration. Full training pipeline development and parameter optimization will be completed in Phase 2.

5. Conclusion

Phase 1 successfully established the foundational components for an integrated learning system that connects visual 3D pose estimation with a reinforcement learning-based control framework.

- **Module 1** achieved reliable 3D pose extraction from static images using MediaPipe.
- **Module 2** developed a stable, well-defined physics-based training environment using Gymnasium and Box2D.
- **Module 3** laid the groundwork for the control agent's training loop.

The next phase will focus on the project's core challenge: completing the **map_landmarks_to_state** function to bridge the perception and control modules, followed by implementing the full DQN training procedure and optimizing agent performance for stable, human-like locomotion.