Stable Reinforcement Learning for Humanoid Locomotion

Roger Perramon Marquès

Universidad Autònoma de Barcelona & PAL Robotics

Abstract

Humanoid locomotion for robots has been normalized as nowadays we see videos online about robots doing amazing tasks. However, it remains a challenging problem in robotics, as every bipedal robot has unique mechanical and control characteristics.
This project presents the research and development of a stable and safe Humanoid Locomotion policy for a biped robot. By integrating advanced robot control techniques and reinforcement learning, the aim is to achieve a policy ready for real-world robots. Our approach focuses on a human-like walking posture. Ensuring that the robot maintains a similar walking trajectory as a person, dynamic balance, robustness against disturbances. Also smooth actions coordinating joint movements to minimize abrupt accelerations, reducing mechanical stress, and ensuring the well-being of the robot actuators.

Keywords: Reinforcement Learning, Humanoid Locomotion, Bipedal Robots, Robot Control, MuJoCo

1 Introduction

1.1 Challenge and Background

One of the most challenging aspects of humanoid robotics is achieving stable and safe locomotion. Unlike robots wheeled robots, for bipedal robots it is a challenge itself to navigate through an environment as they must maintain balance while walking, adapting to various external forces while coordinating multiple joints to do so. Despite all these challenges, it is feasible and has been done numerous times.

Motivated by this complexities, this project aims to create a real-world humanoid locomotion policy and an interface to replicate this for other bipedal robots with minimal modifications.

One of the biggest challenges faced in this project is that,as far as we know, neither of the robots used for development has ever been controlled by a Reinforcement Learning Policy, and no prior research has successfully demonstrated such control. This lack of existing work made the integration of RL into these platforms particularly challenging.

This raises key research questions such as:

- How can a reinforcement learning policy be designed to ensure stable and robust locomotion on real humanoid platforms that have not been previously trained with such methods?
- What level of abstraction or interface is needed to generalize the learned locomotion policy across different bipedal robots with minimal retraining?

1.2 Objective

The primary goal of this project is to:

Develop a robust, adaptable, and safe locomotion policy for humanoid robots using reinforcement learning.

This goal is supported by the following parts:

- **Develop a robust and adaptable locomotion policy** for a humanoid robot with human-like morphology, capable of stable walking under various conditions.
- **Enable recovery from external disturbances** while maintaining the desired trajectory, a particularly challenging task given that the target robots have not previously been controlled using reinforcement learning.
- **Minimize abrupt accelerations and mechanical stress** on the robot's actuators, ensuring that the learned policy is safe and suitable for real-world deployment.
- **Design a generalizable interface and supporting tools** that allow the same policy architecture to be applied and trained on other humanoid biped robots with minimal modifications.

2 Theoretical Background

2.1 Control Techniques

Bipedal locomotion in humanoid robotics is the ability of robots to walk on two legs imitating human movement. It allows robots to move through complex environments like stairs and difficult terrain, making them useful in healthcare, industry, and in multiple situations. This type of movement also attracts research in biomechanics and prosthetics because it provides insights into human gait, balance, and motor control, which can inform the design of assistive devices and rehabilitation technologies.

Traditional solutions include methods like Zero Moment Point (ZMP)[24] , Model Predictive Control (MPC)[23], Whole-Body Control (WBC)[10], and Hybrid Zero Dynamics (HZD)[26]. These techniques focus on predicting motion, maintaining stability, and managing joint coordination.

Recently, Reinforcement Learning (RL) has gained traction as a promising alternative. By learning from experience, RL enables robots to adapt to complex environments and disturbances. The next sections will explore RL methods in detail and compare them to classical approaches.

2.1.1 Zero Moment Point

Zero Moment Point is a fundamental concept in biped locomotion control, essential for maintaining the stability of humanoid robots. The ZMP is the point on the ground where the combined forces of gravity and inertia create no tipping moment. Keeping the ZMP within the robot's support polygon (the area under its feet) is vital for preventing falls during dynamic movement.[25].

ZMP was first introduced by Miomir Vukobratović in the 1960s and formalized in the early 1970s. His work laid the foundation for controlling robot stability by maintaining the ZMP in a stable location. This principle was applied in early humanoid robots like the WABOT series in the 1970s, which demonstrated the potential for bipedal walking. In the 1990s, robots like Honda's ASIMO adopted ZMP to achieve dynamic and stable walking, becoming a practical application of the concept. The HRP-2 robot, developed in the 2000s, extended ZMP control to tasks like climbing stairs and walking on uneven surfaces.

2.1.2 Model Predictive Control

Model Predictive Control is an advanced control strategy that has gained significant traction in the field of humanoid biped locomotion due to its ability to handle multi-variable systems and constraints. Originally developed in the chemical process industry in the 1970s, it has since been adapted for systems like bipedal robots, where predicting future motion is crucial. MPC works by simulating how the robot will move over a short future time window and solving an optimization problem to choose the best control actions at each moment.

This approach is ideal for handling complex conditions such as joint limits, changing terrain, and stability constraints. By considering these factors in real time, MPC enables humanoid robots to adjust their foot placement and posture dynamically[5]. For example, some robots use MPC to walk over rough terrain, while HRP-4 applies it to plan smooth, balanced steps. MPC has also been combined with whole-body control in robots like MIT's Cheetah 3, enhancing its ability to navigate unpredictable environments.

2.1.3 Whole-Body Control

Whole-Body Control is a control technique in humanoid robotics that manages all joints and limbs of a robot together, treating the entire body as a coordinated system. Unlike simpler methods that control limbs independently, WBC enables complex and balanced movements like walking while manipulating objects or recovering from disturbances[9]. This coordination allows robots to move more naturally and efficiently.

Technically, WBC works by solving an optimization problem that considers the robot's full-body dynamics, constraints like joint limits and contact forces, and task objectives such as maintaining balance or following a trajectory.

2.1.4 Hybrid Zero Dynamics

Hybrid Zero Dynamics is a control method specifically designed for bipedal walking. It merges hybrid system theory, which accounts for both continuous motion and sudden changes like foot impacts, with zero dynamics, a concept that simplifies the robot's behavior by focusing on internal stability once key outputs, like leg motion, are controlled.

Developed in the early 2000s, HZD creates stable, periodic walking patterns by guiding the robot to follow a predefined motion while ensuring it can handle the alternating phases of walking (swing and stance). The robot switches between smooth motion (when feet are on the ground) and discrete transitions (when a foot hits or leaves the ground)[8], and HZD manages both seamlessly.

2.2 Reinforcement Learning for Biped Walking

RL has emerged as a promising approach for controlling biped locomotion in humanoid robots, offering a data driven method to learn and adapt walking behaviors. Unlike classical approaches such as ZMP, MPC, WBC, and HZD, which rely on predefined models and mathematical formulations, RL leverages trial-and-error interactions with the environment to develop control policies. This ability to autonomously discover efficient and robust walking strategies enables RL to handle the complexities and uncertainties of biped walking, making it a valuable tool for tasks that are challenging to encode explicitly through traditional control methods. A more detailed comparison of control tecniques can be found in Appendix: **??**

Furthermore, RL methods can adapt to unmodeled dynamics and external disturbances[16]. These capabilities allow RL based controllers to achieve more flexible and robust locomotion compared to classical model based methods, especially in complex or unpredictable environments. Next, we briefly present the most widely used RL policies found in the literature.

2.2.1 Soft Actor–Critic

Soft Actor–Critic (SAC) learns a policy that picks actions from a probability distribution by balancing reward and randomness. In practice it maximises

$$\mathbb{E}\big[Q_{\min}(s,a)\big] + \alpha\,\mathbb{E}\big[H\big(\pi(\cdot \mid s)\big)\big], \tag{1}$$

where $Q_{\min}$ is the smaller of two value estimates and H is the entropy. This extra entropy term helps the robot explore new gaits and avoid getting stuck in a single pattern. SAC is robust on rough or changing ground because it learns from old data in a replay buffer, but it needs several neural networks and the entropy weight α can take time to tune.

2.2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) updates its policy in small steps by clipping large changes in the objective:

$$\mathbb{E}\Big[\min\big(r_t\,A_t,\ \mathrm{clip}(r_t, 1-\epsilon, 1+\epsilon)\,A_t\big)\Big], \tag{2}$$

with $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ the ratio of new to old policy probability and A_t the advantage. This clipping keeps learning stable, so training rarely blows up even on complex walking tasks. PPO is easy to implement and tune, but because it only uses on-policy data, you must collect fresh samples after each update. That makes it less sample efficient than off-policy methods, and you still have to choose a good clip size ϵ.

In robot locomotion, we need a method that is both stable on real hardware and easy to tune when testing on physical platforms. From the two algorithms PPO's reliance on fresh on-policy rollouts gives it the edge, it gives the policy the hability to consistently adaptt to the robot's true dynamics, friction changes, or sensor noise, without requiring a large replay buffer. Its clipped objective

keeps each update small, preventing sudden changes in walking behavior that could damage actuators or cause falls. With just a single policy and value network to tune, PPO is more straightforward to implement on embedded controllers, making it a practical choice for safely learning stable policies on real robots[13].

3 Technological Framework

3.1 Talos Robot

The development will be performed with the TALOS[17] robot, showed in Figure 1, although it is intended to be used for the development of policies for other humanoid robots. TALOS is a state-of-the-art humanoid robot designed for advanced research in locomotion, manipulation, and human-robot interaction by PAL Robotics. It's height is 175 cm and weight 95 kg, TALOS is a full-size humanoid with a maximum capacity of 6 kg per arm when fully extended. This capability allows it to interact with industrial tools.

Figure 1: *Talos Robot.*

It is equipped with two high-performance Intel i7 processors, torque sensors in all joints (except the head, wrists, and grippers), an RGB-D camera, torque sensors on the feet and wrists, an inertial measurement unit (IMU), and stereo speakers. These features make TALOS one of the most advanced humanoid research platforms available today.

3.1.1 Robot Description and Control

- **Torso:** Two degrees of freedom (DOF) with torque control up to ± 200 Nm.
- **Head:** Two DOF with limited torque ranges to allow smooth head movements.
- **Arms:** Seven-DOF arms, each capable of force control, with high torque limits in the proximal joints for lifting heavy objects.
- **Grippers:** Independent actuators with low force control ranges for delicate manipulation.
- **Legs:** Six-DOF per leg, with high torque capabilities (up to ± 400 Nm in knee joints), allowing robust and stable bipedal locomotion.

3.2 Kangaroo Robot

Development will also be conducted using the Kangaroo robot[18],Figure 2, developed by PAL Robotics and introduced in 2021 as a next-generation research platform for agile bipedal locomotion. Kangaroo is engineered with a strong focus on dynamic movements such as jumping and running, combining lightweight construction (12.5 kg per leg) with innovative actuation technologies. Unlike traditional humanoid designs, Kangaroo features all actuators close to the torso through closed linkage mechanisms, providing impact protection and optimizing gear ratios for either high-speed or high-torque operation.

Figure 2: *Kangaroo Robot.*

The robot's legs are powered by 12 linear ball-screw actuators, six per leg, driven by brushless motors for high-efficiency, low-friction linear motion. It features a total of 76 Degrees of Freedom (DoF), 12 actuated and 64 passive, making it one of the most mechanically sophisticated biped platforms to date. However, control strategies are built on a simplified kinematic model to ensure real-time feasibility in both simulation and learning.

3.2.1 Robot Description and Control

- **Hip Joints:** Three DoF per leg, Yaw, Roll, and Pitch, the latter two using a differential actuator configuration to enhance rotational range and force without increasing inertia.
- **Leg Length Joint:** One DoF per leg, achieved through a complex four-bar linkage mechanism enabling precise vertical foot displacement while maintaining foot orientation.
- **Ankle Joints:** Two DoF per leg, Roll and Pitch, also using a differential actuator design for fine control of foot stability and orientation.
- **Actuation:** All joints are powered by linear ball-screw modules with integrated passive joints to enable complex motion profiles.
- **Model Simplification:** To facilitate simulation, a simplified 14-DoF model (all actuated) is employed, abstracting passive elements while preserving the robot's mechanical fidelity and dynamic behavior.

3.3 Unified Robot Description Format

Unified Robot Description Format[20] [22](URDF) is an XML-based format introduced with the Robot Operating System (ROS) in 2009 used to define the structure and properties of a robot. It is widely used in the Robot Operating System (ROS) to describe a robot's kinematics, dynamics, and appearance for simulation, control, and visualization.

3.3.1 URDF Descriptions

Links: Represent physical parts of the robot.

Visual properties: 3D shape, color, and texture. Collision properties: Defines the shape used for physics based interactions. Inertial properties: Mass, center of mass, and inertia tensor for realistic physics simulation.

Joints: Define how links are connected and how they move relative to each other.

- Fixed: No movement (rigid connection).
- Revolute: Rotational movement around a single axis.
- Prismatic: Linear movement along a single axis.
- Continuous: Like a revolute joint but with unlimited rotation .
- Floating: Allows six degrees of freedom (translation + rotation).
- Planar: Allows movement in a plane (2D translation + rotation).

Meshes and collision: Meshes in URDF are 3D models used to represent a robot's shape, for visualization or collision detection. Instead of simple geometric shapes like boxes, meshes provide more detailed and realistic representations of robot parts. These meshes contain large numbers of sides which make physics engines less efficient and more memory expensive, especially for collisions.

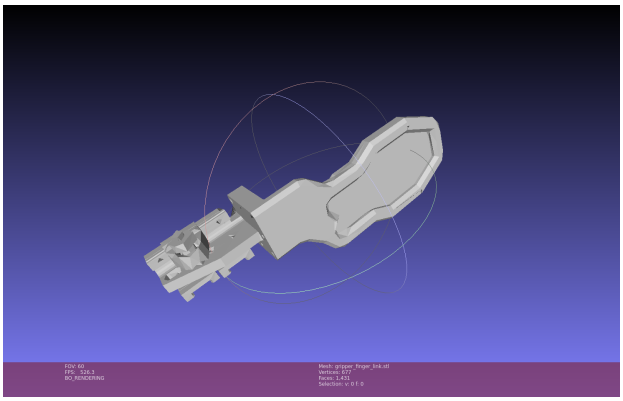

Figure 3: *Finger mesh.*

The image depicted in Figure 3 shows a mesh of a finger from the Talos robot. This mesh alone contains 677 vertices, as this is a small mesh. The bigger and more complex meshes have up to +8.000 vertices. This description is what makes them really expensive for computation.

3.4 Brax

Brax is a fast and fully differentiable physics engine used for research and development of robotics, human perception, materials science, reinforcement learning, and other simulation-heavy applications[7].

Brax is written in JAX and is designed for use on acceleration hardware. It is both efficient for single-device simulation, and scalable to massively parallel simulation on multiple devices, without the need for data centers. In my case we are using the MuJoCo XLA (MJX) simulator as seen in Figure 4.

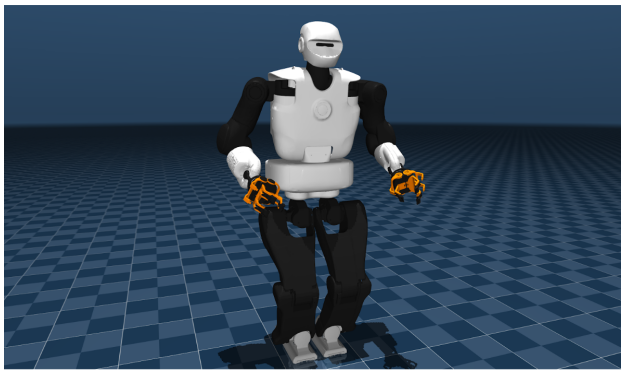

Figure 4: *Talos in MuJoCo.*

3.4.1 MuJoCo XLA

MJX is a just-in-time (JIT) compiled, highly optimized backend for MuJoCo simulations, using Google's XLA (Accelerated Linear Algebra) compiler to improve simulation performance. It transforms MuJoCo's physics computations into highly efficient, hardware-accelerated operations using JAX framework (MJX)[11][14] .

3.4.2 JAX

JAX is a high performance numerical computing library developed by Google that provides automatic differentiation (autograd), Just-In-Time compilation, and GPU/TPU acceleration for Python programs[1]. It is used in machine learning, deep learning, and scientific computing because of its efficiency. It is built on top of NumPy but improves its capabilities with features like vectorization,and JIT compilation to accelerate computations on modern hardware.

4 Proposed Methodology

4.1 Evaluation of Control Signals

Different control strategies are used to achieve precise and stable motion in humanoid robots. The three methods considered are Direct Position Control, Torque Control, and Proportional-Integral-Derivative Control, each with distinct characteristics and applications.

4.1.1 Direct Position

Direct Position Control commands the desired joint positions without considering external forces or torque dynamics. It uses trajectories and position controllers to achieve accurate joint movements.

It is a very simple control method to implement although, not so much when using neural networks as controller. This occurs because neural networks often produce highly variable control signals, requiring an effective Low Pass Filter (LPF) to smooth them. However, even with filtering, signal noise and vibrations make this method unsuitable for this application.

4.1.2 Torque Control

Torque Control directly regulates the torque applied at each joint, allowing for compliant motion and force-sensitive interactions. It enables the robot to respond dynamically to external disturbances by adjusting the force exerted at each joint in real time.

This allows smooth and natural motion, similar to human movement and is essential for dynamic balance and locomotion, as it accounts for contact forces.

Torque control was once considered the future of Reinforcement Learning-based control, as suggested by earlier studies [3]. However, more recent research has shown that torque control is highly sensitive to noise, where even minimal disturbances in sensors or feedback can significantly alter the policy's actions and result in a complete loss of control[15].

4.1.3 Proportional-Integral-Derivative Control

Proportional-Integral-Derivative (PID) control for torque regulation computes the required joint torque based on position error, ensuring accurate and stable motion. The controller calculates the torque as a combination of proportional, integral, and derivative terms, given by:

$$\tau(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt}, \qquad (3)$$

where $e(t)$ represents the position error, and K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively. This approach enables smooth and adaptive motion by dynamically adjusting torque in response to deviations from the desired trajectory, compensating for external disturbances and system dynamics.

4.2 Collisions

As mentioned earlier, collision meshes with a high number of sides reduce the efficiency of physics engines and increase memory usage, especially during collision processing. To address this, we replaced all collision meshes with capsules, as they are the most efficiently processed shapes in MJX, the GPU-accelerated Mujoco simulator[4].

To test this we used en environment for the Franka robot[6] to learn to stay on the home position. The

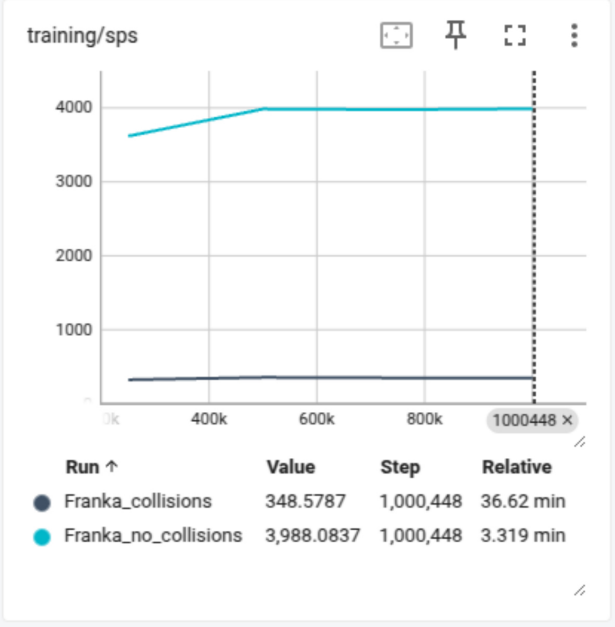

Figure 5: *Steps per second in Franka training.*

Figure 5 shows that the same Franka robot in simulation can perform ten times the amount of physics steps without any collision meshes than when applying collisions.

4.2.1 Capsules

To create the capsules to optimize the performance of the training simulator, we used a package that through optimization finds the smallest capsule that fits for every mesh specifically. Capsules for the robot description are composed by two spheres and a cylinder together, its size is determined by length, being the distance between the two sphere centers, and the radius of these, to determine the orientation it is possible to use a quaternion or two points in space that the capsule will go from one point to another. The result is shown in Figure 6.

4.3 Reward Validation

When training with reinforcement learning, all reward terms are applied at the same time, sometimes it is difficult to debug and weight them all at once. So to ensure each reward term is working as expected, we validated them individually with different environments.

4.3.1 Regularization Reward

Regularization reward is made to ensure that the joints that are not used for the main or other tasks keep a desired position or keeping the closest possible position while performing the task, also penalizing the velocity of the joints.

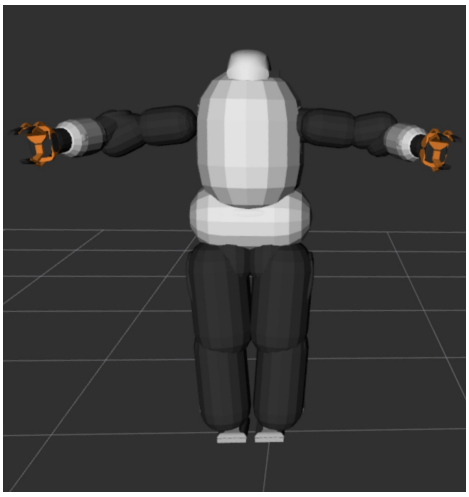

Figure 6: *Talos capsules.*

$$R_{\text{pose}} = w \cdot \exp\left(-\frac{1}{10}\left\|\mathbf{W}\cdot\begin{bmatrix}\mathbf{p}_{\text{target}}-\mathbf{p}\\ \text{quat\_diff}(\mathbf{q}_{\text{target}},\mathbf{q})\\ \boldsymbol{\theta}_{\text{target}}-\boldsymbol{\theta}\\ \dot{\mathbf{q}}\end{bmatrix}\right\|\right)$$

To validate this function the chosen robot is the Franka Research 3 from Franka Robotics, the goal of this environment is to keep the initial position of the Franka Research 3 for the whole duration of the episode. At first it was not accurately still in position so it was found that the reward function or weights were not correct. Then the weight of the joint velocities penalty was reduced 10x and was performing almost perfectly. The reward function shows it in Figure 7

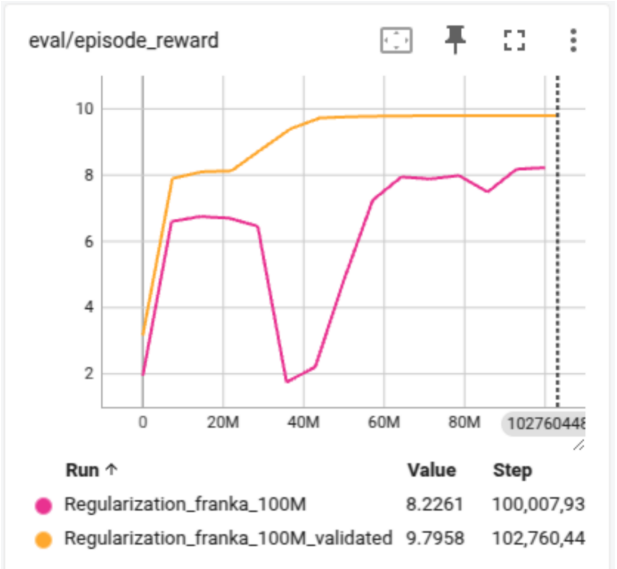

Figure 7: *Regularization reward weight tuning.*

To finally validate the original function we applied the changes with the knowledge acquired with the previous environments with an exact replica of the original environment without the floating base. This change

makes the robot base fixed in space and consequently not being affected by gravity as a whole but each link still is under the gravity force. With this we achieved the Talos robot to stay in its original position.

4.3.2 Energy Reward

Energy reward, or penalization it is used to reduce as much as possible the variance in torque applied to the joints. To validate this reward function we used the previously validated regularization reward.

$$R_{\text{energy}} = w \cdot \exp\left(-\frac{\|\mathbf{u}\|}{5}\right)$$

The regularization reward, although it was staying in the desired position, had small vibrations. The main purpose of this reward function is to remove or reduce this vibration as much as possible. So we re-trained the same environment as previously done, but added the reward for minimizing actions. As a result, the robot was significantly more stable, and there were almost no visible vibrations.

4.3.3 Tracking Reward

The tracking reward guides the robot to follow reference velocity commands, including both linear and angular components. This type of reward is the key part in tasks involving locomotion, navigation, or dynamic movement, where the robot must align its actual motion with targets given on the observation.

The reward function encourages the robot to minimize the squared error between the commanded and actual velocities, and it uses an exponential decay to produce smooth gradients during learning.

$$R_{\text{tracking}} = \exp\left(-\frac{1}{\sigma}\left\|\begin{bmatrix}\mathbf{v}^{xy}_{\text{cmd}} - \mathbf{v}^{xy}_{\text{robot}} \\ \omega^{z}_{\text{cmd}} - \omega^{z}_{\text{robot}}\end{bmatrix}\right\|^2\right)$$

where:

- $\mathbf{v}^{xy}_{\text{cmd}}$ is the commanded linear velocity on the XY plane.
- $\mathbf{v}^{xy}_{\text{robot}}$ is the robot's actual linear velocity on the XY plane.
- ω^{z}_{cmd} is the commanded yaw angular velocity.
- $\omega^{z}_{\text{robot}}$ is the robot's actual yaw angular velocity.
- σ is the tracking sensitivity (is different for every one).

This reward formulation was validated using a Google Barkour robot[2], Figure 8 in a velocity command-following environment. During early experiments, tracking performance was unstable at higher commanded velocities. Reducing the sensitivity parameter σ allowed the reward function to tolerate minor deviations while still encouraging accurate trajectory following, resulting in more robust and natural locomotion.

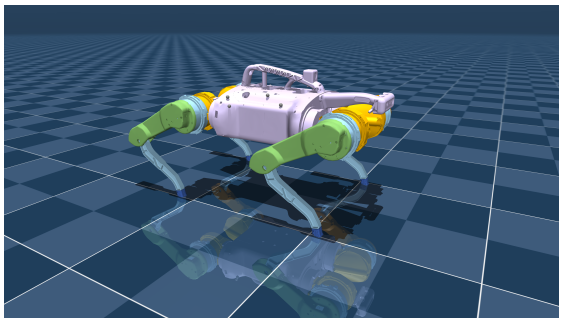

Figure 8: *Google Barkour.*

4.4 Reward Function

The policy is trained using a weighted sum of four key rewards: regularization, energy minimization, velocity tracking, and gait phase tracking. Each component encourages stability, efficiency, and accurate motion.

$$R_{\text{total}} = w_1 \cdot R_{\text{tracking}} + w_2 \cdot R_{\text{pose}} + w_3 \cdot R_{\text{energy}} + w_4 \cdot R_{\text{feet\_phase}}$$

$$R_{\text{tracking}} = \exp\left(-\frac{1}{\sigma}\left\|\begin{bmatrix}\mathbf{v}^{xy}_{\text{cmd}} - \mathbf{v}^{xy}_{\text{robot}} \\ \omega^{z}_{\text{cmd}} - \omega^{z}_{\text{robot}}\end{bmatrix}\right\|^2\right)$$

$$R_{\text{pose}} = w \cdot \exp\left(-\frac{1}{10}\left\|\mathbf{W} \cdot \begin{bmatrix}\mathbf{p}_{\text{target}} - \mathbf{p} \\ \text{quat\_diff}(\mathbf{q}_{\text{target}}, \mathbf{q}) \\ \boldsymbol{\theta}_{\text{target}} - \boldsymbol{\theta} \\ \dot{\mathbf{q}}\end{bmatrix}\right\|\right)$$

$$R_{\text{energy}} = w \cdot \exp\left(-\frac{\|\mathbf{u}\|}{5}\right)$$

$$R_{\text{feet\_phase}} = \exp\left(-\frac{1}{\epsilon}\sum_{i=1}^{N}\left(h_i - h_{\text{ref},i}\right)^2\right)$$

Each of these rewards targets a specific aspect of the robot's performance:

- **Regularization Reward** ensures unused or secondary joints maintain stable and desirable positions, preventing erratic behavior and enforcing a natural posture.
- **Energy Reward** penalizes high or rapidly changing torques, which encourages smoother, more efficient motions and reduces joint vibrations.
- **Tracking Reward** guides the robot to follow external velocity commands, both linear and angular, forming the core behavior of command-following locomotion.
- **Gait Phase Tracking Reward** aligns the motion of the robot's feet with a phase-based target pattern, encouraging smooth and periodic stepping behavior.

Gait Phase Tracking Reward[27]
This reward encourages the feet of the robot to follow a target vertical trajectory based on a predefined gait phase. The reward is computed by comparing the height of the feet with a reference swing height curve and penalizing the deviation.

$$R_{\text{feet\_phase}} = \exp\left(-\frac{1}{\epsilon}\sum_{i=1}^{N}\left(h_i - h_{\text{ref},i}\right)^2\right)$$

Where:

- h_i is the actual height of foot i
- $h_{\text{ref},i}$ is the reference swing height given by the gait generator
- N is the number of feet
- ϵ is a scaling factor for sensitivity

This reward was validated using a quadruped setup in a command-based locomotion environment. The inclusion of the gait phase reward significantly improved stepping consistency and symmetry, especially at higher commanded speeds. The combination of this with the Tracking Reward produced smoother and more natural movement transitions.

Combined Policy The final policy is learned through reinforcement learning using the total reward:

$$R_{\text{total}} = w_1 R_{\text{tracking}} + w_2 R_{\text{pose}} + w_3 R_{\text{energy}} + w_4 R_{\text{feet\_phase}}$$

Where w_1, w_2, w_3, and w_4 are weighting terms tuned through experimentation.

4.5 Sim2Real

4.5.1 Challenges in Sim2Real transfer

Transferring skills, knowledge, or models from a simulated environment to real-world applications can be challenging due to the following factors:

- **Reality gap:** The discrepancy between the simulated environment and the real world is known as the reality gap. This gap can result from differences in physical properties, sensor noise, or actuator dynamics, which can lead to a decline in performance when transferring a model from simulation to reality.
- **Computational complexity:** Running accurate and high-fidelity simulations can be computationally expensive and slow, which might limit the ability to train and test models efficiently.
- **Model limitations:** Simplified assumptions in a simulation, such as frictionless surfaces or idealized sensors, can lead to discrepancies when transferring models to real-world scenarios.

4.5.2 Training

To test sim2real a policy we used the TIAGo[19] arm, showed in Figure 9 to create a training environment in Brax. The environment was made to perform inverse kinematics with the PAL Robotics TIAGo Robot.

The reward function used is the following:

$$R_{\text{target}} = \exp\left(-\frac{\|\mathbf{p}_{\text{target}} - \mathbf{p}_{\text{current}}\|}{0.5}\right)$$

where:

- $\mathbf{p}_{\text{target}}$ is the target position.
- $\mathbf{p}_{\text{current}}$ is the current position of the arm.

$$R_{\text{velocity}} = -\sum_i \mathbb{I}\left(|\Delta\theta_i| > 0.02\right)$$

where:

- $\Delta\theta_i$ is the velocity change in the joint.
- $\mathbb{I}$ is the indicator function, returning 1 if the condition is true and 0 otherwise.

$$R_{\text{acceleration}} = 2\sum_i \left(\mathbf{u}_i - \mathbf{u}_i^{\text{last}}\right)$$

where:

- $\mathbf{u}_i$ is the current control action for joint i.
- $\mathbf{u}_i^{\text{last}}$ is the previous control action.

$$R_{\text{limits}} = 5 \cdot \sum_i \max(0, \text{safe\_margin} - \left|\mathbf{q}_i - \frac{\min_i + \max_i}{2}\right|)^2$$

where:

- $\mathbf{q}_i$ is the position of joint i.
- $\min_i, \max_i$ are the joint's minimum and maximum limits.
- safe_margin is a small margin from the limits to avoid penalties.

The total reward is a weighted sum of all the individual rewards:

$$R_{\text{total}} = 4 \cdot R_{\text{target}} + R_{\text{velocity}} - R_{\text{limits}} - R_{\text{acceleration}}$$

4.5.3 Inference Node

To test a trained policy we used a ROS2 node that controls a robot arm using the Policy Manager package we created to train. The node subscribes to the robot's joint states (position and velocity) and target position, this position and velocity are published by a **Gazebo** simulation running at the same time. Also target position is published by another node developed that creates an interactive marker that can be controlled with **RViz**. Then with the appropriate observations runs inference and publishes those actions to the robot's joint controller.

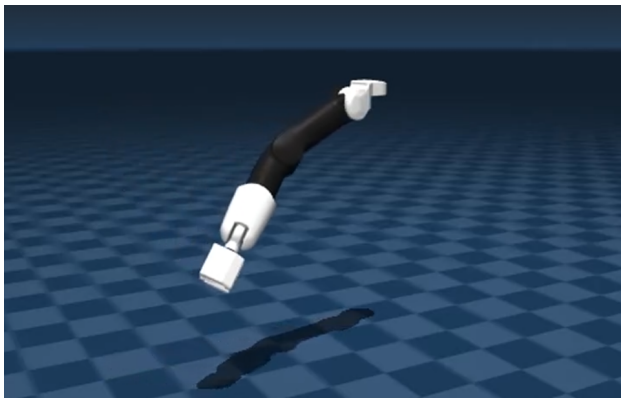

Figure 9: *Tiago arm.*

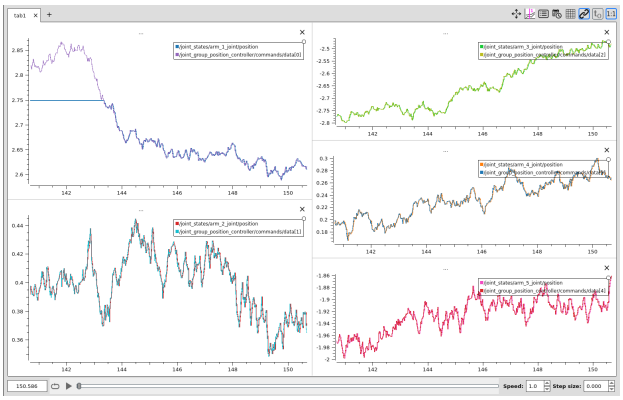

Figure 11: *Noisy Signal.*

4.5.4 Gazebo

Gazebo is an open-source 2D/3D robotics simulator. Integrates with ROS (Robot Operating System), enabling real-time control and testing of robotic algorithms. It also allows the creation of custom robot models and supports multi-robot simulations.

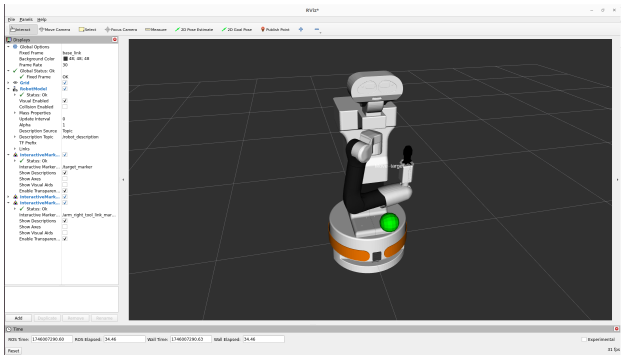

Figure 10: *TIAGo in RViz with the interactive marker.*

4.5.5 RViz

RViz is a 3D visualization tool within the Robot Operating System (ROS) used to visualize robot sensor data, state, and the environment like in Figure 10, where we can visualize the robot and the marker to publish the target point. It allows developers to see data from cameras, lidar, and other sensors, along with robot models and joint states. RViz streams real-time data, enabling interactive debugging and testing.

4.5.6 Signal Filters

In signal processing, a filter is a device or process that removes some unwanted components or features from a signal. In this case, for robot control filters are used to smooth the signal that the actuators receive so it can handle actions correctly. Several filters are employed, with the most commonly used being:

- **Low-pass filter**: Removes high-frequency noise by allowing low-frequency signals to pass through, smoothing out rapid fluctuations.
- **Median filter**: Replaces each data point with the median of its neighbors to remove spikes and impulsive noise.
- **Moving Average Filter**: Averages neighboring data points to smooth the signal and reduce high-frequency noise.

In our case, analizing the raw signal in Figure 11 we just needed to remove close to zero values as we could not get zero outputs from the Neural Network. These small values create noise when the arm is supposed to be completely static, so to clean the signal we changed values lower than 0.01 directly to 0.0 and the same when the previous outputs have a different sign also are set to 0.0. With just this filters to the outputs we achieved a viable signal.

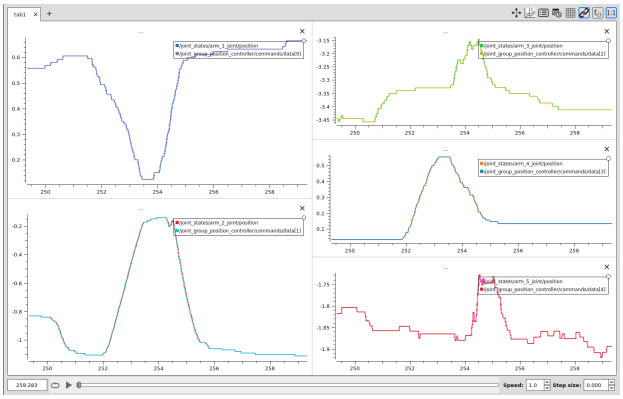

Figure 12: *Clean signal.*

This filtered joint signals shown in the plots in Figure 12 are suitable for real robot control because they exhibit smooth, continuous behavior with minimal noise and closely track the commanded positions. This indicates that the actuators are accurately following the desired trajectories, ensuring stable and reliable motion execution without abrupt changes or oscillations.

5 Discussion of results

5.1 Network Architecture

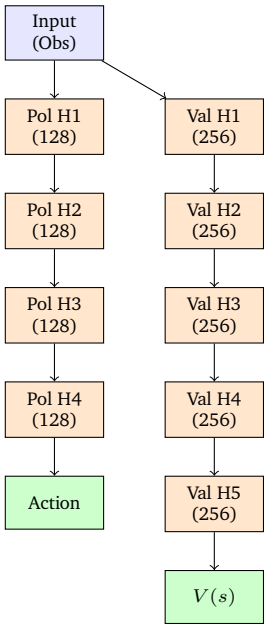

Figure 13: *PPO Architecture: Policy and Value Networks*

The PPO agent uses two separate neural networks: a policy network and a value network. The policy network takes an observation, passes it through four fully connected layers with 128 units each and with sigmoid activations, and outputs continuous actions scaled with a tanh activation. The value network also processes the same observation input but through a deeper architecture with five fully connected layers of 256 units each, producing a single scalar that estimates the state value, this is shown in Figure 13. This separation allows efficient learning of both the action policy and value function, critical for PPO's stability and performance.

5.2 Talos

The study began solely with Talos Robot, the whole reward function and parameters were tuned using this robot, but also keeping in mind that one of the goals of this project is to be able to easily train different humanoid robots.

Once the reward function was validated, we used it in combination with the Brax simulation environment. This led to the following results, depicted in Figure 14.

Looking at the reward graph, we observed that stable learning only occurred when using direct torque control. However, as we discussed earlier in Section 4.1, this method isn't ideal for our use case and doesn't scale well to more complex reinforcement learning tasks.

Although these results were perceived as a failure, we did achieve a stable Policy for Talos Robot. That confirmed the reward function itself was designed correctly. And on top of that, we successfully tested our

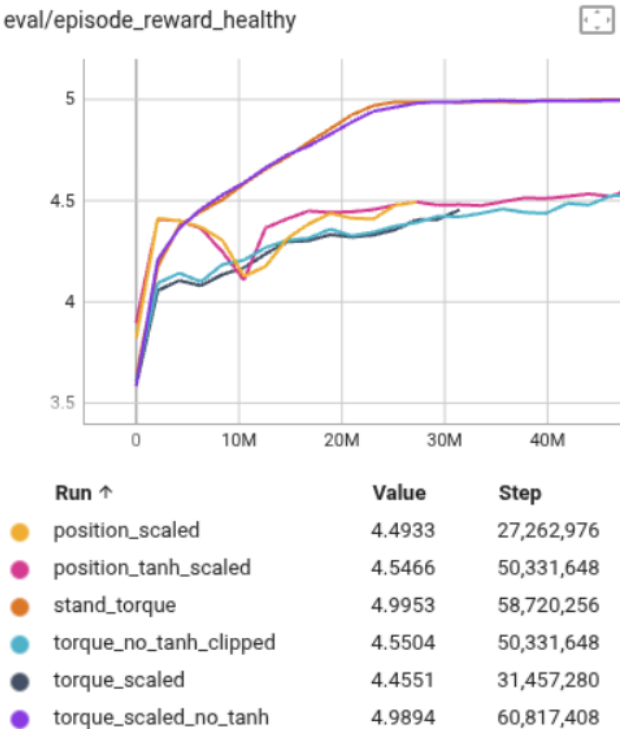

Figure 14: *Talos reward learning.*

training interface, and encouraged us to move on and train another humanoid robot.

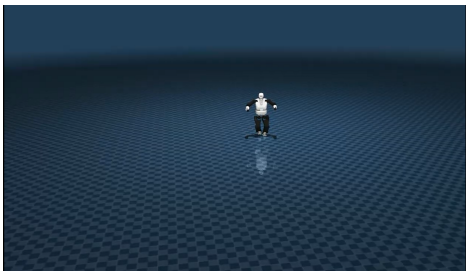

Figure 15: *Talos walking with learned policy (click image to watch video).*

By analyzing the video in Figure 15, we can see that the robot is following the commanded velocity vector accurately, and the resulting gait appears to be acceptable. However, it's also clear that the regularization term in the reward function isn't weighted strongly enough, and will need to be adjusted to improve the quality of the movements.

5.3 Kangaroo

The training results for the Kangaroo robot were much better right from the start. After our experience with Talos, we were able to reuse most of the training setup and reward function, with only small changes.

Although the reward function was designed for Talos, Kangaroo moves in a different way, so we had to fine-tune some of the reward weights. This helped the robot

learn more natural and stable walking, based on how it reacted during training.

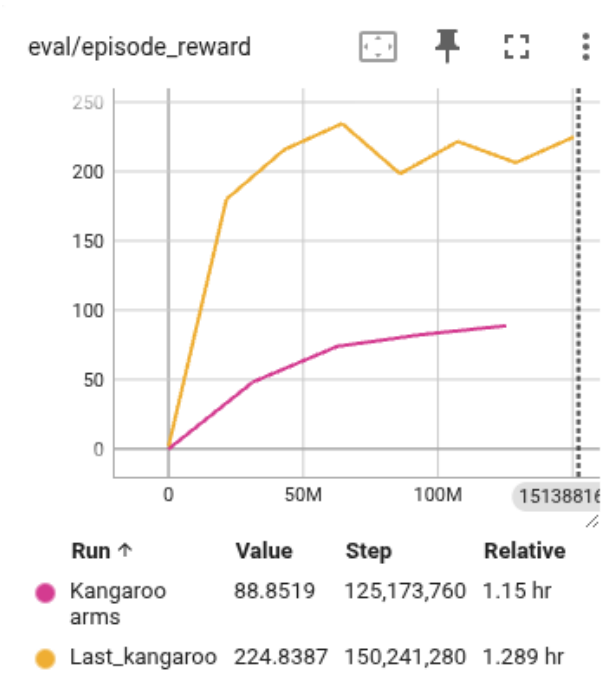

Figure 16: *Kangaroo reward learning.*

Once these changes were made, the training process went smoothly as we can observe in Figure 16. The learning curve improved quickly, and the robot began walking in a consistent and stable way observable in Figure 17 video. Using our control method and the PID controller, Kangaroo can follow the velocity commands and walk correctly.

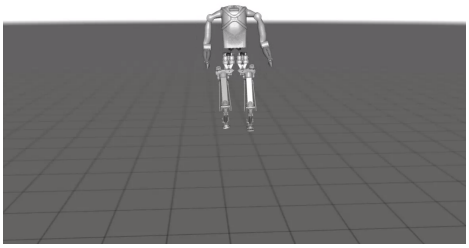

Figure 17: *Kangaroo walking with learned policy (click image to watch video).*

6 Conclusions

This project aimed to develop a robust, adaptable, and safe locomotion policy for humanoid robots using reinforcement learning. Based on the results, we can conclude that the approach we followed is valid and can be used to train stable walking behaviors on different humanoid robots.

We first worked with the Talos robot to design and validate the reward function and training pipeline. Even though we faced some challenges—mainly related to the use of torque control and the specific limitations of the Talos simulation—we still managed to train a working policy. This was a key milestone, as it confirmed that the reward function was correctly designed and that our training tools were working as expected.

Then, we moved on to the Kangaroo robot. Thanks to the general setup and reusable code, we didn't need to make many changes to start training. Just a few adjustments to the reward weights were enough to adapt it to the Kangaroo's dynamics. The training process went much smoother than with Talos, and the robot quickly learned to walk in a stable and natural way. The walking behavior was consistent, and the robot could follow the commanded velocity reliably.

The results show that our architecture and training process are general and adaptable. We were able to apply the same method to two different humanoid robots, which confirms the idea that our interface can be extended to other robots with little effort.

This work has been quite successful, and even though the project is now finished, the process will continue. The policies and tools developed will be used in future work to test and deploy the walking behavior on a real robot.

6.1 Comparison to Initial Objectives

Let's now compare what we achieved to the original goals of the project:

- **Develop a robust and adaptable locomotion policy:** This was achieved. We trained stable policies for both Talos and Kangaroo, even though they have different designs and dynamics.
- **Enable recovery from disturbances and maintain trajectory:** Partially achieved. We did test external disturbances, and the robots were able to follow the velocity commands correctly. However it wasn't trained with disturbances, it was learned as a consequence of itself losing balance.
- **Minimize abrupt accelerations and mechanical stress:** Partially achieved. The walking looked smooth, but in the case of Talos, we noticed that the regularization term in the reward function needs to be stronger. More tuning is needed to fully address actuator stress.
- **Create a generalizable architecture and tools:** Achieved. We reused the same training setup, reward function, and policy architecture for two different robots. Only minor changes were needed, proving that the system is flexible and general.

7 Limitations and future research

7.1 Training Talos limitations

7.1.1 Exploding Gradient

One of the main limitations we encountered during the initial training attempts with the Talos robot was that the learning process consistently crashed due to an exploding gradient problem. After investigating the issue, we discovered that BRAX did not include basic gradient management features such as gradient clipping or a properly integrated optimizer at the time. This was surprising, considering that Brax had already been publicly available for over three years. We reported the problem by opening an issue on the project's repository, and the developers responded that gradient clipping had just been implemented. Once we integrated this new feature into our training setup, the instability was resolved, and the training was able to progress normally

7.1.2 Training Challenges with PID and Torque Control on Talos

During training on the Talos robot, PID control failed to produce stable locomotion despite extensive parameter tuning, primarily because PID relies on accurate models and lacks the flexibility needed for exploratory reinforcement learning. Switching to direct torque control allowed the policy to work, by commanding joint forces directly, enabling successful walking in simulation. However, this approach is unsuitable for real-world deployment as discussed in Section 4.1.

7.1.3 Simulation vs. Reality

Another major problem is that the simulation model of Talos is not completely realistic. Things like joint friction, armature, motor delays, or how the robot touches the ground are either simplified or not modeled accurately. Because of this, the robot in simulation behaves differently from the real one. A policy that works well in simulation might take advantage of these simplifications and end up failing or even causing damage when used on the real robot. This also affects the reward function: behaviors that seem good in simulation don't always give the same results in real life.

7.1.4 Safety Concerns

Using torque control in the real robot also raises serious safety concerns. Because the robot has strong joints and no built-in safety layers when using torque control, even a small error or disturbance can lead to dangerous behavior. In reinforcement learning, where the robot is constantly exploring new actions, these risks are even higher. It's easy for the robot to end up in situations where it moves too quickly or applies too much force, which could break parts of the robot or create hazards in the lab environment.

7.2 Kangaroo Limitations

A key limitation when trying to run policies on the real Kangaroo robot is related to how the joints are actuated. In simulation, we control the joints directly and works in any way, but on the real robot, the actuators are not at the joints. Instead, they are placed closer to the base and move the joints through transmissions, like pistons. This causes problems because the control signal doesn't go straight to the joint, and the transmission changes how the signal behaves.

If we control in position, the transmissions handle it well, but the movements become stiff and jerky. This can be dangerous for the robot, especially if the policy sends sharp or noisy commands. One alternative is using impedance control, which is like position control but with soft PID gains. This makes the actuators act more like springs, which is safer and smoother. However, controlling in torque is still unreliable. Even with tuned controllers, torque signals often don't produce stable behavior due to the complexity of the transmissions.

We discussed this with partners at PAL Robotics, and they mentioned that their humanoid robot uses dynamic gain adjustment to make torque control work. Since the joints aren't directly actuated, they compute new gains at every step to match the joint behavior more accurately. This approach could be useful for future work, but it requires accurate modeling of the transmissions, which we don't yet have in simulation.

8 Acknowledgements

I would like to express my sincere gratitude to my thesis tutor, Dr. Javier Panadero, for his guidance and support throughout this project. I also want to thank the engineering team at PAL Robotics for their valuable insights and collaboration, as well as all the professors of my bachelor's degree for providing the knowledge and foundation that made this work possible.

9 Bibliography

References

[1] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. Accessed 27/02/2025. 2018. URL: http://github.com/jax-ml/jax.

[2] Ken Caluwaerts et al. "Barkour: Benchmarking animal-level agility with quadruped robots". In: *arXiv preprint arXiv:2305.14654* (2023).

[3] Shuxiao Chen et al. *Learning Torque Control for Quadrupedal Locomotion*. Accessed 27/02/2025. 2023. arXiv: 2203.05194 [cs.RO]. URL: https://arxiv.org/abs/2203.05194.

[4] DeepMind. *MuJoCo XML Reference*. https://mujoco.readthedocs.io/en/stable/XMLreference.html. Accessed: 2025-05-15. 2024.

[5] Hossam El-Hussieny et al. "Real-time deep learning-based model predictive control of a 3-DOF biped robot leg". In: *Scientific Reports* 14 (2024), p. 16243. DOI: 10.1038/s41598-024-66104-y.

[6] Franka Emika. *Franka Research 3*. Accessed: 2025-03-27. 2025. URL: https://franka.de/products/franka-research-3.

[7] C. Daniel Freeman et al. *Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. Version 0.12.1. Accessed 27/02/2025. 2021. URL: http://github.com/google/brax.

[8] Jessy W. Grizzle and Eric R. Westervelt. "Hybrid Zero Dynamics of Planar Bipedal Walking". In: *The International Journal of Robotics Research* 21.10 (2002), pp. 929–940. DOI: 10.1177/027836402320690555.

[9] Benjamin Henze. *Whole-Body Control for Multi-Contact Balancing of Humanoid Robots: Design and Experiments*. Vol. 143. Springer Tracts in Advanced Robotics. Springer, 2021. ISBN: 9783030872118. DOI: 10.1007/978-3-030-87212-5.

[10] O. Khatib. "A unified approach for motion and force control of robot manipulators: The operational space formulation". In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53. DOI: 10.1109/JRA.1987.1087068.

[11] Sotetsu Koyamada et al. "Mjx: A framework for Mahjong AI research". In: (2025). Accessed 27/02/2025.

[12] Bart van Marum et al. "Revisiting reward design and evaluation for robust humanoid standing and walking". In: *arXiv preprint arXiv:2404.19173v2 [cs.RO]* (2024). Submitted on 30 Apr 2024, last revised 30 Aug 2024. DOI: 10.48550/arXiv.2404.19173. URL: https://doi.org/10.48550/arXiv.2404.19173.

[13] James Mock and Suresh Muknahallipatna. "A Comparison of PPO, TD3 and SAC Reinforcement Algorithms for Quadruped Walking Gait Generation". In: *Journal of Intelligent Learning Systems and Applications* 15 (Jan. 2023), pp. 36–56. DOI: 10.4236/jilsa.2023.151003.

[14] MuJoCo Community. *MuJoCo MJX*. Accessed 27/02/2025. 2025. URL: https://mujoco.readthedocs.io/en/stable/mjx.html.

[15] Hiroshi Otomo et al. "Adversarial Attacks on Torque-Controlled Legged Robots via Deep Reinforcement Learning". In: *arXiv preprint arXiv:2205.10098* (2022). URL: https://arxiv.org/abs/2205.10098.

[16] Aravind Rajeswaran et al. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2017.

[17] PAL Robotics. *Talos Robot*. Accessed: 2025-05-02. 2025. URL: https://pal-robotics.com/robot/talos/.

[18] PAL Robotics. *Talos Robot*. Accessed: 2025-05-02. 2025. URL: https://pal-robotics.com/robot/kangaroo/.

[19] PAL Robotics. *TIAGo Robot*. Accessed: 2025-05-02. 2025. URL: https://pal-robotics.com/robot/tiago/.

[20] ROS Community. *URDF - Unified Robot Description Format*. Accessed 27/02/2025. 2025. URL: https://wiki.ros.org/urdf.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[22] Daniella Tola and Peter Corke. "Understanding URDF: A Survey Based on User Experience". In: *arXiv* (2023). Accessed 27/02/2025. arXiv: 2302.13442 [cs.RO]. URL: https://arxiv.org/abs/2302.13442.

[23] Constantin Volosencu. *Model Predictive Control - Theory and Applications*. July 2023. DOI: 10.5772/intechopen.1001523.

[24] Miomir Vukobratovic and Davor Juricic. “Contribution to the Synthesis of Biped Gait”. In: *IEEE Transactions on Biomedical Engineering* BME-16.1 (1969), pp. 1–6. DOI: 10.1109/TBME.1969.4502596.

[25] Miomir Vukobratović and Branislav Borovac. “Zero-Moment Point—Thirty Five Years of its Life”. In: *International Journal of Humanoid Robotics* 1.1 (2004), pp. 157–173. DOI: 10.1142/S021984360400027X.

[26] E.R. Westervelt, J.W. Grizzle, and D.E. Koditschek. “Hybrid zero dynamics of planar biped walkers”. In: *IEEE Transactions on Automatic Control* 48.1 (2003), pp. 42–56. DOI: 10.1109/TAC.2002.806653.

[27] Kevin Zakka et al. *MuJoCo Playground: An open-source framework for GPU-accelerated robot learning and sim-to-real transfer*. 2025. URL: https://github.com/google-deepmind/mujoco_playground.