COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

TÍTULO DEL TRABAJO

Autor: Pablo Gómez Martínez

Director: Lucía Güitta López

Co-Director: Álvaro Jesús López López

Madrid

Junio 2025

Desarrollo y Entrenamiento de un Robot Bípedo con Técnicas de Aprendizaje Por Refuerzo

Autor: Pablo Gómez Martínez
Director: Güitta López, Lucía.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Resumen

Este trabajo presenta el diseño y construcción de un robot bípedo de bajo coste, entrenado mediante aprendizaje por refuerzo profundo. Mediante una metodología basada en simulación utilizando MuJoCo y PyTorch se compararon algoritmos para control continuo: DDPG, D4PG, SAC y MPO. Se diseñó la función de recompensa, logrando una locomoción estable y natural. El modelo final muestra resultados prometedores y establece una plataforma para futuras transferencias a la realidad. Este proyecto se centró en el diseño, los algoritmos y los resultados clave de la creación de un robot bípedo, con el objetivo de democratizar la investigación en robótica humanoide.

Palabras clave: aprendizaje por refuerzo profundo, robot humanoide, SAC, D4PG, DDPG, MPO, locomoción bípeda, diseño de función de recompensa, MuJoCo

1. Introducción

El objetivo del proyecto es democratizar la locomoción robótica avanzada utilizando hardware accesible y técnicas modernas de inteligencia artificial. Los robots bípedos requieren estrategias de control complejas para mantener el equilibrio y la estabilidad. Inspirado por plataformas académicas como Cassie [1] y la investigación de DeepMind sobre aprendizaje por refuerzo en control continuo [2], este proyecto intenta reproducir y probar métodos similares en hardware no profesional.

2. Definición del Proyecto

El objetivo fue diseñar y entrenar un robot bípedo utilizando aprendizaje por refuerzo (RL) en simulación. Se construyó un robot personalizado con piezas impresas en 3D y servomotores, modelado digitalmente en Fusion 360 y simulado en MuJoCo [3]. El proyecto compara cuatro algoritmos de RL para control continuo: DDPG [4], D4PG [5], SAC [6], y MPO [7], usando una arquitectura consistente y observando su rendimiento bajo el mismo entorno y función de recompensa.Los objetivos secundarios incluyen:

- Diseñar una función de recompensa modular y extensible para caminar [8][9].
- Explorar la aleatorización del entorno para la transferencia de simulación a mundo real [10].
- Construir un código abierto para liberar públicamente.

3. Descripción del Sistema.

El robot cuenta con 12 servomotores controlados por un Jetson Nano [11] y simulado usando MuJoCo. Su gemelo digital fue exportado automáticamente desde Fusion 360 a URDF y convertido a un XML compatible con MuJoCo utilizando fusion2urdf [12], una librería de código abierto. El espacio de observación incluye posición y orientación del torso, velocidades y posiciones y velocidades de las articulaciones, muestreadas a 100 Hz.

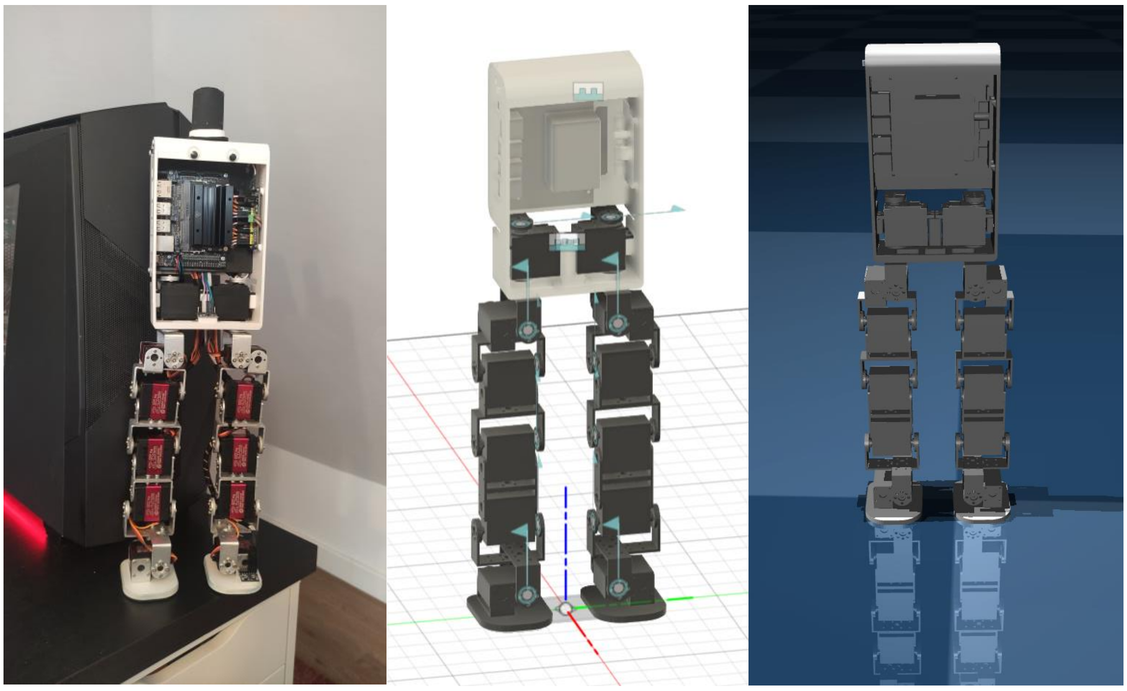

Figura 1: Diseño del robot

Diseño final del robot en la realidad, Fusion 360 y MuJoCo respectivamente.

La arquitectura de control consiste en una red neuronal MLP de 2 capas (256,256) con una arquitectura actor-crítico, con modelos LSTM y CNN-1D implementados para su uso futuro en entornos más complejos donde se necesiten observaciones pasadas para un buen rendimiento. [2][13].

4. Resultados

La primera etapa del esfuerzo experimental se centró en la ingeniería de la función de recompensa. Se introdujo iterativamente una serie de componentes de recompensa. Esto culminó en una función de recompensa modular que logra que un robot camine de forma efectiva y natural [8][9].

Término de Recompensa	**Definición**	**Peso**
Sobrevivió un paso	1	0.001
Velocidad	$e^{-5\cdot\lvert v_{xy}-c_{xy}\rvert^2}$	0.15
Torque	$e^{-0.02\cdot\frac{1}{N}\sum\frac{\lvert t_{motor}\rvert}{t_{max}}}$	0.01
Diferencia de acción	$e^{-0.02\cdot\sum\lvert a_t-a_{t-1}\rvert}$	0.01
Aceleración del torso	$e^{-0.01\cdot\sum\lvert b_{x\,y\,z}\rvert}$	0.05

Orientación Yaw	$e^{-30 \cdot qd(q_{yaw}, c_{yaw})}$	0.02
Orientación Pitch y Roll	$e^{-30 \cdot qd(q_{pitch,roll}, c_{pitch,roll})}$	0.04
Orientación de los pies	$e^{-30 \cdot \sum qd(q_{feet,ypr}, c_{feet,ypr})}$	0.3
Centrado del torso	$e^{-20 \cdot \lvert p_{xy} - \frac{p_{r\_feet,xy} + p_{l\_feet,xy}}{2} \rvert^2}$	0.1
Contacto del pie	$\begin{cases} \Delta t_{td} \text{ } if \text{ } \mathbb{1}_{td,f} \text{ } and \text{ } p_{f,x} > p_{f',x} \text{ } for \text{ } any \text{ } f \in [left, right] \\ 0 \text{ } otherwise \end{cases}$	$1.5^{\dagger}$

Table 1. Elementos de la Función de Recompensa Final

c = un comando; q = un cuaternión; p = una posición; b = aceleración del torso; $qd(\cdot)$ = función de distancia entre cuaterniones; $\mathbb{1}_{td}$ = variable booleana que indica un nuevo contacto con un pie distinto del anterior; Δt_{td} = diferencia temporal entre el último contacto premiado y el actual; † = nota: al ser esta la única recompensa dispersa, su peso es significativamente mayor que los demás términos.

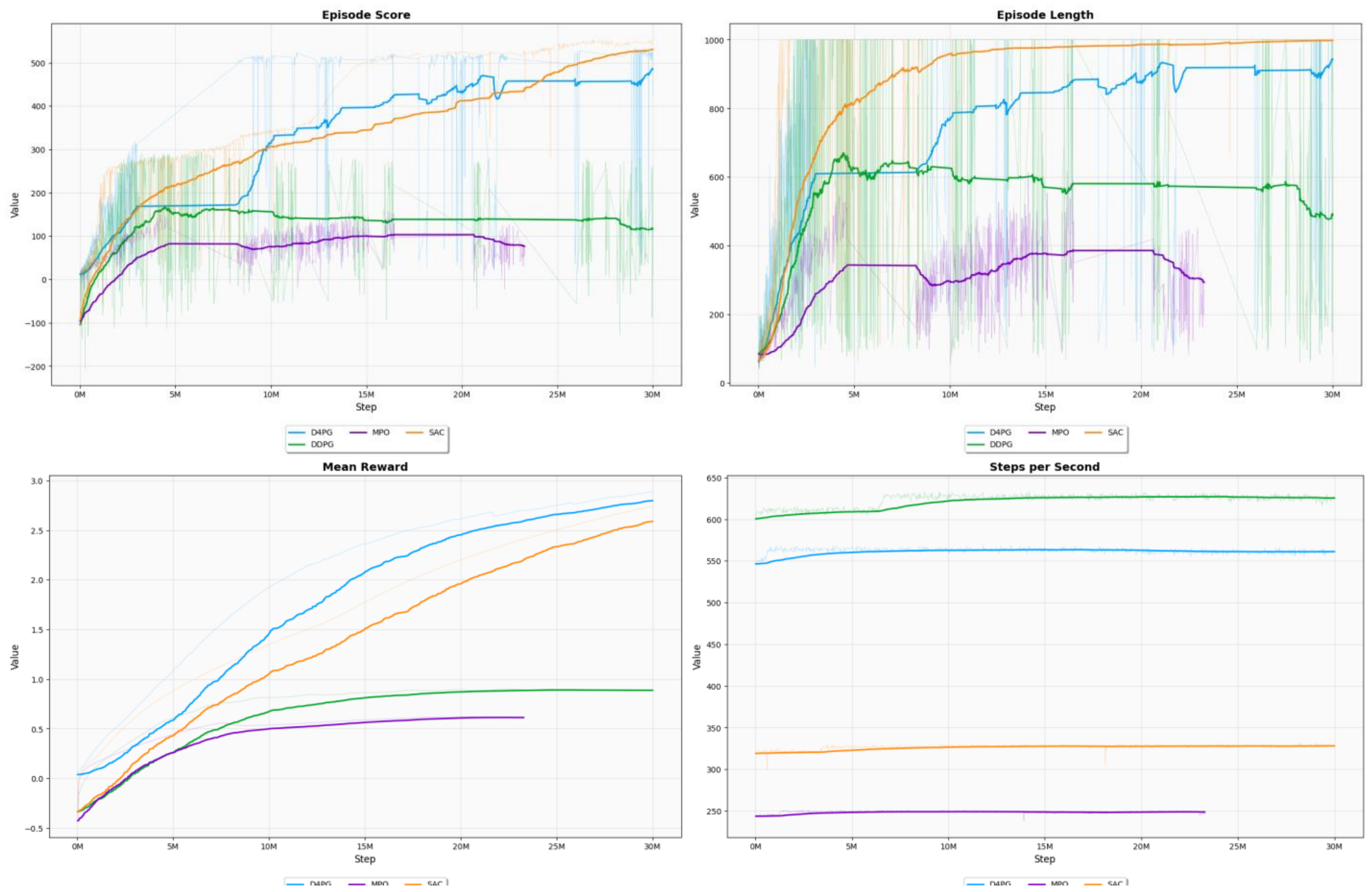

Figura 2: Resultados finales de los algoritmos

Gráficas que muestran los resultados tras ejecutar DDPG, D4PG, MPO y SAC. Se muestran, en orden, las puntuaciones finales obtenidas por episodio a lo largo de los pasos, la duración de los episodios a lo largo de los pasos, la recompensa media por paso y los pasos por segundo. Todas las gráficas han sido suavizadas para mostrar mejor los resultados, ya que la salida real es muy ruidosa.

Cada algoritmo fue entrenado durante 30 millones de pasos en un entorno con un terreno plano. Las métricas principales analizadas fueron la recompensa final, la duración del episodio, la velocidad de entrenamiento y la calidad y estabilidad de la marcha. Los resultados fueron los siguientes:

- **SAC** logró la recompensa más alta y la marcha más natural, con episodios constantes de 1000 pasos.
- **D4PG** ofreció el mejor equilibrio entre velocidad y rendimiento, convergiendo rápidamente con una marcha estable.
- **MPO** mostró un mayor tiempo de entrenamiento y peor rendimiento, pero potencial en entornos más complejos.
- **DDPG** fue rápido y simple, pero inestable, con una calidad de marcha deficiente.

El código se encuentra disponible en el siguiente repositorio de GitHub:
https://github.com/AsterisCrack/BipedRobot

5. Conclusiones

Este trabajo demuestra que es posible entrenar políticas de locomoción bípedo estables utilizando técnicas modernas de aprendizaje por refuerzo en hardware no profesional. Al diseñar una función de recompensa modular y comparar cuatro algoritmos clave de RL, el proyecto ofrece un camino reproducible para futuros investigadores en robótica a bajo coste.

Los objetivos futuros incluyen pruebas en el mundo real, integración de modelos temporales y aleatorización del dominio para la transferencia simulación-realidad.

6. Referencias

[1] Gong, Yukai, Ross Hartley, Xingye Da, Ayonga Hereid, Omar Harib, Jiunn-Kai Huang, Jessy Grizzle. "Feedback Control of a Cassie Bipedal Robot: Walking, Standing, and Riding a Segway." arXiv:1809.07279, 2018.

[2] Haarnoja, Tuomas, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, Michael Bloesch, Kristian Hartikainen, Arunkumar Byravan, Leonard Hasenclever, Yuval Tassa, Fereshteh Sadeghi, Nathan Batchelor, Federico Casarini, Stefano Saliceti, Charles Game, Neil Sreendra, Kushal Patel, Marlon Gwira, Andrea Huber, Nicole Hurley, Francesco Nori, Raia Hadsell, Nicolas Heess. "Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning." arXiv:2304.13653, 2023.

[3] E. Todorov, T. Erez and Y. Tassa, “MuJoCo: A Physics Engine for Model-Based Control.” IEEE Xplore, 2012

[4] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. "Continuous control with deep reinforcement learning." arXiv:1509.02971, 2015.

[5] Barth-Maron, Gabriel, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap. "Distributed Distributional Deterministic Policy Gradients." arXiv:1804.08617, 2018.

[6] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." arXiv:1801.01290, 2018.

[7] Abdolmaleki, Abbas, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, Martin Riedmiller. "Maximum a Posteriori Policy Optimisation." arXiv:1806.06920, 2018.

[8] Marum, Bart van, Aayam Shrestha, Helei Duan, Pranay Dugar, Jeremy Dao, Alan Fern. "Revisiting Reward Design and Evaluation for Robust Humanoid Standing and Walking." arXiv:2404.19173, 2024.
[9] Tassa, Yuval, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, Martin Riedmiller. "DeepMind Control Suite." arXiv:1801.00690, 2018.
[10] Ha, Sehoon, Joonho Lee, Michiel van de Panne, Zhaoming Xie, Wenhao Yu, Majid Khadiv. "Learning based legged locomotion; state of the art and future perspectives." arXiv:2406.01152, 2024.
[11] Swaminathan, Tushar Prasanna, Christopher Silver, Thangarajah Akilan. "Benchmarking Deep Learning Models on NVIDIA Jetson Nano for Real-Time Systems: An Empirical Investigation." arXiv:2406.17749, 2024.
[12] syuntoku14, “fusion2urdf.” GitHub, 2025
[13] Lin, Sixu, Guanren Qiao, Yunxin Tai, Ang Li, Kui Jia, Guiliang Liu. "HWC-Loco: A Hierarchical WholeBody Control Approach to Robust Humanoid Locomotion."

ENGLISH VERSION

Abstract

This work presents a custom built and designed low-cost biped robot trained using deep reinforcement learning. A simulation-based methodology based on MuJoCo and Pytorch was developed to test and compare continuous deep reinforcement learning algorithms: DDPG, D4PG, SAC, and MPO. Special emphasis was placed on reward function engineering, achieving stable and natural locomotion. The final model shows promising results and establishes a platform for future sim-to-real transfer. This project was aimed towards the design, algorithms, and key outcomes of creating a biped robot, aiming to democratize humanoid robot research by simplifying the design and implementation process with a low-cost platform.

Keywords: deep reinforcement learning, humanoid robot, SAC, D4PG, DDPG, MPO, bipedal locomotion, reward function design, MuJoCo

1. Introduction

The project aims to democratize advanced robotic locomotion by using accessible hardware and modern AI techniques. Biped robots require complex control strategies to maintain balance and stability. Inspired by academic platforms like Cassie [1] and DeepMind's research on reinforcement learning in continuous control [2], this project attempts to reproduce and test similar methods on hobby-grade hardware.

2. Project Definition

The goal was to design and train a biped robot using reinforcement learning (RL) in simulation. A custom robot was built with 3D-printed parts and hobby servos, digitally modeled in Fusion 360, and simulated in MuJoCo [3]. The project compares four continuous control RL algorithms, DDPG [4], D4PG [5], SAC [6], and MPO [7], using a consistent architecture, observing their performance under the same environment and reward function.

Secondary objectives included:

- Designing a modular and extensible reward function for walking [8][9].
- Exploring environment randomization for sim-to-real transfer [10].
- Building a codebase for future public release.

3. System description

The robot features 12 servo motors controlled by a Jetson Nano [11] and simulated using MuJoCo. Its digital twin was automatically exported from Fusion 360 to URDF and converted to MuJoCo XML using fusion2urdf [12], an open-source library. The observation space includes torso position, orientation, velocities, and joint positions and velocities sampled at 100 Hz.

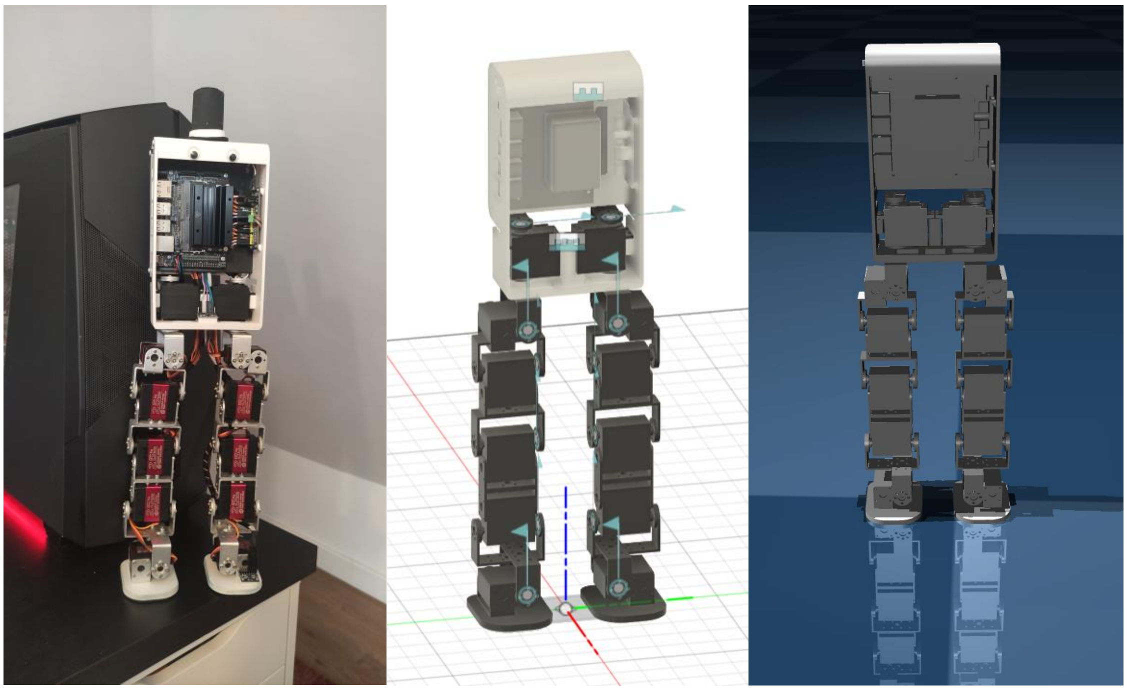

Figure 2: Robot design

The final hardware design of the robot in reality, Fusion 360 and MuJoCo respectively.

The control architecture consists of a 2-layer MLP (256,256) actor-critic architecture, with LSTM and 1D-CNN models implemented for future use in more complex environments where past observations are needed for good performance. [2][13].

4. Results

The first stage of experimental effort was directed toward reward function engineering. A sequence of reward components was introduced iteratively. This culminated in a modular reward function that achieves effective and natural walking [8][9].

Reward Term	**Definition**	**Weighting**
Survived a step	1	0.001
Velocity	$e^{-5\cdot\lvert v_{xy}-c_{xy}\rvert^2}$	0.15
Torque	$e^{-0.02\cdot\frac{1}{N}\sum\frac{\lvert t_{motor}\rvert}{t_{max}}}$	0.01
Action Difference	$e^{-0.02\cdot\sum\lvert a_t-a_{t-1}\rvert}$	0.01
Base Acceleration	$e^{-0.01\cdot\sum\lvert b_{x\,y\,z}\rvert}$	0.05
Yaw orientation	$e^{-30\cdot qd(q_{yaw},c_{yaw})}$	0.02
Pitch and Roll orientation	$e^{-30\cdot qd(q_{pitch,roll},c_{pitch,roll})}$	0.04
Feet orientation	$e^{-30\cdot\sum qd(q_{feet,ypr},c_{feet,ypr})}$	0.3

Torso centering	$e^{-20\cdot\lvert p_{xy}-\frac{p_{r\_feet,xy}+p_{l\_feet,xy}}{2}\rvert^2}$	0.1
Feet contact position	$\begin{cases}\Delta t_{td} \; if \; \mathbb{1}_{td,f} \; and \; p_{f,x} > p_{f',x} \; for \; any \; f \; \in \; [left, right] \\ 0 \; otherwise\end{cases}$	1.5†

Table 2. Elements of the Final Reward Function

c = a command; q = a quaternion; p = a position; b = base acceleration; $qd(\cdot)$ = quaternion distance function; $\mathbb{1}_{td}$ = a Boolean variable indicating a touchdown in the current timestep with a different foot from the last touchdown; Δt_{td}= time difference between the current and the last touchdown reward given; † = note due to the feet contact reward being the only sparse reward the weight is significantly bigger than other terms.

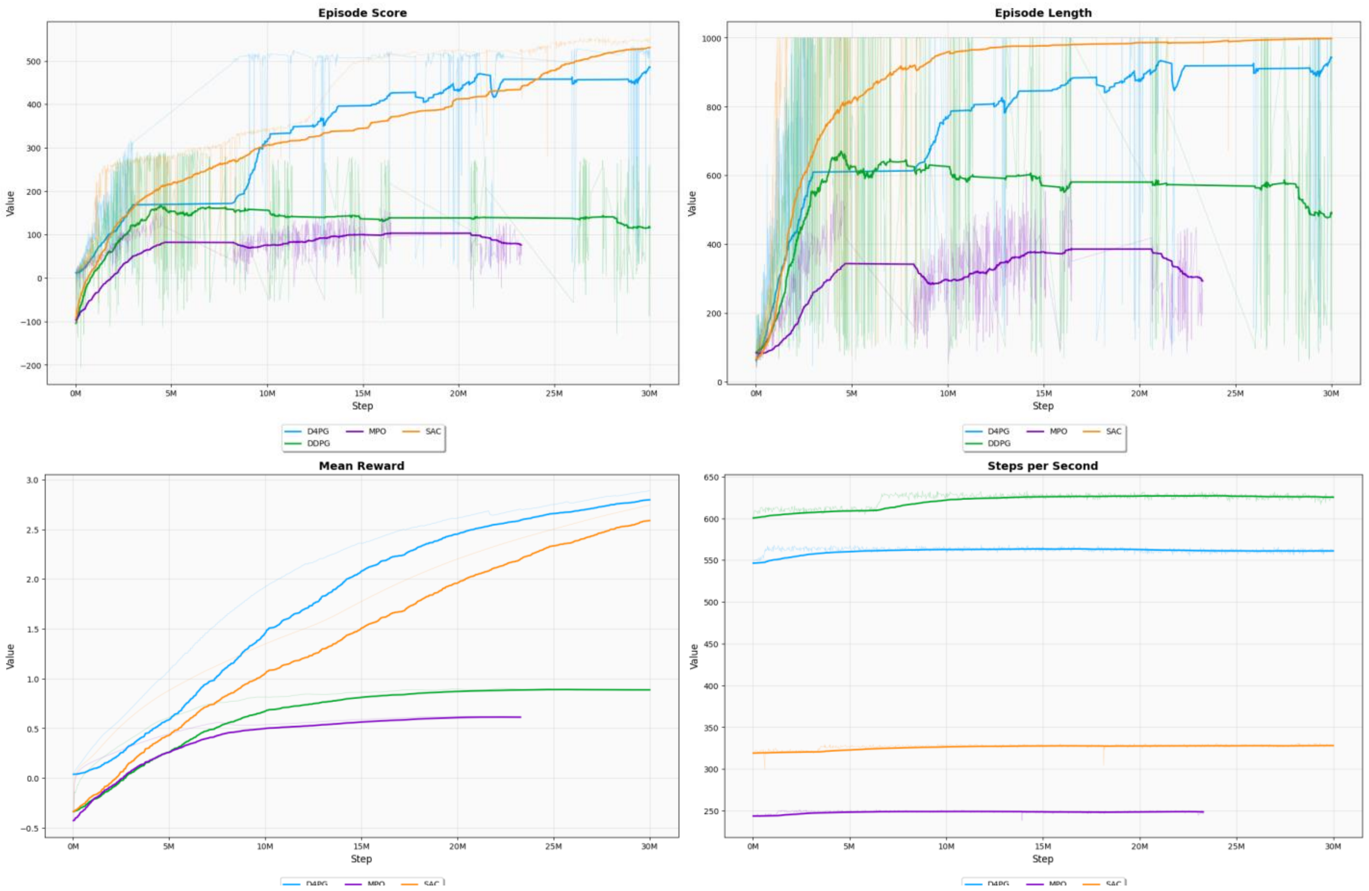

Figure 2: Results for the final algorithms

Graphs showing the results after executing DDPG, D4PG, MPO and SAC. They show, in order, the final scores obtained per episode over the steps, the episode lengths over the steps, the mean reward per step, the steps per second. All graphs have been smoothed to better demonstrate findings since the real output is very noisy.

Each algorithm was trained for 30 million steps in a flat-terrain environment. The main metrics analyzed were final reward, episode length, training speed, and qualitative gait and stability. The results are the following:

- **SAC** achieved the highest reward and most natural gait, with consistent 1000-step episodes.

- **D4PG** offered the best speed-performance tradeoff, converging quickly with stable walking.
- **MPO** showed higher training time and worse performance, but potential in more complex environments.
- **DDPG** was fast and simple but unstable, with poor final gait quality.

The code can be found in the following GitHub page:
https://github.com/AsterisCrack/BipedRobot

5. Conclusions

This work demonstrates that it is possible to train stable bipedal locomotion policies using modern reinforcement learning techniques on hobby-grade hardware. By designing a modular reward function and benchmarking four key RL algorithms, the project offers a reproducible path for future robotics researchers at a low cost.

Future goals include real-world testing, integration of temporal models, and domain randomization for sim-to-real transfer.

6. References

[1] Gong, Yukai, Ross Hartley, Xingye Da, Ayonga Hereid, Omar Harib, Jiunn-Kai Huang, Jessy Grizzle. "Feedback Control of a Cassie Bipedal Robot: Walking, Standing, and Riding a Segway." arXiv:1809.07279, 2018.

[2] Haarnoja, Tuomas, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, Michael Bloesch, Kristian Hartikainen, Arunkumar Byravan, Leonard Hasenclever, Yuval Tassa, Fereshteh Sadeghi, Nathan Batchelor, Federico Casarini, Stefano Saliceti, Charles Game, Neil Sreendra, Kushal Patel, Marlon Gwira, Andrea Huber, Nicole Hurley, Francesco Nori, Raia Hadsell, Nicolas Heess. "Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning." arXiv:2304.13653, 2023.

[3] E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A Physics Engine for Model-Based Control." IEEE Xplore, 2012

[4] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. "Continuous control with deep reinforcement learning." arXiv:1509.02971, 2015.

[5] Barth-Maron, Gabriel, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap. "Distributed Distributional Deterministic Policy Gradients." arXiv:1804.08617, 2018.

[6] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." arXiv:1801.01290, 2018.

[7] Abdolmaleki, Abbas, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, Martin Riedmiller. "Maximum a Posteriori Policy Optimisation." arXiv:1806.06920, 2018.

[8] Marum, Bart van, Aayam Shrestha, Helei Duan, Pranay Dugar, Jeremy Dao, Alan Fern. "Revisiting Reward Design and Evaluation for Robust Humanoid Standing and Walking." arXiv:2404.19173, 2024.

[9] Tassa, Yuval, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, Martin Riedmiller. "DeepMind Control Suite." arXiv:1801.00690, 2018.

[10] Ha, Sehoon, Joonho Lee, Michiel van de Panne, Zhaoming Xie, Wenhao Yu, Majid Khadiv. "Learning based legged locomotion; state of the art and future perspectives." arXiv:2406.01152, 2024.
[11] Swaminathan, Tushar Prasanna, Christopher Silver, Thangarajah Akilan. "Benchmarking Deep Learning Models on NVIDIA Jetson Nano for Real-Time Systems: An Empirical Investigation." arXiv:2406.17749, 2024.
[12] syuntoku14, “fusion2urdf.” GitHub, 2025
[13] Lin, Sixu, Guanren Qiao, Yunxin Tai, Ang Li, Kui Jia, Guiliang Liu. "HWC-Loco: A Hierarchical WholeBody Control Approach to Robust Humanoid Locomotion." arXiv:2503.00923, 2025.

Table of Contents

Figure index

Table index

1. INTRODUCTION

The development of humanoid robots has long captured the imagination of researchers and engineers due to their potential to operate in human-centered environments and perform tasks in ways familiar and intuitive to people. Among the various challenges in humanoid robotics, bipedal locomotion remains one of the most complex, requiring real-time dynamic balance, control of multiple degrees of freedom, and adaptation to uneven terrain. The goal of this project is to tackle these challenges using a low-cost, accessible approach based on reinforcement learning techniques. The proposed bipedal robot is built from affordable, off-the-shelf components and aims to lay the groundwork for open-source development of walking behaviors using modern machine learning methods.

In this work, we take a practical approach to a problem typically reserved for highly specialized research groups with expensive equipment. Our robot uses hobby-grade servo motors, 3D-printed structures, and an affordable microcontroller-based control system. In contrast to high-end platforms such as Cassie [1] or Honda's ASIMO [2], which rely on sophisticated sensor arrays, precision machining, and proprietary software, our project investigates how far one can go in teaching a robot to walk using a combination of simple hardware and intelligent state-of-the-art algorithms. The core of this exploration lies in the application of reinforcement learning, particularly continuous deep reinforcement learning algorithms to learn effective locomotion strategies.

1.1. CONTEXT AND MOTIVATION

Humanoid robots have evolved significantly in recent decades. From passive dynamic walkers to real-time balance controllers powered by deep reinforcement learning, the research community has made remarkable advances in modeling, control, and execution. Robots such as Cassie are now capable of impressive feats, including running, jumping, and navigating unstructured environments. However, the barrier to entry for this type of development remains prohibitively high for individual researchers, hobbyists, and students due to cost, complexity, and lack of open access to proprietary systems.

This project is motivated by the desire to democratize humanoid locomotion research, making it more accessible to those outside large institutions. By working with inexpensive and widely available parts along with simple to understand and modular code, the aim is to bridge the gap between academic research and DIY innovation. The ultimate goal is to contribute to an ecosystem where anyone with basic engineering knowledge can experiment with bipedal robots and reinforcement learning, paving the way for broader educational and research applications.

Moreover, the rapid development of open-source tools such as MuJoCo and PyTorch has made it increasingly feasible to simulate and train walking algorithms efficiently, even on modest hardware. Inspired by DeepMind’s work on MPO for continuous control tasks on hobby grade robots [3], this project explores similar principles in a drastically constrained environment.

1.2. OBJECTIVES

Main Objective

The primary objective of this project is to design and train a reinforcement learning (RL) model capable of controlling the locomotion of a custom-built biped robot. Leveraging recent advances in deep reinforcement learning and physics-based simulation environments, this project aims to lay the groundwork for achieving dynamic and stable bipedal walking using low-cost hardware components and DIY fabrication techniques.

While the design and construction of the real-world robot will be thoroughly explained, the project will focus on evaluating and comparing different state-of-the-art reinforcement learning algorithms in terms of their efficiency, stability, and learning performance when applied to bipedal locomotion. The outcome will help identify the most suitable algorithms for training biped robots in resource-constrained conditions, without relying on high-end sensors or actuators.

Secondary Objectives

Although the initial plan included implementing full sim-to-real transfer, constraints due to the limited resources provided limited the scope of the project. Instead, the following secondary objectives were pursued, both of which are crucial foundational steps toward achieving those long-term goals in the future:

1. **Exploration and Experimentation on the Reward Function:**

   A significant portion of the project was dedicated to researching and designing an effective reward function for learning a natural gait for the robot. Since the reward function is a key component in reinforcement learning, determining how to guide the agent toward stable, efficient and natural walking behavior proved essential. Various formulations were tested; such as velocity tracking, energy efficiency, torso stability, and foot placement accuracy; based on state-of-the-art work, to understand their impact on learning dynamics and final performance.

2. **Research on Next Steps Toward Sim-to-Real Transfer:**

   While a full sim-to-real implementation was not completed, the project includes an exploration of techniques required to enable this transition. In particular, it investigates the use of environment randomization (e.g., varying mass, friction, sensor noise) as a method to improve the robustness of learned policies when moving from simulation to real-world execution. This work helps lay the groundwork for future integration of real hardware by identifying the major gaps and proposing practical steps to bridge them.

1.3. PLANNING AND ECONOMIC VIABILITY

The project was structured to minimize costs and development time while ensuring meaningful experimentation. The following strategies were adopted:

- **Hardware**: The robot is built using 3D-printed PLA components along with aluminum braces, included with most servo motors. It is controlled by an NVIDIA Jetson Nano [4]; a low-cost microcontroller targeted towards AI research. Each leg includes 6 degrees of freedom powered by standard hobby servos, keeping the total mechanical cost under €500.

- **Simulation**: Training is performed in simulation using MuJoCo [5], which allows rapid iteration without risking physical hardware. Simulated training runs on a consumer-grade PC or cloud GPU, significantly reducing the need for expensive robotic test benches.

- **Development Schedule**: **Planning and Schedule:** The project was initially structured into seven main phases distributed across the academic year, from October to June. These phases were: (1) robot design, (2) state-of-the-art review, (3) simulation environment setup, (4) algorithm implementation, (5) training and optimization, (6) sim-to-real transfer, and (7) analysis and report writing.

The early stages progressed according to plan. The robot design, literature review, and simulation setup were successfully completed within the first semester (October–December). However, the subsequent phases experienced schedule deviations.

The algorithm implementation phase began in January as planned and extended into March. However, the training and optimization phase, which started in February, required significantly more time than initially anticipated. The schedule was designed considering the compute power the university promised they would provide; unfortunately, these resources were never provided so training had to be done locally with a laptop. As a result, training and optimization occupied most of the time originally allocated for sim-to-real transfer and final analysis, which had to be deprioritized.

Consequently, although a theoretical exploration of sim-to-real techniques was conducted, no physical deployment was achieved. Similarly, the final report was completed under tighter time constraints, overlapping with late-stage training experiments.

Despite these shifts, the project was successful in achieving its main objective and laying the groundwork for future development, particularly in sim-to-real transfer and real-world testing.

- **Cost Overview**:
    - Servos (12x): ~€240
    - Control board and power: ~€250
    - 3D printing materials: ~€20
    - Miscellaneous components (screws, wires, sensors): ~€20
    - Total: ~€500–550
- **Viability**: This budget-friendly design demonstrates that meaningful work on biped locomotion does not require industrial-level investment, which reinforces the goal of open accessibility.

2. STATE OF THE ART

2.1. HUMANOID PLATFORMS AND TRENDS

Recently a lot of new humanoid/bipedal robots have emerged. For example, Figure 01, Tesla Optimus, Agility Robotics' Cassie and Digit, Unitree H1 or Apptronik Apollo [14]. All these robots include very advanced algorithms and hardware, now featuring high-torque, back drivable electric actuators [15] and onboard vision, enabling agile locomotion and manipulation. These platforms apply the vision of "embodied intelligence" [16], which means integrating advanced AI with physical hardware.

2.2. REINFORCEMENT LEARNING FOR LOCOMOTION

Since the creation of the first deep RL methods, legged locomotion has significantly advanced. The state of the art in reinforcement learning for legged locomotion includes:

- **Model-based RL (MBRL):** Learns environment dynamics to improve sample efficiency [17].
- **On-policy algorithms (PPO, TRPO):** Simple algorithms are still being used in training in parallelized simulation for real-time gait control [15].
- **Hierarchical RL:** For example, HWC-Loco employs a high-level planner to manage switching between velocity-tracking, balance, and safety recovery policies in different scenarios [18].
- **Offline RL & world models:** RWM-O trains entirely from pre-collected datasets, reducing dangerous real-world exploration [17]. Imitation learning, through adversarial motion priors and human motion capture, guides policy training to simplify the sim-to-real gap [16][19].

2.3. SIMULATION AND AI-GENERATED ENVIRONMENTS

Modern simulation engines like NVIDIA's Isaac Gym and DeepMind's MuJoCo are essential for training legged locomotion policies over diversely randomized terrains in a faster and safer manner than real-life simulation [15]. State-of-the-art research extends this by using generative AI to produce synthetic environments. For example, MIT's LucidSim uses LLMs to generate varied 3D terrains and textures, enabling "visual parkour" policy training [20].

2.4. VISION–LANGUAGE–ACTION MODELS

One of the fastest growing and most interesting trends in robotics consists of integrating vision, language, and control via multimodal models. A few examples include:

- **PaLM-E** embeds vision and proprioceptive data into a transformer-based LLM. This enables the model for planning, question answering, and multi-step manipulation tasks [21].
- **Transformers** have been trained to predict next-action tokens using a mixture of datasets, including physics simulation, mocap and YouTube videos. This model achieved zero-shot real-world biped walking after only around 27 hours of data [22].
- **MoRE** fuses vision-language instruction models with RL through adapter modules and Q-learning fine-tuning, achieving generalization to novel commands [23].

2.5. EMBODIED MANIPULATION AND INTELLIGENCE

Locomotion controllers are just the beginning while creating a robot. The final goal for a functional robot is to manipulate its environment as humans do. Simultaneous locomotion and manipulation performance has been improving in recent years. For example, FLAM uses human motion-based captures to shape rewards for joint posture and task success, enhancing co-learning of walking and manipulation [19]. Broadly, the use of pre-trained perception and language models like CLIP, SAM and LLMs with RL is driving robots toward generalist embodied intelligence [19][21]

RL robotics is rapidly advancing, and large research groups are creating practical and generalist models to operate robots. However, these technologies are not easily accessible to small researchers, and it is not easy to learn deep reinforcement learning from the ground. That is why the goal of this project is to simplify the development and research of the first stages towards more complex robotics controllers.

3. METHODOLOGY

This chapter presents the technical foundation of the project. It details the process followed to design the biped robot, digitalize it into a simulation-compatible format, and apply various reinforcement learning algorithms to train it for stable locomotion.

3.1. TECHNICAL DESCRIPTION

3.1.1. ROBOT DESIGN AND DIGITALIZATION

The biped robot used in this project was entirely custom-built using low-cost, readily available components, with the aim of democratizing bipedal robotic research. The mechanical design was carried out using Fusion 360, where custom parts were created to be 3D-printable in a hobby grade FDM 3D printer. Also, the robot's legs were built using the aluminum braces included with the servo motors. Finally, to strengthen the connection between the legs and the torso, a construction with skateboarding bearings was made. The decision to use this type of piece was based on the goal of making the robot unexpensive and easy to build.

The selected servo motors are 12 RDS3225 servo motors. These servos were chosen because they have 25kg of torque, which is enough for a small robot, while also being cheap and reliable. However, these servos do not include any type of encoder, which might be necessary in the future. The selected microcontroller was the NVIDIA Jetson Nano. This microcontroller was designed for AI applications and includes a GPU designed for AI models, making it the perfect choice for this project. As for the sensors, an IMU (inertial measurement unit) is present in the center of the torso to gather the inertial data needed as input for the model.

To simulate the robot, a URDF (Unified Robot Description Format) model was automatically generated using a Fusion 360 extension [6]. This allowed for easy conversion into a MuJoCo compatible XML file. This step was crucial, as it avoided the need for manual calculation of link inertias and mass properties—data which Fusion 360 could export directly based on material and geometry. The final design and adaptation in these platforms can be seen in Figure 3: Robot design.

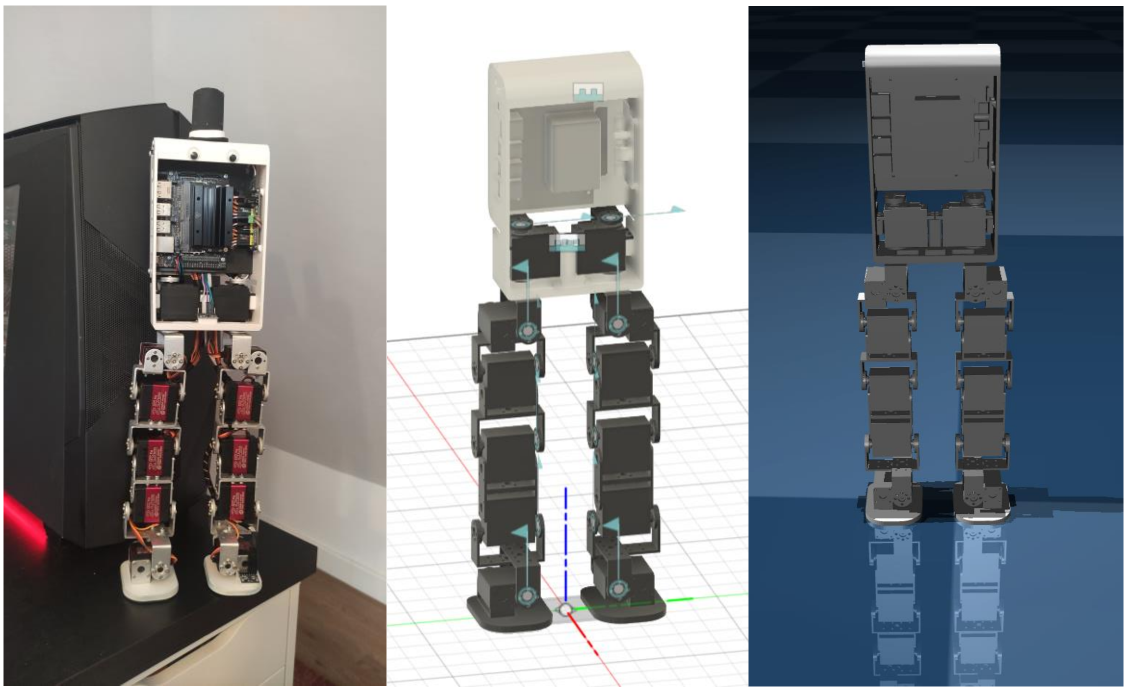

Figure 3: Robot design

The final hardware design of the robot in reality, Fusion 360 and MuJoCo respectively.

3.1.2. Simulation with MuJoCo

The simulation was performed using MuJoCo [3] (Multi-Joint dynamics with Contact), a physics engine designed for accurate and efficient simulation of robots and articulated structures. MuJoCo offers high-speed simulation and precise contact modeling, making it suitable for training reinforcement learning policies.

The simulation process was organized into three progressively complex environments:

1. **Flat terrain, constant velocity**: The initial environment had a flat plane and a fixed forward velocity goal. This was used to test the algorithms and design the reward function.
2. **Randomized environment**: To simulate sim-to-real transfer, environmental parameters (e.g., friction, mass perturbation) were randomized during training. This improves robustness and adaptability.
3. **Random velocity and direction**: In the final environment, the agent received a random target velocity vector (speed and direction) and had to learn how to reach it dynamically.

All these environments shared the same terminal states. First, if the robot center of mass reaches a height lower than 20 cm it is considered that it fell, and the episode is terminated unsuccessfully. However, if it reaches a certain number of steps, in this case 1000, it is considered that the robot's gait is stable and it is terminated successfully.

Moreover, a functionality was added to execute MuJoCo simulations in sequence and in parallel. Having various simulations at the same time allows for efficient usage of compute power and enables faster training. In the end, 8 worker groups with 16 workers per group were selected for training.

3.1.3. Observation Space

The observation vector was sampled at 100 Hz from the torso (where the real robot's IMU is located) and included information necessary for balance and locomotion. The observation format was:

Index	Feature	Description
0-2	x, y, z	Position of the torso
3-6	qx, qy, qz, qw	Orientation (quaternion)
7-n	joint_positions	Current joint angles
n+1 to n+3	vx, vy, vz	Linear velocity of the torso
n+4 to n+6	qvx, qvy, qvz	Angular velocity of the torso
...	joint_velocities	Angular velocity of all joints

The absence of servo encoders in the real robot posed a future challenge: either the system must be upgraded to include them, or the observation space must be adapted to remove joint state dependency.

Due to the complexity of the problem, environments 2 and 3 require past data, so in this environments the observation space will include a number of past observations-actions pairs along with the present observations.

3.2. MODEL DESIGN

To enable comparisons across different reinforcement learning algorithms, a consistent architecture was adopted for the actor and critic networks.

3.2.1. MLP Architecture

For the base environments (constant forward velocity), a Multilayer Perceptron (MLP) with two hidden layers of size (256, 256) and ReLU activations were used for both the actor and the critic.

3.2.2. Temporal Models: LSTM and 1D-CNN

For more complex environments that require memory, two temporal models were implemented and should be explored in the future:

- **LSTM (Long Short-Term Memory)** layers were added before the MLP to handle time sequences of past observations.
- A **1D Convolutional Neural Network (1D-CNN)** was also implemented to capture patterns in the recent history of observations, offering faster training with slightly less flexibility than LSTM.

This design allows the same input and output formats across models and facilitates benchmarking.

3.3. Training and Validation

3.3.1. Algorithms Explored

Four actor-critic reinforcement learning algorithms were selected for evaluation, all suited for continuous control tasks:

- **DDPG (Deep Deterministic Policy Gradient):**
  DDPG [13] is an off-policy, actor-critic algorithm that learns deterministic policies for continuous control by combining Q-learning with a policy gradient. It trains quickly but newer algorithms made it obsolete.
- **D4PG (Distributed Distributional DDPG):**
  D4PG [24] builds on DDPG by using a distributional critic, n-step returns, and distributed experience collection, improving training stability and performance. It offers a strong balance between speed and reward quality, making it ideal for rapid experimentation.
- **MPO (Maximum a Posteriori Policy Optimization):**
  MPO [7] frames reinforcement learning as probabilistic inference, alternating between policy improvement and supervised policy fitting steps. Although more robust and theoretically grounded, it is computationally intensive and slower to converge in practice. Also, tuning the bigger amount of hyperparameters can be difficult.
- **SAC (Soft Actor-Critic):**
  SAC [8] is a stochastic, entropy-regularized actor-critic algorithm that encourages exploration by maximizing both reward and policy entropy. It achieves state-of-the-art stability and performance but requires more time to train compared to other methods.

Each algorithm used is suited for continuous action spaces, all of them are off-policy algorithms, with actor-critic architectures and distinct optimization strategies. All models were trained with the same MLP architecture: (256, 256) for both actor and critic, to allow for a fair and consistent comparison. They will be described in greater detail in ANNEX I.

3.3.2. Reward Function Design

Reward design was one of the most critical components for achieving stable and natural-looking walking behavior. Using D4PG for its stability and performance, the reward function was iteratively designed and tested.

3.3.3. Evaluation Metrics

The evaluation of each algorithm was based on:

- **Reward curve:** The reward over time is arguably the most important metric to evaluate the different algorithms. With it we evaluate not only the final reward reached but also the speed at which each algorithm learns.

- **Episode length curve:** In the case of this project, the reward is not the only important metric to be maximized. Since we want the gait to be stable, a good metric to use is the episode length. Longer episodes consistently mean the robot does not fall as often, hence, a more stable gait.

  The combination of these two metrics is ideal to test the algorithms for a biped robot. The reward curve is the most important, but in some cases the design of the reward function is not tuned correctly, and it can allow for reward hacking, as will be explained later. Also, in some cases it is preferred to prioritize stability rather than perfecting components for the reward. For these reasons, this project will be looking at both curves to compare the algorithms.

- **Steps per second and training time:** To evaluate the algorithms' speed, it is essential to compare the algorithm speed. Each algorithm's training speed differs greatly. For that, the training time and the steps per second will be compared.
- **Qualitative walking behavior (naturalness, symmetry):** Finally, as a subjective metric, the naturalness, symmetry and overall quality of the gaits will be compared. This is a more visual and subjective metric, but it is interesting to look at the differences in results.

3.3.4. Codebase

All code was written in Python, following a modular and extensible architecture. This allowed for easy swapping of algorithms, networks, environments, and reward functions. The goal is to release the platform as open source, so that researchers and enthusiasts can contribute and develop new ideas without requiring expensive hardware. The algorithms and distributed environments were modified from Tonic RL Library [25].

The code can be found in the following GitHub page: https://github.com/AsterisCrack/BipedRobot

4. EXPERIMENTS

This section describes in detail the experimentation process, including the objectives, resources used, configuration, algorithm evaluation, reward function tuning, and performance assessment. All experiments were carried out exclusively in the first environment (flat plane, forward locomotion) due to hardware limitations. The goal was to establish a strong foundation and identify the most effective reinforcement learning algorithm and reward function for a stable and natural biped gait.

4.1. OBJECTIVES

The main goals of the experimentation phase were:

- To compare the performance of four state-of-the-art continuous control reinforcement learning algorithms: Maximum a Posteriori Policy Optimization (MPO), Soft Actor-Critic (SAC), Deep Deterministic Policy Gradient (DDPG), and Distributed Distributional DDPG (D4PG).
- To iteratively design, test, and refine the reward function in order to obtain a stable, natural and efficient walking pattern.
- To assess training speed, convergence rate, and the final reward performance of each algorithm in a controlled simulated environment.

Due to hardware constraints, experiments were limited to a flat-terrain environment with fixed target velocities, and only simple MLP-based architectures were trained.

4.2. DATA

No external datasets were used. All training data was generated online through interaction with the MuJoCo simulation environment. The robot receives observations at 100 Hz as described in 3.1, including position, orientation, velocity, and joint states.

The environment provides a reward signal and a next observation after each action, completing the loop for reinforcement learning.

4.3. CONFIGURATION

Hardware:

- **Laptop model:** MSI Prestige 15
- **CPU:** Intel Core i7-10710U
- **RAM:** 16GB
- **GPU:** NVIDIA GeForce GTX 1650 Max-Q

Software stack:

- **Simulation:** MuJoCo physics engine
- **Programming language:** Python 3.11
- **Frameworks:** PyTorch, Gym

4.4. REWARD FUNCTION EXPERIMENTATION

A central component of this work involved the careful design and iterative refinement of the reward function, which guides the learning process of the reinforcement learning model. The reward function determines what behaviors are encouraged and which are penalized, making it a critical factor in shaping the quality, stability, and naturalness of the biped robot's locomotion.

In early experiments, it was observed that poorly designed rewards often led to unnatural or unstable gaits, or to the agent exploiting loopholes in the reward signal (e.g., shaking limbs rapidly to produce reward spikes without meaningful movement, standing still to avoid the penalization of failing). To mitigate this, experimentation was made following a modular and incremental reward design strategy, beginning with minimal terms and gradually incorporating additional components as needed to address specific behavioral deficiencies.

4.4.1. Design Philosophy and Use of Exponential Terms

All the reward components were formulated using exponential functions of the form $e^{-k * error^2}$ or $e^{-k * |error|}$. This structure provides several desirable properties:

- **Boundedness**: Rewards are naturally scaled between 0 and 1, preventing gradient explosion.
- **Smoothness**: The exponential curve penalizes large errors more harshly while allowing small errors to contribute meaningfully.
- **Gradient-based encouragement**: Gradual improvements in behavior lead to measurable reward improvements, crucial for stable convergence.

This approach was inspired by prior work on locomotion reward shaping, particularly in simulated robots, such as in the DeepMind Control Suite [11].

4.4.2. Iterative Term Introduction and Effects

The D4PG algorithm was used to test and refine the reward, based on its stability and speed during learning. Below is an overview of how terms were progressively introduced and what they achieved.

1. ***Velocity tracking***

Ensures the robot walks at a commanded velocity. For testing purposes, set at 0.5 m/s in x direction, but this velocity command is randomly chosen in more complex environments. This is the most basic reward element. It encourages forward locomotion in the desired direction. Without this term, the agent learned to stand still.

When adding more complex terms, their penalization might surpass the reward of moving forward. Instead of increasing the reward for this term, which might cause gradient explosion, a conditional term was added to give a big negative reward if the velocity is less than a minimum.

2. ***Torque and Control Smoothness***

To encourage energy efficient and smooth actions, two terms were added to minimize torque and minimize action difference between steps. These terms Reduced jerky limb movements, leading to more natural gaits. Adding these terms helped prevent the robot from using extreme joint torques that could cause instability or mechanical failure. It will also increase the life expectancy of the motors.

3. ***Posture and Orientation Stability***

The robot now moved forward, but in an asymmetric and unstable manner. To solve this, torso stability is encouraged by adding a term to maintain a straight upright posture, with minimal roll or pitch deviations. These terms significantly reduced falling episodes due to instability, hence speeding up the training. Pitch and roll stabilization helped develop a more balanced and natural gait.

4. ***Height tracking***

Most reward functions for biped robots [10] include a term for commanding a certain height, normally the base height of the standing robot. However, after experimentation, this term was not necessary in the end since the robot learned to maintain a certain stable height on its own.

5. *Feet Orientation*

The robot, however, tended to walk with the side of the feet or in "tippy toes". Ideally, the robot should use the full bottom of the feet as contact to improve stability. To solve this problem, a similar term to the torso orientation one was added for each foot.

6. *Torso Centering*

After the last element of the reward function, more natural gaits were emerging. However, the robot tended to lean too forward. According to the double pendulum walking model [9], a stable walk should always have the center of mass between the two feet in the sequence. To solve this issue, a new term was created to encourage the torso to be placed between the two feet.

7. *Feet contact Alternation*

Finally, after a very stable walking gait emerged. However, it was visually off-putting due to the asymmetry of the gait. In biological walkers, gaits usually involve alternating the legs that are forward and back at every step, creating a symmetric and visually pleasing gait. The emerging gait, moreover, presented an asymmetric gait where one leg was always placed behind the other, using very quick and short steps.

To solve this, different approaches were explored. The final approach involves a sparse reward when one-foot touches down in front of the other. This reward needs to depend on the time difference between the current and the previous feet contact reward given. This is because, with a constant reward, the training showed a form of reward hacking where it would do a lot of fast small steps to maximize the reward.

4.5. Performance Analysis

After experimenting on the reward function, finally, experiments were done on the different algorithms. This point will analyze the performance of all 4 algorithms.

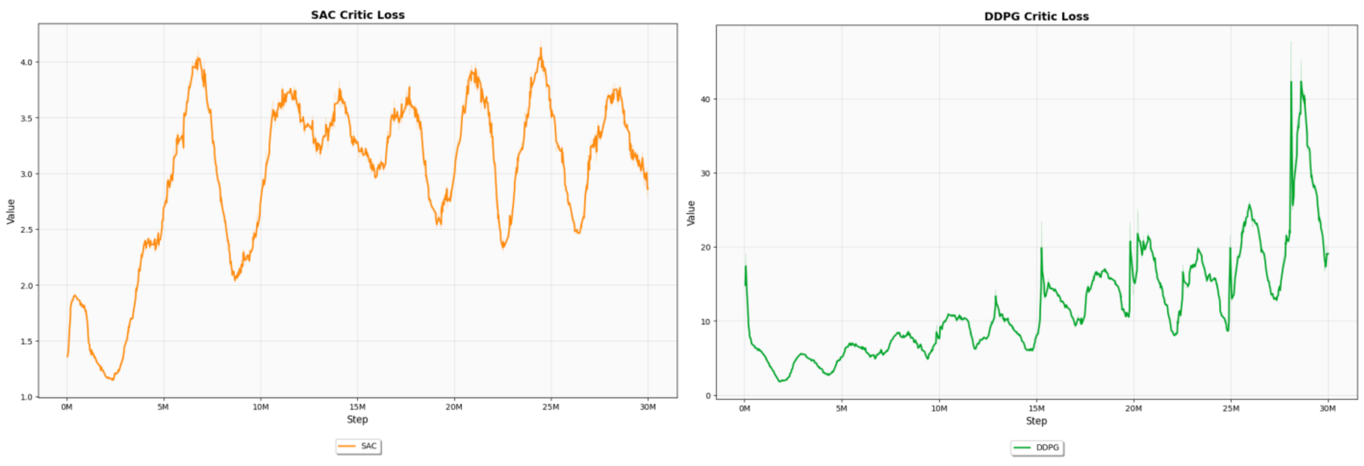

Figure 4: Critic loss curves of SAC and DDPG without exploration decay

These algorithms without adding exploration decay show unstable training.

In the first times executing the different algorithms, a recurrent issue emerged. Training was very unstable in DDPG, D4PG and SAC, as seen in Figure 4. This is because MPO is the only algorithm that tunes exploration to stabilize learning. In the other algorithms, exploration is a hyperparameter that needs to be tuned.

In the end, the solution to stabilize learning was a combination of two techniques. First, decreasing all reward terms to keep the reward low, ideally less than 1. This helps the gradients not to explode while learning. Second, the exploration was kept high in the beginning and was exponentially decreased during training. This helps collect information about the environment first and then tune the policy in later steps.

4.5.1. DDPG

DDPG is the oldest and simplest algorithm of the ones used, and its performance reflected it. Thanks to its simplicity, it was the fastest algorithm with an average of 627 steps per second. It was also the fastest to converge, as the mean reward stopped improving at roughly 20 million steps.

The reward curve is steep, meaning training is fast. However, it stopped growing quickly and it didn't achieve a high final reward. The gait is not very stable, as most episodes don't finish successfully, with an average episode length of 600 out of the 1000 steps. Visually, the gait is very unstable, which leads to the robot trying to continuously balance itself instead of moving forward at a constant speed.

4.5.2. D4PG

D4PG builds upon DDPG making some important changes. These changes result in a more robust algorithm that keeps the good aspects of DDPG and fixes its issues. Same as DDPG, D4PG is a fast algorithm, with an average of 563 steps per second. However, it converges later than DDPG, making the overall training of this algorithm longer. Moreover, not only is D4PG fast, but also its performance is very good. The final reward curve is the steepest and, opposite to DDPG, the reward keeps increasing, achieving the second highest final reward, only below SAC. Also, the gait is very stable and symmetrical. The episode length curve is also very steep, matching the reward, and in the final training steps it reaches the 1000 steps consistently.

4.5.3. MPO

MPO is a very advanced algorithm and is capable of learning very complex environments and policies as proven by DeepMind [3]. However, this added complexity makes the algorithm very slow. With the same configuration as the others, MPO trained at only 248 steps per second on average. It also takes a lot of steps to converge, comparable to D4PG and SAC.

The results were also surprisingly poor. Although being better than the simplest DDPG, the episode length and reward curves were less steep than the other algorithms, meaning its sample efficiency is the worst amongst all algorithms. The episode length reaches 1000 in some iterations, but the reward is not as high, and the final gait isn't as stable as D4PG or SAC. MPO has potential for being a very good algorithm with environments with more complexity which requires a better balance on exploration.

Another issue faced with MPO is the complexity of tuning its hyperparameters. It is the algorithm with most hyperparameters amongst the four tested, and it is very sensitive for all. It requires a delicate tuning of all of them, which is a hard and lengthy process. Probably, we have not found the optimal hyperparameters and that is one of the reasons why the performance is so poor.

4.5.4. SAC

SAC is usually the baseline algorithm used for benchmarking, even in the current SOTA. The reasoning behind it is very clear by seeing its performance. SAC's speed falls short compared to D4PG, with an average of 327 steps per second. Also, the reward curve is more linear, and it looks like it will take a very large number of steps to fully converge, making the overall training of this algorithm quite long. However, this is not an issue since its performance is so good. The final reward curve increases almost at a constant rate, achieving the highest final reward, possibly surpassing D$PG by a lot if enough training time was allowed.

Also, the gait is the most stable and symmetrical. The episode length curve doesn't match the reward curve so closely since it is the steepest amongst all the algorithms and at about 10 million steps it reaches the 1000 steps almost always. This means that the learning process followed by SAC first created a stable gait that does not fall and then consistently modified this gait to maximize reward gain.

5. RESULTS

After the previously mentioned experimentation on the rewards, the final reward function consists of the following terms:

Reward Term	Definition	Weighting
Survived a step	1	0.001
Velocity	$e^{-5\cdot\lvert v_{xy}-c_{xy}\rvert^2}$	0.15
Torque	$e^{-0.02\cdot\frac{1}{N}\sum\frac{\lvert t_{motor}\rvert}{t_{max}}}$	0.01
Action Difference	$e^{-0.02\cdot\sum\lvert a_t-a_{t-1}\rvert}$	0.01
Base Acceleration	$e^{-0.01\cdot\sum\lvert b_{x\,y\,z}\rvert}$	0.05
Yaw orientation	$e^{-30\cdot qd(q_{yaw},c_{yaw})}$	0.02
Pitch and Roll orientation	$e^{-30\cdot qd(q_{pitch,roll},c_{pitch,roll})}$	0.04
Feet orientation	$e^{-30\cdot\sum qd(q_{feet,ypr},c_{feet,ypr})}$	0.3
Torso centering	$e^{-20\cdot\lvert p_{xy}-\frac{p_{r\_feet,xy}+p_{l\_feet,xy}}{2}\rvert^2}$	0.1
Feet contact position	$\begin{cases}\Delta t_{td}\ if\ \mathbb{1}_{td,f}\ and\ p_{f,x} > p_{f',x}\ for\ any\ f\ \in\ [left,right] \\ 0\ otherwise\end{cases}$	$1.5^{\dagger}$

Table 3. Elements of the Final Reward Function

c = a command; q = a quaternion; p = a position; b = base acceleration; $qd(\cdot)$ = quaternion distance function; $\mathbb{1}_{td}$ = a Boolean variable indicating a touchdown in the current timestep with a different foot from the last touchdown; Δt_{td} = time difference between the current and the last touchdown reward given; † = note due to the feet contact reward being the only sparse reward the weight is significantly bigger than other terms.

With this reward function, all algorithms were executed, achieving the results shown in Figure 5: Results for the final algorithms. The results show that the best algorithm overall is SAC, achieving not only the highest score and episode length, but a very stable training with a constant growth in score. This growth does not stop in the 30 million steps allowed for training, which indicates potential for even better performance.

However, it is somewhat slow. For that reason, D4PG is the best algorithm for testing and experimentation. The final performance of the model is not as good as D4PG, but with almost double the training speed and a steepest reward curve at first, D4PG is the best algorithm for quick experimentation, making the process much faster and, in consequence, cheaper.

For all these reasons, the recommended workflow for designing a biped robot with RL consists of using D4PG to tune the reward function and, at last, use SAC to train the final model.

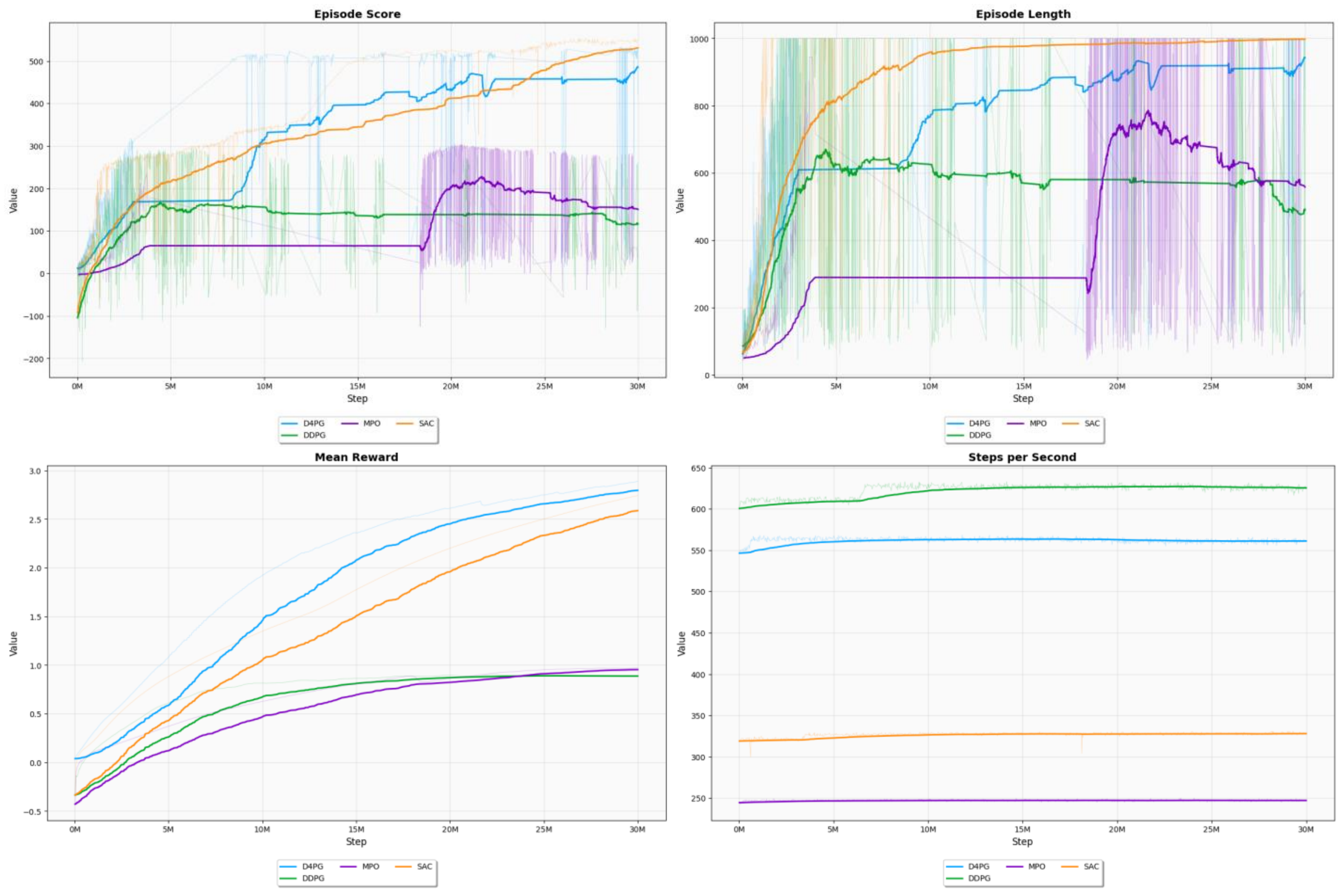

Figure 5: Results for the final algorithms

Graphs showing the results after executing DDPG, D4PG, MPO and SAC. They show, in order, the final scores obtained per episode over the steps, the episode lengths over the steps, the mean reward per step, the steps per second. All graphs have been smoothed to better demonstrate findings since the real output is very noisy.

6. CONCLUSION AND FUTURE WORK

This project has demonstrated the feasibility of applying deep reinforcement learning to control a custom-built, low-cost biped robot in simulation. Through a combination of mechanical design, simulation with MuJoCo, and deep RL algorithms, we achieved stable and natural locomotion using only affordable hardware and open-source tools. The evaluation of four state-of-the-art RL algorithms—DDPG, D4PG, SAC, and MPO—revealed that SAC provided the best overall performance in terms of stability and final reward, while D4PG offered the fastest and most efficient experimentation cycle.

A major contribution of this work lies in the modular design and iterative refinement of the reward function, which was key to shaping effective and natural gaits. By progressively incorporating terms such as velocity tracking, torque penalization, orientation tracking, and foot contact alternation, the final reward function enabled learning gaits that were not only functional but also symmetric and visually natural. The results validate the hypothesis that it is possible to achieve viable locomotion outcomes by focusing on reward engineering and algorithmic selection.

Despite the hardware limitations that restricted training to simulation, the project took preliminary steps toward the secondary objective, sim-to-real transfer, by developing environments with environment randomization techniques. These methods are essential for building robustness in learned policies and will serve as a foundation for future real-world deployment. Additionally, while temporal models like LSTM and 1D CNNs were implemented, their testing alongside environment randomization remains an open task due to computational constraints.

In future work, we aim to extend the experiments to more complex environments and integrate the trained policies onto the physical robot platform. Further evaluation of sim-to-real techniques and temporal models will be critical for this transition. Ultimately, this project establishes a reproducible and extensible platform that can serve both the research community and hobbyist developers in advancing the field of low-cost, intelligent humanoid robotics.

7. BIBLIOGRAPHY

[1] Gong, Yukai, Ross Hartley, Xingye Da, Ayonga Hereid, Omar Harib, Jiunn-Kai Huang, Jessy Grizzle. "Feedback Control of a Cassie Bipedal Robot: Walking, Standing, and Riding a Segway." arXiv:1809.07279, 2018.

[2] Sakagami, Yoshiaki & Watanabe, Ryujin & Aoyama, Chiaki & Matsunaga, Shinichi & Higaki, Nobuo & FujiMura, Kikuo. "The intelligent ASIMO: System overview and integration." IEEE International Conference on Intelligent Robots and Systems. 3. 2478 - 2483 vol.3. 10.1109/IRDS.2002.1041641, 2002.

[3] Haarnoja, Tuomas, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, Michael Bloesch, Kristian Hartikainen, Arunkumar Byravan, Leonard Hasenclever, Yuval Tassa, Fereshteh Sadeghi, Nathan Batchelor, Federico Casarini, Stefano Saliceti, Charles Game, Neil Sreendra, Kushal Patel, Marlon Gwira, Andrea Huber, Nicole Hurley, Francesco Nori, Raia Hadsell, Nicolas Heess. "Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning." arXiv:2304.13653, 2023.

[4] Swaminathan, Tushar Prasanna, Christopher Silver, Thangarajah Akilan. "Benchmarking Deep Learning Models on NVIDIA Jetson Nano for Real-Time Systems: An Empirical Investigation." arXiv:2406.17749, 2024.

[5] E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A Physics Engine for Model-Based Control." IEEE Xplore, 2012

[6] syuntoku14, "fusion2urdf." GitHub, 2025

[7] Abdolmaleki, Abbas, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, Martin Riedmiller. "Maximum a Posteriori Policy Optimisation." arXiv:1806.06920, 2018.

[8] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." arXiv:1801.01290, 2018.

[9] M. Garcia, A. Chatterjee, A. Ruina, and M. Coleman, "The simplest walking model: stability, complexity, and scaling," Journal of Biomechanical Engineering, vol. 120, 1998

[10] Marum, Bart van, Aayam Shrestha, Helei Duan, Pranay Dugar, Jeremy Dao, Alan Fern. "Revisiting Reward Design and Evaluation for Robust Humanoid Standing and Walking." arXiv:2404.19173, 2024.

[11] Tassa, Yuval, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, Martin Riedmiller. "DeepMind Control Suite." arXiv:1801.00690, 2018.

[12] Fujimoto, Scott, Herke van Hoof, David Meger. "Addressing Function Approximation Error in Actor-Critic Methods." arXiv:1802.09477, 2018.

[13] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. "Continuous control with deep reinforcement learning." arXiv:1509.02971, 2015.

[14] Zhang, Qiang, Peter Cui, David Yan, Jingkai Sun, Yiqun Duan, Gang Han, Wen Zhao, Weining Zhang, Yijie Guo, Arthur Zhang, Renjing Xu. "Whole-body Humanoid Robot Locomotion with Human Reference." arXiv:2402.18294, 2024.

[15] Ha, Sehoon, Joonho Lee, Michiel van de Panne, Zhaoming Xie, Wenhao Yu, Majid Khadiv. "Learning-based legged locomotion; state of the art and future perspectives." arXiv:2406.01152, 2024.

[16] Zhang, Qiang, Peter Cui, David Yan, Jingkai Sun, Yiqun Duan, Gang Han, Wen Zhao, Weining Zhang, Yijie Guo, Arthur Zhang, Renjing Xu. "Whole-body Humanoid Robot Locomotion with Human Reference." arXiv:2402.18294, 2024.

[17] Li, Chenhao, Andreas Krause, Marco Hutter. "Offline Robotic World Model: Learning Robotic Policies without a Physics Simulator." arXiv:2504.16680, 2025.

[18] Lin, Sixu, Guanren Qiao, Yunxin Tai, Ang Li, Kui Jia, Guiliang Liu. "HWC-Loco: A Hierarchical Whole-Body Control Approach to Robust Humanoid Locomotion." arXiv:2503.00923, 2025.

[19] Zhang, Xianqi, Hongliang Wei, Wenrui Wang, Xingtao Wang, Xiaopeng Fan, Debin Zhao. "FLAM: Foundation Model-Based Body Stabilization for Humanoid Locomotion and Manipulation." arXiv:2503.22249, 2025.

[20] Yu, Alan, Ge Yang, Ran Choi, Yajvan Ravan, John Leonard, Phillip Isola. "Learning Visual Parkour from Generated Images." arXiv:2411.00083, 2024.

[21] Driess, Danny, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, Pete Florence. "PaLM-E: An Embodied Multimodal Language Model." arXiv:2303.03378, 2023.

[22] Radosavovic, Ilija, Bike Zhang, Baifeng Shi, Jathushan Rajasegaran, Sarthak Kamat, Trevor Darrell, Koushil Sreenath, Jitendra Malik. "Humanoid Locomotion as Next Token Prediction." arXiv:2402.19469, 2024.

[23] Zhao, Han, Wenxuan Song, Donglin Wang, Xinyang Tong, Pengxiang Ding, Xuelian Cheng, Zongyuan Ge. "MoRE: Unlocking Scalability in Reinforcement Learning for Quadruped Vision-Language-Action Models." arXiv:2503.08007, 2025.

[24] Barth-Maron, Gabriel, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap. "Distributed Distributional Deterministic Policy Gradients." arXiv:1804.08617, 2018.

[25] Pardo, Fabio. "Tonic: A Deep Reinforcement Learning Library for Fast Prototyping and Benchmarking." arXiv:2011.07537, 2020.

ANNEX I

8.1. ALGORITHMS

In this annex, each algorithm will be explained in more detail and a pseudocode for each one will be provided.

8.1.1. DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

DDPG [13] is a model-free, off-policy, actor-critic algorithm tailored for continuous action spaces. It extends the Deep Q-Learning ideas to deterministic policies using the Deterministic Policy Gradient Theorem. DDPG serves as the foundation for more advanced algorithms like TD3 and D4PG.

Key innovations of DDPG:

- Uses a deterministic actor $\mu(s|\theta^{\mu})$ to output continuous actions directly.
- Trains a critic to approximate the Q-function $Q(s,a|\theta^{Q})$ using bootstrapped targets.
- Employs target networks and a replay buffer for stable training.

Theoretical Formulation

Policy and Critic Objectives

The critic minimizes the Bellman loss**:**

$$\mathcal{L}_Q(\theta^Q) = E_{(s,a,r,s')}[(Q(s,a|\theta^Q) - y)^2]$$

Where the target is:

$$y = r + \gamma Q'(s', \mu'(s')\big|\theta^{Q'})$$

- Q' and μ' are target networks, updated slowly to stabilize learning.

The actor is trained to maximize the Q-value using the deterministic policy gradient**:**

$$\nabla_{\theta^\mu} J \approx E_{s\sim\mathbb{D}}\left[\nabla_a Q(s,a|\theta^Q)|_{a=\mu(s)} \cdot \nabla_{\theta^\mu}\mu(s|\theta^\mu)\right]$$

Target Networks

- Target networks Q', μ' are soft-updated after every step:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

Where $\boldsymbol{\tau} \ll 1$, e.g. $\tau = 0.005$

Exploration

Since the policy is deterministic, exploration is added via noise during training:

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$$

Pseudocode

```
Initialize actor μ(s|θμ) and critic Q(s,a|θQ)
Initialize target networks: θQ′ ← θQ, θμ′ ← θμ
Initialize replay buffer R

for each training step:
    Select action: a = μ(s|θμ) + noise
    Execute a in environment, observe (r, s′)
    Store (s, a, r, s′) in R

    Sample mini-batch from R
    Compute targets: y = r + γ Q′(s′, μ′(s′))
    Update critic by minimizing:
        L = (Q(s, a) - y)^2

    Update actor using:
        ∇θμ J ≈ ∇a Q(s, a)|a=μ(s) ∇θμ μ(s)

    Update target networks:
        θQ′ ← τ θQ + (1 − τ) θQ′
        θμ′ ← τ θμ + (1 − τ) θμ′
```

8.1.2. DISTRIBUTED DISTRIBUTIONAL DDPG (D4PG)

D4PG [24] is an extension of the DDPG algorithm**,** integrating three powerful improvements:

1. Distributional Critics: Instead of predicting a scalar value $Q(s,a)$, it learns a full distribution over possible returns.
2. N-step Returns: Bootstrapping over multiple steps increases learning stability.
3. Distributed Experience Collection: Parallel actors collect experiences and feed a centralized replay buffer.

These changes make D4PG significantly more stable and performant in complex tasks such as locomotion and manipulation, improving over DDPG and even SAC in many benchmarks.

Theoretical Formulation

D4PG modifies DDPG in three major ways:

Distributional Critic

Instead of approximating $\boldsymbol{Q(s,a) \approx E[Z(s,a)]}$**,** D4PG models the full distribution $\boldsymbol{Z(s,a)}$**.**

1. Distributional Bellman Operator

$$\mathcal{T}^{\pi} Z(s,a) = r(s,a) + \gamma Z\big(s', \pi(s')\big)$$

Where Z is a random variable representing the return.

2. Distributional Loss

$$\mathcal{L}_{\text{critic}} = D_{\text{KL}}(\Phi \mathcal{T}^{\pi} Z' \parallel Z)$$

Where:

- $\Phi \mathcal{T}^{\pi} Z'$: projected target distribution
- Z: current prediction
- D_{KL} : KL divergence

The projection operator **Φ** ensures the target lies on the same support as the predicted distribution.

N-Step Returns

Rather than 1-step TD updates, D4PG uses N-step bootstrapping**:**

$$y = \sum_{n=0}^{N-1} \gamma^n r_{t+n} + \gamma^N Z\big(s_{t+N}, \pi(s_{t+N})\big)$$

This improves stability and learning speed, especially in sparse reward tasks.

Deterministic Actor Update

Just like DDPG, the actor is deterministic and updated via the deterministic policy gradient**:**

$$\nabla_\theta J(\theta) = E_{s\sim\mathcal{D}}\big[\nabla_\theta \pi_\theta(s) \cdot \nabla_a Q(s,a)|_{a=\pi_\theta(s)}\big]$$

In D4PG, the actor uses the mean of the learned return distribution:

$$Q(s,a) = E[Z(s,a)]$$

And uses this in the standard actor gradient.

Pseudocode

```
Initialize actor π_θ and distributional critic Z_ϕ
Initialize target networks θ', ϕ'
Initialize distributed replay buffer
```

```
for each training step:
    # Collect experience in parallel
    For each actor:
        a_t = π_θ(s_t) + noise
        Execute a_t, observe r_t, s_{t+1}
        Store (s_t, a_t, r_t, s_{t+1}) in shared replay

    # Sample N-step transitions
    (s, a, r_1...r_N, s') ~ Replay

    # Critic update:
    Compute target distribution:
        y = r_1 + γ r_2 + ... + γ^N Z'(s', π(s'))
    Project y onto support atoms
    Compute KL loss to critic output Z(s,a)
    Backpropagate and update ϕ

    # Actor update:
    Use ∇_θ Q(s, π(s)) = ∇_θ E[Z(s, π(s))]
    Update θ via deterministic policy gradient

    # Target updates:
    θ' ← τ θ + (1−τ) θ'
    ϕ' ← τ ϕ + (1−τ) ϕ'
```

8.1.3. Maximum a Posteriori Policy Optimization (MPO)

MPO [7] is an off-policy actor-critic reinforcement learning algorithm that formulates the policy optimization process as a coordinate ascent in a KL-regularized EM-like setting. Unlike many actor-critic methods, it separates exploration and optimization through non-parametric policy improvement (E-step) and parametric supervised learning (M-step).

Theoretical Formulation

The optimization objective in MPO is derived from casting RL as a probabilistic inference problem. Given an auxiliary distribution $q(a \mid s)$ over actions, the regularized objective is:

$$\max_{q,\theta} \mathrm{E}_{q(s,a)}[r(s,a) - \alpha \cdot \mathrm{KL}(q(a \mid s) \parallel \pi_\theta(a \mid s))] + \log p(\theta)$$

Where:

- $\boldsymbol{\pi(a \mid s; \theta)}$: the current policy parameterized by θ
- $\boldsymbol{q(a \mid s)}$: non-parametric target distribution
- $\boldsymbol{\alpha}$: temperature (controls soft-max weighting)
- $\boldsymbol{p(\theta)}$: prior over policy parameters (usually Gaussian around current policy)

The algorithm alternates between:

E-step (policy improvement):

Compute a non-parametric distribution $q^*(a \mid s) \propto \pi_\theta(a \mid s) \cdot e^{\frac{Q(s,a)}{\eta}}$

Where η is a temperature term, tuned by dual optimization.

M-step (policy fitting):

We now fix q and update $\boldsymbol{\pi}$ to match it:

$$\max_\pi E_{s\sim\mu_q(s)}\left[E_{a\sim q(a|s)}[\log \pi\,(a|s)]\right] \quad \text{s.t.} \quad E_s[\mathrm{KL}(\pi_{\text{old}}|\pi)] \le \epsilon$$

This is solved via supervised learning where:

- Inputs: states
- Targets: actions weighted by q
- Objective: maximize log-likelihood, regularized by KL to prior policy

This step is crucial to fit the new policy, while keeping it close to the old one for stability.

Pseudocode for MPO:

```
# Initialization
Initialize actor πθ, critic Qϕ, replay buffer D

for each iteration:
    # Sample batch of transitions from replay
    Sample (s, a, r, s') ~ D

    # Estimate Q-function using Retrace
    Fit Qϕ with Expected SARSA or Retrace

    # E-step: compute target distribution q(a|s)
    For each s in batch:
        Sample actions a_i from πθ
        Compute weights w_i ∝ exp(Q(s, a_i)/η)
        Normalize weights to get q(a|s)

    # M-step: update πθ to minimize KL(q(a|s) || π(a|s; θ))
    Compute supervised loss weighted by q
    Update actor parameters θ with gradient descent

    # Update dual variables (α, η) using gradients
```

8.1.4. SOFT ACTOR-CRITIC (SAC)

SAC [8] is a state-of-the-art reinforcement learning algorithm designed for sample-efficient, stable learning in continuous control tasks. Unlike deterministic actor-critic methods such as DDPG, SAC uses a stochastic policy and incorporates entropy maximization to encourage exploration. This results in better stability and robustness across different environments and hyperparameter settings.

SAC combines:

- Off-policy learning for sample efficiency.
- A stochastic actor that maximizes both return and entropy.
- Twin Q-networks to mitigate overestimation bias.
- A reparameterization trick for low-variance policy gradients.

Theoretical Formulation

SAC is based on the maximum entropy RL framework, where the agent maximizes a trade-off between expected return and entropy. The objective function it tries to optimize is:

$$J(\pi) = \sum_{t=0}^{\infty} E_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}\left(\pi(\cdot\,|s_t)\right)\right]$$

- α: temperature parameter that balances reward vs. entropy.
- $\mathcal{H}\big(\pi(\cdot\,|s)\big) = -E_{a\sim\pi}[\log \pi\,(a|s)]$

The entropy term encourages diverse action choices, improving exploration and robustness.

Policy Iteration Framework

SAC follows a soft variant of policy iteration, alternating between soft policy evaluation and soft policy improvement.

1. Soft Policy Evaluation

Given a policy π, we define the soft Bellman backup:

$$Q^{\pi}(s_t,a_t) = r(s_t,a_t) + \gamma E_{s_{t+1}}[V^{\pi}(s_{t+1})]$$

Where:

$$V^{\pi}(s) = E_{a\sim\pi}[Q^{\pi}(s,a) - \log \pi\,(a|s)]$$

The Q-network is trained to minimize:

$$J_Q(\theta) = E_{(s,a,r,s')}\left[\frac{1}{2}\left(Q_\theta(s,a) - \hat{Q}(s,a)\right)^2\right]$$

$$\hat{Q}(s,a) = r + \gamma E_{a' \sim \pi}[Q_{\bar{\theta}}(s',a') - \alpha \log \pi(a'|s')]$$

- Two Q-functions are used: $Q_{\theta_1}, Q_{\theta_2}$
- A target network $\bar{\boldsymbol{\theta}}$ is used for stable bootstrapping.

In SAC, what might stand out the most is the use of two different critics. They are used to reduce positive bias from overestimation in value targets (common in Q-learning), done by taking the minimum of the two:

$$\hat{Q}(s,a) = r + \gamma\left[\min_{i=1,2} Q_{\overline{\theta_i}}(s',a') - \alpha \log \pi(a'|s')\right]$$

This "Double Q" trick was introduced in DDPG's successor TD3 [12] and improves stability and performance.

2. Soft Policy Improvement

The policy π_ϕ is updated using the reparameterization trick:

- Let $a = f_\phi(\epsilon, s)$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$
- Then:

$$J_\pi(\phi) = E_{s \sim D, \epsilon \sim \mathcal{N}}\left[\alpha \log \pi_\phi\left(f_\phi(\epsilon,s)\middle|s\right) - Q_\theta\left(s, f_\phi(\epsilon,s)\right)\right]$$

This results in a low-variance gradient estimate for stochastic policies.

Pseudocode

```
# SAC Algorithm
Initialize: actor π_φ, Q-networks Q_θ1, Q_θ2, target Q⁻ networks, replay buffer D

for each step:
    Sample action a ~ π_φ(a|s)
    Observe r, s′ from environment
    Store (s, a, r, s′) in D

    # Sample batch from D
    Sample (s, a, r, s′) ~ D

    # Critic update
    a′ ~ π_φ(a′|s′), log π = log prob(a′|s′)
    Q_target = r + γ * (min(Q⁻_θ1(s′, a′), Q⁻_θ2(s′, a′)) - α * log π)
    Loss_Q = MSE(Q_θi(s,a), Q_target) for i=1,2
    Update Q_θ1, Q_θ2

    # Actor update (via reparameterization trick)
    a ~ π_φ, log π = log π_φ(a|s)
    Loss_π = α * log π - min(Q_θ1(s,a), Q_θ2(s,a))
    Update φ

    # Update target networks
    θ⁻ ← τθ + (1 - τ)θ⁻
```