POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es

www.industriales.upm.es

INDUSTRIALES

05 TRABAJO FIN DE MASTER

Josep María Barberá Civera

TRABAJO FIN DE MASTER

# ADVANCED LOCOMOTION FOR QUADRUPED ROBOTS THROUGH REINFORCEMENT LEARNING

TRABAJO FIN DE MASTER
PARA LA OBTENCIÓN DEL
TÍTULO DE **MASTER EN AUTOMÁTICA Y ROBÓTICA**

SEPTIEMBRE **2024**

**Josep Mª Barberá Civera**

DIRECTOR DEL TRABAJO FIN DE MASTER:
**Antonio Barrientos Cruz**
**Christyan Mario Cruz Ulloa**

INDUSTRIALES
ETSII | UPM

POLITÉCNICA


ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

MÁSTER EN AUTOMÁTICA Y ROBÓTICA

*Master's Thesis*

# Advanced Locomotion for Quadruped Robots Through Reinforcement Learning

---

Supervisors:
Dr. Antonio Barrientos
Dr. Christyan Cruz

Student:
Josep María Barberá Civera
17048

---

September 2024

*"We can only see a short distance ahead,*
*but we can see plenty there that needs to be done."*
Alan Turing

# Advanced Locomotion for Quadruped Robots through Reinforcement Learning

**by**

**Josep María Barberá Civera**

Submitted to the Escuela Técnica Superior de Ingenieros Industriales
on September 6, 2024, in partial fulfillment of the
requirements for the degree of
**Máster en Automática y Robótica**

# ABSTRACT

Search and Rescue (SAR) operations face significant challenges in hazardous environments where rescuers are at high risk. In post-disaster scenarios, such as building collapses, chemical spills, or earthquake-affected zones, access to critical areas is often delayed for safety reasons. Deploying robots equipped with advanced sensors can provide essential real-time data to rescue teams, reducing risks and improving decision-making. Quadruped robots are particularly suited for SAR operations due to their ability to traverse difficult terrain, offering a clear advantage over wheeled or tracked robots. These robots can gather crucial information via cameras, LIDAR, and thermal imaging, aiding in locating survivors and assessing structural damage.

For quadruped robots to operate effectively in such challenging conditions, robust control strategies are essential. While Model Predictive Control (MPC) has been widely used for precise locomotion, it struggles in unpredictable environments due to high computational demands. Reinforcement Learning (RL) is emerging as a more adaptive and computationally feasible alternative, enabling robots to learn and adjust to complex, dynamic situations typical of SAR missions. This project proposes a structured methodology to enhance quadruped robot locomotion in SAR environments, focusing on RL-based control strategies.

The proposed framework is divided into three phases: training, testing, and deployment. In the training phase, robots learn to walk using Deep RL, specifically the Proximal Policy Optimization (PPO) algorithm, which was optimized through a hyperparameter study. Results showed an improvement of 8.85% in general locomotion and 16.73% in obstacle avoidance by fine-tuning the discount rate and entropy coefficient.

The testing phase integrates these trained policies into the Robot Operating System (ROS) for evaluation, leveraging simulated environments in Gazebo with sensor data fusion from two Intel RealSense cameras. Finally, the deployment phase involved building a physical test circuit and conducting real-world tests with the Aliengo robot. The established baseline for basic locomotion using the manufacturer's control system provides a qualitative reference for future RL-based strategies. The findings of this research highlight the potential of RL to significantly improve robot performance in SAR tasks, with further development needed to enhance robustness and real-world deployment.

# RESUMEN

**Las operaciones de búsqueda y rescate** (en inglés SAR) **se enfrentan a importantes retos cuando se trata de acceder a entornos peligrosos en los que los rescatistas pueden correr grandes riesgos**. En muchas situaciones postdesastre, tales como derrumbes de edificios, vertidos químicos o zonas afectadas por terremotos, el acceso inmediato a zonas críticas suele retrasarse por motivos de seguridad. **La necesidad de respuestas más seguras y precisas en estos entornos ha hecho que se preste cada vez más atención al despliegue de robots que puedan actuar como primera línea de acción**[1]. Estos robots, equipados con numerosos sensores, pueden proporcionar información vital a los equipos de rescate, identificando supervivientes, evaluando la integridad estructural de los edificios y detectando potenciales riesgos antes de que el personal de rescate entre en las zonas peligrosas [2].

**Entre los distintos tipos de robots, los cuadrúpedos son especialmente adecuados para las operaciones SAR** debido a su diseño con patas, que les permite desplazarse por terrenos difíciles y accidentados con mayor eficacia que los robots con ruedas u orugas [3]. Su capacidad para caminar, trepar y atravesar escombros los hace ideales para acceder a edificios derrumbados o zonas con escombros donde los métodos de rescate tradicionales tendrían más dificultades. Estos robots pueden proporcionar datos en tiempo real gracias a sus cámaras, sensore LIDAR o imágenes térmicas, que pueden ser fundamentales para la toma de decisiones en entornos de rescate.

Sin embargo, **para que los robots cuadrúpedos funcionen con eficacia en condiciones tan difíciles, necesitan en primer lugar estrategias de control y locomoción sólidas**. Tradicionalmente, el control predictivo (MPC) ha sido un método muy empleado para la locomoción de estos robots, ya que ofrece gran precisión y robustez [4, 5]. Aunque el MPC ha tenido éxito en muchos entornos estructurados, puede fallar en escenarios impredecibles y complejos, como superficies resbaladizas o zonas con mucha suciedad, esta limitación suele estar motivada en parte por su gran coste computacional. Frente a estas situaciones, **técnicas de inteligencia artificial como el aprendizaje por refuerzo (RL) han empezado a perfilarse como una alternativa más robusta y factible** [6], **a la vez que computacionalmente posible**. El RL permite al robot aprender y adaptarse a situaciones de gran complejidad, lo que lo hace más adecuado para entornos dinámicos y caóticos, característicos de este tipo de misiones.

Para abordar las limitaciones actuales de las operaciones SAR y la necesidad de sistemas de locomoción más avanzados, **este proyecto propone una metodología diseñada para mejorar la locomoción de los robots cuadrúpedos en entornos SAR. El marco desarrollado se estructura en tres fases: entrenamiento, pruebas y despliegue** (ver Figura 0.1).

En la **fase de entrenamiento**, los robots aprenden a caminar mediante técnicas de Deep Reinforcement Learning, donde la política de locomoción es una red neuronal, los pesos de la cual se ajustan gracias a la interacción del robot con el mundo. De este manera el robot aprende poco a poco a mantenerse en pie y seguir comandos de velocidad en entornos cada vez más difíciles. En concreto se emplea el algoritmo llamado Proximal Policy Optimization (PPO) el cual requiere de muchos individuos entrenando en paralelo

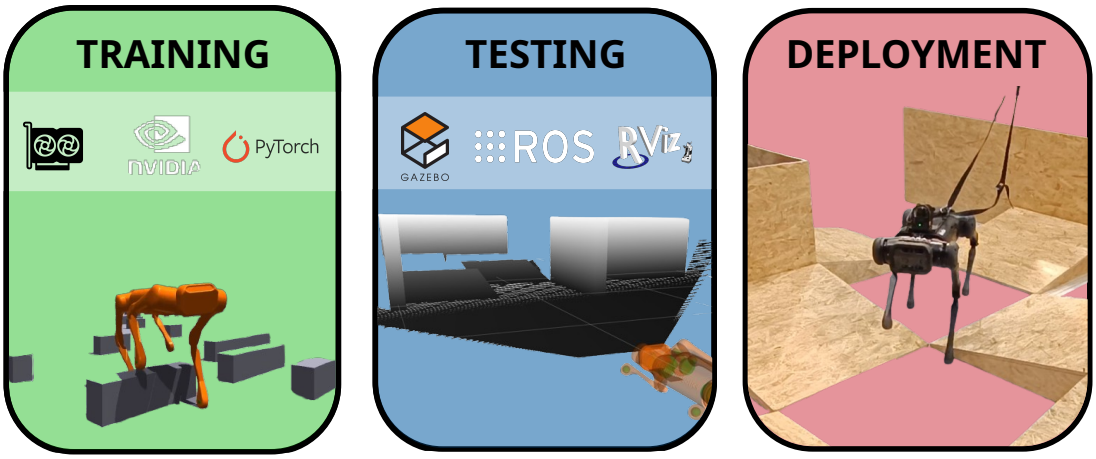


**Figure 0.1:** *Principales fases del framework propuesto: entrenamiento, pruebas y despliegue.*

para obtener así una única política más robusta. **Gracias al uso del entorno de entrenamiento Isaac Gym** desarrollado por NVIDIA, el cual optimiza la paralelización en targetas gráficas (GPUs), **se han logrado replicar trabajos como el de Rudin et al.** (Legged Gym [7]) **y el de Kareer et al.** (ViNL [8]).

**Se han llevado a cabo dos tipos de entrenamiento, uno multipropósito y otro de evasión de obstáculos.** El entrenamiento multipropósito, es decir, para la locomoción en entornos postdesastre, trata de enseñar al robot a que este avance y navegue lo más rápidamente posible sin caerse o sin perder el equilibrio. El entrenamiento para la evasión de obstáculos, busca dotar al robot de la habilidad de avanzar evitando pisar o golpear objetos del suelo, replicando así situaciones con elementos potencialmente peligrosos. Ambas fases se han entrenado gracias el uso de una GPU NIVIDA GeForce RTX 4060, junto con la implementación de Kareer et al.

Como mejora de esta primera etapa del framework, **se ha realizado un estudio hiperparamétrico para la optimización del entrenamiento**. Se han estudiado 11 hiperparámetros del algoritmo PPO probando dos nuevos valores para cada uno de ellos. Tras las simulaciones, **se ha constatado cómo la disminución del *discount rate*** ($\gamma$) **de 0,99 a 0,95 y del *entropy coefficient*** ($c_{\text{entropy}}$) **de 0,01 a 0,005 ofrece una mejora de la media total de entrenamiento respecto al baseline** (entrenamiento con los parámetros por defectos de Legged Gym[7]) **de** $+8,85\%$ **en el entrenamiento multipropósito y de** $+16,73\%$ **en el entrenamiento de evasión de impuestos** (ver Figura 0.2).

Respecto de la **fase de pruebas**, **se ha propuesto una arquitectura en ROS**[1] **en la que puedan evaluarse las políticas entrenadas**. En ella se detallan a nivel general, los paquetes necesarios así como los tópicos (topics) principales para el funcionamiento de la red neuronal. Se ha puesto especial hincapié en la identificación de las entradas y las salidas de la red, de manera que la implementación en ROS se efectiva. Será necesario adecuar la información recibada de los topicos a los que se subscribe el controlador, para que concuerden con el tamaño y forma que espera la red. **Se ha avanzado también en la construcción de cuatro circuitos de obstáculos en Gazebo, la inclusión**

[1]ROS (Robot Operating System) es un conjunto de bibliotecas y herramientas que ayudan a los desarrolladores a crear aplicaciones robóticas.

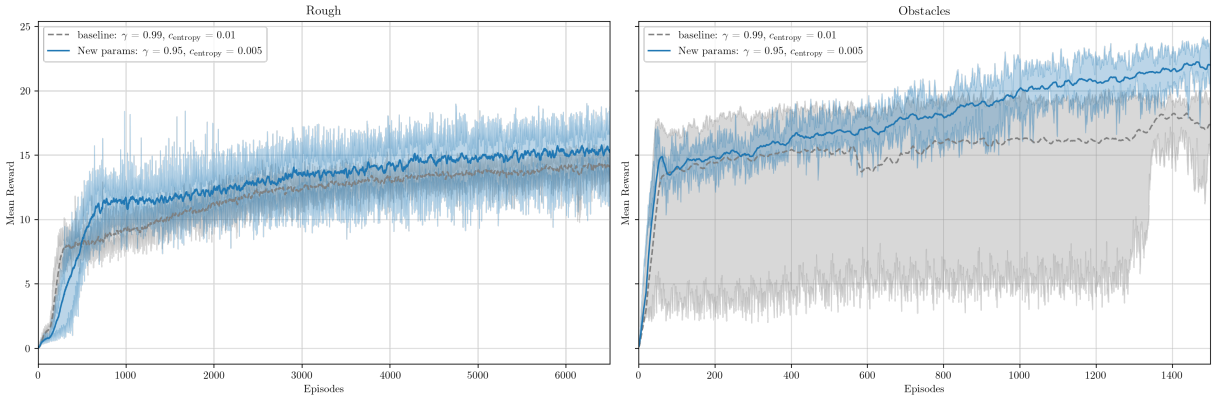


**Figure 0.2:** *Comparación del entrenamiento base y los parámetros optimizados para las etapas de (a) locomoción general y (b) evasión de obstáculos, mostrando una razonable mejora debido a los nuevos valores de los hiperparámetros del "discount rate" ($\gamma$) y el "entropy coefficient" ($c_{entropy}$).*

**de dos cámaras Intel RealSense D435 en el modelo del robot y la fusión de las nubes de puntos que estas ofrecen**. La integración de la red entrenada con la arquitectura de ROS propuesta resulta clave para la posterior implementación en el robot real.

Finalmente **en la fase de despliegue**, **se ha diseñado y construido un circuito de pruebas** alineado con los modelados en Gazebo (ver Figura 0.3). Para ello se han empleado paneles de madera conglomerada tipo OSB de 15 mm de espesor, construyendo las paredes del circuito y 6 rampas de base cuadrada y 75 cm de lado. Posicionando adecuadamente estos bloques se obtienen los 4 circuitos empleado en Gazebo, de manera que puedan compararse fácilmente ambas etapas. **Una vez construidos se ha procedido a probar con el robot real dichos circuitos, estableciendo así un criterio de referencia con el que comparar posteriores políticas de locomoción**.

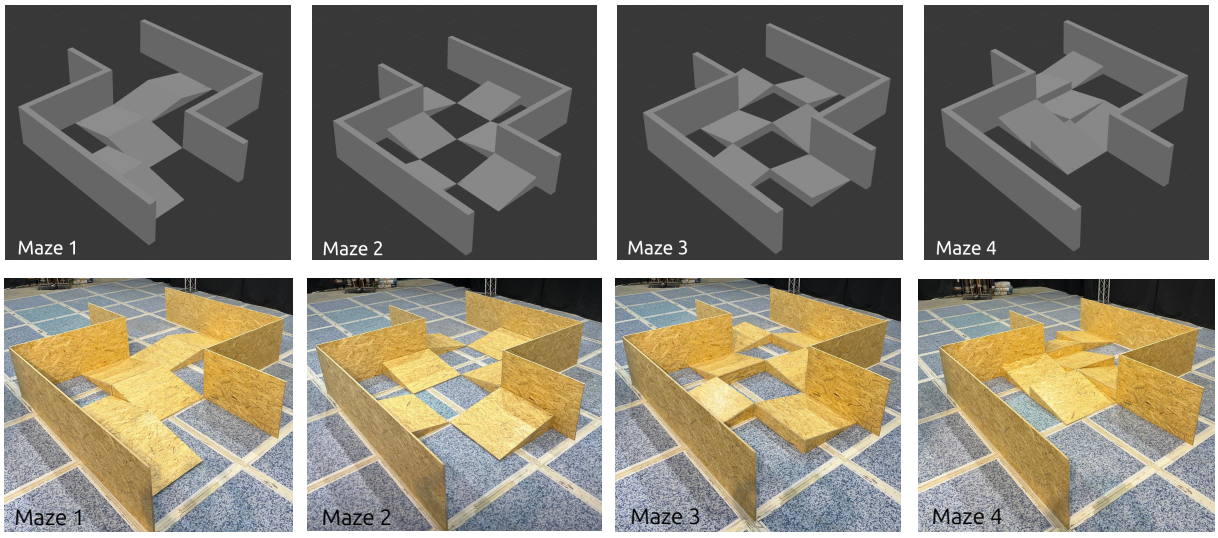


**Figure 0.3:** *Circuitos para las fases de pruebas en Gazebo y el robot real.*

**Se ha empleado el robot Aliengo de Unitree, en concreto se ha evaluado el generador de patrones de marcha que ofrece el fabricante**. Para ello, se ha teleoperado el robot por los cuatro terrenos comprobando qué nivel de dificultad suponen

para esta locomoción básica. Por seguridad dos operarios acompañaban con un sistema de sujección al robot mientras este navegaba por el circuito. Se ha podido constatar (ver Figura 0.4) que los circuitos tienen dificultad creciente en este orden: 2, 1, 4, 3. En concreto, este último circuito no ha sido finalizado por el robot, sufriendo caídas para dos modos de marcha, el modo sport y el modo de ascensión de escaleras.

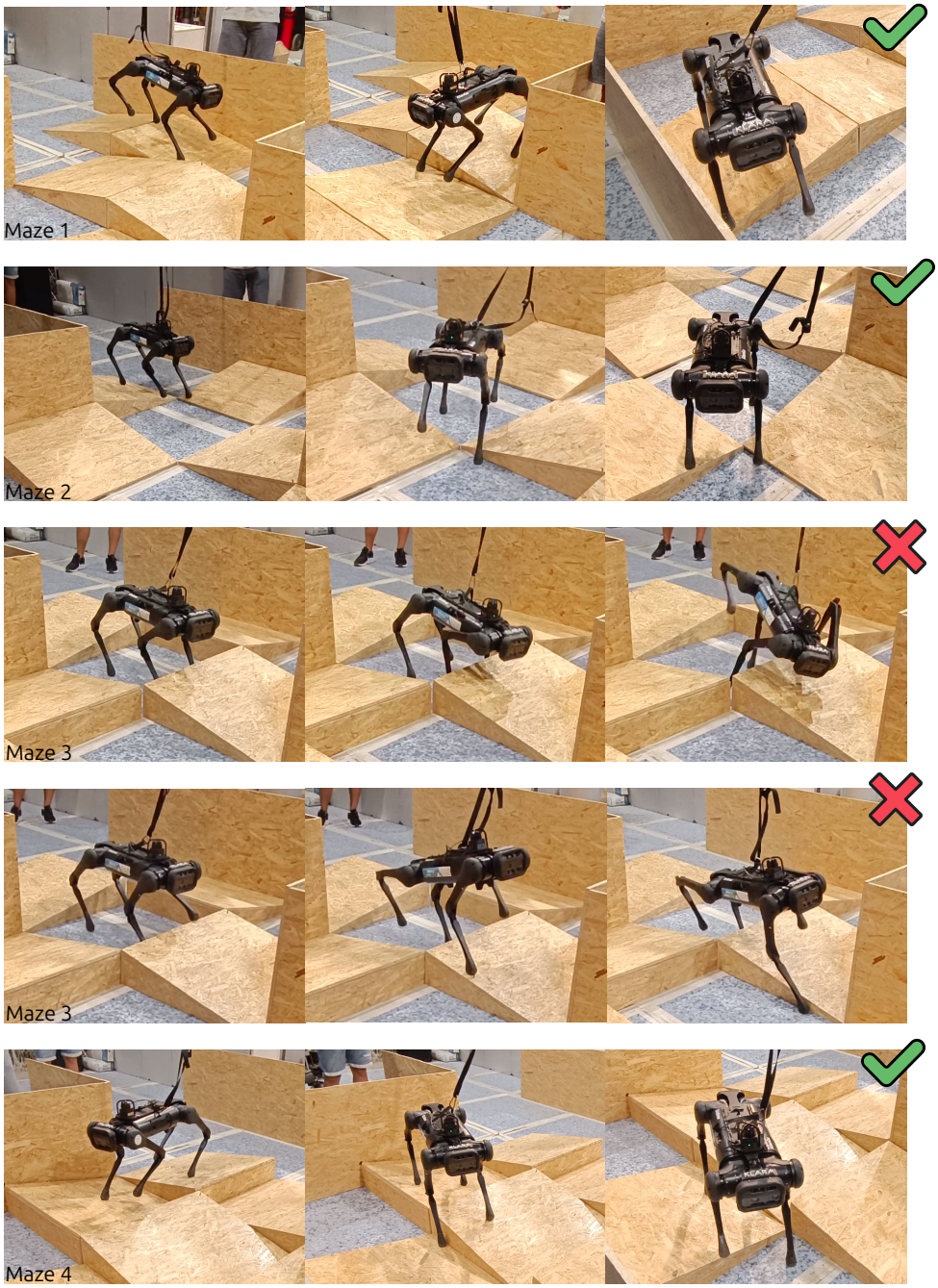


**Figure 0.4:** *Pruebas con el robot Aliengo: navegación a través de cada circuito*

Se ha podido concluir, por tanto, cómo los patrones de marcha actuales son frágiles y carecen de control en bucle cerrado. En estos casos se requiere una teleoperación constante y antenta o un planificador local que se adapte lo mejor posible al terreno buscando vías de fácil navegación.

Los principales resultados de esta investigación indican que el ajuste de los parámetros de entrenamiento de RL puede mejorar significativamente el rendimiento del robot y se proponen como referentes para posteriores trabajos con Legged Gym los valores obtenidos ($\gamma = 0.95$ y $c_{\text{entropy}} = 0.005$). Además, la arquitectura de ROS propuesta junto con los avances en las simulaciones con Gazebo ofrecen gran potencial y puede facilitar futuros trabajos en esta línea. Finalmente, la creación de un circuito de pruebas junto con el establecimiento de una referencia cualitativa para la locomoción básica del robot Aliengo, permiten la evaluación de nuevas políticas de locomoción basadas en RL u otros sistemas de control. El futuro de la robótica SAR depende de que se impulsen este tipo de avances, garantizando que los robots cuadrúpedos puedan ayudar de forma segura y eficiente en las operaciones de rescate y, en última instancia, salvar vidas en entornos de alto riesgo.

## Códigos UNESCO

- [**120304**] - Inteligencia Artificial
- [**330419**] - Robótica
- [**240102**] - Comportamiento Animal
- [**120326**] - Simulación

## Palabras clave:

Deep Reinforcement Learning, robots cuadrúpedos, GPUs, hiperparámetros, Proximal Policy Optimization.

# ACKNOWLEDGEMENTS

First of all, I would like to thank Antonio Barrientos for his constant support, for giving me the opportunity to do this TFM in the Robotics and Cybernetics group and for his continuous interest and selfless help. I would also like to thank Christyan Cruz for his close and always available guidance, without his support this work would not have been possible. Thanks also to the rest of RobCib members and students (David, Jorge, Miguel, ...) for always being open to my questions, for your good humor. Also, a big thanks to Jota for his help in hardware testing, for his continuous support throughout the project, sharing our progress has made everything more bearable.

Many thanks to Jorge Villagrá for his unwavering support. His help in allowing me to focus on completing this project before starting my PhD has been invaluable, and I hope I can express my gratitude to him fully. Thanks to the other members of Autopía, especially to Marcos for his discussions about RL and nomenclature, also to Juanlu, with him the work with ROS has been simple and intuitive, thanks to all of you, I hope and wish to have time to know and appreciate you better.

I want to thank my family for all their support, first of all my parents for always giving me the best, listening to me, advising me, never doubting me and encouraging me to be more generous. To my siblings, for their support, their joy, their company.

Thanks to Pepe, for your friendship, companion of sorrows and joys, because everything is better among friends. Thanks to Miquel, for being there, for always advising me and setting an example. Thanks to all the rest of my friends, from Tomás Bretón, Turmalina, AF3. Thanks for Camilo (yes... I still owe you a hamburger...), thanks for Rafa and Javi, this summer has been unforgettable.

Thanks to all of you

# Table of Contents

# LIST OF TABLES

# LIST OF FIGURES

# CODES INDEX

# 1 INTRODUCTION

This chapter explains the main reasons behind this research, its motivation and underlying ideas (see Section 1.1). It also outlines the main objectives set at the beginning of the investigation and describes how this report is organized (see Section 1.2). Finally, it summarizes the principal contributions based on the outlined objectives, and provides references and links to additional materials created throughout this study (see Section 1.3).

## 1.1 Motivation

Quadruped robots have garnered significant interest in recent times due to their versatile applications and capabilities. On the one hand, quadruped robots have the ability to access highly unstructured terrain, which has recently made them the preferred choice over wheeled or tracked robots for search and rescue missions [3] (see Figure 1.1.a). Furthermore, the addition of robotic manipulator arms (as shown in Figure 1.1.b) makes them more suited for carrying out tasks in hazardous or extreme environments like opening doors, closing valves or grasping objects [9]. These capabilities have been highlighted by recent improvements in actuators and locomotion systems [10], as well as the availability of increased computing capabilities in control devices. This is evidenced by the latest robotic demonstrations where the locomotion and manipulation possibilities of quadruped robots are shown [11, 12].

a) b)

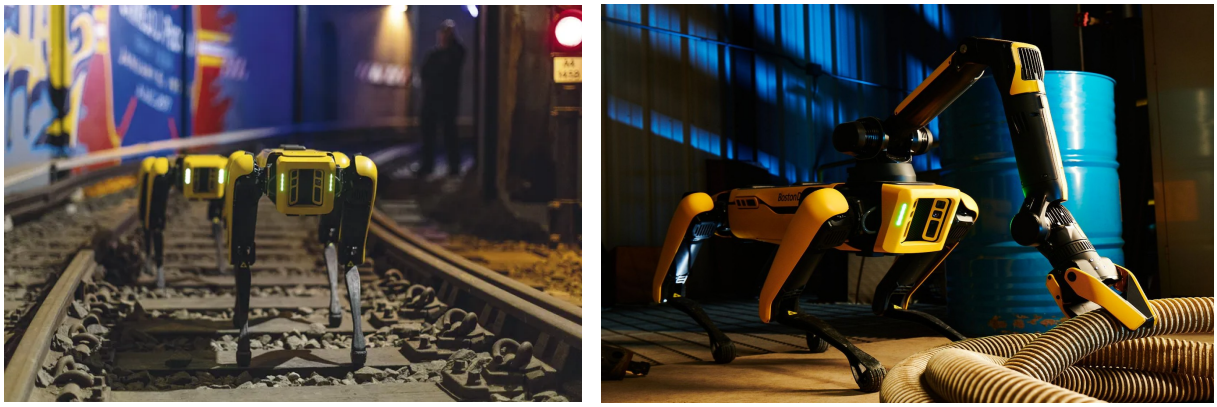

**Figure 1.1:** *Spot, the quadruped robot from Boston Dynamics. (a) During an underground training mission with the New York Fire Department [13]. (b) Equipped with a robotic arm in an industrial environment [14].*

Despite the high expectations placed on quadruped robots, their current capabilities remain limited [15]. These systems are mostly demonstrated and tested in laboratories or under controlled conditions, which means that their applicability in real world scenarios is rather limited. Therefore, further research is needed to improve their robustness and performance in multi-terrain or post-disaster environments. But also pushing them up to the next level, making them self-sufficient and independent learning.

Given the highly non-linear nature of quadruped locomotion, Model Predictive Control (MPC) formulations have been commonly used [5]. MPC integrates optimal control,

dead time handling, stochastic control, and multivariable control, making it a robust and reliable technique for biped or quadruped robot locomotion. However, this robustness comes at a cost, as MPC is computationally intensive and often slow, limiting its real-world applicability.

In contrast, Reinforcement Learning (RL) and other Data-Driven approaches have gained popularity in addressing complex problems in quadruped locomotion and robot manipulation. As highlighted by Lee *et al.* in [16], RL bypasses the need for a physical model by using a neural network trained on specific control policies or large datasets. This training can occur either online or offline. RL's ability to quickly adapt and optimize makes it a fast and cost-effective alternative, ideal for practical applications where speed and affordability are critical.

### 1.1.1 Promising Areas in Robotics and Machine Learning

Also, as a source of inspiration for our research topic and recognizing the significant contributions that quadruped robots are expected to make to science and technology, we consider it appropriate to highlight some promising areas that are likely to have a major impact on robotics in the coming years. Figure 1.2 illustrates six interrelated lines of research, encompassing robot learning, training, and deployment.

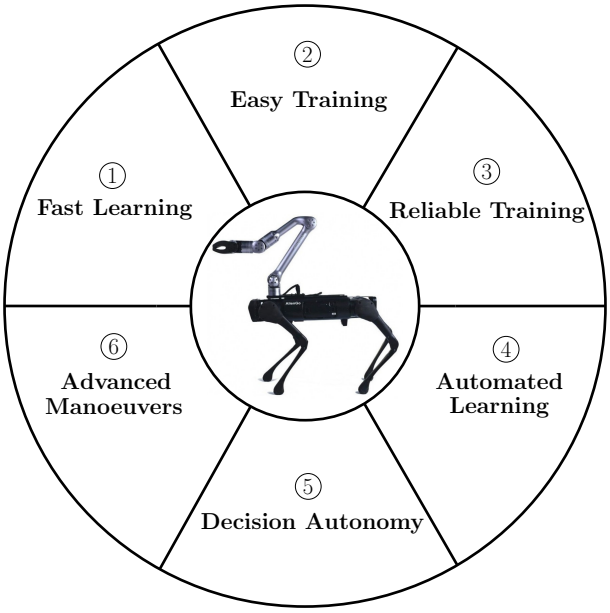


**Figure 1.2:** *Six major impact research lines for robotics in the coming years.*

① Simulations will be much faster, to such an extent that it will be possible to train the Neural Networks (NNs) directly with the data obtained in real time by the robot during task execution. Research will be needed on new algorithms and architectures for super-fast NNs.

② Reinforcement Learning must evolve to a new semi-automated form in which low-level training policies are generated from high-level language. In addition, forms of imitation training (follow me and learn) will enable robots to function in all kinds

of environments. It will be necessary to combine programmed guidelines by written language, voice commands, and visual information.

(3) The NNs must be fully reliable. Efforts should be made to impose restrictions on the NNs, ensuring their outputs are bounded by any input.

(4) Learning should be a continuous process. Initially, the robot must be able to know everything about itself (importing dimensions, masses, inertias, etc., directly from the URDFs). As a result of (1), the robot will be capable of updating its behavior in real time by interacting with and learning from the environment in new situations.

(5) Long-term memory will allow the robot to recognize past situations and act accordingly. In addition, it will need to fuse knowledge, both from what it has learned during the mission deployment and from what it has learned on other missions. Knowledge "condensation" or maturation processes will be necessary for the development of intelligent memory to allow the robot to continuously improve and become more autonomous in performing tasks.

(6) In addition to all of the above, it is necessary to bring robots to a higher capacity to interact in an advanced and complex way with the environment. Limbs (legs and manipulators) will allow for more natural and efficient maneuvers. For example, they will enable the robot to climb steep slopes using natural grips or to improve stability using the manipulator.

### 1.1.2 Search and Rescue

Since the early 21st century, advancements in science and technology, particularly in robotics, have significantly expanded the capabilities of Search And Rescue (SAR) operations[17]. The deployment of robots in these operations has proven invaluable, especially in scenarios where human access is either limited or entirely restricted due to high-risk conditions[1]. Such conditions may arise from natural disasters like earthquakes, wildfires, or avalanches, as well as from man-made events such as terrorist attacks that result in building collapses or the need to disarm explosive devices. In these critical situations, the prompt search and rescue of injured individuals is paramount, and robots offer vital assistance [18]. Equipped with advanced sensors and detection systems that surpass the capabilities of human rescuers, robots can efficiently locate and aid in the rescue of victims, enhancing the overall effectiveness and safety of SAR missions[2].

Some examples of robots used in SAR missions include:

- **Inuktun robots:** Following the September 11, 2001, terrorist attacks on the Twin Towers, and the devastation caused by Hurricane Katrina, Inuktun robots (as seen in Figure 1.3.a) were deployed for inspection and search tasks in confined and hazardous spaces. These robots were instrumental in navigating the narrow, debris-filled environments where human access was limited.
- **EMILY (Emergency Integrated Lifesaving Lanyard):** EMILY is an aquatic robot designed to save individuals who are drowning or trapped in rough waters

a) b) c)

**Figure 1.3:** *Examples of Search and Rescue robots: (a) Inuktun robot, designed for operation in confined spaces [19]; (b) EMILY, an aquatic robot used for water rescues [20]; (c) a drone used in aerial search and rescue missions [21].*

(refer to Figure 1.3.b). It can swiftly reach victims, serve as a flotation device, and even tow them to safety, making it an invaluable tool in water rescue operations.

- **UAVs (Unmanned Aerial Vehicles):** UAVs have been widely utilized in SAR missions. For example, the National Park Service uses drones to search for missing persons in areas like the Grand Canyon. They were also used to map earthquake damage in Nepal in 2015 (see Figure 1.3.c), aiding in prioritizing rebuilding efforts by providing real-time aerial assessments of the affected regions.

In this context, quadruped robots start to being used as SAR robots. For example, at the SpaceX facilities, after failure tests of their prototypes, a Spot robot has been observed inspecting the damages and collecting data [22], a task that could not be performed without its help until several hours later, leading to a potential loss of information.

### 1.1.3 CESAR and EXTRA-BRAIN Projects

Given that this work is part of the Robotics and Cybernetics group within the Center for Automation and Robotics CAR (a joint center of UPM and CSIC), it naturally aligns with several of the research projects to which the group is affiliated. Two of these projects, in particular, have significant relevance to quadruped locomotion.

- **CESAR**: **C**ollaborativ**E** **S**earch **A**nd **R**escue **R**obot (PID2022-142129OB-I00), whose primary mission is to enhance the capabilities of small and medium-sized search and rescue robots to assist rescue teams in post-disaster operations. Specifically, its initial focus is on improving the exploration capabilities of these robots, which includes the enhancement and evaluation of gait adaptability in commercial quadruped robots.
- **EXTRA-BRAIN**: **EX**plainable **TR**ustworthy **AI** for Data Intensive Applications (HORIZON-CL4-2023-HUMAN-01-01 Research and Innovation Action) focuses on improving brain-inspired perception in autonomous robots, specifically for SAR operations. In this context, the project aims to enhance the agility and safety of

these robots in disaster zones, ensuring better real-time decision-making and safe navigation through challenging conditions.

Align with these projects it is fundamental to make significant progress in enhancing the locomotion capabilities of commercial robots, such as those from the manufacturer Unitree, beyond their basic functionalities. To achieve this, we will focus on applying AI data intensive methods for robotics, specifically through the paradigm of Reinforcement Learning.

## 1.2 Aim, Objectives and Thesis Organization

### 1.2.1 Aim and Objectives

The aim of this research is to progress in the locomotion of quadruped robots using Reinforcement Learning (RL), specifically in the context of Search and Rescue missions. To achieve this, five specific objectives have been outlined:

1. **Explore the quadrupedal locomotion problem**
   (a) Contextualize quadrupedal robot research, along with its historical background and morphological aspects.
   (b) Conduct a study on RL as a promising tool for robotic training.
   (c) Perform a state-of-the-art review focused on the specific task of quadrupedal locomotion.
2. **Evaluate and optimize the RL training process**
   (a) Replicate outstanding RL works for quadrupedal locomotion.
   (b) Conduct a hyperparameter study to enhance performance.
3. **Propose a new ROS architecture**
4. **Design and build a new test circuit for robotic locomotion**
5. **Organize all as a framework for quadrupedal Locomotion**
   (a) Review existing frameworks and identify areas for improvement.
   (b) Include new training optimization.
   (c) Include new ROS architecture.
   (d) Define the baseline performance for commercial robots in the built test circuit.

### 1.2.2 Thesis Organization

The structure of this thesis is as follows:

- **Chapter 2** (Literature Review) addresses the first objective: understanding the quadrupedal locomotion problem. This chapter begins by examining the state of quadruped robots in terms of citations, publications, and patents. Next, it briefly reviews the history of quadrupedal robots, outlining their key characteristics and capabilities. It then introduces the fundamental concepts of Reinforcement Learning, with a focus on the PPO algorithm. Finally, the chapter concludes with a concise state-of-the-art review on quadrupedal locomotion, discussing the use of MPC and, more extensively, RL.
- **Chapter 3** (Methodology) covers the third objective, as well as the fourth and part of the fifth. It begins by analyzing the available tools for addressing the quadrupedal locomotion problem in robotics, alongside a brief review of relevant frameworks. The chapter then introduces our new framework and details its main components: training, testing (with a ROS-based architecture), and deployment (supported by the construction of a test circuit).
- **Chapter 4** (Results) addresses the second objective. It first explains the simulations conducted and the hyperparameter study performed. Finally, it presents a qualitative assessment of the robot's gait patterns for each of the constructed circuits.
- **Chapter 5** (Conclusions and Future Work) concludes the thesis by summarizing the key findings and outlining future objectives for continued research.

## 1.3 Contributions

The main contributions related to the outlined objectives are as follows:

- **Objective 1:** Explore the quadrupedal locomotion problem
  - A bibliometric and patent study on quadruped robots.
  - An introduction to the history and hardware of quadrupedal robots.
  - A brief overview of the fundamentals of Reinforcement Learning.
  - A literature review on advancements in quadrupedal robot locomotion.
- **Objective 2:** Evaluate and optimize the RL training process
  - Optimization of hyperparameters for RL training using PPO. The new studied values of the `discount rate` ($\gamma = 0.95$) and the `entropy coefficient` ($c_{\text{entropy}} = 0.005$) resulted in overall average improvements of +8.85% in the first training phase and +16.73% in the second phase.

- **Objective 3:** Propose a new ROS architecture
  - A block diagram illustrating the inputs and outputs of the system.
  - Initial iterations in Gazebo and RViz for positioning the robot within the circuits, integrating two depth cameras into the robot, and merging the point clouds from both cameras.
- **Objective 4:** Design and build a new test circuit for robotic locomotion
  - Construction of a 3.5 m x 3 m test circuit using 15 mm OSB wood panels.
- **Objective 5:** Organize everything into a framework for quadrupedal locomotion
  - Establishment of a qualitative baseline for the default gait patterns of the Aliengo robot during navigation through the constructed obstacle circuit.

The code for preparing, launching, and plotting the hyperparameter study, along with various software documentation files, is available at https://github.com/jbarciv/Hyper-ViNL.

# 2 LITERATURE REVIEW

This chapter provides an in-depth overview of Quadruped Robots (QR). It begins with an introduction to the context of robotics research using structured databases (refer to Section 2.1). Next, Section 2.2 delves deeper into QR by examining relevant patents. Section 2.3 reviews the historical development of QR and highlights the world's leading manufacturers. Section 2.4 covers the core principles of Reinforcement Learning algorithms, comparing the most common techniques. Finally, Section 2.5 presents a state-of-the-art study on the use of RL for QR locomotion.

## 2.1 Bibliometric Study on Quadruped Robots

Scientific research today is perhaps easier than it was a century ago. Thanks to advances in technology and the use of the internet, the number of publications has increased rapidly, and they have become more easily accessible. It is therefore necessary to have the right tools and methods to make the most of these resources. This section deals with a bibliometric study based on structured databases on legged and quadruped robotics.

The first step is to search structured databases such as Web of Science (WoS) [23] and Scopus [24] which currently rank among the most trustworthy bibliographical databases [25, 26]. This study will be focused on WoS results, but the same research would also be feasible in the case of using Scopus.

It is common for these websites to have advanced search engines that allow the use of the boolean operators. This helps us to perform more precise searches or queries. As mentioned above, we will focus on "quadruped robots", but we also want to continue with two more specific topics, which will help us to appreciate the differences regarding the available data between the general context of quadruped robotics and others more specific subfields. These special topics are locomanipulation through reinforcement learning and its fusion with MPC with the help of Physics Informed Reinforcement Learning. They have been selected because in the field of quadruped robots they are today the most state-of-the-art problems.

The queries used in each theme are detailed below:

- **Legged robots**: two options are studied and compared.

```
topic = "legged* robot*"
```

**Listing 2.1:** *Legged Robot*

```
topic = "quadruped* robot*"
```

**Listing 2.2:** *Quadruped Robot*

- **Locomanipulation with quadruped robots using reinforcement learning techniques**:

```
topic =
("quadruped* robot*" OR "legged robot*") AND
("mobile manipulation" OR "locomanipulation" OR
("Deep" AND ("Reinforcement Learning" OR
"Learning")) OR "Isaac Gym")
```

**Listing 2.3:** *Quadruped Robot RL Locomanipulation*

- **Physics Informed Reinforcement Learning and MPC in quadruped robots locomanipulation**: in this case the next two queries will be necessary.

```
topic =
("Physic*-informed" AND "Neural Networks") OR
("Physic*-informed" AND "Reinforcement Learning")
```

**Listing 2.4:** *PINNs and PIRL*

```
topic =
"quadruped* robot" AND (("optimal" AND "control") OR
"MPC" OR ("model predictive" AND "control"))
```

**Listing 2.5:** *RL and MPC*

The second step after using one of the previous queries, either in WoS or Scopus, is to select the information needed for the study. Given that we are carrying out a bibliometric study, we are firstly interested in the number of annual publications and annual cites.

Finally, plotting the data will permit us to visualize better its meaning. In our case with the help of the python library "Matplotlib" [27] we have implemented a customized Python script to represent the information. The data was stored in Python dictionaries where it was possible to programmatically fill in the years in which there was no information, associating a null value of publications with that year. Apart from the annual publications, these structured databases offer us much more information concerning the author, the cites, affiliations, etc. It is possible to export this data and analyze it, for example with the use of the R library "bibliometrix" [28].

First, annual citations for the "legged robotics" has been acquired from WoS, please refer to following Figure 2.1. It can be seen from the graph that the number of citations is growing exponentially. This is largely due to the increase in publications in recent decades.

Data relating to annual publications for each specific query have been displayed in the form of bar charts. The more general topic is presented first. Figure 2.2 gives an overview of the results obtained from WoS for publications on "legged* robot*" (in blue) and "quadruped* robot*" (in orange) covering the period from the mid-1980s to 2023. The number of publications on each topic has been analyzed in the context of the wider field of robotics, with relative values expressed as percentages[2]. Note how quadruped robots represent a subfield within legged robotics. Furthermore, it can be seen how it is becoming increasingly important in the sector, accounting for more than half of all

[2]This relative values are computed as $\frac{\text{No. of publications about a specific topic within robotics}}{\text{No. of publications about robots}} \cdot 100\%$

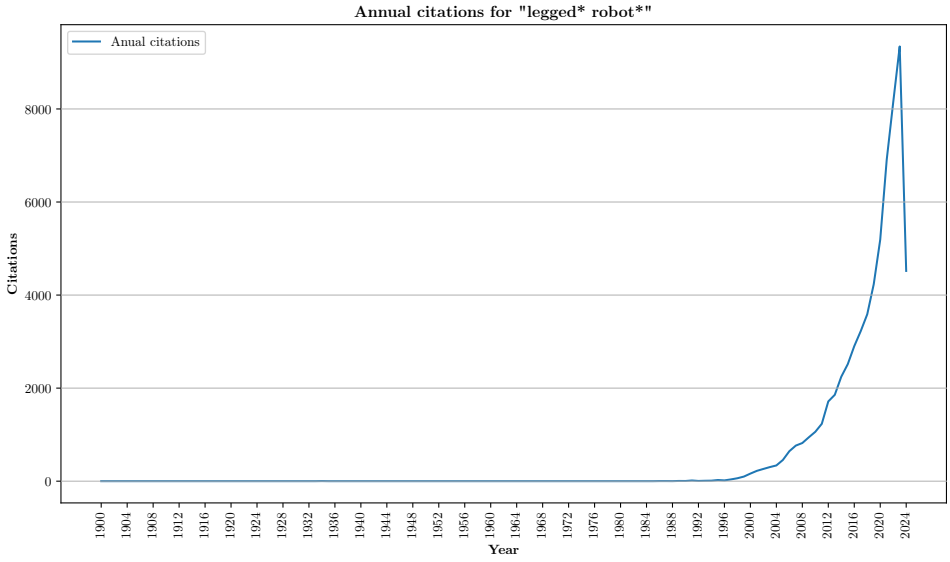


**Figure 2.1:** *Annual citations from* ***Web of Science (WoS)*** *for "legged* robot*" query.*

research in the area over the last ten years. As can be seen, legged robotics began before quadruped robotics, which did not begin until the mid-1980s. In the early years there was a great growth in research with legged robots, with quadruped robotics being in the minority, this changed slightly in the 2000s when legged robotics had its first boom, with quadruped robotics accounting for half of the publications. Around 2008, interest in these areas declined, but quickly picked up again around 2012, when the biggest boom so far was reached. Currently, interest in legged robotics remains close to that of 2012, but at lower levels, and the same is happening with quadrupeds. While legged robotics seems to be on a general downward trend, quadrupeds have been gaining more and more interest within the sector, accounting for almost three quarters of all legged robotics research last year.

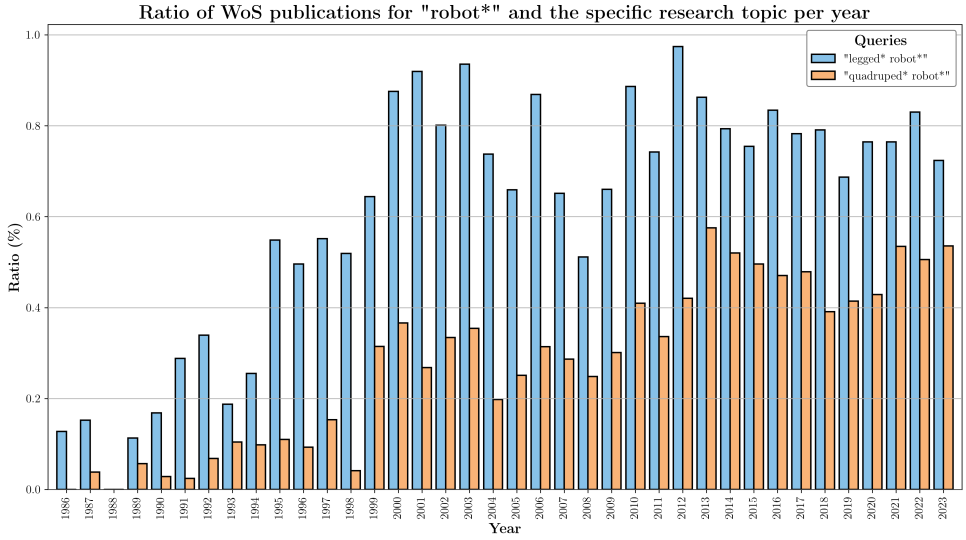


**Figure 2.2:** *Annual publications from* ***Web of Science (WoS)*** *for "legged* robot*" (blue) and "quadruped* robot*" (orange) queries, the orange one being next to the blue one. The values are expressed as percentages relative to the general field of robotics ("robot*").*

The same process has been repeated for more specific lines of research. Firstly, for "locomanipulation with quadruped robots through the use of reinforcement learning", in this case the data obtained is shown in Figure 2.3. The results are more heterogeneous, as they include publications related to legged or quadruped robotics and their association with reinforcement learning methods, manipulation and locomanipulation, deep learning, and even Nvidia RL simulators [29] due to its widespread use with quadruped robots. The available information starts in 2009 and shows sporadic activity until 2015. There is then a notable increase in development within the sector up to 2018, after which research follows a linear trajectory up to the present. This trend highlights an emerging field characterized by both growing interest and a narrow previous history. The reason for that can be found in the emergence of intelligent learning methods, which previously had no industrial relevance or practical results, but recently has been boosted by advances in technology and computing capabilities.

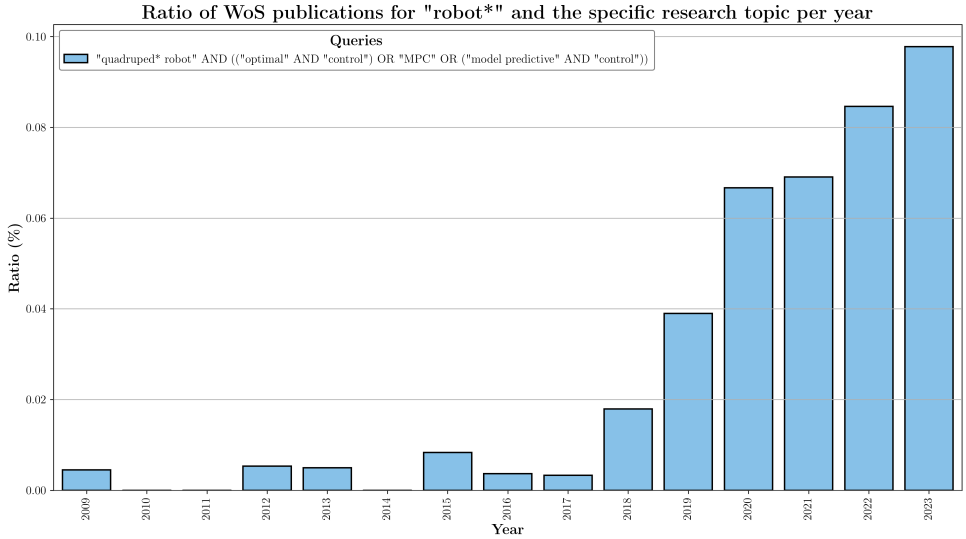


**Figure 2.3:** *Annual publications from* ***Web of Science (WoS)*** *for the Query 2.3. The values are expressed as percentages relative to the general field of robotics ("robot*").*

Similarly, Figures 2.4 and Figure 2.5 present the results for the field of "Physics Informed Reinforcement Learning (PIRL) and Model Predictive Control (MPC) in quadruped robots locomanipulation", which has been divided into two searches: one for the concept of PIRL and the other for MPC. In both scenarios, data appear limited, especially when novel concepts are explored. This scarcity is accentuated as the concept under investigation becomes increasingly innovative.

In the case of the search on PIRL, it can be seen in Figure 2.4 that it presents a high degree of novelty given that it only started 5 years ago. Since its inception it has gained a lot of importance in the scientific community with higher ratios than those seen so far in the previous searches. Also the "Physics Informed Neural Networks (PINNs)" research field has been searched within the PIRL.

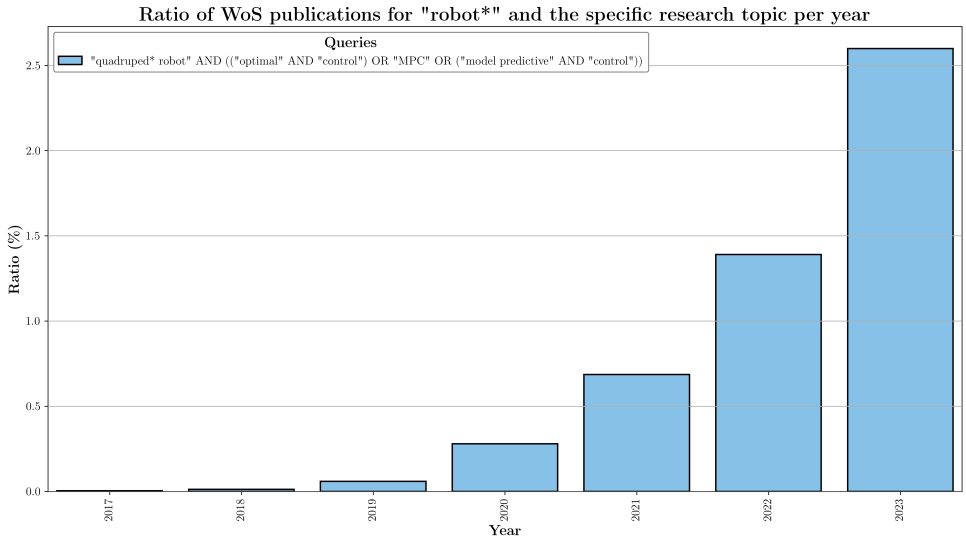


**Figure 2.4:** *Annual publications from* ***Web of Science (WoS)*** *for the Query 2.4. The values are expressed as percentages relative to the general field of robotics ("robot*").*

On the other hand, MPC is a well-known control paradigm that has been implemented in many systems since the 1970s [30]. When searching for publications in this field applied to the locomotion and locomanipulation of quadruped robots, the amount of information obtained is somewhat greater than in previous searches. If you look closely at Figure 2.5, you can see that until 2015 it was not a topic of great interest. The years 2016 and 2018 are interspersed between years with many publications, we believe that this phenomenon may be due to the fact that these years were years of development of the necessary knowledge as a preliminary step to subsequent publications. Since 2020 the sector has become more mature, due in part to the generalization of this area around the world, with many more entities now conducting research and enabling a continuum of publications. It is an area of great current interest.

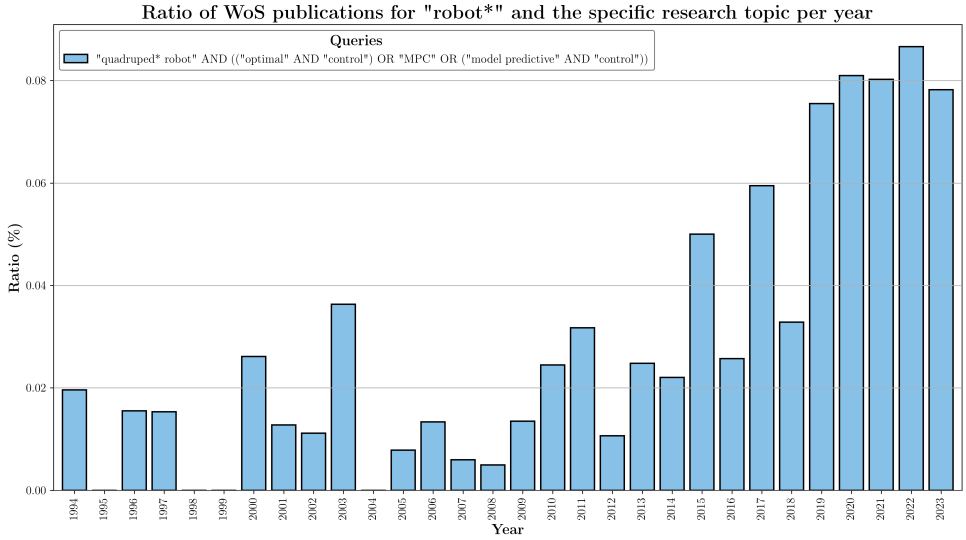


**Figure 2.5:** *Annual publications from* ***Web of Science (WoS)*** *for the Query 2.5. The values are expressed as percentages relative to the general field of robotics ("robot*").*

## 2.2 Patent Study on Quadruped Robots

Continuing with the bibliometric review, this section delves into the study of patent databases. The underlying motivation for this analysis is based on the fact that **a substantial amount of scientific and technical knowledge is accessible exclusively through patent documents**. In particular, patents offer the advantage of presenting information in a globally standardized format, which facilitates the tracking of relevant technologies and their categorization into groups or families. The following information is organized as follows. First, the chosen research topic is described. Next, the main patent databases consulted along with the methodology used are explained. Finally, the results obtained and conclusions regarding patents are presented.

Unlike the previous study, we hope to obtain results that are more focused on the hardware of these robots, their possible morphologies and locomotion mechanisms, among others. This information is complementary to that obtained in publications more focused on algorithms and programming.

Of all the possible patent databases, the following two have been selected (click on them to access the website):

- WIPO Patentscope
- Patent Lens

In them, a search was carried out with the following query:

```
topic = ("quadruped*" AND "robot*") NOT ("humanoid*" OR "legged")
```

Depending on the platform, it has been possible to include these restrictions to a greater or lesser extent. In general, we always restrict ourselves to the title, being more flexible in the abstract.

### 2.2.1 WIPO Patentscope

The World Intellectual Property Organization (WIPO) offers an open access patent database called *Patentscope*. It is very similar to *Patents Lens* but focuses more on international patent applications under the Patent Cooperation Treaty (PCT) and generally offers fewer resources and tools.

The full query has been entered without any problems and **627** results have been obtained. The most relevant information is included in the Table 2.1.

It is worth highlighting the **incredible difference in the number of patents** on quadruped robots **between China and the rest of the countries**. On the other hand, we can see that **it is a growing technology, as the first patents were not issued until 2015**, and in 2021 the figures almost tripled and remained at the same level in the following years.

| Countries | | Applicants | | Inventors | | IPC code | | Publication Dates | |
|---|---|---|---|---|---|---|---|---|---|
| China | 577 | HANGZHOU YUSHU TECH CO LTD | 32 | WANG XINGXING | 30 | B62D | 283 | 2015 | 7 |
| PCT | 16 | SHANDONG UNIVERSITY | 27 | LI CHAO | 26 | B25J | 191 | 2016 | 1 |
| United States of America | 11 | DEEPROBOTICS CO LTD | 10 | ZHU QIUGUO | 25 | G05D | 95 | 2017 | 19 |
| Japan | 7 | HANGZHOU DEEPROBOTICS TECH CO LTD | 10 | LI YIBIN | 20 | G05B | 35 | 2018 | 37 |
| Republic of Korea | 6 | BEIJING INSTITUTE OF TECH | 9 | RONG XUEWEN | 14 | G01C | 23 | 2019 | 48 |
| India | 4 | CHINA NORTH VEHICLE RESEARCH INSTITUTE | 9 | LI XUESHENG | 13 | G06T | 22 | 2020 | 80 |
| European Patent Office | 3 | | | YANG ZHIYU | 13 | G06V | 21 | 2021 | 129 |
| United Kingdom | 2 | DELU POWER TECH (CHENGDU) CO LTD | 9 | JIANG LEI | 11 | F15B | 15 | 2022 | 113 |
| France | 1 | GUANGDONG POWER GRID COMPANY | 8 | WANG CHUNLEI | 11 | G01S | 15 | 2023 | 161 |
| | | HARBIN UNIVERSITY OF SCIENCE AND TECH | 8 | YANG YA | 11 | G06N | 15 | 2024 | 16 |
| | | FOSHAN POWER SUPPLY BUREAU GUANGDONG POWER GRID CO | | | | | | | |

**Table 2.1:** *Summary table obtained from Patentscope [31] with the above query 2.2. It shows five columns with the associated figures, i.e. number of citations by country, applicant organizations, authors of the invention, classification code and year of publication.*

### 2.2.2 Patent Lens

*Patent Lens* offers much more information and with greater reliability. It is a very powerful tool for searching for patents, offering an immediate, very visual and detailed analysis. The previous query has been used again, and similar results have been obtained but with higher numbers.

In particular, **1023** patents have been obtained. The main graphs of the analysis provided by Lens are shown in Figures 2.6-2.8, which correspond to the most relevant applicant entities, the number of patents per year, and the top inventors in the sector, respectively.

**With regard to the applicant entities** which ask for a patent publication in quadruped robots, is worth noting that **the vast majority are Chinese universities as well as Chinese manufacturing companies** (e.g. Deeprobotics Co LTD [32]).

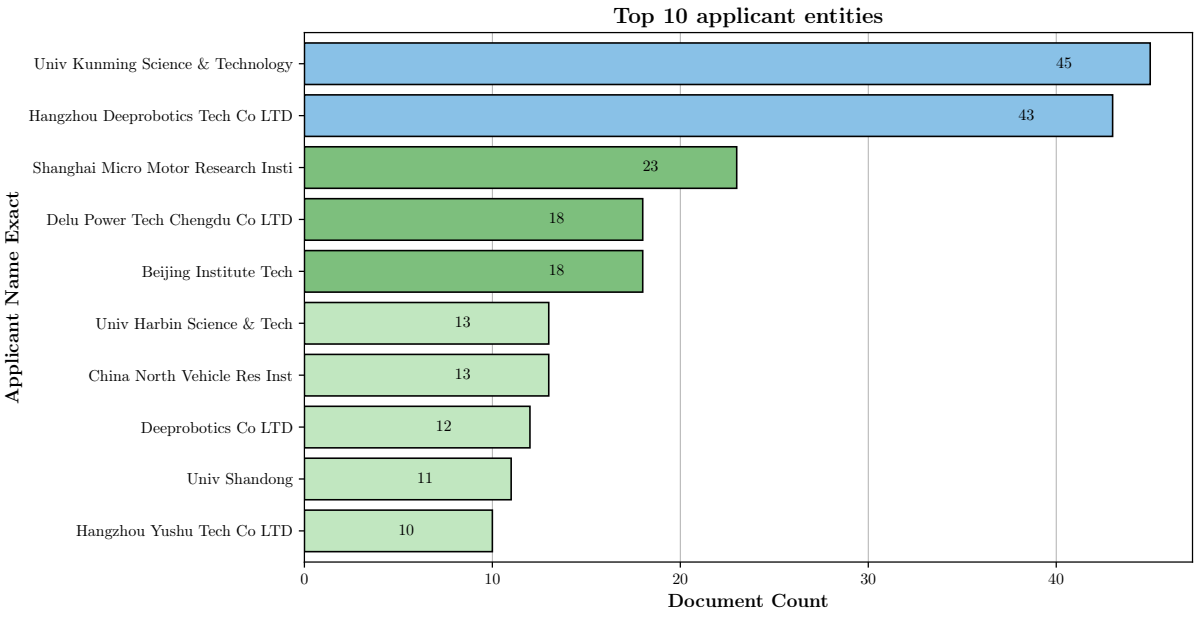


**Figure 2.6:** *Bar chart showing the top ten applicant entities.*

On the other hand, in terms of number of patents per year, the drastic increase can be seen again from 2021 onwards. In this case (see Fig. 2.7), **the first patents begin in the early 2000s, but the sector does not consolidate until twenty years later**.

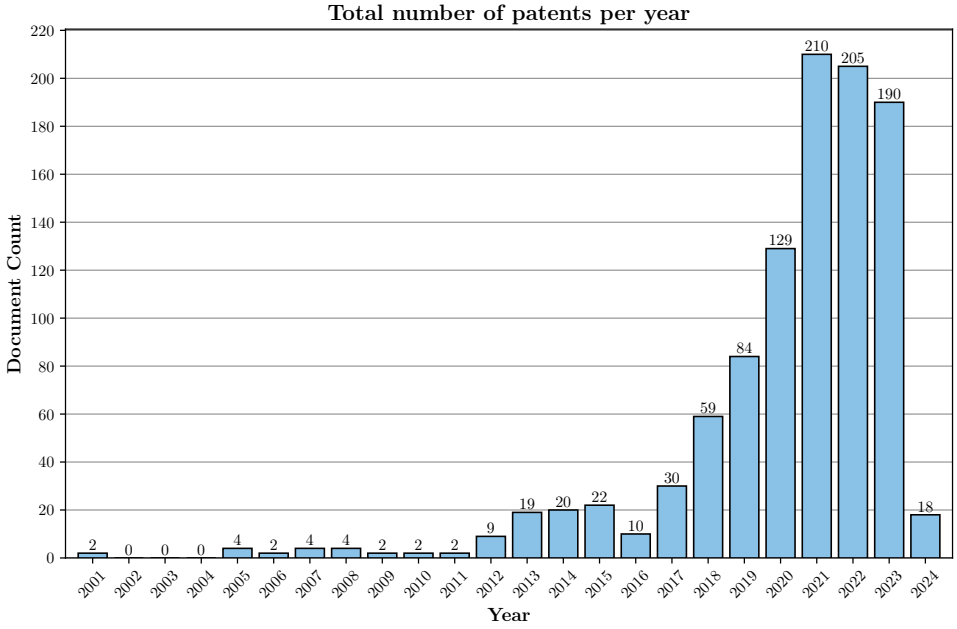


**Figure 2.7:** *Bar plot showing the number of patents per year.*

**Regarding to the most relevant inventors** in the quadruped robots, is surprising how **all of them are of oriental origin**.

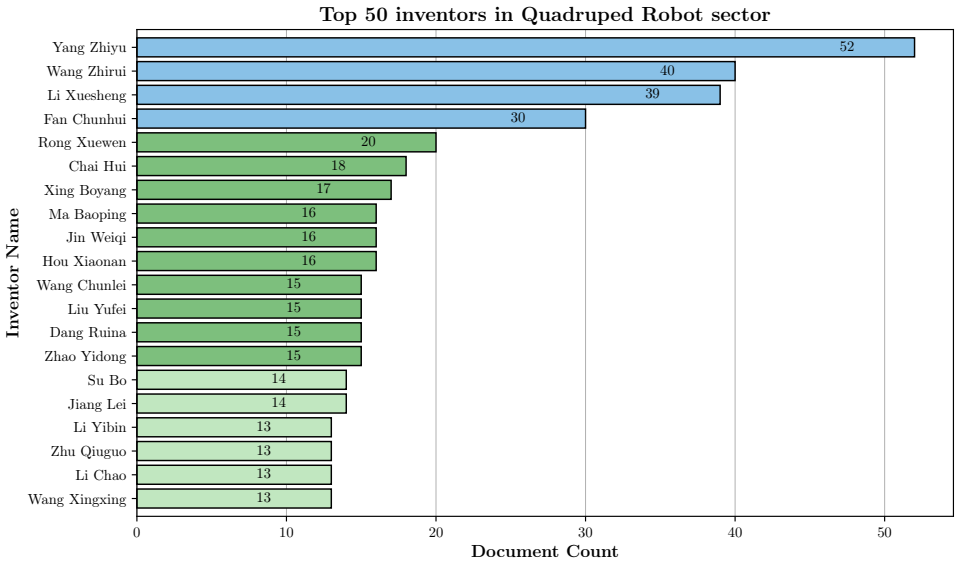


**Figure 2.8:** *Horizontal bar chart with the fifteen most relevant inventors in the sector.*

To conclude, regarding the quadruped robot sector, we can state that **it is a booming technology**. Since its appearance at the beginning of the 21st century, it has grown slightly until the year 2021, when the number of patents has tripled and has remained at that rate ever since. In the almost 25 years since the 2000s, the sector has been

consolidating and is currently growing rapidly. In particular, **China leads this field of research and innovation, accompanied by renowned entities such as *Boston Dynamics* and *Google***. The difference, however, between the number of patents in China and the rest of the world is overwhelmingly different. Europe needs to think about that.

## 2.3 Quadruped Robots: history and hardware platforms

In addition to analyzing trends in quadruped robotics through databases and patent records, it is essential to contextualize our study within the history of these robots. While many previous studies have explored this subject in greater depth than this report, in this section we provide a short but concise overview of the historical evolution of quadruped robotics up to the present day. For a more comprehensive understanding and analysis of the topic, please consult any of the following sources:

- A **detailed state-of-the-art review** with a focus on structural and hardware improvements:

  P. Biswal and P. K. Mohanty, "Development of quadruped walking robots: A review", *Ain Shams Engineering Journal*, vol. 12, no. 2, pp. 2017-2031, 2021. doi: 10.1016/j.asej.2020.11.005.

- A **comprehensive review of previous studies** in the field of quadrupedal mobile robots (QMR). It collects, compiles and synthesizes the different topologies studied, the actuation mechanisms used, the kinematic and dynamic studies, the gait analysis, the control algorithms and more. It updates the history of quadruped robotics with the latest developments. And finally, it analyzes the future trends of the sector:

  H. Taheri and N. Mozayani, "A study on quadruped mobile robots", *Mechanism and Machine Theory*, vol. 190, 2023, Art. no. 105448. doi: 10.1016/j.mechmachtheory.2023.105448.

- An **insightful video lecture on the history of walking robots**. It analyzes the methods used in the past and those used today, showing where possible their use on real platforms:

  S. N. Y. Kolathya, "History of Walking Robots", *International Center for Theoretical Sciences, Bengaluru (India)*, YouTube, 30 Oct. 2022. [Online]. Available: https://www.youtube.com/live/1G6cWc_KvOE?feature=shared.

Furthermore, we include a subsection dedicated to hardware platforms (exemplified by the *Aliengo* robot from *Unitree*), highlighting its morphology, key characteristics, and nominal capabilities.

### 2.3.1 History

The history of quadruped robots dates back to the late 19th and early 20th centuries, when the first walking mechanisms appeared, purely mechanical and without any type of algorithm controlling them. The "mechanical horse" by Rygg, created in 1893 (refer to Fig. 2.9.a), is well-known for converting the rider's pedaling into the horse's leg movements via pedals [33]. However, it wasn't until the 1960s that the first quadruped robot in history was built, named "Phony Pony" (see Fig. 2.9.b), developed by the University of Southern California [34].

In 1965, the established company *General Electric* wanted to test this emerging technology and commissioned the construction of a manned hydraulic quadruped weighing over a ton [35], as can be seen in Figure 2.9.c. Following this, Japanese professors Hirose and Kato introduced the *KUMO* and *TITAN* series (Figures 2.9.d and 2.9.e respectively), the first generation of mobile quadruped robots to incorporate advanced control systems and successfully climb stairs for the first time in history [36]. In the early 1980s, Raibert and his MIT team sparked a robotics revolution by developing planar and three-dimensional machines that could run and hop like a one-legged kangaroo. Their patented balance and dynamics principle outperformed all legged systems, surpassing even biological robots, and set a landmark for dynamic stability in four-legged robot motion control [33] (refer to Fig. 2.9.f).

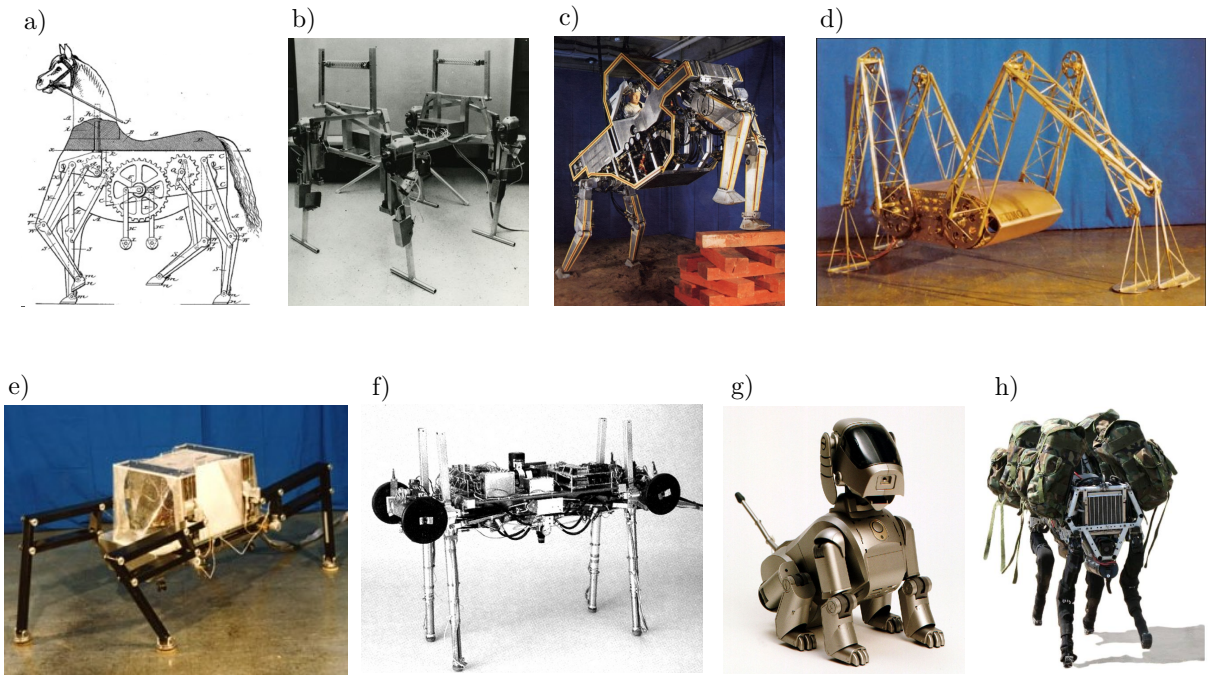


**Figure 2.9:** *Historical quadruped robots: (a) Rygg's mechanical horse. (b) Phony Pony. (c) General Electric walking truck. (d) KUMO-I. (e) TITAN-III. (f) Raibert's quadruped robot (MIT). (g) AIBO quadruped robot from Sony. (h) Big Dog from Boston Dynamics.*

After several generations of new quadruped robots, the late 1990s saw the appearance of Sony's *AIBO* series, the first commercial model, opening a new door for experimentation by software developers (see Fig. 2.9.g). In 2010, a significant milestone achieved by *Boston Dynamics* (BD) revolutionized the sector by using its hydraulic *BigDog* in military

applications (refer to Fig. 2.9.h) and presenting new models that exhibited unprecedented movements for legged robots.

Both the Spot robot, manufactured by BD (see Fig. 2.10.a), and *ANYmal* by *ANYbotics*, a spin-off from *ETH Zurich* (refer to Fig. 2.10.b), deserve mention. These robots have begun to facilitate the transition from laboratories to the industrial and service sectors. Since then, many companies and research centers have continued to invest in these robots, aiming to make them more affordable and accessible, as well as advancing their motor capabilities.

Asian companies such as *Unitree*, *DeepRobotics*, and *Xiaomi* (Figures 2.10.c, 2.10.d and 2.10.e respectively), along with the American company *Ghost Robotics* (see Fig. 2.10.f), are the most competitive and driven to push these robots to their limits, some even in the military field. Other research platforms, such as *HyQReal* (refer to Fig. 2.10.g) from the *Istituto Italiano di Tecnologia* (IIT) and the *Cheetah* and *Mini Cheetah* from the MIT (see Fig. 2.10.h), have also recently emerged as promising real-world hardware robots, moving these leading research centers up at the forefront of innovation.

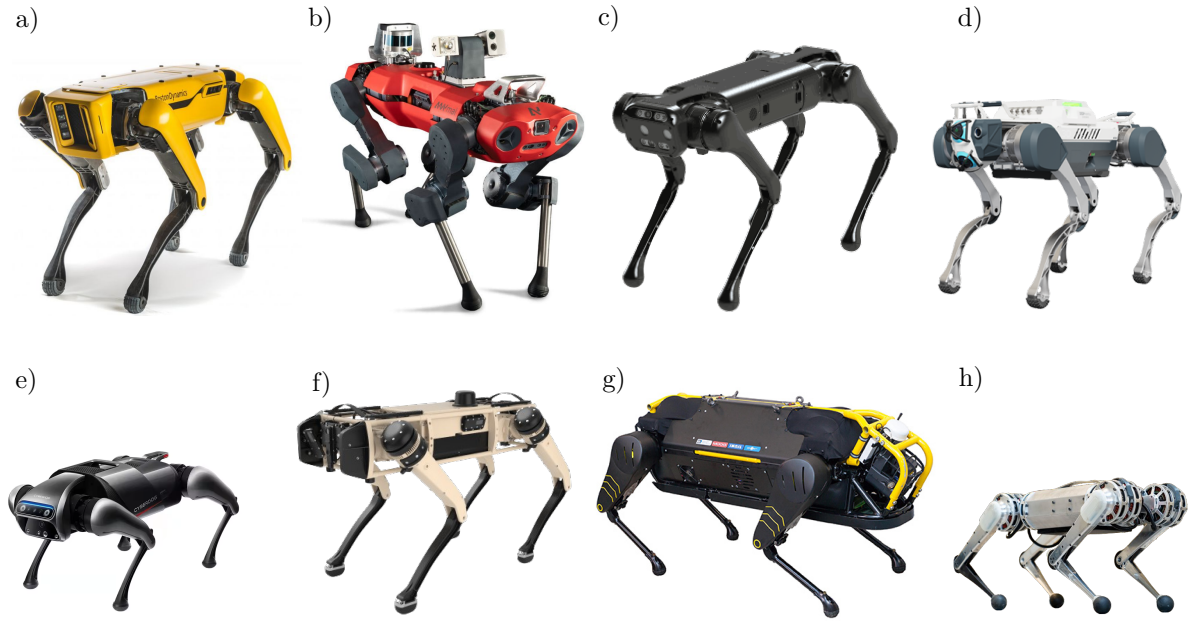


**Figure 2.10:** *Today's robots: (a) Spot from Boston Dynamics. (b) ANYmal from ANYbotics. (c) Aliengo from Unitree. (d) X30 from Deep Robotics. (e) Cyberdog from Xiaomi. (f) Vision 60 from Ghost Robotics. (g) HyQReal from IIT. (h) Mini Cheetah from MIT.*

### 2.3.2 Hardware Platforms

Figure 2.10 illustrates that the morphology of modern quadruped robots is generally consistent, typically exhibiting a *half-symmetric* design, as defined by [37]. The key differences among these robots lie in their size, weight, and the technology used. For instance, some manufacturers utilize drive belts to move the calf joint, while others opt for direct connection to an electric motor or employ parallel mechanisms.

In the following paragraphs, we will outline the fundamental terminology related to

quadruped morphology, highlight the basic characteristics of interest from a research perspective, and discuss the primary capabilities that define these robots. The *Aliengo* quadruped robot, developed by *Unitree*, will serve as our reference platform, as it is the hardware used in our subsequent experiments. Nonetheless, the concepts discussed can be generalized to other quadrupeds, where specific values may differ, but the overall data structure remains consistent.

- **Morphology:** As illustrated in Figure 2.11, each leg of the quadruped has three Degrees of Freedom (DoF), resulting in a total of 12 DoFs for the entire robot. The end effector of the foot is referred to as the *calf*, the middle section of the leg as the *thigh*, and the uppermost part as the *hip*. We differentiate the limbs based on their position, distinguishing between the front and rear, as well as the left and right sides, giving us four limbs: Front Left (FL), Front Right (FR), Rear Left (RL), and Rear Right (RR). The figure also depicts the world frame, denoted as $\{W\}$, and the base frame, denoted as $\{B\}$, which corresponds to the robot's body.

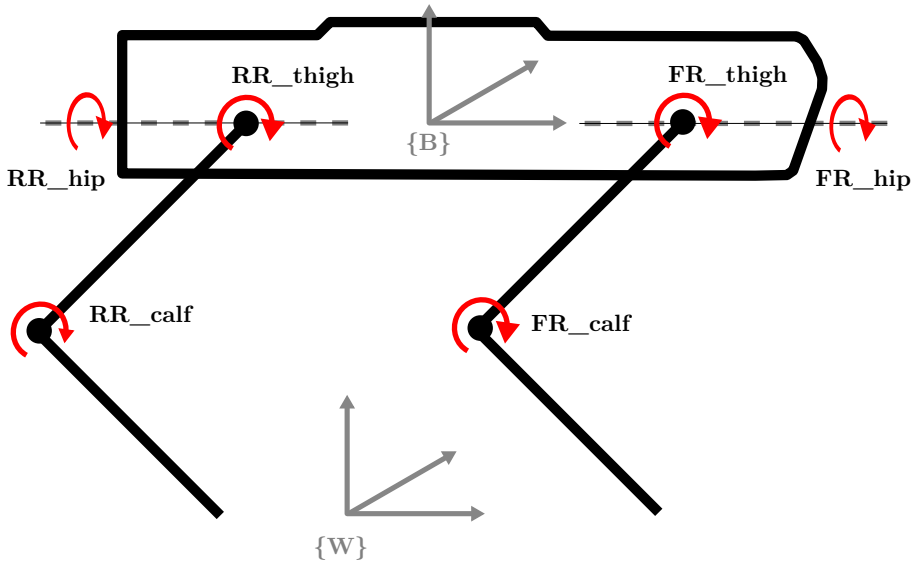


**Figure 2.11:** *Basic terminology for the Aliengo joints. The figure highlights the right side of the robot, with the left side being analogous.* $\{W\}$ *denotes the world reference frame, while* $\{B\}$ *represents the body frame.*

- **Characteristics:** From a simulation perspective, it is essential to measure all the physical properties of the robot. These values are crucial not only for creating realistic simulations but also for the training process. The performance of the real robot will directly depend on how closely the simulated robot matches reality. For this reason, as we will discuss later, it is common practice to randomize the friction coefficient, apply forces to the robot, or add random weight to its nominal mass during training. These techniques help bridge the gap between simulation and real-world performance.

  Table 2.2 summarizes the key parameters of the Aliengo robot. Some of these values were sourced from the official Unitree GitHub [38], where the URDF[3] files for all

[3]URDF stands for Unified Robot Description Format, an XML file used in robotics to describe a robot's physical and visual characteristics.

their robots are available. In addition to the characteristics highlighted in Table 2.2, further parameters can be found in the URDF file, which are also listed in Table A.1 in Appendix A.

| Parameter | Symbol | Value | Units |
|---|---|---|---|
| Mass | $m$ | 22 | kg |
| Body Inertia | $I_{xx}$ | 0.033 | kg·m$^2$ |
| | $I_{yy}$ | 0.16 | kg·m$^2$ |
| | $I_{zz}$ | 0.17 | kg·m$^2$ |
| Body Length | $l_{body}$ | 0.647 | m |
| Body Width | $w_{body}$ | 0.31 | m |
| Body Height | $h_{body}$ | 0.15 | m |
| Leg Link Lengths | $l_1$ | 0.25 | m |
| | $l_2$ | 0.25 | m |

**Table 2.2:** *Aliengo physical robot parameters.*

- **Capabilities:** From a functional perspective, it is crucial to assess the robot's capabilities as specified by the manufacturer. Some of these, such as the maximum motor torque, represent hard limits that cannot be exceeded without risking damage to the robot. However, other constraints, like those related to the manufacturer-provided controllers, may be less rigid and could potentially be surpassed through improved control systems or algorithms. The table below summarizes the nominal capabilities of the Aliengo robot.

| Parameter | Value | Units |
|---|---|---|
| Payload | 12 | kg |
| Max Speed | 1.67 | m/s |
| Obstacle Passing | $< 18$ | cm |
| Ground Clearance | $\approx 35$ | cm |
| Climbing Ability | 25° to 30° | degrees |
| Turning Radius | 0 | - |
| Maximum Torque | 40 | Nm |
| Maximum Joint Velocity | 26.5 | rad/s |
| Joint Reduction Ratio | $\approx 10$ | - |
| Motor Encoder Resolution | 15 | bit |
| Endurance Time: | | |
| - Normal life | 210 | mins |
| - Keep walking | 120 | mins |
| - Standing still | 270 | mins |
| - Sleep mode | 320 | mins |

**Table 2.3:** *Aliengo nominal capabilities.*

As previously noted, the robot's *obstacle passing* and *climbing ability* can be enhanced, as demonstrated in recent studies like [39]. Achieving these improvements involves not only refining the training parameters but also rethinking the training process itself, particularly in the context of reinforcement learning (RL), as illustrated by [40].

## 2.4 Reinforcement Learning

Given the central role of reinforcement learning (RL) in this study, this section provides a concise introduction adapted to the general reader. It outlines the basic components and operational framework of RL. The section then delves into the fundamental equations that underlie RL. Since Proximal Policy Optimization (PPO) is the primary algorithm employed in this research, the final part of the section focuses on PPO, with brief references to alternative algorithms. The following sources have been consulted for this section:

- *(primarily)*: A concise introduction to RL terminology and foundational equations:

  J. Achiam, "Spinning Up in Deep Reinforcement Learning", *2018*. Available: https://github.com/openai/spinningup.

- *(briefly)*: A seminar series of eight videos providing a high-level overview of reinforcement learning, featuring key algorithms and notable applications.

  S. L. Brunton, "Machine Learning Meets Control Theory", *Reinforcement Learning*, Cassyni, 2021. Available: https://doi.org/10.52843/cassyni.x2t0sp.

### 2.4.1 Introduction

In Artificial Intelligence (AI), Machine Learning (ML) focuses on developing algorithms that autonomously improve through experience [41]. Among the three main types of ML (Supervised, Unsupervised, and Reinforcement), the latter stands out for its biologically inspired approach, where the agent learns through trial and error. This method is particularly effective at addressing the computational challenges associated with achieving long-term goals, as it allows the agent to continuously refine its strategies based on feedback from its interactions.

In the context of robotics, particularly in quadrupedal locomotion, the high-dimensional range of possible situations (or states) to consider during training makes RL the preferred methodology. Additionally, the typical heterogeneous observations of these states necessitate the use of Neural Networks. These networks, serving as general function estimators, have recently proven to be a promising solution when combined with Deep Learning (DL) techniques [42]. The integration of DL with RL is commonly referred to as Deep Reinforcement Learning (DRL).

### 2.4.2 Key Concepts and Terminology

The primary components of Reinforcement Learning (as can be seen in Figure 2.12) are the **agent** and the **environment**. The environment represents the world in which the agent exists and with which it interacts. During each interaction, the agent observes a (potentially incomplete) view of the world's current state and selects an action to perform. The environment then undergoes changes either in response to the agent's action or independently.

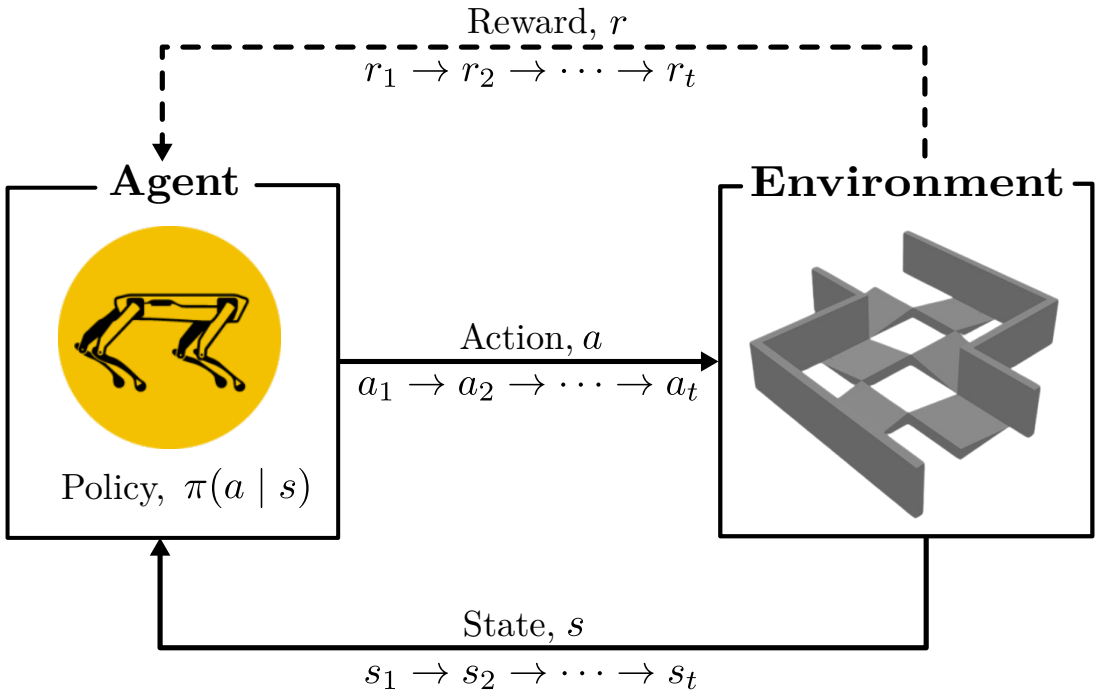


**Figure 2.12:** *RL General Framework. The agent (e.g., a quadruped robot) governed by a policy ($\pi$) interacts with the environment. The robot receives observations from the environment, referred to as the state ($s$) if they represent the full context of the world, and selects the next action ($a$) to perform. Based on this action, the agent may receive a reward ($r$), which is often sparse, reflected by the dashed arrow. This process is repeated at each time step ($t$) throughout the simulation. Main diagram inspired by [43]. Spot icon sourced from [44].*

The agent also receives a reward signal from the environment, which provides feedback on how favorable or unfavorable the current state of the world is. The agent's objective is to maximize the cumulative reward, known as the return. Reinforcement Learning techniques are strategies that the agent can use to learn behaviors that help it achieve this objective.

***States and Observations:***

A **state** ($s$) from the set of all valid states ($\mathcal{S}$) provides a full representation of the world, containing all relevant information. In contrast, an **observation** ($o$) offers a partial view of a state, potentially leaving out certain details. In robotics, for example, a visual observation might be captured as an RGB matrix corresponding to pixel values, while a robot's state could be expressed through its joint angles and velocities.

When an agent has access to the entire state of the system, the scenario is described as **fully observed**. Conversely, if the agent only has access to a limited observation, the scenario is referred to as **partially observed**.

***Action Spaces:***

Various environments permit different types of actions. The collection of all possible actions within a specific environment is commonly referred to as the **action space** ($\mathcal{A}$). Certain environments, such as board games, feature **discrete action spaces**, where the agent has a limited set of moves to choose from. In contrast, environments where an agent

controls a robot in the physical world have **continuous action spaces**, where actions are represented as real-valued vectors.

***Markov Decision Processes:***

Up to this point, we have described the agent's environment in a more informal and intuitive manner. However, in the realm of academic literature, this interaction is often formalized using a standard mathematical model known as a **Markov Decision Process** (MDP). An MDP is a framework that encapsulates the key elements of decision-making in an environment, and it is characterized by a 5-tuple $\langle S, A, R, P, \rho_0 \rangle$, where:

- $S$ is the set of all possible states the environment can be in,
- $A$ is the set of all actions available to the agent,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, which provides a reward $r_t = R(s_t, a_t, s_{t+1})$ when the agent transitions from state $s_t$ to state $s_{t+1}$ after taking action $a_t$,
- $P : S \times A \to \mathcal{P}(S)$ is the state transition probability function, where $P(s'|s, a)$ represents the probability of transitioning to state $s'$ given that the agent is in state $s$ and takes action $a$,
- $\rho_0$ is the initial state distribution, describing the probability distribution over the states at the start of the process.

The concept of an MDP is fundamental because it assumes the system adheres to the **Markov property**, meaning that the next state and the resulting reward depend solely on the current state and action, with no influence from prior history. This assumption simplifies the decision-making process, allowing the agent to focus on the immediate present when choosing actions, rather than considering the entire sequence of past states and actions.

However, as we've mentioned, in many cases, accessing the full state of the system is not feasible. Instead, the agent receives observations that only partially reveal the state. To model such scenarios, we use a **Partially Observable Markov Decision Process** (POMDP), which generalizes the MDP to account for situations where the agent cannot directly observe the true state of the environment. In a POMDP, the agent relies on these observations to infer information about the state, and the framework is defined by a 7-tuple $\langle S, A, R, P, \Omega, O, \rho_0 \rangle$, where the new two terms are:

- $\Omega$ is the set of possible observations the agent can receive.
- $O : S \times A \to \mathcal{P}(\Omega)$ is the observation probability function, with $O(o|s', a)$ being the probability of observing $o$ given that the agent took action $a$ and arrived at state $s'$.

In POMDP the agent must make decisions based on a belief, which is a probability distribution over possible states, updated as new observations are made. The POMDP framework is crucial for modeling decision-making problems in environments where information is incomplete or uncertain, such as in robot locomotion.

***Policies:***

A **policy** is a guideline followed by an agent to determine the actions it should take. Policies can be **deterministic**, typically represented by $\mu$:

$$a_t = \mu(s_t),$$

or they can be **stochastic**, often denoted by $\pi$:

$$a_t \sim \pi(\cdot|s_t).$$

Since a policy effectively acts as the agent's decision-making mechanism, the terms "policy" and "agent" are sometimes used interchangeably, such as in the phrase "The policy aims to maximize reward."

In DRL, it is usual to work with **parameterized policies**: these are policies whose outputs are functions based on a set of parameters (for example, the weights and biases in a neural network) that can be modified through an optimization algorithm to adjust the agent's behavior.

We typically denote the parameters of such a policy by $\theta$ or $\phi$, and write this as a subscript to the policy symbol to indicate the relationship:

$$a_t = \mu_\theta(s_t)$$
$$a_t \sim \pi_\theta(\cdot|s_t)$$

As we are working with non-deterministic problems, we will focus on **stochastic policies**. In DRL, the two most prevalent types of stochastic policies are **categorical policies** and **diagonal Gaussian policies**. Categorical policies are applicable in **discrete action spaces**, whereas diagonal Gaussian policies are suited for **continuous action spaces**.

There are two critical computations that are essential when working with and training stochastic policies:

- Sampling actions from the policy,
- and computing the log-likelihood of specific actions, $\log \pi_\theta(a|s)$.

We will now explain the procedures for both categorical and diagonal Gaussian policies:

1. **Categorical Policies**: A categorical policy functions similarly to a classifier that selects from discrete actions. To construct the neural network for a categorical policy, you would proceed in the same way as you would for a classifier: start with the observation as input, pass it through a series of layers (which might include convolutional or densely-connected layers, depending on the input type), and finish with a final linear layer that outputs *logits*[4] for each possible action. These logits

[4] *Logits* refer to the raw, unnormalized predictions generated by the last layer of a neural network before applying an activation function.

are then passed through a *softmax*[5] function to transform them into probabilities.

(a) **Sampling**: Once you have the action probabilities, you can use built-in functions in frameworks like PyTorch or TensorFlow to sample actions.

(b) **Log-Likelihood**: Let's denote the output layer of probabilities as $P_\theta(s)$. This vector has as many entries as there are possible actions, allowing us to use the action as an index. The log likelihood for a specific action $a$ can then be calculated by indexing into this vector:

$$\log \pi_\theta(a \mid s) = \log \left[P_\theta(s)\right]_a .$$

2. **Diagonal Gaussian Policies**: A multivariate Gaussian distribution (also known as a multivariate normal distribution) is characterized by a mean vector, $\mu$, and a covariance matrix, $\Sigma$. In the special case of a diagonal Gaussian distribution, the covariance matrix has non-zero entries only on its diagonal, allowing it to be represented as a vector.

   In a diagonal Gaussian policy, a neural network is used to map observations to the mean actions, $\mu_\theta(s)$. There are two common approaches to representing the covariance matrix in this context:

   - **First approach**: The log standard deviations, $\log \sigma$, are represented by a single vector that is independent of the state; these $\log \sigma$ values are standalone parameters.
   - **Second approach**: A neural network maps states to log standard deviations, $\log \sigma_\theta(s)$. This network might share some layers with the network that produces the mean.

   In both methods, log standard deviations are outputted instead of standard deviations directly. The reason is that log standard deviations can freely take values in $(-\infty, \infty)$, while standard deviations must be nonnegative. Training is simplified without the need to enforce non-negativity constraints. Standard deviations can be easily obtained by exponentiating the log standard deviations, so nothing is lost by this representation.

   (a) **Sampling**: Given the mean action $\mu_\theta(s)$ and the standard deviation $\sigma_\theta(s)$, along with a noise vector $z$ sampled from a spherical Gaussian distribution ($z \sim \mathcal{N}(0, I)$), an action can be sampled as follows:

   $$a = \mu_\theta(s) + \sigma_\theta(s) \odot z,$$

   where $\odot$ denotes the elementwise product of two vectors.

   (b) **Log-Likelihood**: For a $k$-dimensional action $a$, the log-likelihood under a diagonal Gaussian distribution with mean $\mu = \mu_\theta(s)$ and standard deviation $\sigma = \sigma_\theta(s)$ is given by:

   $$\log \pi_\theta(a \mid s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2\log \sigma_i\right) + k \log 2\pi\right).$$

[5]The *softmax* function transforms a vector of $K$ real numbers into a probability distribution across $K$ possible outcomes. It is commonly used as the final activation function in a neural network to normalize the network's output into a probability distribution over the predicted classes. Specifically, the probability of action $j$ in a categorical distribution with logits $x_j$ is given by $p_j = \exp(x_j)/\sum_i \exp(x_i)$

***Episodes:***

An **episode** $\tau$ (often referred to as trajectory or rollouts) represents a sequence of states and actions in an environment:

$$\tau = (s_0, a_0, s_1, a_1, \dots).$$

The initial state of the environment, $s_0$, is sampled from a distribution over possible start states, often denoted by $\rho_0$:

$$s_0 \sim \rho_0(\cdot).$$

The transitions between states (i.e., how the environment evolves from state $s_t$ at time $t$ to state $s_{t+1}$ at time $t+1$) are determined by the dynamics of the environment and depend solely on the most recent action, $a_t$. These transitions can be deterministic:

$$s_{t+1} = f(s_t, a_t),$$

or they can be stochastic:

$$s_{t+1} \sim P(\cdot \mid s_t, a_t).$$

Actions are selected by the agent according to its policy.

***Return and Reward:***

The **reward** function $R$ plays a crucial role in RL. It is typically defined based on the current state of the environment, the action that has just been taken, and the resulting next state:

$$r_t = R(s_t, a_t, s_{t+1})$$

However, it is often simplified to depend solely on the current state, $r_t = R(s_t)$, or on the state-action pair, $r_t = R(s_t, a_t)$.

The agent's objective is to maximize some form of cumulative reward over an episode, though this can be interpreted in different ways. We will denote all these cases by $R(\tau)$, and the context will usually clarify which specific case is intended, or it might not matter since the same equations generally apply.

One type of return is the **finite-horizon undiscounted return**, which represents the sum of rewards accumulated within a fixed number of steps:

$$R(\tau) = \sum_{t=0}^{T} r_t.$$

Another type of return is the **infinite-horizon discounted return**, which sums all rewards the agent ever receives, but applies a discount based on how far into the future the rewards are obtained. This approach introduces a **discount factor** $\gamma \in (0, 1)$:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t.$$

The discount factor is both conceptually appealing and mathematically useful. Intuitively, receiving a reward now is better than receiving it later. Mathematically, summing an

infinite series of rewards may not converge to a finite value and can be difficult to handle in equations. However, with a discount factor and under reasonable conditions, the infinite sum converges.

In deep RL, the distinction between finite-horizon and infinite-horizon returns often becomes blurred. For example, we may optimize for undiscounted returns while still applying discount factors when estimating value functions.

One of the central challenges in RL is the "credit assignment problem", first identified by Marvin Minsky in the 1960s [45]. Given that rewards are sparse and infrequent, it is often difficult to determine which specific actions led to the received reward. This issue complicates the learning process, requiring extensive data and numerous trials to identify effective action sequences.

Dense rewards facilitate faster learning, similar to playing with a more knowledgeable opponent, while sparse rewards are less informative, necessitating many examples to learn an optimal policy. To address the challenge of sparse rewards, practitioners often employ "reward shaping", where intermediate rewards are introduced to guide the agent toward the final goal.

### 2.4.3 The RL Problem

Regardless of whether we choose to measure return using infinite-horizon discounted rewards or finite-horizon undiscounted rewards, the ultimate goal in reinforcement learning is to identify a policy that maximizes the expected return when the agent follows it.

To discuss expected return, we first need to consider the probability distributions over possible episodes.

Assume that both the environment's transitions and the agent's policy are stochastic. In this scenario, the probability of a trajectory spanning $T$ steps is given by:

$$P(\tau \mid \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t)\pi(a_t \mid s_t).$$

The **expected return**, regardless of the specific return measure, is denoted by $J(\pi)$ and is calculated as:

$$J(\pi) = \int_{\tau} P(\tau \mid \pi)R(\tau) = \underset{\tau\sim\pi}{\mathbb{E}}[R(\tau)].$$

The main optimization task in reinforcement learning is then to find the policy $\pi^*$ that maximizes $J(\pi)$, expressed as:

$$\pi^* = \arg\max_{\pi} J(\pi),$$

where $\pi^*$ represents the optimal policy.

### 2.4.4 Value Functions

In reinforcement learning, it is often important to understand the value of a state or a state-action pair. The "value" refers to the expected return when starting from that state or state-action pair and then following a specific policy indefinitely. Value functions are a fundamental component of nearly every RL algorithm.

There are four key functions to consider:

- **On-Policy Value Function, $V^\pi(s)$:** the expected return is provided by this function if you start in state $s$ and always follow policy $\pi$:

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s\right]$$

- **On-Policy Action-Value Function, $Q^\pi(s,a)$:** This function represents the expected return when starting in state $s$, taking a specific action $a$ (which may not follow the policy), and then continuing to follow policy $\pi$ thereafter:

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s, a_0 = a\right]$$

- **Optimal Value Function, $V^*(s)$:** This function denotes the expected return when starting in state $s$ and consistently following the optimal policy:

$$V^*(s) = \max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s\right]$$

- **Optimal Action-Value Function, $Q^*(s,a)$:** This function represents the expected return when starting in state $s$, taking any action $a$, and then following the optimal policy from that point onward:

$$Q^*(s,a) = \max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s, a_0 = a\right]$$

Although most RL methods revolve around estimating value functions, it is not mandatory for solving RL problems. Alternative approaches, such as *genetic algorithms* or *simulated annealing*, explore static policies over time without estimating value functions. However, RL methods that learn from interaction with the environment often outperform evolutionary methods in efficiency and effectiveness.

Before continuing, it may be relevant to review the following notes:

1. When value functions are discussed without reference to time, they generally refer to the expected return over an infinite horizon with discounted rewards. In contrast, value functions for finite-horizon, undiscounted returns must include time as a parameter.
2. Two key relationships between the value function and the action-value function are often encountered:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}\left[Q^\pi(s,a)\right],$$

and

$$V^*(s) = \max_a Q^*(s, a).$$

These relationships follow directly from the definitions.

3. In policy optimization, we use the *arg max* operator, which is distinct from the *max* operator used in value functions. Here's the difference:

   - The argmax is defined as:

     $$\arg\max_x f(x) = \{x : f(x) \geq f(y)\, \forall y \in X\},$$

     where $X$ is the domain of possible values for $x$. This is the set of points $x$ where $f(x)$ reaches its maximum value. The set can be empty, a single point, or multiple points.
   - The max operator returns the maximum value of the function itself:

     $$\max_x f(x) = \{f(x) : f(x) \geq f(y)\, \forall y \in X\}.$$

     Unlike argmax, max cannot have multiple values.

   A key identity linking argmax and max is:

   $$f\left(\arg\max_x f(x)\right) = \max_x f(x).$$

### 2.4.5 The Optimal Q-Function and the Optimal Action:

There exists a fundamental relationship between the optimal action-value function, denoted as $Q^*(s, a)$, and the selection of actions made by the optimal policy. The optimal action-value function $Q^*(s, a)$, by definition, quantifies the expected cumulative return when the agent begins in a given state $s$, takes a specific action $a$, and subsequently adheres to the optimal policy for all future time steps. In essence, $Q^*(s, a)$ captures the long-term value of executing action $a$ in state $s$, while assuming that the most effective strategy is followed from that point onward. This connection is crucial because the optimal policy selects actions based on maximizing $Q^*(s, a)$, ensuring that the best possible outcome is achieved in terms of expected returns.

Therefore, having $Q^*$, the optimal action, $a^*(s)$ can be directly determined by:

$$a^*(s) = \arg\max_a Q^*(s, a).$$

It is important to note that several actions could potentially maximize $Q^*(s, a)$, meaning that each of these actions would be considered optimal. In situations like this, the optimal policy may choose randomly from among these actions. However, there will always be at least one optimal policy that selects a specific action deterministically.

### 2.4.6 Bellman Equations

All four value functions satisfy specific self-consistency equations known as Bellman equations. The core concept behind these equations is:

> "The value at your starting state is the reward you expect to receive there, plus the value of the next state you transition to" [46].

For on-policy value functions, the Bellman equations are:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[ r(s,a) + \gamma V^{\pi}(s') \right],$$

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s'\sim P} \left[ r(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi} \left[ Q^{\pi}(s',a') \right] \right],$$

where $s' \sim P$ denotes sampling the next state $s'$ according to the environment's transition dynamics $P(\cdot \mid s,a)$, and $a \sim \pi$ and $a' \sim \pi$ indicate sampling actions based on the policy $\pi$ given the states $s$ and $s'$, respectively.

For the optimal value functions, the Bellman equations are:

$$V^{*}(s) = \max_{a} \mathop{\mathbb{E}}_{s'\sim P} \left[ r(s,a) + \gamma V^{*}(s') \right],$$

$$Q^{*}(s,a) = \mathop{\mathbb{E}}_{s'\sim P} \left[ r(s,a) + \gamma \max_{a'} Q^{*}(s',a') \right].$$

The key distinction between the Bellman equations for on-policy value functions and those for optimal value functions lies in the presence of the max over actions. This max indicates that to act optimally, the agent must choose the action that maximizes the expected value whenever it has the opportunity to make a decision.

### 2.4.7 Advantage Functions

In reinforcement learning, we don't always need to assess how good an action is in absolute terms; sometimes, it is more useful to understand how much better a particular action is compared to the average action. This relative measure is captured by the advantage function.

The advantage function $A^{\pi}(s,a)$ for a given policy $\pi$ quantifies how much better it is to take a specific action $a$ in state $s$ compared to simply choosing an action according to the policy $\pi(\cdot \mid s)$, assuming that the policy $\pi$ is followed thereafter. Formally, the advantage function is defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s).$$

Beware that the advantage function plays a critical role in policy gradient methods.

### 2.4.8 RL Algorithms

Having established the fundamental terminology and notation in RL, we can now delve into a more advanced exploration of the field. Here we provide an overview of the landscape of modern RL algorithms and discusses the critical trade-offs involved in algorithm design.

#### *A Taxonomy of RL Algorithms*

Categorizing RL algorithms is challenging due to their modular nature, which makes it difficult to fit them into a single taxonomy. Figure 2.13 presents a broad, though not exhaustive, diagram of modern RL algorithms. The Proximal Policy Optimization (PPO) algorithm, highlighted in red, is one of the most popular approaches and is the focus of the current master's thesis research. Other relevant topics for RL algorithms, such as exploration strategies, transfer learning, and meta-learning, are important but will not be covered in this document for the sake of simplicity.

In RL, one of the key decisions in algorithm design is whether the agent **should have access to, or learn, a model of the environment**. A model, in this context, refers to a function that predicts state transitions and rewards based on the agent's actions.

The advantage of using a model is that it enables the agent to plan by simulating future scenarios and making informed decisions based on the potential outcomes of different actions. This ability to "think ahead" can significantly improve sample efficiency, as the agent can explore multiple options mentally before taking actual actions, potentially leading to faster and more effective learning.

However, the downside is that a true model of the environment is rarely available. When an agent must learn the model from scratch through experience, it faces several challenges. The most significant is the risk of model bias, where inaccuracies in the learned model can mislead the agent, causing it to perform well according to the model but poorly in the real environment. Learning an accurate model is inherently difficult, and even with substantial computational resources and time, success is not guaranteed.

Algorithms that utilize a model are known as **model-based methods**, while those that do not are referred to as **model-free methods**. Although model-free methods miss out on the potential efficiency gains from using a model, they are generally simpler to implement and more robust in practice.

Another crucial decision in designing an RL algorithm is determining **what the agent should learn**. Common options include learning policies (which can be stochastic or deterministic), action-value functions (Q-functions), value functions, or even models of the environment.

#### *Model-Free RL:*

In the context of model-free RL, one of the most critical decisions is determining what the agent should learn. This choice significantly influences the design and effectiveness of the algorithm. There are two primary approaches:

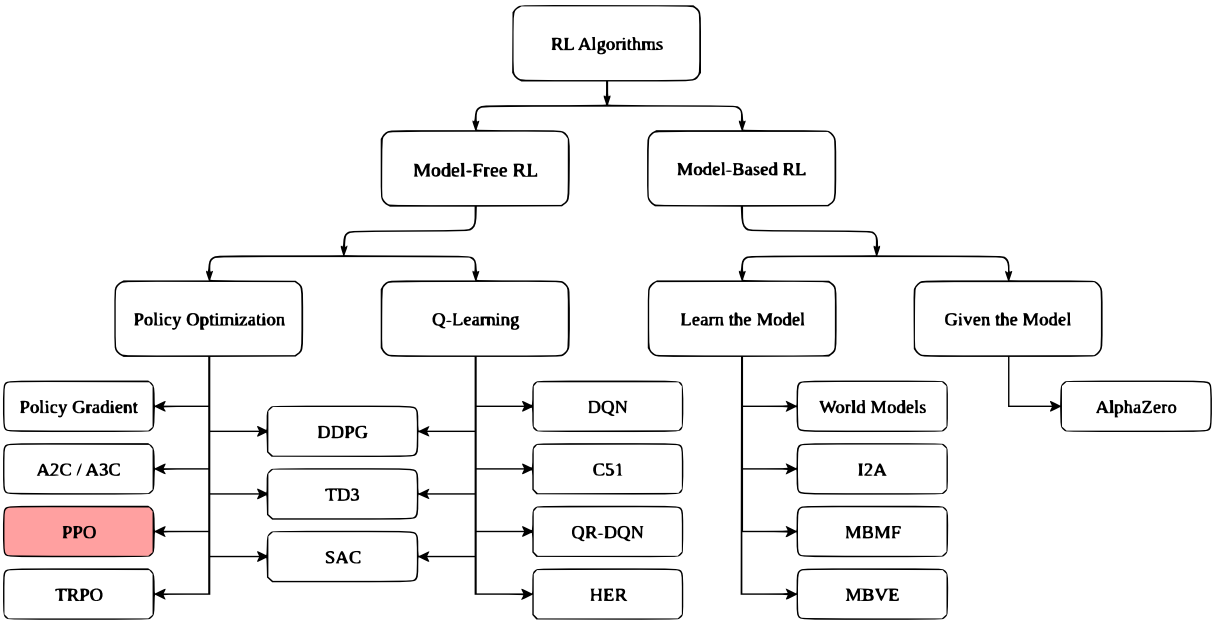


**Figure 2.13:** *A practical and broad classification of modern RL algorithms. The Proximal Policy Optimization (PPO) algorithm is highlighted due to its significance in this thesis. Image modified from[46].*

1. **Policy Optimization:** Policy optimization methods focus on directly learning a policy, typically represented as $\pi_\theta(a \mid s)$, where $\theta$ are the parameters of the policy. These methods aim to optimize $\theta$ either by directly maximizing the performance objective $J(\pi_\theta)$ through gradient ascent or by maximizing local approximations of $J(\pi_\theta)$. This optimization is generally performed on-policy, meaning the agent updates its policy using data collected under the most recent version of that policy. Additionally, policy optimization often involves learning an approximator $V_\phi(s)$ for the value function $V^\pi(s)$, which helps guide the policy updates.

    - **Proximal Policy Optimization (PPO)** is a standout example of policy optimization. PPO optimizes performance by maximizing a surrogate objective function, providing a more conservative and stable update compared to other methods like A2C/A3C, which directly apply gradient ascent. PPO's balance of stability and performance has made it one of the most widely used algorithms in modern RL.

2. **Q-Learning:** Q-learning methods focus on learning an approximator $Q_\theta(s, a)$ for the optimal action-value function $Q^*(s, a)$. This approach typically uses an objective function based on the Bellman equation and is performed off-policy, meaning it can leverage data collected at any point during training, regardless of the current policy. The policy is derived from the Q-function by selecting actions that maximize $Q_\theta(s, a)$, i.e., $a(s) = \arg\max_a Q_\theta(s, a)$. Examples include DQN, which popularized deep RL, and C51, which extends Q-learning to model the distribution of returns.

Policy optimization methods, like PPO, are advantageous because they directly optimize for the desired outcome, leading to stable and reliable learning. However, they tend to be less sample-efficient compared to Q-learning methods, which, when they work, can reuse data more effectively and achieve higher sample efficiency. The trade-off is

that Q-learning is more prone to instability due to its indirect approach to optimizing agent performance, particularly when dealing with complex environments or when using function approximation.

***Model-Based RL:***

Model-based RL introduces a different set of considerations, as it involves learning or using a model of the environment. Unlike model-free methods, which have clearer categories, model-based methods vary widely in their approach. Here, we briefly comment on some common strategies:

- **Pure Planning:** The most straightforward model-based approach involves using **Model-Predictive Control** (MPC) to plan actions without explicitly learning a policy. The agent computes an optimal plan based on the current model and executes the first action, discarding the rest of the plan as it re-evaluates at each step. This approach is used in tasks where precise control and planning over short horizons are critical.
- **Expert Iteration:** This method involves using a planning algorithm to generate better actions than the current policy could, effectively acting as an "expert". The policy is then updated to mimic the expert's actions, gradually improving over time. **AlphaZero** and the **ExIt** algorithm are notable examples.
- **Data Augmentation for Model-Free Methods:** Here, model-based elements are integrated into model-free methods by using simulated (or "fictitious") experiences to supplement real data. This can either enhance the learning process or entirely replace real experiences, as seen in approaches like **MBVE** and **World Models**.
- **Embedding Planning Loops into Policies:** In this strategy, the planning process is embedded directly into the policy as a subroutine, allowing the policy to decide when and how to use the plan. This approach can mitigate the issues of model bias, as the policy can learn to rely on the model only when it is beneficial. **Imagination-Augmented Agents (I2A)** is an example of this technique.

### 2.4.9 Policy Optimization

Having studied the main algorithms used in RL, we would like to focus on the case of model-free RL and in particular on Policy Optimization algorithms. Because PPO is the algorithm used in the present study, we believe it is convenient to introduce in general the Policy Optimization algorithms and then focus on the PPO. In the following explanations we will deal with the main ideas without always going into the smallest detail, but we will always point out the best sources, in our opinion, to continue the study in those topics that cannot be dealt with in greater depth in this paper.

Policy optimization provides a direct method for enhancing an agent's behavior by focusing on the policy, which dictates the actions the agent takes. Unlike value-based approaches, which estimate the value of state-action pairs and derive policies indirectly,

policy optimization targets the refinement of the policy itself. This approach often leads to a more straightforward and stable learning process.

At the heart of policy optimization is the **policy gradient theorem**, which provides a mathematical framework for adjusting the policy parameters $\theta$ in the direction that maximizes expected returns. For a stochastic policy $\pi_\theta(a|s)$, the goal is to maximize the objective function $J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)]$, where $\tau$ represents a trajectory through the environment and $R(\tau)$ is the cumulative reward. The policy gradient, $\nabla_\theta J(\pi_\theta)$, is derived using the log-derivative trick, leading to an expression that can be computed through sampling:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right].$$

This formulation allows for the use of gradient ascent to iteratively improve the policy. In practice, algorithms estimate this gradient using data collected from the environment and update the policy accordingly.

Modern policy optimization methods build upon this foundation with various techniques designed to improve performance and stability. Proximal Policy Optimization (PPO) is one of the most prominent algorithms in this space, known for its balance between simplicity and effectiveness. PPO uses a surrogate objective function that constrains policy updates, preventing large, destabilizing changes. This results in a more conservative update process, making learning more stable compared to earlier methods like Vanilla Policy Gradient (VPG) or Trust Region Policy Optimization (TRPO). Another popular algorithm is Advantage Actor-Critic (A2C/A3C), which simultaneously learns a policy and a value function, reducing variance in the policy gradient estimates.

An important aspect of policy optimization is the use of baselines, such as the value function $V^\pi(s)$, to reduce the variance of gradient estimates. By subtracting a baseline from the reward function, the algorithm focuses on the relative advantage of actions, leading to faster and more reliable convergence. A popular method for baseline estimation is Generalized Advantage Estimation (GAE), which provides a smooth trade-off between bias and variance.

For those looking to dive deeper into policy optimization, the original papers on PPO by Schulman et al. [47], TRPO [48], and A3C [49] are excellent starting points. Additionally, recent works on GAE [50] offer advanced techniques for improving policy optimization performance.

### 2.4.10 PPO

Proximal Policy Optimization (PPO) is a widely used algorithm in Reinforcement Learning, developed by John Schulman [47] in 2017. It is renowned for its simplicity and effectiveness in improving policy performance without risking significant drops in efficiency. PPO was designed to tackle a central challenge in policy optimization: *how to make large improvements to a policy using available data while avoiding steps that could severely degrade performance.* While Trust Region Policy Optimization (TRPO) addresses this issue

using a complex second-order optimization method, PPO simplifies the process by employing first-order methods, along with additional techniques to keep the new policy close to the old one. This results in an algorithm that is easier to implement and, empirically, performs on par with TRPO.

PPO comes in two primary variants: PPO-Penalty and PPO-Clip. In PPO-Penalty, the policy update is formulated similarly to TRPO by enforcing a constraint on the KL-divergence [51] between the new and old policies. Specifically, PPO-Penalty modifies the objective function by adding a penalty term proportional to the KL-divergence.

PPO-Clip, the more commonly used variant, eliminates the KL-divergence term entirely. Instead, it employs a clipping mechanism directly in the objective function to control how much the new policy can diverge from the old one. The objective function for PPO-Clip updates policies via:

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[ L_{\text{Clip}}(s, a, \theta_k, \theta) \right],$$

with $L_{\text{Clip}}$ given as:

$$L_{\text{Clip}}(s, a, \theta_k, \theta) = \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad \text{clip} \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right),$$

where $\epsilon$ is a small hyperparameter that controls the allowable deviation of the new policy from the old one. The clipping mechanism ensures that policy updates do not result in large, destabilizing changes, even when the advantage $A^{\pi_{\theta_k}}(s, a)$ suggests significant improvements.

To understand how this works, consider two scenarios:

- **Positive Advantage:** When $A^{\pi_{\theta_k}}(s, a) > 0$, increasing $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ (i.e., making action $a$ more probable in state $s$) improves the objective. However, the clipping term ensures that if $\pi_\theta(a|s)$ increases too much (i.e., $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} > 1 + \epsilon$), the objective is capped, preventing further increases and thus limiting how much the new policy can differ from the old one.
- **Negative Advantage:** When $A^{\pi_{\theta_k}}(s, a) < 0$, decreasing $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ (i.e., making action $a$ less probable) is beneficial. The clipping mechanism similarly caps this decrease when $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} < 1 - \epsilon$, preventing the policy from drastically reducing the likelihood of certain actions.

This clipping mechanism acts as a regularizer, effectively removing the incentive for the policy to change too drastically between updates. The hyperparameter $\epsilon$ directly influences the extent of this regularization, making it a crucial aspect of tuning PPO.

***Exploration vs. Exploitation:***

PPO trains a stochastic policy in an on-policy manner, meaning it explores by sampling actions according to the latest version of its policy. Initially, the policy explores broadly,

but as training progresses, it increasingly exploits the actions that have yielded high rewards. This natural shift from exploration to exploitation is a key aspect of PPO, although it can sometimes lead to convergence on local optima if not managed carefully.

PPO's design makes it a robust and flexible choice for various reinforcement learning tasks. Its simplicity and effectiveness have made it a cornerstone in the field, with widespread use in both research and applications. For a deeper understanding, the original PPO papers by Schulman et al. [47] provide detailed insights, while Heess et al. presents in [52] a comprehensive empirical study of the behaviors acquired by PPO agents in complex environments (but utilizing PPO-Penalty rather than PPO-Clip).

## 2.5 Locomotion for Quadruped Robots: state of the art

Quadruped robotics has seen a significant resurgence in recent years, driven by the collaborative efforts of companies, research centers, and universities. These legged robots, particularly quadrupeds, are uniquely adept at navigating unstructured environments where wheeled or tracked robots struggle, making them a promising alternative for rescue missions and exploration in high-risk areas [53]. Consequently, research has heavily focused on improving locomotion, with an additional interest in manipulation capabilities, to enable quadruped robots to operate effectively in diverse environments like urban, natural, and post-disaster.

Locomotion methodologies in this field can be broadly categorized into model-based approaches, such as Model Predictive Control (MPC), and model-free approaches, like Reinforcement Learning (RL).

### 2.5.1 Model Predictive Control for Locomotion:

The robustness of the MPC approach has made it widely adopted in robotics, particularly in early locomotion research, where simplified models and manually defined gait patterns were used to achieve dynamic balance and movement in legged robots [54, 55, 56]. Over time, layered control architectures were developed, enabling robots to navigate various terrains such as rough, soft, and slippery surfaces [57, 58, 59, 60, 61, 62]. More recently, advancements have been made in high-speed locomotion, with whole-body control techniques allowing simultaneous real-time modeling of both a robot's dynamics and its kinematic constraints [63]. For instance, [64] demonstrated a whole-body control system that enabled the Mini Cheetah quadruped robot to reach speeds exceeding 3.5 m/s. Additionally, modern techniques like Regularized Predictive Control have further enhanced locomotion, achieving capabilities such as higher speeds and more natural tight turns, as shown in [65].

Despite the accuracy that these controllers can achieve in quadruped locomotion [66, 53], they are fundamentally constrained by the limitations of the underlying model, which is never perfect or fully computationally tractable. Specifically, MPC often fails in scenarios involving slippage or occlusions in terrain perception. Additionally, MPC relies heavily on predetermined gait patterns, which restricts any additional forms of contact between the

robot and its environment. Deep Reinforcement Learning (DRL) is emerging as a viable solution to many of the challenges where MPC falls short, and has recently demonstrated its effectiveness in enabling robust locomotion for quadrupeds [67, 68].

### 2.5.2 Reinforcement Learning for Locomotion:

As computational power and hardware acceleration have significantly advanced [69], the adoption of RL techniques utilizing neural networks has become increasingly prevalent. For example, in 2018, Tan et al. leveraged RL with dynamics randomization to develop trotting and galloping controllers for the Minitaur robot [70], which achieved stable movement over flat surfaces at a constant speed and direction. Building on these results, Hwangbo et al. [71] refined the training by incorporating a speed-tracking controller for the ANYmal robot, enabling it to reach velocities of up to 1.5 $m/s$. Later, in 2020, Xie et al. utilized sim-to-real RL for the locomotion of the Cassie bipedal robot [72]. That same year, additional improvements in robustness were made for the ANYmal robot across varied terrains by employing a teacher-student learning method [16]. Similar research efforts have also been applied to lower-cost robots, such as Unitree's A1, which is comparable in size to the Mini Cheetah, with a focus on locomotion across different environments [73, 74].

In 2022, two parallel studies from MIT by Ji et al. [75] and Margolis et al. [76] demonstrated successful training of agile running behaviors for the Mini Cheetah robot using RL (see Figure 2.14). Both studies reported highly favorable outcomes, concluding that online system identification greatly improves sim-to-real transfer, particularly for high-speed locomotion.

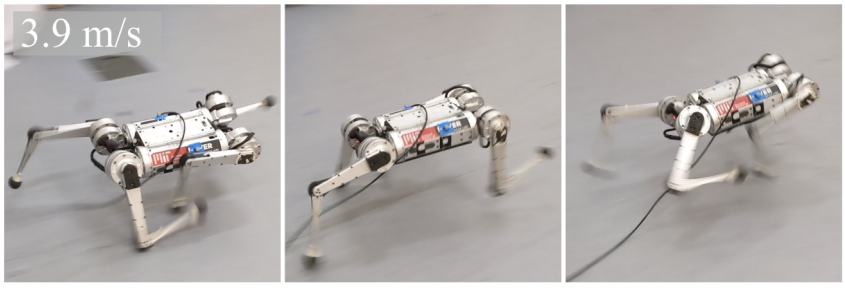


**Figure 2.14:** *Use of RL for Agile policies for fast troting in Mini Cheetah robot. [76].*

Meanwhile, Nikita et al. in [7] developed a framework for multipurpose legged-robot locomotion based on Nvidia's physical simulator (Isaac Gym [29]). This framework leverages GPUs to train policies in as little as 20 minutes. Recently, this system has been employed to train advanced policies capable of agile, parkour-style movements. Building on this work, Holler et al. in [77] created a comprehensive RL-based architecture for the ANYmal robot. Their system integrates various advanced locomotion techniques with a planner and a robust perception system capable of reconstructing height maps, even in environments with occlusions or noise (refer to Figure 2.15). Parallel advances have also been made for low-cost robots. For instance, Cheng et al. [39] developed a system that contrasts Holler et al.'s approach by avoiding reliance on height maps. Instead, their training focuses on using a single depth camera, achieving precise actions despite its low frequency and noisy data.

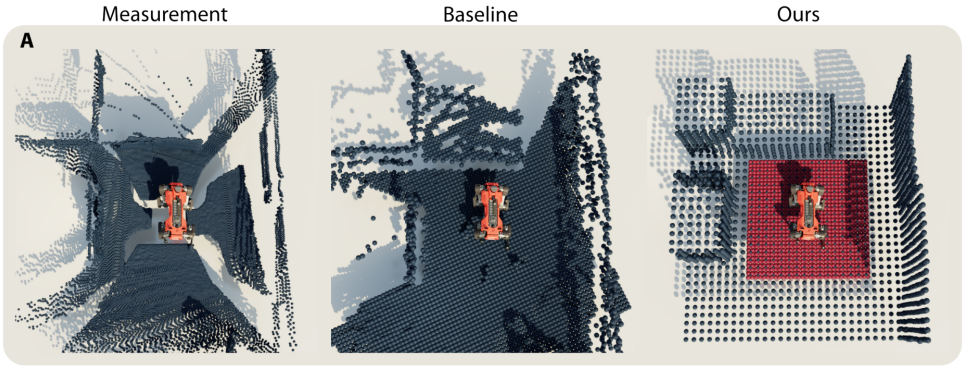


**Figure 2.15:** *A robust perception system for reconstruction of noisy height maps [77].*

Similarly, Zhuang et al. [78] designed an autonomous learning system for low-cost robots that performs parkour skills using vision-based policies. Their method involves a two-stage training process: initial training with soft dynamic constraints, followed by fine-tuning with hard dynamic constraints. [79] also contributed to this field by developing an integrated locomotion system for the A1 robot, using a single depth camera for egocentric vision to navigate challenging terrain (as can be seen in Figure 2.16). This approach also employed a two-stage (teacher-student) training process. Kareer et al. [8] presented related work on the use of egocentric vision, focusing on refining general locomotion policies for obstacle avoidance and indoor navigation. Their system achieves sim-to-sim zero-shot transfer across different environments without fine-tuning.

**Figure 2.16:** *An RL-based locomotion system for the A1 robot, utilizing solely egocentric vision to navigate challenging terrains [79].*

Lastly, other notable efforts have explored the use of quadrupedal robot legs beyond locomotion, such as [80], which focuses on training policies for basic manipulation tasks like pushing a button or kicking a ball.

# 3 METHODOLOGY

This chapter explores the challenges of locomotion in the current context, detailing its key requirements and the available solutions. Section 3.1 briefly introduces the locomotion problem. Section 3.2 summarizes the theoretical, hardware, and software demands necessary for effective quadruped locomotion research. Finally, Section 3.3 reviews alternative approaches and Section 3.4 presents our locomotion framework, highlighting how the explained tools are integrated into our solution.

## 3.1 Introduction

As discussed in the previous chapter, the history of quadruped robots dates back to the early 20th century. However, it wasn't until the early 21st century that these robots achieved sufficient maturity and practicality for use beyond the laboratory. Several interrelated factors have contributed to this progress. First, **a new paradigm was needed to formally address the challenges of locomotion**. Among the many proposals, **those based on Reinforcement Learning (RL) have proven to be the most robust**, offering feasible solutions to many of the associated problems (i.e. [42, 47]). Additionally, **this new branch of Machine Learning**, emerging alongside the recent AI boom, **has been made possible by advances in technology, particularly in computational power and acceleration capacity**. These developments have enabled faster learning, quicker simulations, and more efficient data acquisition, leading to more effective feedback loops. Finally, **the construction of lighter, more robust robots with improved actuators has made the challenges of quadruped locomotion more manageable** and, to some extent, solvable.

In the following sections, we will discuss the new tools available. In particular, the new RL paradigm and the chosen Proximal Policy Optimization (PPO) algorithm, trying to highlight its strengths and limitations. We will also provide an overview of the main robot manufacturers, explain the hardware selected for our project, and detail the reasons behind our choices. Then, we will explore other available software implementations and their key features. Lastly, our proposed framework will detailed.

## 3.2 Tools

We consider three key areas for studying the available tools: Artificial Intelligence (or Computer Science) (3.2.1), technological development, particularly in hardware (processors and electric actuators) (3.2.2), and the development of the Internet, with a focus on open-source software and extensive libraries (3.2.3).

### 3.2.1 RL: A New Paradigm

The foundations of RL were laid in the mid-20th century by Richard Sutton [81], building on the earlier work of Bellman [82]. This pioneering research opened up new possibilities for tackling complex problems involving decision-making in multivariable stochastic environments. With the establishment of this new paradigm, RL algorithms initially found a testing ground in board games and arcade video games. However, the influence of these algorithms extended well beyond gaming, proving their remarkable efficacy in fields such as robotics and control. Their applications now encompass a wide range of tasks, from object manipulation and environmental interaction to navigation and autonomous vehicle driving.

Among the various algorithms developed, **Proximal Policy Optimization (PPO) [47] has emerged as a standout due to its widespread applicability, data efficiency, robustness, and simplicity of implementation**. These characteristics make it particularly well-suited for tasks involving both discrete and continuous action spaces, **which is why we selected this algorithm for our training processes**.

However, adjusting the numerous hyperparameters can be difficult and time-consuming, prompting us to conduct a hyperparametric study (see Section 4.1.2). Below, we introduce its key hyperparameters and their applicable ranges (see Table 3.1). In the next chapter, we will delve deeper into the significance of each parameter and its impact on quadruped locomotion.

| Symbol | Hyperparameter | Default Value* | Range** |
|---|---|---|---|
| $n_{\text{epochs}}$ | `num_learning_epochs` | 5 | [3, 20] |
| $n_{\text{batches}}$ | `num_mini_batches` | 4 | [1, 32] |
| $\epsilon$ | `clip_param` | 0.2 | [0.1, 0.3] |
| $\gamma$ | `gamma` | 0.99 | [0.8, 0.9997] |
| $\lambda$ | `lam (GAE)` | 0.95 | [0.9, 1.0] |
| $c_{\text{value}}$ | `value_loss_coef` | 1.0 | [0.5, 1.0] |
| $c_{\text{entropy}}$ | `entropy_coef` | 0.01 | [0.0, 0.01] |
| $\alpha$ | `learning_rate` | 1e-3 | [1e-5, 1e-3] |
| $\|g\|_{\max}$ | `max_grad_norm` | 1.0 | [0.5, 1.0] |
| $\text{KL}_{\text{desired}}$ | `desired_kl` | 0.01 | [0.003, 0.03] |
| seed | `seed` | 1 | $(-\infty, +\infty)$ |

**Table 3.1:** *Principal Hyperparameters of the PPO Algorithm [83]. *Default values come from [84]. **A concise study for PPO hyperparameters ranges obtained from [85].*

### 3.2.2 Hardware

Two types of hardware are important to consider here: the acceleration hardware used for training and the quadruped robots themselves.

- **Graphical Processing Units (GPUs):** have revolutionized the ability to parallelize training, significantly enhancing the speed and scale of learning processes. A

decade ago, acquiring GPUs was prohibitively expensive, not only because the technology was less advanced, but also because GPU architectures were typically hybrids with CPUs. In these systems, while GPUs accelerated certain computations, other tasks still had to be processed by CPUs, creating a bottleneck. Advanced training, therefore, required large clusters with hundreds or even thousands of GPUs and CPU cores working together [86].

However, **advancements such as Isaac Gym [29] have dramatically streamlined and accelerated this process. Now, entire learning workflows can be efficiently managed on a single GPU** or multiple GPUs without the need for extensive CPU-GPU communication, which previously slowed things down. As a result, solutions like Isaac Gym have become increasingly accessible to small teams and independent researchers. With just one or a few GPUs, they can now conduct cutting-edge research and train models at a level comparable to that of large corporations or research institutions.

The pace of advancement in this field is so rapid that, despite their accessibility to the average consumer, these components quickly become outdated, necessitating upgrades every few years. As of this study, NVIDIA's top-performing graphics card is the `GeForce RTX 4090 Laptop`. **For our research, we utilized an `GeForce RTX 4060 Laptop`**, which delivered impressive results in both speed and computing power. Specifically, **we achieved average simulation times of approximately** 10 **minutes for** $1,024$ **environments over** $1,000$ **iterations**.

- **Quadruped Robots:** *Boston Dynamics*, a spin-off from MIT's Leg Laboratory founded by Marc Raibert in 1992, is widely regarded as the pioneer in developing advanced quadruped robots. Their early prototypes, comparable to military-grade projects, demonstrated exceptional performance in demanding field environments. This early success accelerated the advancement of both hardware and software technologies, providing a strong foundation for rapid progress in the field. Consequently, today's market offers a wide array of similar technologies from numerous manufacturers, often at significantly lower costs. A list of available hardware can be found at the end of Section 2.3.1.

  **For this study, equipment from the Chinese company *Unitree* was utilized**. The proliferation of companies like **Unitree has made robust hardware platforms accessible to smaller research groups without the burden of prohibitive expenses**. Moreover, significant differences between companies and products should be considered. For instance, Boston Dynamics' Spot robot restricts user access to low-level control, offering only high-level functionalities. Additionally, the Spot's manipulator can interact only with objects on the floor, limiting its versatility. In contrast, **Unitree's robots are fully open to user customization, allowing control over positions and joint velocities at a low level, and even enabling modifications to each motor's controllers**. This openness, coupled with their lower price, has made Unitree the preferred choice for this study.

### 3.2.3 Software

Over the past decade, the development of Deep Learning (DL) has led to the emergence of highly versatile and user-friendly software tools. Among these, *TensorFlow* (2015) from

the Google Brain Team and *PyTorch* (2016) from Facebook's AI Research Lab stand out as the most robust and widely used open-access frameworks. Leveraging these tools, more specialized frameworks have been developed. In our case, Isaac Gym (built on PyTorch for training RL policies) has been the primary framework used. Specifically, we have relied on two key works that make use of it: *Legged Gym* by Nikita et al. [7] and *ViNL* by Kareer et al. [8]. Below, we detail some of their main features in terms of software and their innovations compared to other frameworks.

- **Legged Gym:** A key strength of this approach lies in its capability to train a perception policy in under 20 minutes using a single GPU. This is accomplished through a game-like curriculum that dynamically adjusts task difficulty according to the policy's performance. Additionally, it utilizes Nvidia's Isaac Gym simulation platform, enabling both training and simulation entirely on the GPU, supporting the parallel simulation of thousands of robots. The simulation is powered by the PhysX physics engine, which was also developed by Nvidia for Isaac Gym.

  The Deep Reinforcement Learning algorithm employed is PPO (Proximal Policy Optimization), which enables a policy to gather data in batches before the next policy update. The volume of data collected is regulated by the hyperparameter *batch size*. Furthermore, PPO includes a *critic* that estimates the infinite-horizon sum of discounted future rewards (see Section on Return and Reward for more details).

  In Legged Gym, individual legged robots, whether quadrupedal or bipedal, are trained to follow commands for linear speed and heading, while maintaining a stable height from the robot's base, and learning to navigate complex terrains. Specifically, five types of terrains are featured: flat, sloped, randomly rough surfaces, discrete obstacles, and various stair configurations. These terrains are organized into 8-meter squares arranged in a grid (illustrated in Figure 3.1). The robots begin at the center of each square, receiving randomized heading and speed commands that remain constant throughout an episode. Initially, the policy is trained on simpler terrain configurations (e.g., lower inclination, fewer obstacles) before progressively tackling more challenging conditions.

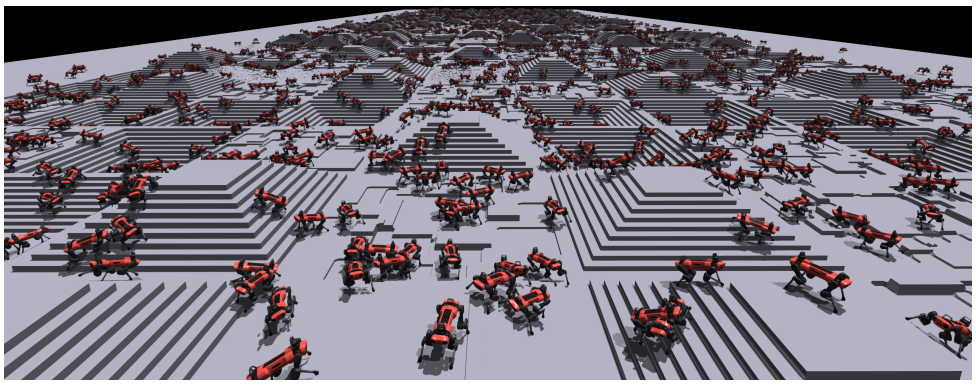

**Figure 3.1:** *Tiled squares for the 5 types of terrain in Legged Gym framework. Quadruped robots learn to walk in this general purpose game-inspired curriculum. Sourced from [7].*

The policy receives the robot's proprioceptive measurements as well as information

from the terrain around the robot's base. This observation space at timestep $t$ consist of the terms summarized in Table 3.2.

| Symbol | Description |
|---|---|
| $\mathbf{q}_t \in \mathbb{R}^{12}$ | Joint positions |
| $\dot{\mathbf{q}}_t \in \mathbb{R}^{12}$ | Joint velocities |
| $\mathbf{q}_{t-1} \in \mathbb{R}^{12}$ | Previous joint commands |
| $\mathbf{v}_b \in \mathbb{R}^3$ | Base linear velocity |
| $\boldsymbol{\omega}_b \in \mathbb{R}^3$ | Base angular velocity |
| $\mathbf{g} \in \mathbb{R}^3$ | Projected gravity vector |
| $\mathbf{v}_b^* \in \mathbb{R}^3$ | Commanded velocities |
| $\mathbf{H} \in \mathbb{R}^{108}$ | Terrain map |

**Table 3.2:** *Legged Gym Policy Observations (inputs)*

In addition, the policy is guided by a total reward function, which is the sum of nine weighted terms, as detailed in Table 3.3. The primary objectives of these terms are to ensure the robot follows the commanded velocities while minimizing undesirable movement in other axes. The reward function also imposes penalties for excessive joint torques, high joint accelerations, abrupt changes in joint position commands, and collisions with the terrain or between the robot's legs. Furthermore, the robot is incentivized to take larger steps to promote a more natural walking gait. A single policy is trained using these rewards across all terrain types.

| Term | Definition | Weight |
|---|---|---|
| Linear velocity tracking | $\phi(\mathbf{v}^*_{b,xy} - \mathbf{v}_{b,xy})$ | $1\,dt$ |
| Angular velocity tracking | $\phi(\boldsymbol{\omega}^*_{b,z} - \boldsymbol{\omega}_{b,z})$ | $0.5\,dt$ |
| Linear velocity penalty | $-\mathbf{v}^2_{b,z}$ | $4\,dt$ |
| Angular velocity penalty | $-\lvert\lvert\boldsymbol{\omega}_{b,xy}\rvert\rvert^2$ | $0.05\,dt$ |
| Joint motion | $-\lvert\lvert\ddot{\mathbf{q}}\rvert\rvert^2 - \lvert\lvert\dot{\mathbf{q}}\rvert\rvert^2$ | $0.001\,dt$ |
| Joint torques | $-\lvert\lvert\boldsymbol{\tau}\rvert\rvert^2$ | $0.00002\,dt$ |
| Action rate | $-\lvert\lvert\dot{\mathbf{q}}^*\rvert\rvert^2$ | $0.25\,dt$ |
| Collisions | $-n_{\text{collision}}$ | $0.001\,dt$ |
| Feet air time | $\sum_{f=0}^{4}(\mathbf{t}_{\text{air},f} - 0.5)$ | $2\,dt$ |

**Table 3.3:** *Definition of reward terms, with* $\phi(x) := \exp\left(-\frac{||x||^2}{0.25}\right)$. *The z-axis is aligned with gravity. Sourced from supplementary material in [7].*

The policies output desired joint positions ($q_t^*$) as actions, which are transmitted to the motors where PD controllers generate the necessary torques.

To facilitate a successful sim-to-real transfer, the training process involves randomizing ground friction, adding noise to the observations (as done in [7] based on actual measured data), and applying perturbations to the robot. Specifically, every 10 seconds during an episode, the robot is either nudged or its base is accelerated by up to $\pm 1$ m/s in the x and y directions. These strategies are designed to help the robot develop more stable postures.

- **ViNL:** Visual Navigation and Locomotion focuses on training two distinct policies: one for locomotion and another for navigation, enabling robots to traverse indoor environments using only egocentric vision while avoiding obstacles. In this case, the use of legged robots clearly offers a significant advantage over wheeled robots, as they can step over obstacles rather than having to navigate around them.

  For the locomotion policy, ViNL builds upon the Legged Gym framework, using the Aliengo robot and introducing specific modifications to the training process. The key differences between ViNL and Legged Gym are outlined below:

  – ViNL utilizes the Aliengo robot (as previously mentioned) instead of the ANYmal robot.

  – The primary goal of ViNL is to deploy a policy that relies predominantly on egocentric visual information. Unlike Legged Gym, where the policy uses a discretized height map generated by mapping the robot's environment, ViNL focuses on using depth cameras (RGB-D images). In contrast, Legged Gym (for real hardware) relies on Lidar sensors to create such environmental maps.

  – ViNL introduces a three-stage learning process. The first stage involves learning general locomotion skills using the same curricula as Legged Gym. In the second stage, the policy is refined for flat environments with obstacles by adding a reward term that heavily penalizes contact with obstacles, encouraging the robot to avoid them while following velocity commands. During these two stages, the policy has access to the height map around the robot.

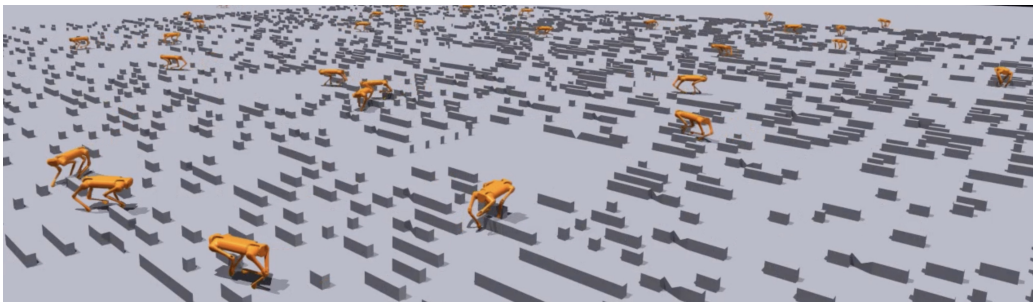

**Figure 3.2:** *ViNL's second training stage. Robots learn to walk in cluttered environments. Sourced from [8].*

  The third stage shifts to supervised learning, where a teacher-student distillation approach is used. In this stage, the student policy has access only to depth images while learning to reconstruct the height map from them (what is called in the paper Learning by Cheating [87]).

  – ViNL employs a height map $H \in \mathbb{R}^{187}$, which has a higher resolution compared to the height map used in Legged Gym ($H \in \mathbb{R}^{108}$).

  – In ViNL the local map is not concatenated directly with the rest of observations (as Legged Gym does), it is first encoded separately with the use of a MLP encoder with three layers (with 128, 64 and 32 hidden units).

  – ViNL also trains a high-level visual navigation policy using 3D reconstructions of real-world indoor environments, leveraging the Habitat-Lab simulator [88]. This policy is trained with kinematic control, where the commands issued specify the linear and angular velocity of the robot's center of mass.

– In ViNL, both policies are combined in simulation to tackle visual navigation tasks in obstacle-rich environments, employing zero-shot transfer.

On the other hand, beyond DL software, robotics requires essential platforms like ROS. ROS plays a crucial role in the development of robotic systems by offering significant modularity and making our implementations accessible to the broader robotics community. Additionally, visualization and simulation environments like RViz [89] and Gazebo [90], which are closely integrated with ROS, allow us to test and refine designs before deployment, reducing the risk of errors that could cause harm or damage to the robots.


**Figure 3.3:** *From left to right: a) Gazebo logo. b) RViz logo. c) ROS Noetic, the final version of ROS 1 before the transition to ROS 2. d) ROS 2 Humble, the most stable release to date.*

## 3.3 Other Frameworks

In addition to the points discussed so far, it is important to highlight alternative approaches to quadruped locomotion and some particularly promising work in this area.

First, as previously mentioned, many researchers [78, 8, 39, 79] opt for egocentric vision to avoid relying on height maps, which require sensors like LiDAR or depth cameras to generate point clouds, but need for more complex neural networks like RNNs[6]. The challenge with height maps lies in the noisiness of the data and frequent occlusions. To address this, methods using visual locomotion often employ teacher-student distillation. In this approach, a teacher model is trained with privileged information, including noise-free scanned dots from height maps (see Figure 3.4). The student model is then trained through supervised learning to mimic the teacher's behavior but with less information and intentionally added noise, enhancing the student's robustness.

Conversely, some researchers focus on improving height map reconstruction or filtering techniques [77]. Given the demands of real-time point cloud processing and height map

[6]RNN stands for Recurrent Neural Networks, a type of neural architecture commonly used for processing temporal or sequential data. A notable example is the Long Short-Term Memory (LSTM) RNN, which is often employed in scenarios where a robot must navigate using vision. In such cases, objects may temporarily disappear from the robot's field of view, but the system needs to remember their existence. LSTM networks enable robots to retain this kind of memory, allowing them to act more intelligently in complex environments.

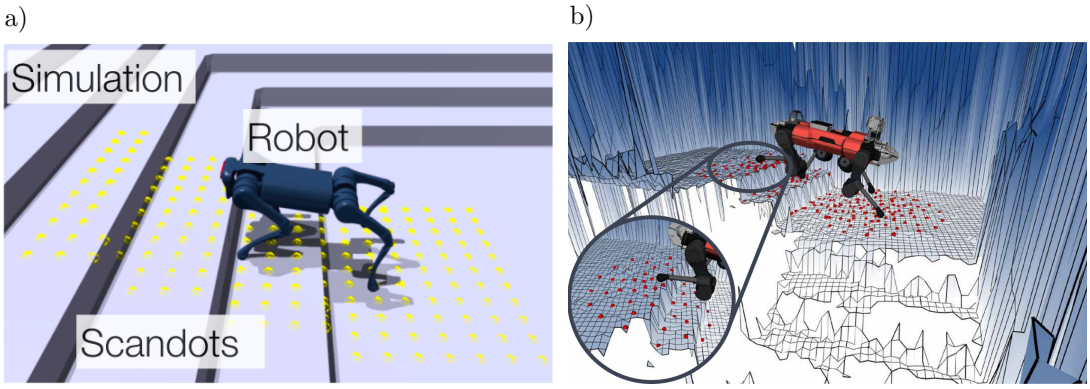


**Figure 3.4:** *Scanned dots obtained from simulation (a) or from real height map reconstruction (b). Sourced from [79] and [6] respectively.*

generation, these methods often require dedicated GPUs to ensure timely map availability. Another possible strategy, as seen in ViNL, is the simultaneous or interdependent training of navigation and locomotion policies [77].

The Legged Robotics group at ETH has made significant contributions using reinforcement learning. Their publications [71, 91, 53, 16, 40, 9, 6, 92, 53, 93, 77] extensively utilize Isaac Gym, teacher-student distillation, and complex neural architectures, including RNNs and advanced encoders (e.g., the one featured in [6] and illustrated in Figure 3.5). These studies are highly recommended for detailed examination and gradual replication[7].

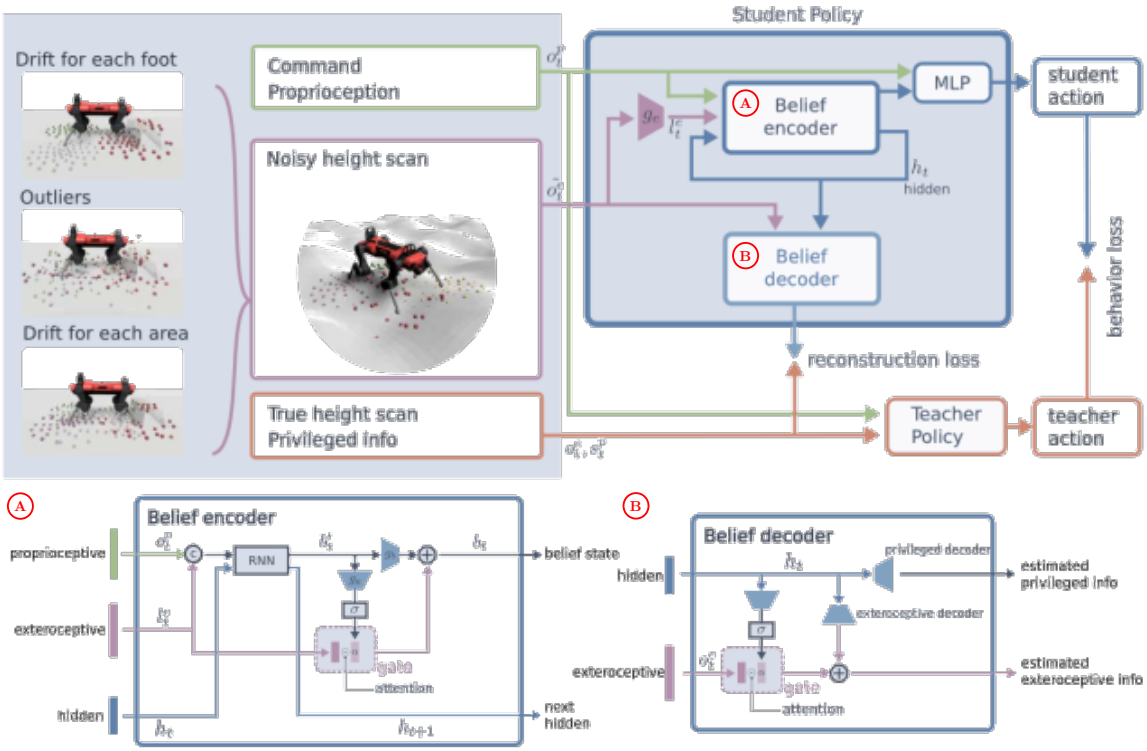


**Figure 3.5:** *Student Policy Architecture from [6]. The Belief encoder and decoder internal components are highlighted.*

[7]Regarding the Belief State Encoder/Decoder implementation please refer to this GitHub repository: https://github.com/lucidrains/anymal-belief-state-encoder-decoder-pytorch/tree/main.

## 3.4 Proposed Framework

Once the primary tools and methods necessary for addressing the problem of quadruped locomotion have been established, we proceed to thoroughly characterize the selected methods and their implementation details. Our framework is divided into three main stages: training, testing, and deployment.

- **Training** involves developing a locomotion policy through reinforcement learning using algorithms such as PPO in simulation environments like Isaac Gym. This stage also encompasses training strategies like teacher-student distillation and sim-to-real transfer techniques.
- **Testing** refers to evaluating these policies in simulated environments, such as those provided by ROS [94]. This includes testing sensory fusion, robot models, new simulated environments in Gazebo, control architecture, loop closure speeds, and signal matching.
- **Deployment** encompasses the design and construction of real-world test environments, executing the simulated ROS architecture on the actual robots, as well as data collection and performance testing.

Guided by this architecture, as depicted in Figure 3.6, we summarize our chosen framework below.

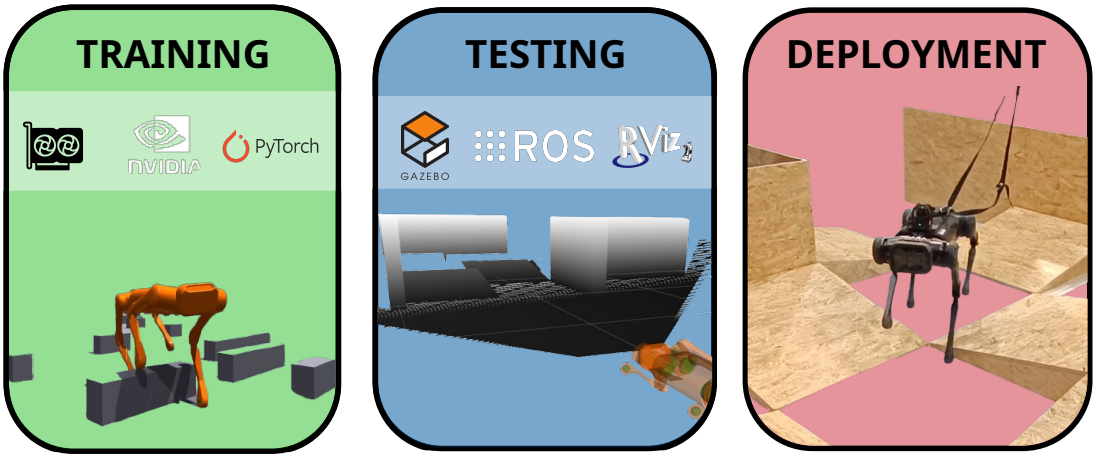


**Figure 3.6:** *Three main stages in our framework: training, testing and deployment.*

### 3.4.1 Training

As mentioned in Section 3.2.3 of this chapter, Legged Gym and ViNL serve as our primary references for policy training. It is important to clarify the hierarchy in the use of these frameworks. Our main tool is the implementation by Kareer et al. with ViNL, which is itself primarily based on Legged Gym, utilizing Isaac Gym as the underlying simulator. Thus, Legged Gym and Isaac Gym are the core training components, while ViNL serves as a specific application tailored to our use case, particularly with respect to the Aliengo

robot. However, future work may require exploring other approaches similar to ViNL that also rely on Isaac Gym as the simulation foundation.

It is worth noting that Isaac Gym is currently undergoing several upgrades and migrations. While these changes do not render it obsolete, they have led to reduced support from the NVIDIA team. NVIDIA now recommends using Isaac Sim, a more versatile platform that builds on the features of Isaac Gym while offering greater integration with other systems, such as ROS. Isaac Sim provides a more general-purpose solution compared to the more specialized Isaac Gym. Although we won't delve into further details here, it is advisable for readers to stay updated on the status of these NVIDIA frameworks. Other tools, such as Isaac Lab and Omniverse, have been excluded from our work.

In summary, our work utilizes ViNL, particularly the locomotion policy training as far as we do not need to permorf in this study the navigation. Within locomotion, we focus on the first two stages: the first stage, general-purpose locomotion, is particularly relevant for SAR missions, while the second stage, obstacle avoidance, is also valuable for handling less common scenarios, such as casualty care, explosive ordnance disposal, or navigation in hazardous environments. Although the third stage of supervised training has been examined in detail,[8] it is not of particular interest to us. Our focus is on a fusion approach that leverages multiple sources of information, rather than relying solely on vision. We believe that incorporating additional data, such as height maps, offers a more comprehensive and effective solution. This approach presents a promising avenue for future exploration.

In the first two locomotion stages, an actor is trained with access to a local height map surrounding the robot ($\mathbf{H} \in \mathbb{R}^{187}$). Unlike in Legged Gym, as previously discussed, the trained policy[9] features a slightly different architecture. It utilizes the inputs[10] and outputs shown in Figure 3.7 and incorporates an encoder to condense the height map. The encoder (shown in light blue in the figure) is a three-layer MLP with 128, 64, and 32 hidden units, while the actor and critic MLP networks each have three layers with 256 hidden units per layer.

Table A.2 (in Annexes) presents key general training configurations related to the environment. These, along with other minor details, can be found in the environment's configuration files. They can be adjusted to accommodate a new robot configuration or training strategy as needed.

Finally, the remaining sim-to-real techniques are retained from the original Legged Gym implementation in ViNL. Since ViNL is focused purely on sim-to-sim research, these techniques have not been adapted for the Aliengo robot and require further exploration in future work.

[8] The following link provides details on the debugging study conducted regarding the issues encountered during the third training stage: https://github.com/SimarKareer/ViNL/issues/7.

[9] Note that when we refer to training a policy in DRL, we are actually referring to training a neural network. In DRL, the objective is not to derive an analytical equation that optimizes performance but rather to learn the weights of a neural network that approximates the solution to these theoretical equations, especially given the large action space. This network, given certain observations of the state, provides the best action to maximize the reward.

[10] Note that the order of the inputs is crucial for the network's subsequent inference, particularly during the testing and deployment stages.

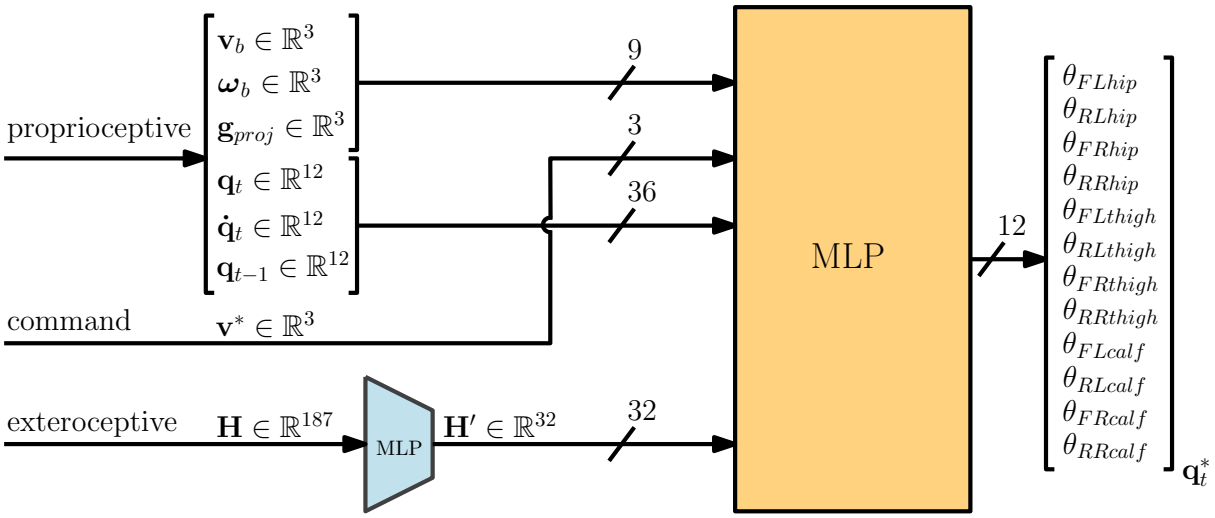


**Figure 3.7:** *Basic actor architecture. Inputs consist of proprioceptive and exteroceptive data, along with velocity commands, arranged in the specific order used. Outputs correspond to the next 12 joint positions of the robot (naming is consistent with Figure 2.11). The encoder is shown in light blue, while the actor MLP is depicted in light orange. The total input size is* $\in \mathbb{R}^{80}$.

### 3.4.2 Testing

Once the policy has been trained and visualized in the training simulator with satisfactory results, it is crucial to proceed to the next phase: testing. In this phase, the robot's control architecture will be constructed using ROS, where the locomotion policy will be implemented.

Currently, **our work utilizes ROS 1 Noetic on `Ubuntu 20.04`**. However, future developments may potentially transition to ROS 2 Humble, which offers significant improvements, such as better support for real-time systems, enhanced security features, and improved communication protocols. This would align with the evolving needs of robotic systems and ensure that our architecture remains cutting-edge.

#### *ROS Architecture*

Figure 3.8 illustrates the overall ROS architecture to be implemented. The manufacturer provides the core ROS packages (`unitree_ros`) [95, 96] for simulating and controlling the robot. Additionally, a custom package or node must be developed to create the neural network, load the trained weights, and perform inference to determine the robot's joint positions (`rl_controller`). Since our training relies on a height map of the robot's surroundings, sensors that capture point clouds of the environment are required, which are then converted into height data. The *Elevation Mapping* package from ETH [97, 98] allows us to generate a `grid_map` containing the height information around the robot. The two front-facing depth cameras (see Figure 3.9.a) are used to capture point clouds [99, 100], which are fused [101] to enhance robustness before being processed by the height map generator. Finally, the manufacturer's low-level controller uses PD controllers to follow the angular references of each joint according to the neural policy (`rl_controller`). For initial testing, pure teleoperation via joystick speed commands can be employed as a local

planner. In later iterations, a trajectory or waypoints can be generated for each Gazebo test map, and a local planner can be used to follow the designated path.

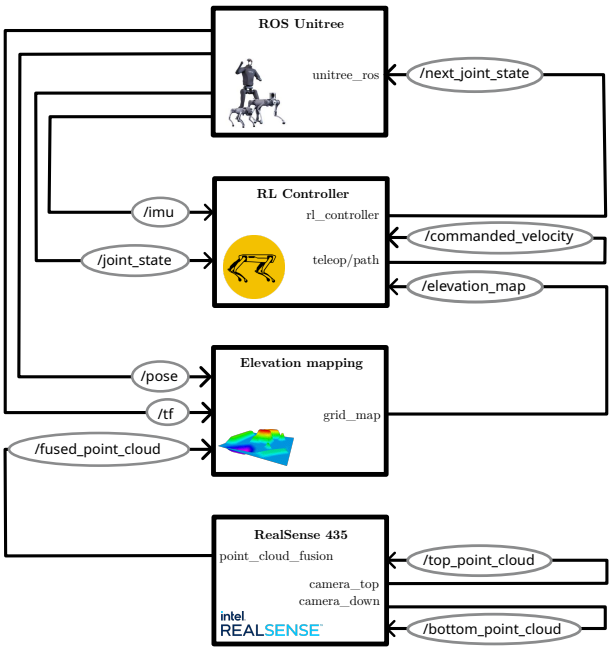


**Figure 3.8:** *Proposed ROS architecture for RL controller deployment. This architecture serves as the testing component of our RL locomotion framework. It leverages the* ***unitree_ros*** *low-level controllers [95, 96], a* ***grid_map*** *generated from point cloud data [97, 98], and the RealSense Depth camera ROS libraries [99, 100].*

Utilizing the repositories mentioned above, we successfully visualized the Aliengo robot in Gazebo and integrated two Intel RealSense 435 cameras (as shown in Figure 3.9). Since the exact relative positions of these cameras with respect to the robot's base were not available, they were positioned to the best of our ability (see Figures 3.9.b and 3.9.c). Code 1 provides the xacro snippet used to include the cameras in the Aliengo URDF, where their positioning relative to `trunk`, which functions as the `baselink`, is detailed. As a result, if the robot were to be trained using RGB images, any inaccuracies in the camera positions could potentially impact its performance in simulation and, consequently, in real-world scenarios. This would necessitate retraining the policy to account for the possible uncertainties in camera positioning. However, since our system relies on height maps generated from point clouds via the *Elevation Mapping* ROS package, it is sufficient to accurately reference the cameras relative to the robot's base, which is the origin from which the heights in the `grid_map` are measured.

```
<!-- ######### Camera Intel RealSense 435 (down) ######### -->
<xacro:property name="M_PI" value="3.1415926535897931" />
<xacro:include filename="$(find
    ↪ realsense2_description)/urdf/_d435.urdf.xacro"/>
<xacro:sensor_d435  parent="trunk" name="D435_camera_down"
    ↪ topics_ns="D435_camera_down">
     <origin rpy="0 ${M_PI/15} 0" xyz="0.312 0.013 -0.035"/>
</xacro:sensor_d435>
<!-- ######### Camera Intel RealSense 435 (top) ######### -->
<xacro:include filename="$(find
    ↪ realsense2_description)/urdf/_d435.urdf.xacro"/>
<xacro:sensor_d435  parent="trunk" name="D435_camera_top"
    ↪ topics_ns="D435_camera_top">
```

```
    <origin rpy="0 ${-M_PI/11} 0" xyz="0.325 0.013 0.024"/>
</xacro:sensor_d435>
<!-- ################################################### -->
```

**Listing 3.1:** *Xacro snippet used to include the Intel RealSense cameras in the Aliengo URDF.*

As shown in Figures 3.9.a and 3.9.b, the cameras have been positioned as closely to their real-world counterparts as possible. The lower camera provides a view of the terrain directly in front of the robot, while the upper camera offers a broader perspective of the area ahead.

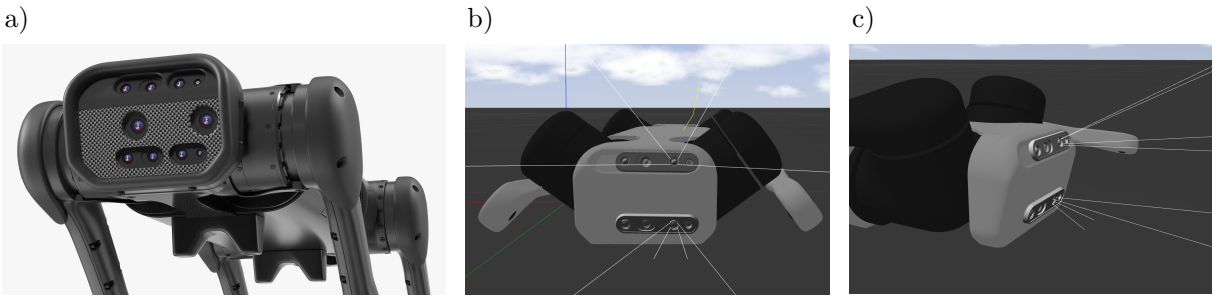


**Figure 3.9:** *Intel RealSense cameras positioning in the default Unitree URDF model. This was accomplished using the code in Listing 3.1. (a) displays the actual positions of the cameras, while (b) and (c) provide different perspectives of the robot's output after the cameras have been positioned.*

Figure 3.10 illustrates the coordinate frame tree for one of the cameras, generated using the ROS `tf` library. The manufacturer provides a detailed model of the lenses, including their relative positions and their placement within the housing. Depending on the requirements, this model could be simplified if using only depth.

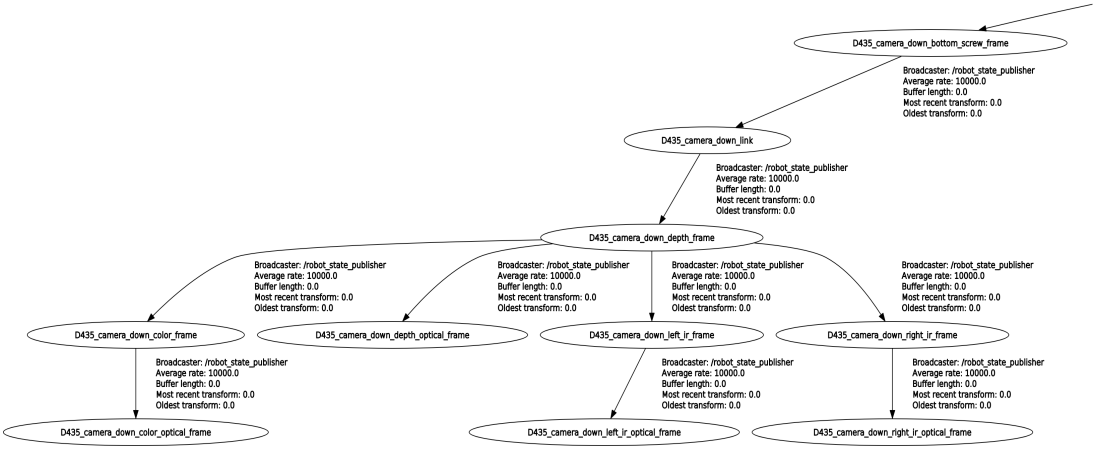


**Figure 3.10:** *ROS coordinate frame tree for down-facing camera in Aliengo. It has been generated using the ROS `tf` library.*

Figure 3.11 illustrates the robot's positioning within a test environment (which will be discussed later). On one side, Figure 3.11.a shows the images captured by the robot, while Figure 3.11.b displays the generated point clouds. Specifically, the lower part of Figure 3.11.b presents the point clouds from both cameras, with the merged output (`/fused_point_cloud`) being visualized.

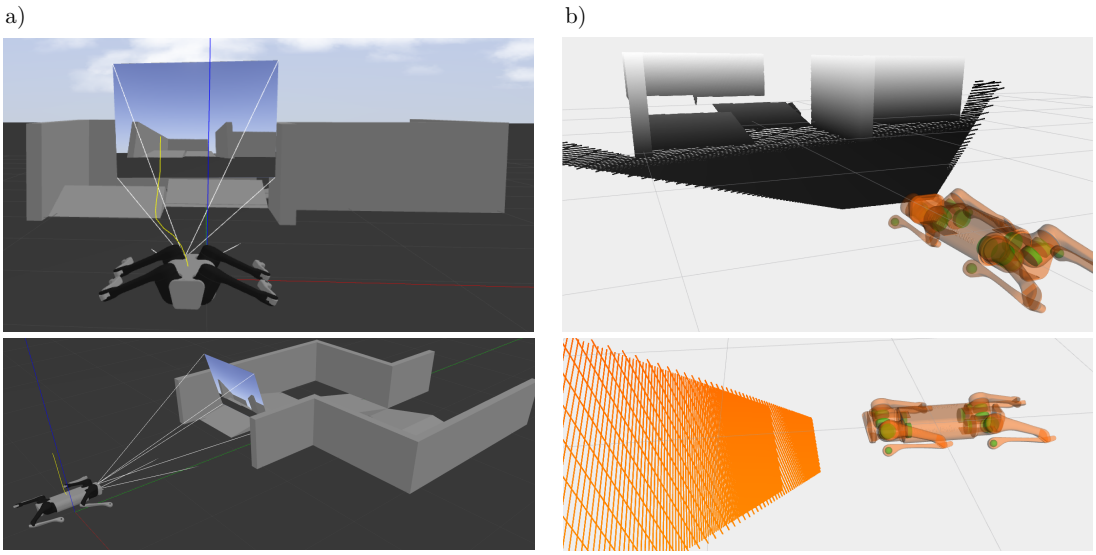


**Figure 3.11:** *Robot's positioning within a test environment in Gazebo. (a) displays the images captured by the simulated robot (and cameras), while (b) shows the generated point clouds, specifically the merged output from both cameras (`/fused_point_cloud`).*

Finally, Figure 3.12 also includes the topics and nodes graph generated using `rqt_graph`. On the right side, it can be seen the node responsible for fusing the point clouds, which subscribes to the depth data published by the Intel RealSense D435 top and bottom cameras (`/D435_camera_down/depth/color/points` and `/D435_camera_top/depth/color/points`).

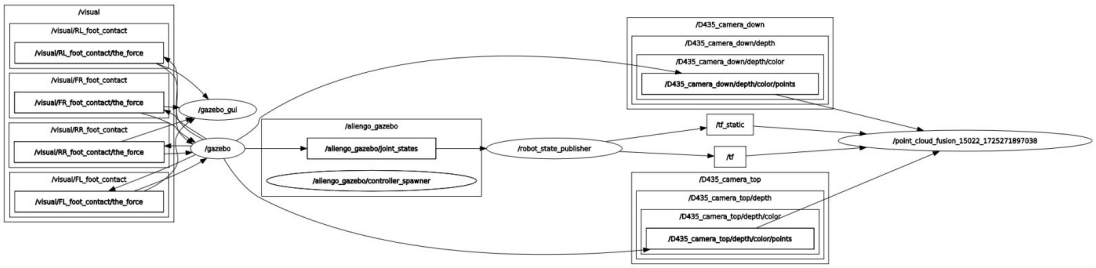


**Figure 3.12:** *Topics and nodes graph generated using `rqt_graph`.*

### *Gazebo Maps*

The maps shown in Figure 3.13 were designed for testing the robustness and performance of the RL-trained policies in Gazebo. Four types of maps, each with varying levels of difficulty, were created with geometry specifically tailored for eventual real-world construction. For instance, the design includes six ramps, corresponding to the number planned for actual construction (see next Section 3.4.3). By replicating these simulation models in the real world, we can gain greater confidence in the robot's future real-world performance.

The `stl` files were modelled with *Autodesk Inventor* and imported to Gazebo with the help of the following folder structure (see Listing 3.2) and `model.config` (see Listing 3.3) and `model.sdf` (see Listing 3.4) templates.

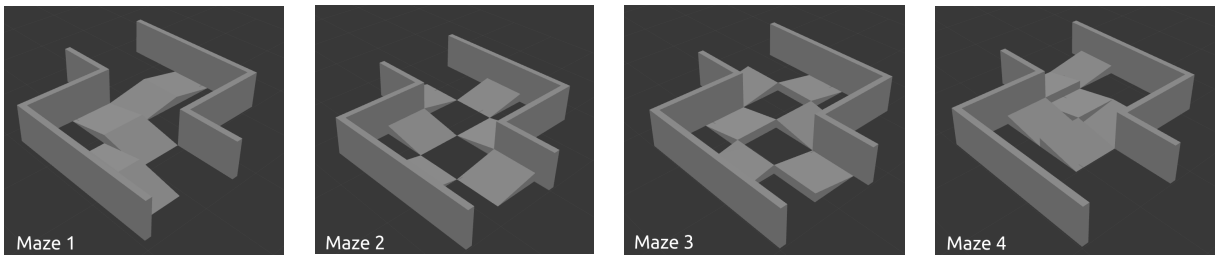


**Figure 3.13:** *Gazebo maps designed for testing the RL-trained model in ROS.*

```
home
|- .gazebo/models
    |- meshes
    |  |- maze_1.stl
    |- model.config
    |- model.sdf
```

**Listing 3.2:** *Folder structure for Gazebo worlds creation.*

```
<?xml version="1.0"?>
<model>
<name>Maze 1</name>
<version>1.0</version>
<sdf version="1.6">model.sdf</sdf>
<author>
    <name>Your Name</name>
    <email>yourmail@mail.com</email>
</author>
<description>
    A meaningful description.
</description>
</model>
```

**Listing 3.3:** *`model.config` template for Gazebo world creation.*

```
<?xml version="1.0"?>
<sdf version="1.6">
<model name="maze_1">
    <static>true</static>
    <link name="link">
    <visual name="visual">
        <geometry>
        <mesh>
            <uri>model://maze_1/meshes/maze_1.stl</uri>
            <scale>0.001 0.001 0.001</scale>
        </mesh>
        </geometry>
    </visual>
    <collision name="collision">
        <geometry>
        <mesh>
            <uri>model://maze_1/meshes/maze_1.stl</uri>
            <scale>0.001 0.001 0.001</scale>
        </mesh>
        </geometry>
    </collision>
    </link>
```

```
</model>
</sdf>
```

**Listing 3.4:** *`model.sdf` template for Gazebo world creation.*

It is also important to consider parameters such as `<scale>` and `<static>`, among others. Additionally, you can import other prebuilt environments[11] Further instructions on these settings can be found in our GitHub project repository[12].

### 3.4.3 Deployment

After completing the simulation phase with ROS, the next crucial step is deploying the architecture on the actual hardware. This phase often presents various challenges, as it is common for execution times in simulation to differ from those on the robot's hardware and sensors. This discrepancy can necessitate significant debugging or restructuring of the code. Moreover, even with sim-to-real randomization techniques and robustness-focused actions, it is essential to assess how much measurement noise affects policy execution. In some cases, this may require retraining the model to improve performance.

Additionally, it is important to design and build test circuits or benches that can effectively demonstrate the specific capabilities of our implementation. In our case, while we couldn't complete the deployment on the robot itself, we designed and constructed an obstacle course for testing multipurpose locomotion policies.

#### *Circuit Construction*

As the third phase of our framework, we proposed deploying the ROS architecture on the real robot. While this phase has not yet been fully executed, significant progress has been made in designing and constructing a modular test circuit that will facilitate future testing of this deployment.

Figure 3.14 shows the elevation and plan views of the test circuit sketches. The circuit has a Z-shape inspired from [102], with corridor widths of 1.5 meters. The total area covered is 3.5 meters in length and 3 meters in width. Inside, there is space for up to 12 blocks, each measuring 75 cm $\times$ 75 cm, which in our case will serve as ramps.

As can be seen in Figure 3.15, the side walls have a height of 62.5 cm. Both the walls and the ramps are constructed using 15 mm thick OSB wood panels. The walls were joined together with metal brackets and small 3 cm long bolts, while the ramps were assembled using 3 mm $\times$ 3 cm wood screws. Since the boards have standard dimensions, the circuit is geometry was adapted accordingly. As shown in Figure 3.15, all the side panels were obtained from just two boards with only three cuts.

[11] A good repository for finding these is available at https://github.com/macc-n/gazebo_worlds. We have selected two possible environments for testing locomotion: an easy environment https://github.com/macc-n/gazebo_worlds/blob/master/gazebo/screenshots/office_cpr_construction.jpg and a hard environment https://github.com/macc-n/gazebo_worlds/blob/master/gazebo/screenshots/rubble.jpg.

[12] Please refer to the following GitHub repository for more details about our conducted simulations and results, like videos and code details: https://github.com/jbarciv/Hyper-ViNL

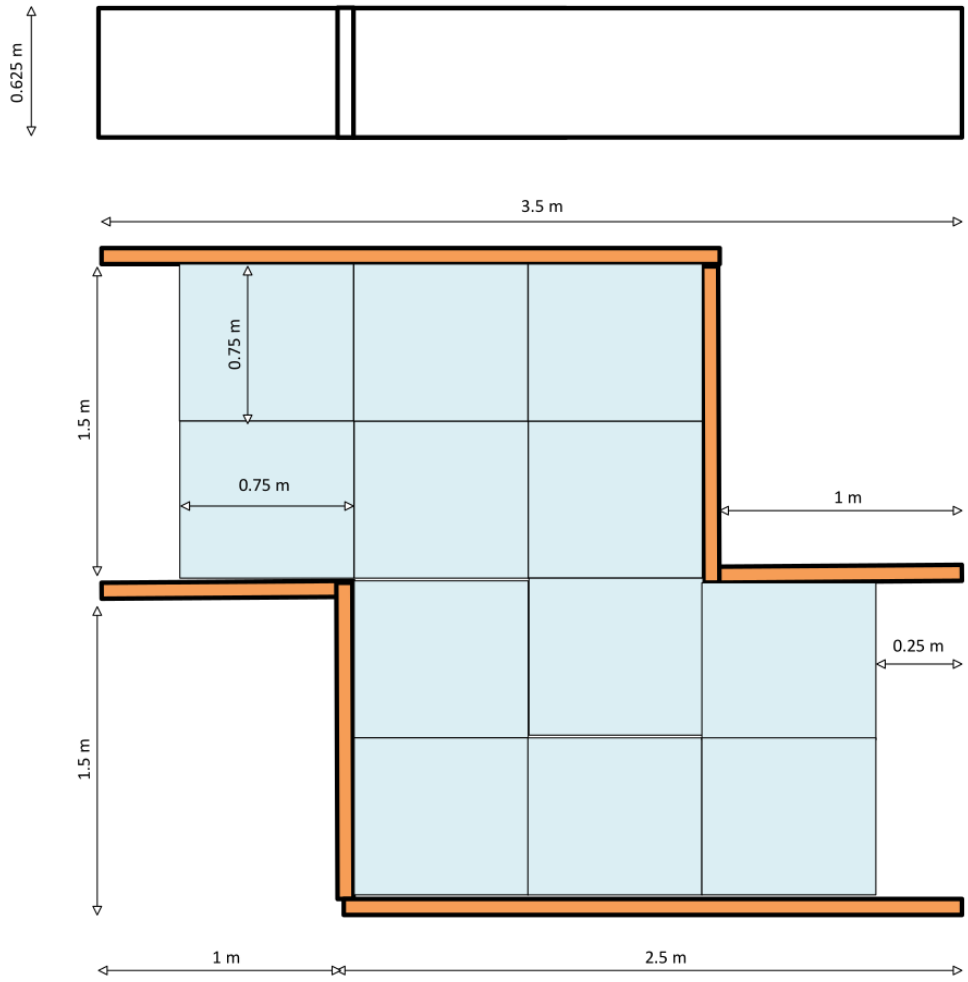


**Figure 3.14:** *Elevation and plan views of the test circuit sketches. The circuit has a Z-shape inspired from [102].*

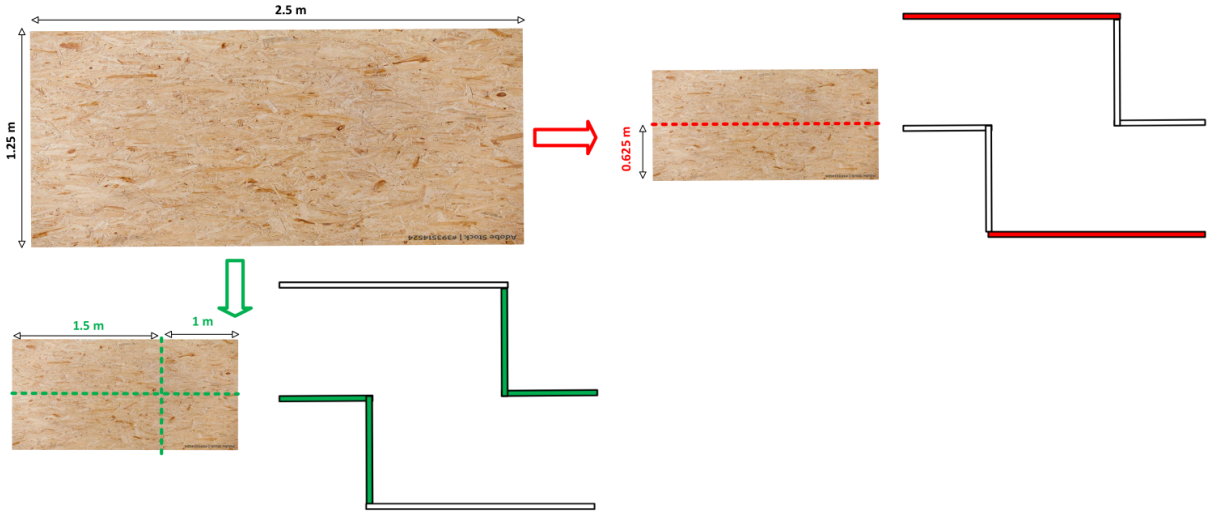


**Figure 3.15:** *Construction of the circuit's side panels. Only two raw boards are needed.*

As mentioned earlier, in our setup, all the blocks in the circuit are ramps with the geometry shown in Figure 3.16. The limiting factors here are the ramp height and slope,

which, according to the manufacturer's specifications (see Table 2.3), should not exceed heights of 18 cm and slopes of 30 degrees. For safety reasons, and to accommodate robots with lower capabilities than the Aliengo, we chose not to push these limits. Consequently, the ramps have a height of 15 cm and a square base of 75 cm, resulting in a slope of 11.31 degrees."

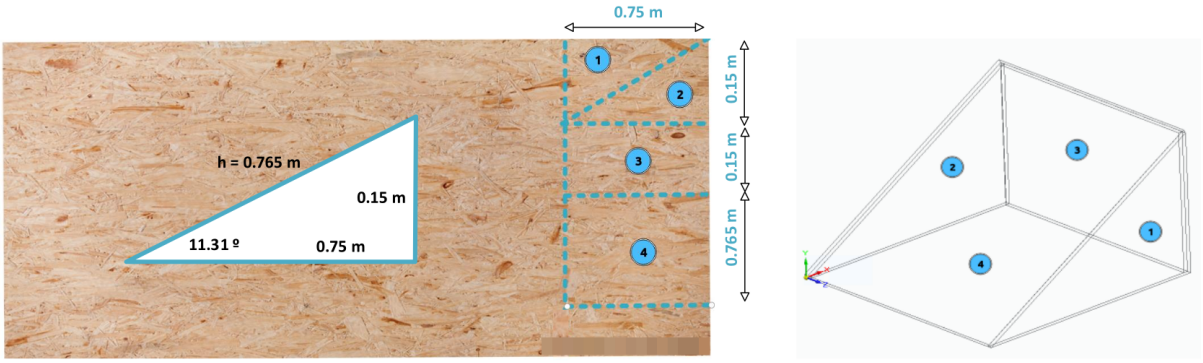


**Figure 3.16:** *Cutting areas for the construction of square base ramps.*

Figure 3.17 shows the four designs constructed in reality, corresponding to the 3D models previously shown in Figure 3.13.

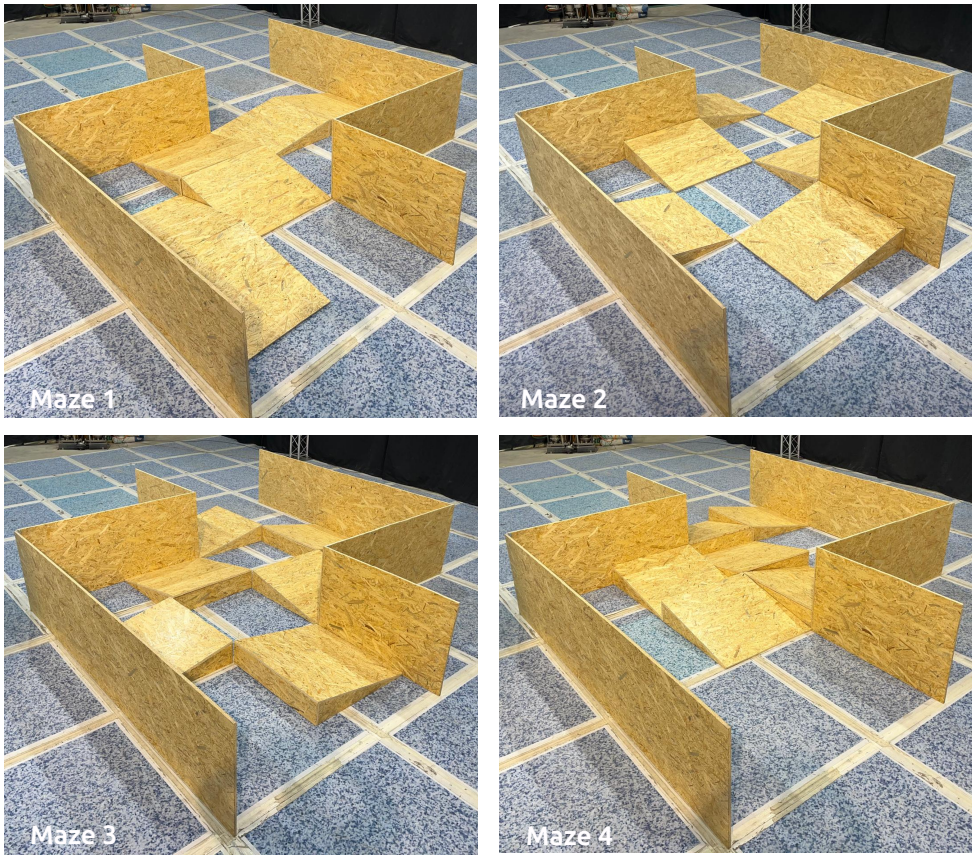


**Figure 3.17:** *Final result of the test circuit constructed using OSB panels.. Due to its modular design, four possible block arrangements are shown. The digital counterpart is depicted in Figure 3.13.*

Finally, Figure 3.18 presents photographs taken during the construction of the ramps,

with dimensions provided for clarity. Note that the floor tiles measure 1 meter square, offering a helpful reference for visualizing the size of the ramps. The construction took place at the CAR-Arena facilities within the School of Industrial Engineering at the Technical University of Madrid. It is worth noting the benefits of using chipboard for this project. Firstly, its cost is lower due to its widespread use in construction. Additionally, its rough texture makes it particularly suitable for locomotion tasks with robots. Finally, the wood joints were easily assembled, and when a sufficient number of screws are used, the structure's robustness is also ensured.


**Figure 3.18:** *Images taken during the construction of the ramps, with the actual dimensions highlighted in red. We have made an effort to provide visual references to aid in accurately visualizing the geometries. Notably, the floor tiles are 1 meter square, offering a clear reference for scale*

The modular design allows for the creation of various patterns or geometries to test different aspects of robot locomotion. For instance, while circuit 2 (maze 2) encourages

the use of the center of the corridor, circuit 3 (maze 3) seems to discourage it. Due to limited wood resources, only six ramps have been built. While this limitation could be addressed in future work, the remaining spaces can be utilized for other types of obstacles or tests.

It is important to note that the results obtained have been highly satisfactory and hold significant relevance for future tests and potential use cases involving quadrupeds or other types of robots.

# 4 RESULTS

This chapter consolidates the results and key contributions from the conducted research. The content is organized into two sections. Section 4.1 details the initial training of the locomotion policy in Isaac Gym, covering the simulations conducted, the hyperparameter study, and the best training outcomes. Section 4.2 presents the tests conducted using the default gait patterns of the Aliengo robot in the constructed test circuit.

## 4.1 Training Optimization

First, we conducted a baseline simulation using default settings for training parameters, rewards, hyperparameters, environments, and controllers. Next, we carried out an extensive hyperparameter study. Finally, we present the newly proposed training parameters and compare the resulting simulations. The section concludes with a list of outstanding tasks and a discussion of potential issues that may arise as the framework evolves.

### 4.1.1 Baseline Simulations

Using the default ViNL configuration [8], we replicated the training processes of phases 1 and 2, which we will refer to as `Rough` and `Obstacles`, respectively. In the first phase, training occurs in a multipurpose environment, while in the second phase, the terrain is flat but includes obstacles of varying heights and lengths. These phases are consecutive, meaning the training from the first phase is further refined during the obstacle avoidance task in the second phase. As noted by [8], this approach is advantageous because the gait patterns developed after training on `Rough` are more natural than if training had been initiated directly on `Obstacles`. This outcome is influenced by the reward structure, which remains positive for `Rough`, but with `Obstacles` adding a significant penalty for foot contact with the cluttered blocks (see Equation 4.1.1).

$$r_{\text{contact}} = \begin{cases} -1 & \text{if any leg in contact} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

To quantify and visualize the training process, the average reward was plotted. For each episode, the reward was calculated as the mean of all the rewards obtained by the 1024 individuals. Following the recommendations of [103], the simulations were repeated at least 5 times. Having a set of simulations allowed us to plot the range between the maximum and minimum rewards for each episode, the mean reward across simulations, and the moving average with a window of 20 episodes. In Figure 4.1, the light blue area represents the range of rewards, the dark blue line shows the mean rewards, and the black line indicates the moving average. Additionally, the overall mean is displayed in red, and the maximum mean reward achieved is highlighted. Figure 4.1 presents the results for both training phases, with each simulation run 10 times for a total of 5000 episodes per stage.

The graph on the left shows that **the training in the first phase has not yet converged after** 5000 **iterations, exhibiting an almost linear growth**. The wide range

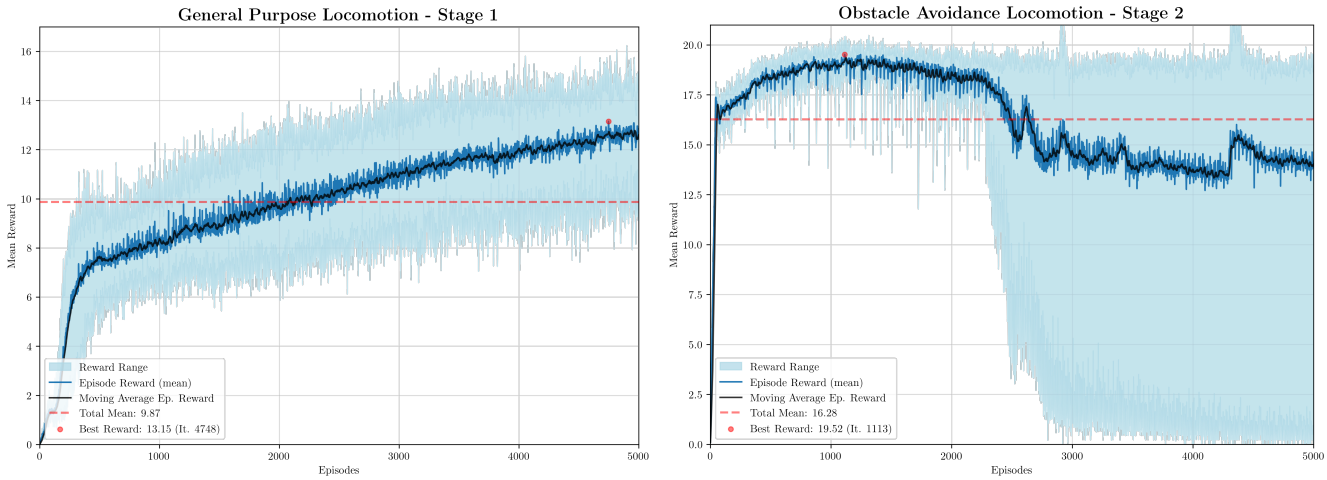


**Figure 4.1:** *Training results using the default ViNL parameters over* 5000 *iterations. The graph on the left shows the first training phase (`Rough`), while the graph on the right depicts the second training phase (`Obstacles`).*

of results across different simulations further emphasizes the need for multiple training runs in this case, allowing for the selection of the policy with the highest reward. In contrast, the graph on the right depicts **the training of the second phase**, where the model has already specialized and **reached its maximum reward within** 1000 **iterations**. However, **the remaining 4000 iterations appear to be counterproductive**, generally leading to overfitting, as indicated by the widening light blue band and the subsequent decrease in the average reward.

To check the convergence of the first training phase, the simulation was extended to 10000 episodes. The results, withoud reruns, are shown in Figure 4.2, where the light blue range is no longer present, and the blue line represents the reward for a single simulation.

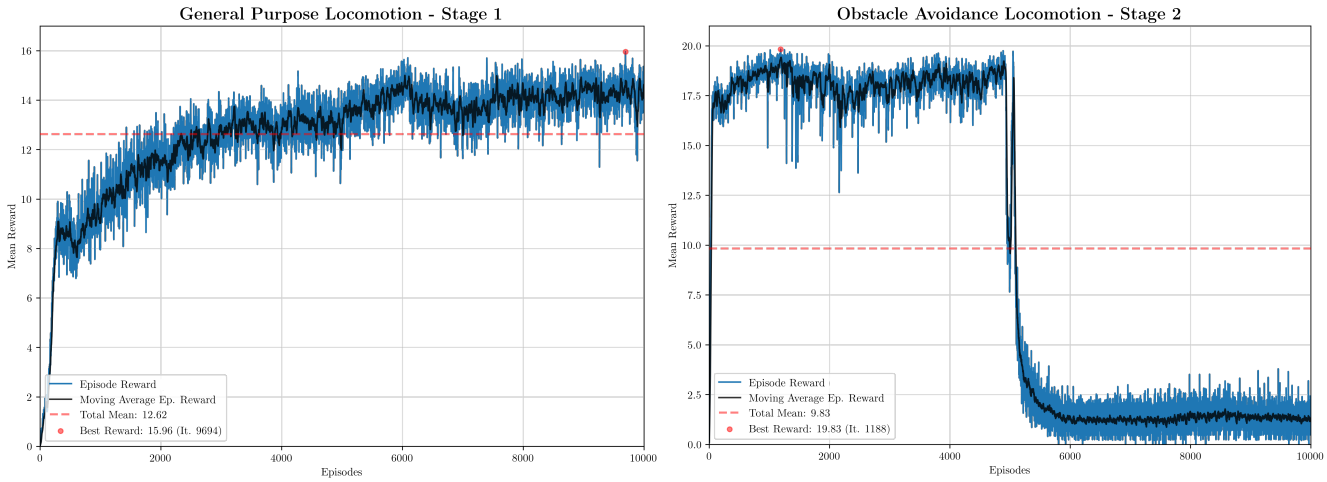


**Figure 4.2:** *Training results using the default ViNL parameters over* 10000 *iterations.*

In the left graph of Figure 4.2, it can be seen that **the first training phase reaches convergence after** 6000 **iterations**, achieving an average reward of approximately 14. On the right, the second graph (training phase 2) shows that the policy specializes and reaches its maximum reward after just 1000 iterations, consistent with Figure 4.1. Ad-

ditional episodes do not improve performance and eventually lead again to overfitting. Therefore, **for `Rough`, 6000 to 6500 episodes are sufficient for convergence, while for `Obstacles`, 1000 to 1500 episodes are adequate for specialization and achieving the maximum reward**.

After completing both trainning stages, the trained weights can be loaded to perform inference, allowing to visualize the execution of the policies through the simulator. Figure 4.3 shows the results for different training sessions.

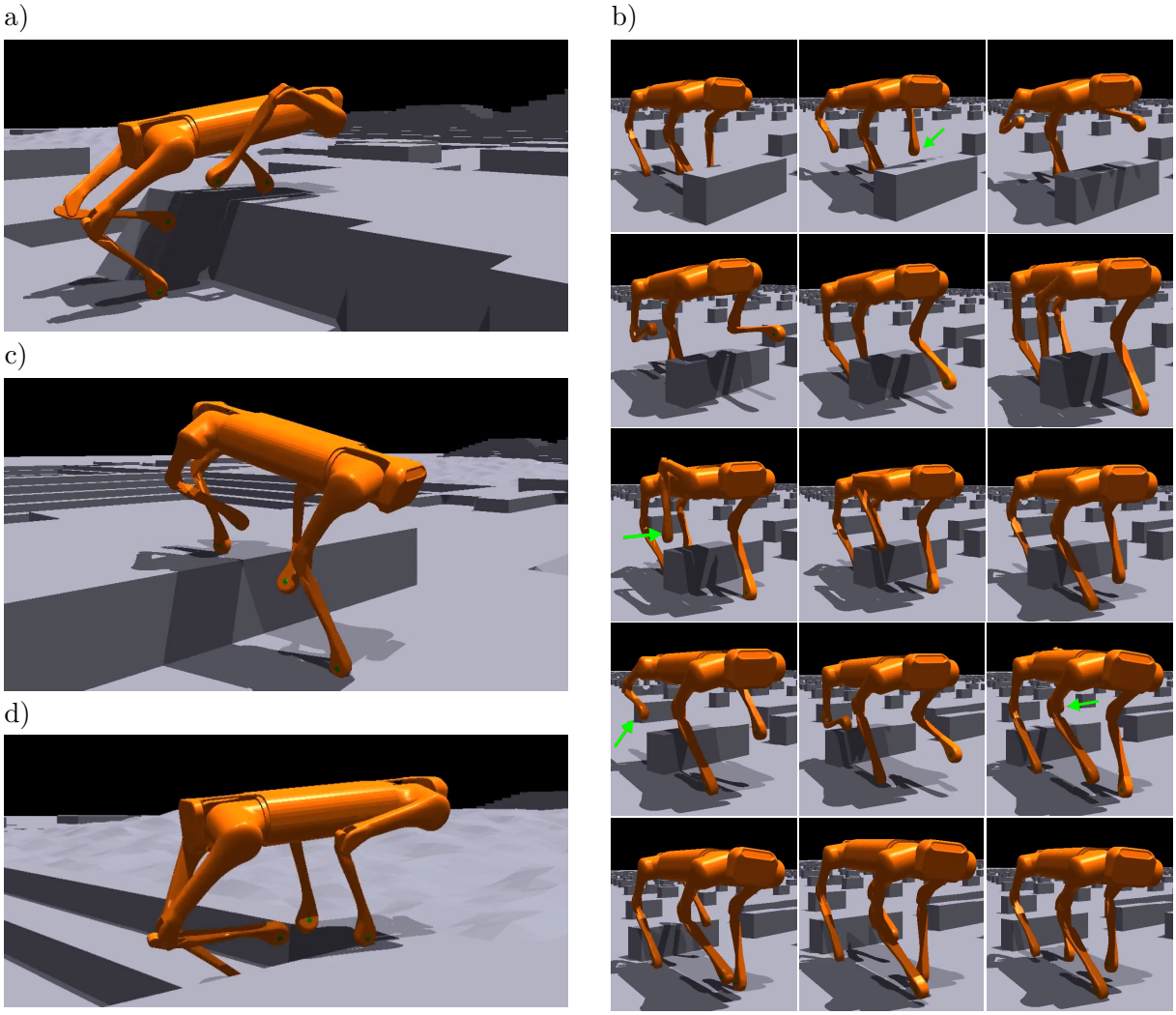


**Figure 4.3:** *Baseline behavior visualized with Isaac Gym during inference of trained policies for both stages. On the left: a) shows the robot climbing a steep step over 25 cm high, c) shows the robot stepping backward, and d) shows the robot transitioning from rough terrain to a staircase. On the right: b) presents a series of snapshots highlighting the learned skill of obstacle avoidance.*

Images a), c), and d) in Figure 4.3 depict challenging situations where the robot responds effectively. Specifically, a) shows the robot climbing a steep step over 25 cm high; c) also shows the robot ascending a step, but this time backward (which is even more difficult); and d) demonstrates the robot transitioning from rough terrain to a staircase, again moving backward. These three images correspond to the simulation on the left side of

Figure 4.2, specifically to the weights obtained after episode 7400, with an average reward of 15.35.

On the other hand, the set of images in Figure 4.3.b) shows the robot evading an obstacle by sequentially abducting all four legs to move over it. The moments when each of the four legs lifts to clear the obstacle are highlighted with a gree arrow pointing the associated leg. These images correspond to the simulation on the right side of Figure 4.1, specifically to the weights obtained after episode 1100, with an average reward of 20.22.

In general, **the training performed in this first phase successfully produced valid locomotion policies for both `Rough` and `Obstacles`** (see previous Footnote 12). The default hyperparameters proposed by Legged Gym (see Table 3.1) were used. Regarding simulation times, **with the `RTX 4060 GPU` (laptop version) used, the average time was about 10 minutes for** 1000 **episodes, simulating** 1024 **robots in parallel**. This means it would take approximately 1 hour and 20 minutes to train 6500 episodes for `Rough` and 1500 episodes for `Obstacles`.

### 4.1.2 Hyperparameters Study

As previously discussed in Section 3.2.1, while PPO offers numerous benefits, its primary challenge lies in the complexity of tuning its many hyperparameters. This study aims to provide deeper insights into training locomotion policies in quadrupeds, thereby enhancing our understanding of the underlying issues and maximizing the potential of these simulations. The novelty of this research, which to our knowledge is unmatched in the existing literature, serves as a key motivation for its execution.

Table 4.1 summarizes the two experiments conducted for each set of base hyperparameters, resulting in a total of 20 simulations. The objective was to explore the full spectrum of the hyperparameter range while closely examining the vicinity of the base values. Additionally, two further simulations were performed to assess the impact of modifying the random seed.

| Symbol | Range* | Baseline** | Experiment 1 | Experiment 2 |
|---|---|---|---|---|
| $n_{\text{epochs}}$ | [3, 20] | 5 | 3 | 10 |
| $n_{\text{batches}}$ | [1, 32] | 4 | 2 | 8 |
| $\epsilon$ | [0.1, 0.3] | 0.2 | 0.1 | 0.3 |
| $\gamma$ | [0.8, 0.9997] | 0.99 | 0.95 | 0.9997 |
| $\lambda$ | [0.9, 1.0] | 0.95 | 0.9 | 0.99 |
| $c_{\text{value}}$ | [0.5, 1.0] | 1.0 | 0.5 | 0.75 |
| $c_{\text{entropy}}$ | [0.0, 0.01] | 0.01 | 0.005 | 0.0075 |
| $\alpha$ | [1e-5, 1e-3] | 1e-3 | 5e-4 | 2e-3 |
| $\|g\|_{\text{max}}$ | [0.5, 1.0] | 1.0 | 0.5 | 0.75 |
| $\text{KL}_{\text{desired}}$ | [0.003, 0.03] | 0.01 | 0.005 | 0.0075 |
| seed | $(-\infty, +\infty)$ | 1 | 42 | 789 |

**Table 4.1:** *Hyperparameters experiments for PPO Algorithm. *PPO hyperparameters ranges were obtained from [85]. **Baseline values come from [84].*

To automate the simulation process, a Python script was developed to define the matrix of parameters to be studied. This script generates a unique configuration file, `legged_robot_config.py`, for each parameter, tailored specifically for Legged Gym. This file includes the default simulation values and the relevant hyperparameters. Following this, a Shell script is employed to execute an experiment for each configuration file. The experiment involves training the robot in two phases: Phase 1 (`Rough`) and Phase 2 (`Obstacles`). Each phase is run five times, with 1000 episodes per run, totaling 10 minutes per simulation. As a result, a full experiment takes approximately 1 hour and 40 minutes to complete. We have not conducted longer simulations (e.g., 6500 episodes to full convergence) to prevent the study from becoming temporarily unfeasible.

During the first phase of training, the seed is consistently set to one (with the exception of the two experiments specifically examining seed variation). For the second phase, however, the seed is randomized for each new simulation. The network weights, are saved every 50 episodes, providing 20 models by the end of a simulation. The best-performing model from the first phase is then saved and used to initialize the actor's weights in the second phase. However, this approach has proven problematic, as certain locomotion anomalies (unnatural gaits) have been carried over during retraining in the second phase. In retrospect, it would be advisable to use different weights for each simulation in the second phase to promote variability and prevent undesirable behaviors from being inherited. This will be a key consideration for future studies. Both the python and shell script developed for the automatic hyperparameter study can be found in our GitHub repository (see previous footnote 12).

The 11 hyperparameters examined in this study are detailed below. For each hyperparameter, we present the combined results from both the `Rough` and `Obstacles` phases, as outlined in Table 4.1, alongside the baseline performance. We will first provide a brief discussion of the significance of each hyperparameter and its general impact on training through reinforcement learning. Following this, we will analyze its specific effects on the locomotion task, drawing insights from the accompanying graphs. To maintain conciseness, the figures detailing each individual simulation are included in the annexes (please refer to Figures A.1 and A.2).

- **`num_learning_epochs`**: this parameter controls the number of times the neural network is updated using the same batch of data each iteration. Increasing this value can improve the quality of the learned policy but may lead to overfitting if set too high.

  Regarding the training in `Rough`, as can be seen in Figure 4.4 (left), a higher number of epochs accelerates initial training (achieving a higher reward in fewer episodes), but this advantage is matched by the baseline within 1000 iterations. On the other hand, a lower number of epochs appears to lead to a wider range of results, indicating greater variability or instability in the training process. **We can conclude that in this first stage, reducing the number of epochs is detrimental, while increasing it does not seem to improve training outcomes (at least within the first** 1000 **iterations)**.

  In contrast, for the training in `Obstacles` (see Figure 4.4 right), the difference is more pronounced, with the baseline value yielding the best results.

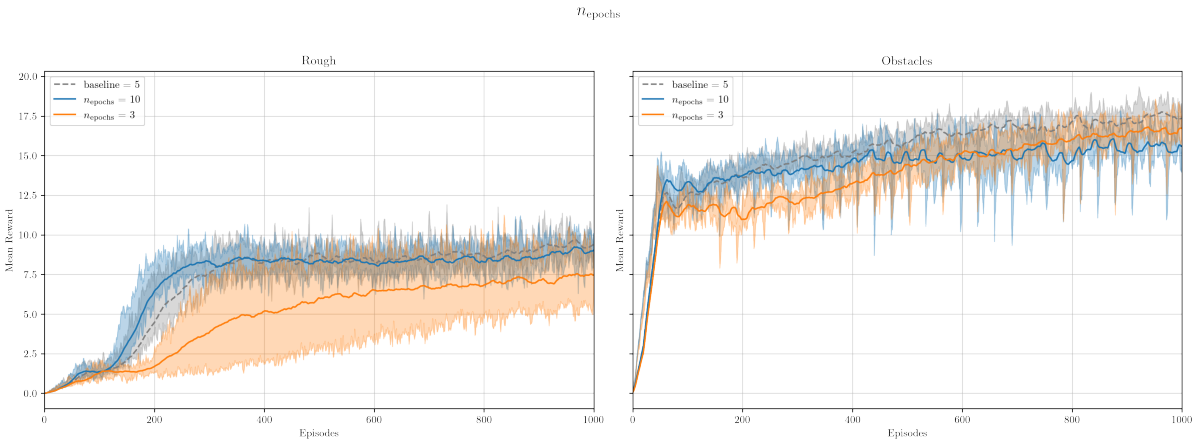


**Figure 4.4:** *Comparison of the number of learning epochs with the baseline for the rough and obstacles training stage: The left graph shows* **`Rough`** *training with the baseline value (5) in a dashed grey line, compared to two new experiments, blue (10) and orange (3). The right graph presents the same comparison for the* **`Obstacles`** *stage.*

- **`num_mini_batches`**: it determines the number of smaller batches that the training data is split into for more granular updates. More mini-batches can lead to more stable and reliable updates but require more computational resources.

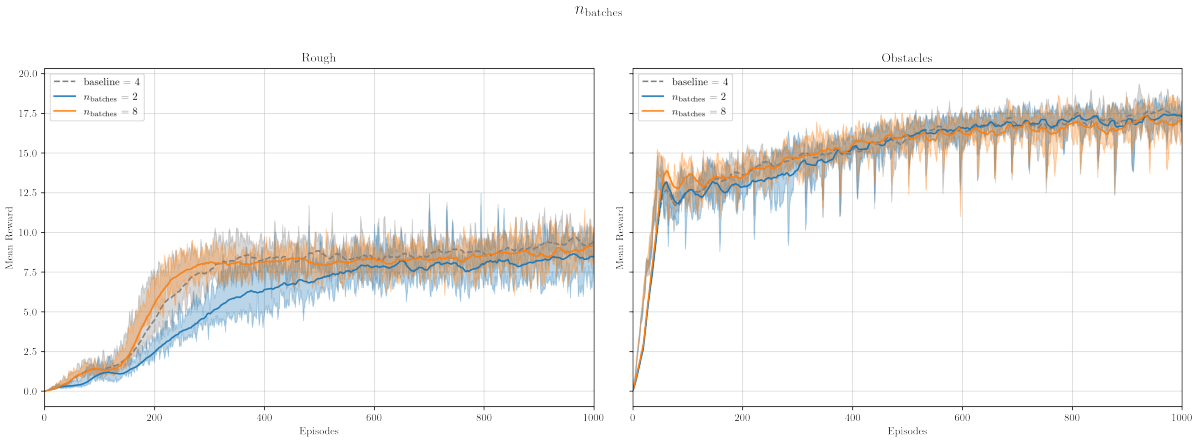


**Figure 4.5:** *Comparison of the number of mini batches with the baseline for the rough and obstacles training stage: The left graph shows* **`Rough`** *training with the baseline value (4) in a dashed grey line, compared to two new experiments, blue (2) and orange (8). The right graph presents the same comparison for the* **`Obstacles`** *stage.*

**For the `Rough` phase**, as can be seen in Figure 4.5 (left), a higher number of mini-batches results in smoother learning curves and faster progress during the first 400 episodes. Conversely, a smaller number of mini-batches slows down the learning process. Interestingly, the smoothness of the curves does not significantly differ between the baseline and the reduced mini-batch configuration. However, after 1000 iterations, **the baseline appears to provide a more optimal number of mini-batches in terms of the reward achieved**.

**In the `Obstacles` phase**, as can be seen in Figure 4.5 (right), the differences between the curves are less pronounced. Based on the graphs, **modifying the baseline number of mini-batches does not offer a clear advantage**.

- **`clip_param`**: this limit the size of the policy updates to prevent large, destabilizing updates. Clipping helps maintain the stability of training by ensuring updates do not deviate too much from the previous policy.

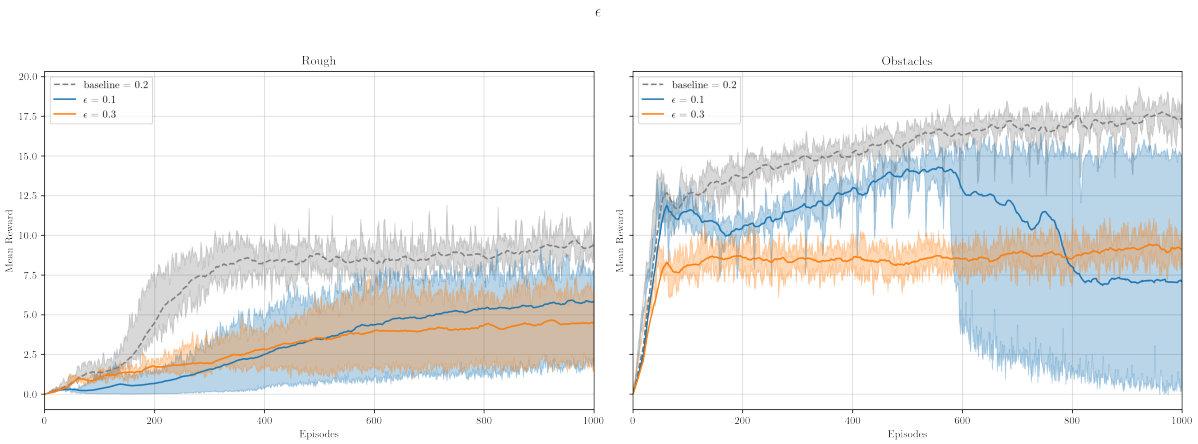


**Figure 4.6:** *Comparison of the clip parameter with the baseline for the rough and obstacles training stage: The left graph shows* **`Rough`** *training with the baseline value (0.2) in a dashed grey line, compared to two new experiments, blue (0.1) and orange (0.3). The right graph presents the same comparison for the* **`Obstacles`** *stage.*

**In the `Rough` stage**, as can be seen in Figure 4.6 (left), **the baseline $\epsilon$ value clearly outperforms the two new values explored**. The alternative values exhibit slower learning and greater variability, likely due to suboptimal exploration/-exploitation balance, resulting in widely different rewards across the various simulations. **Similarly, in the `Obstacles` training** (refer to Figure 4.6 right), **the baseline $\epsilon$ value again shows a clear advantage.** A lower Clip parameter in this phase leads to a wide range of reward values in the final 400 episodes, as it restricts the agent's exploration capacity, causing it to become trapped in suboptimal policies.

- **`gamma`**: This parameter represents the discount factor, which influences how much weight is placed on future rewards during the training process. Higher values of gamma give more importance to long-term rewards, while lower values tend to focus on short-term gains.

  **In the `Rough` phase of training**, as can be seen in Figure 4.7 (left), **a lower value of $\gamma$ results in better long-term learning**, despite slower progress during the first 400 iterations. The baseline value, while showing less variability and faster initial learning, ultimately achieves a lower reward after 1000 iterations. On the other hand, a $\gamma$ **value higher than the baseline proves to be catastrophic**, as it causes the model to overly prioritize future rewards, leading to the neglect of immediate ones and effectively blocking learning.

  **In the `Obstacles` phase**, as it is shown in Figure 4.7 (right), **the difference between lower $\gamma$ values and the baseline is less pronounced than in the previous phase**. Here, a lower $\gamma$ value introduces greater instability in training, characterized by large fluctuations. **However, by** 1000 **iterations, this lower value begins to outperform the baseline**. Conversely, a higher $\gamma$ value remains detrimental, as it continues to hinder learning.

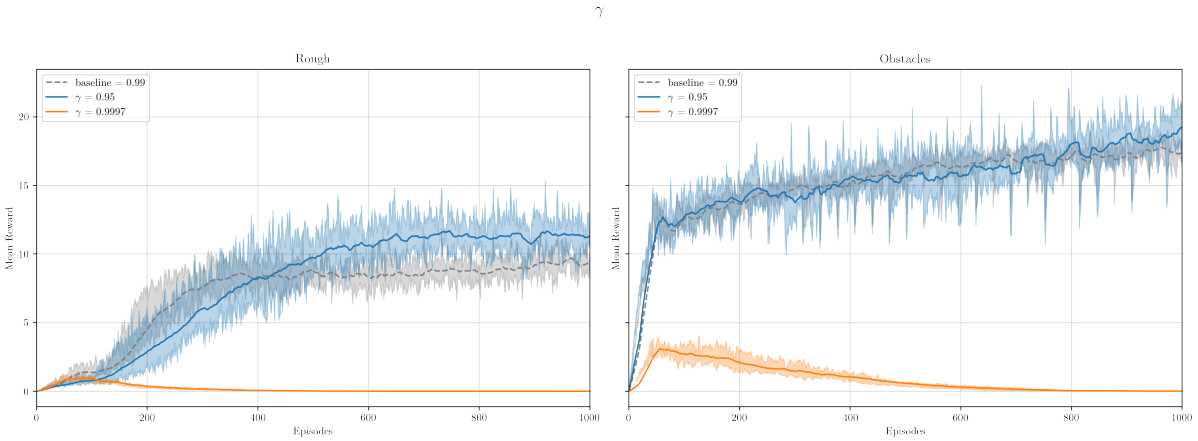


**Figure 4.7:** *Comparison of gamma with the baseline for the rough and obstacles training stage: The left graph shows* `Rough` *training with the baseline value (0.99) in a dashed grey line, compared to two new experiments, blue (0.95) and orange (0.9997). The right graph presents the same comparison for the* `Obstacles` *stage.*

- **`lam (GAE)`**: it is a parameter used in Generalized Advantage Estimation to control the trade-off between bias and variance in advantage estimation. Proper tuning of lambda helps in stabilizing training by smoothing the advantage function, balancing between bias and variance[13].

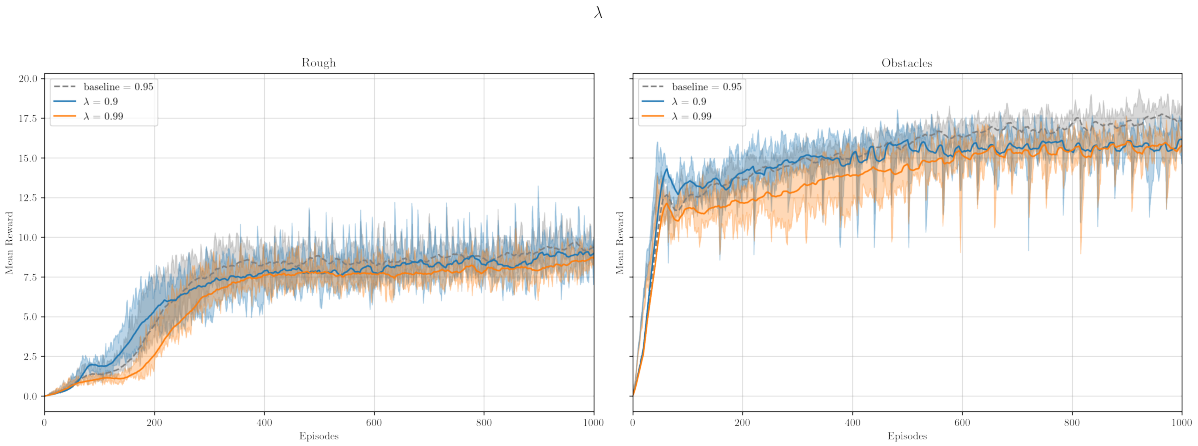


**Figure 4.8:** *Comparison of lambda (GAE) with the baseline for the rough and obstacles training stage: The left graph shows* `Rough` *training with the baseline value (0.95) in a dashed grey line, compared to two new experiments, blue (0.9) and orange (0.99). The right graph presents the same comparison for the* `Obstacles` *stage.*

Figure 4.8 (left) illustrates the comparison of $\lambda$ values during the `Rough` training phase. The baseline slightly outperforms the two alternative values, with the smaller value exhibiting more spiky (unstable) behavior. Figure 4.8 (right) presents the same comparison for the `Obstacles` stage. In this case, the spiky behavior is

[13]This concept reflects the degree to which the agent depends on its current value estimate when updating it. Lower values mean the agent leans more heavily on its existing estimate, which can introduce a high bias. In contrast, higher values place greater emphasis on the actual rewards received from the environment, which can lead to high variance. This parameter balances these two factors, and selecting the optimal value can result in a more stable training process. Obtained from [104] GitHub documentation.

more pronounced for both new values, while the baseline outperforms the others, particularly in the final 400 episodes.

- **`value_loss_coef`**: this controls the weight of the value function loss in the overall loss function used for training. Balancing this helps in ensuring that the value function is learned adequately without dominating the policy learning.

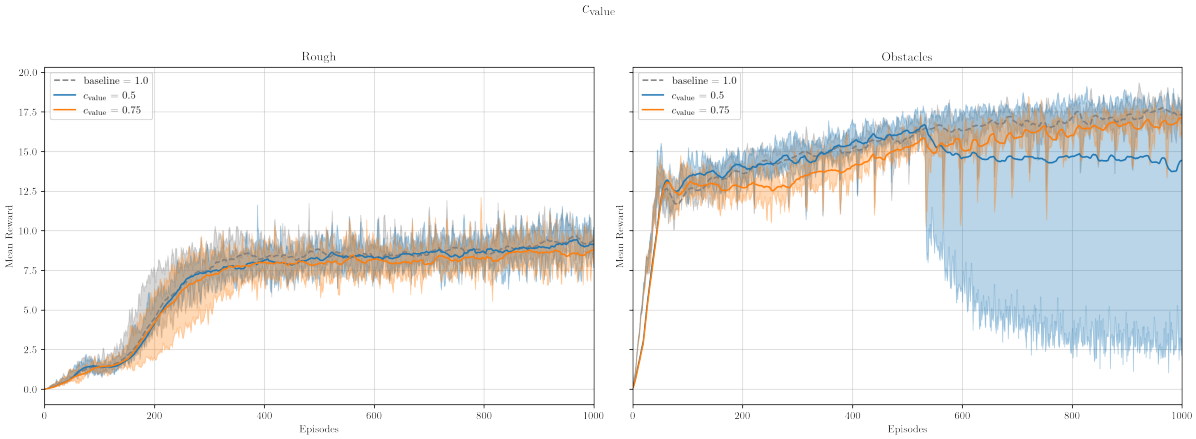


**Figure 4.9:** *Comparison of the value loss coefficient with the baseline for the rough and obstacles training stage: The left graph shows* **`Rough`** *training with the baseline value (1.0) in a dashed grey line, compared to two new experiments, blue (0.5) and orange (0.75). The right graph presents the same comparison for the* **`Obstacles`** *stage.*

In the `Rough` phase, as shown in Figure 4.9 (left), there is no significant difference between the baseline ($c_{\text{value}} = 1$) and a value half the baseline ($c_{\text{value}} = 0.5$). However, slightly lower values ($c_{\text{value}} = 0.75$) exhibit somewhat worse performance compared to the other two. This suggests that **the baseline value remains the most suitable choice for this phase**.

For the `Obstacles` phase (see Figure 4.9, right), the results differ. **The baseline value again achieves the highest reward, while the two alternative values perform slightly worse**. Notably, similar to the behavior observed with $\epsilon$ values of 0.1, halving the value loss coefficient results in increased variability during the final 500 iterations.

- **`entropy_coef`**: it encourages exploration by adding an entropy term to the loss function. Higher entropy leads to more exploration of the action space, which can be crucial for discovering optimal policies, especially in environments with sparse rewards.

  Figure 4.10 (left) compares the `Rough` training phase for different values of the entropy coefficient. **Within the first** 1000 **iterations, a value half the baseline ($c_{\text{entropy}} = 0.005$) achieves better training performance**. For slightly lower values ($c_{\text{entropy}} = 0.0075$), the behavior is very similar to the baseline, with no clear differences.

  In contrast, Figure 4.10 (right) shows that **during the `Obstacles` training phase, the difference between the new values and the baseline becomes more pronounced, with the alternative values yielding better rewards.** This can be attributed to the varying demands of the training scenarios. In the first phase,

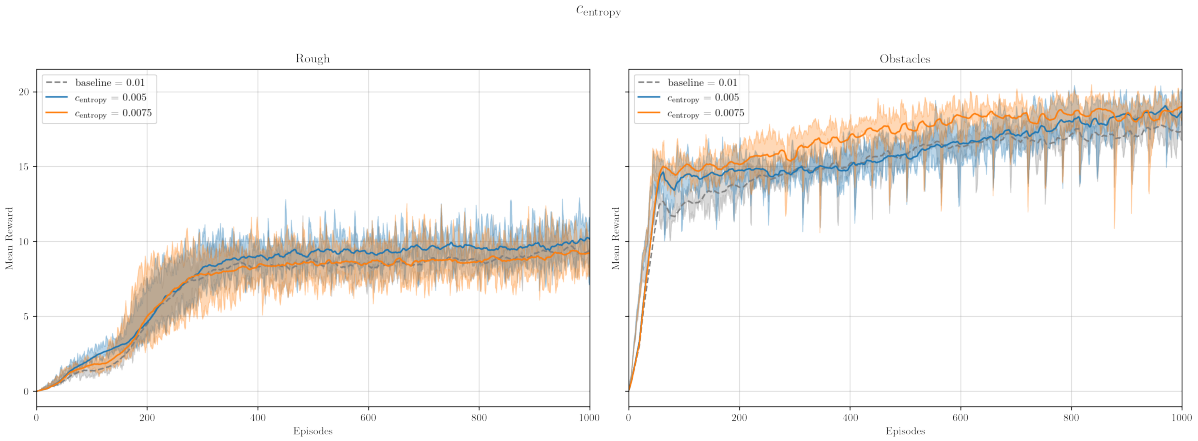


**Figure 4.10:** *Comparison of the entropy coefficient with the baseline for the rough and obstacles training stage: The left graph shows* `Rough` *training with the baseline value (0.01) in a dashed grey line, compared to two new experiments, blue (0.005) and orange (0.0075). The right graph presents the same comparison for the* `Obstacles` *stage.*

where there are multiple terrains with randomized configurations, more extensive exploration is necessary and beneficial (**additional iterations would be needed to determine if values lower than the baseline are indeed more suitable**). However, in the second phase, the environment's novelty is significantly reduced, making it more reasonable to decrease exploration in order to converge to an optimal policy for obstacle avoidance.

- `learning_rate`: it controls the step size during each iteration as the process advances towards minimizing the loss function. A well-chosen learning rate can significantly accelerate training: too high a rate can cause instability, while too low a rate can lead to slow convergence. t is common for the learning rate to be dynamic, meaning it starts high to facilitate initial learning but gradually decreases as the model converges, ensuring it reaches an optimal solution.

  In the `Rough` phase, as shown in Figure 4.11 (left), different learning rate values do not significantly impact the reward compared to the baseline. A higher learning rate ($\alpha = 0.002$) introduces greater variability, resulting in a wider range between the maximum and minimum rewards per episode. In contrast, for the `Obstacles` phase (see Figure 4.11, right), there is a more noticeable difference between the new values and the baseline. Specifically, a lower learning rate ($\alpha = 0.0005$) consistently yields higher rewards over the 1000 iterations, while a higher rate ($\alpha = 0.002$) shows no significant deviation from the baseline. However, in the last 200 iterations, all three profiles converge, displaying approximately the same reward per episode. **Overall, it can be concluded that the baseline learning rate is adequate for training in both stages**.

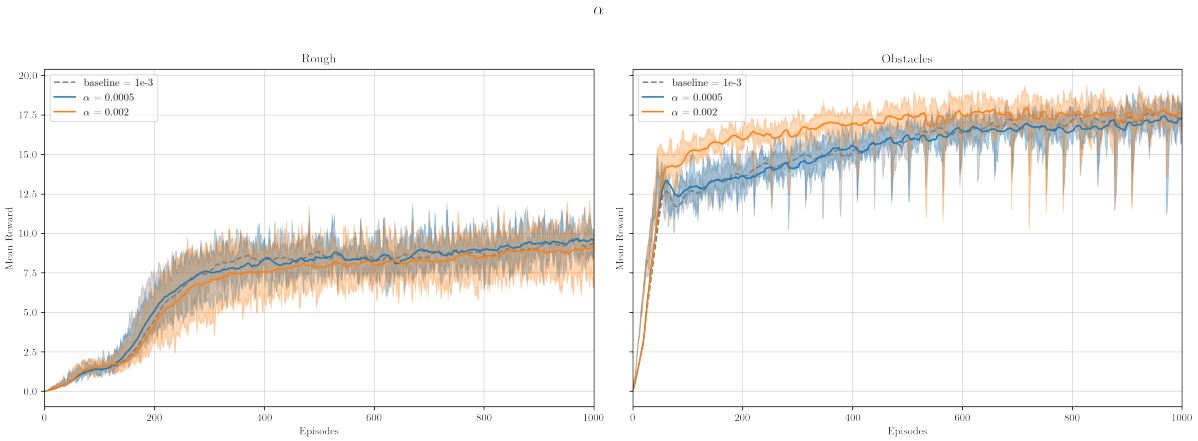


**Figure 4.11:** *Comparison of the number of learning epochs with the baseline for the rough and obstacles training stage: The left graph shows* `Rough` *training with the baseline value (0.001) in a dashed grey line, compared to two new experiments, blue (0.0005) and orange (0.002). The right graph presents the same comparison for the* `Obstacles` *stage.*

- **`max_grad_norm`**: this parameter limits the maximum value of the gradients during backpropagation. Gradient clipping helps prevent the "exploding gradients" problem, ensuring stable updates during training, especially with deep networks.

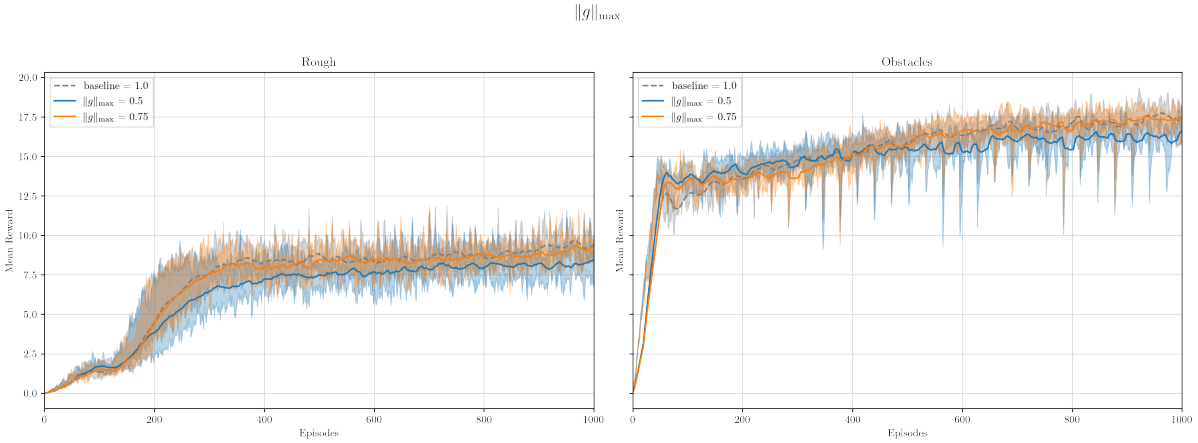


**Figure 4.12:** *Comparison of the number of learning epochs with the baseline for the rough and obstacles training stage: The left graph shows* `Rough` *training with the baseline value (1) in a dashed grey line, compared to two new experiments, blue (0.5) and orange (0.75). The right graph presents the same comparison for the* `Obstacles` *stage.*

Figure 4.12 (left) compares different maximum gradient norm values during the `Rough` phase. It is evident that using a value half the baseline ($\|g\|_{\max} = 0.5$) leads to significantly worse performance compared to the baseline. Meanwhile, a slightly lower value ($\|g\|_{\max} = 0.75$) shows no clear difference over the 1000 iterations of the experiment. Based on these observations, **it is advisable to maintain the baseline value for future simulations**.

Figure 4.12 (right) presents the results for the `Obstacles` phase, showing a very similar pattern. The conclusions drawn for the `Rough` phase apply here as well.

- **`desired_kl`**: it sets a target value for the Kullback-Leiber (KL) divergence between the new and old policies during training. This helps in controlling the policy updates to be conservative, preventing drastic changes that could destabilize the learning process.

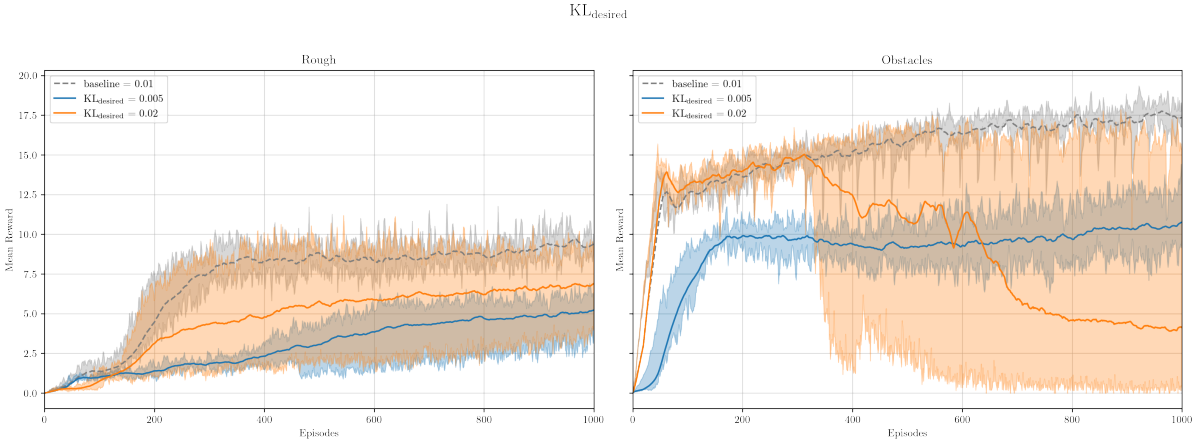


**Figure 4.13:** *Comparison of the desired KL with the baseline for the rough and obstacles training stage: The left graph shows* **`Rough`** *training with the baseline value (0.01) in a dashed grey line, compared to two new experiments, blue (0.005) and orange (0.02). The right graph presents the same comparison for the* **`Obstacles`** *stage.*

For both the `Rough` and `Obstacles` phases, as can be seen in Figure 4.13, the results lead to similar conclusions. Higher-than-baseline values result in greater variability, likely due to more aggressive policy updates. While this approach is intended to accelerate training, it often produces instability and, on average, yields suboptimal and largely non-convergent outcomes. In the `Obstacles` phase, this increased variability is even more pronounced for certain hyperparameters. Conversely, values lower than the baseline result in insufficient policy updates, causing significantly slower training in the `Rough` phase and suboptimal policies in the `Obstacles` phase. In summary, **the baseline values are appropriate for this training; altering them offers no advantage.**

- **`seed`**: it initializes the random number generator used in training. Setting a seed ensures reproducibility of results, allowing the same model and environments setup to yield consistent outcomes across different runs.

  The simulations presented in Figure 4.14 explore the impact of different seeds on training outcomes. In the first phase (`Rough`), there is no significant difference between the seeds, although the Baseline seed performs slightly better than the others. In the second phase (`Obstacles`), the differences between the simulations become more pronounced, with the Baseline seed once again outperforming the other two. Overall, we can conclude that the system is not significantly affected by changing the seed. However, it may be beneficial to conduct exploratory simulations during training to identify seeds that could potentially yield better results.

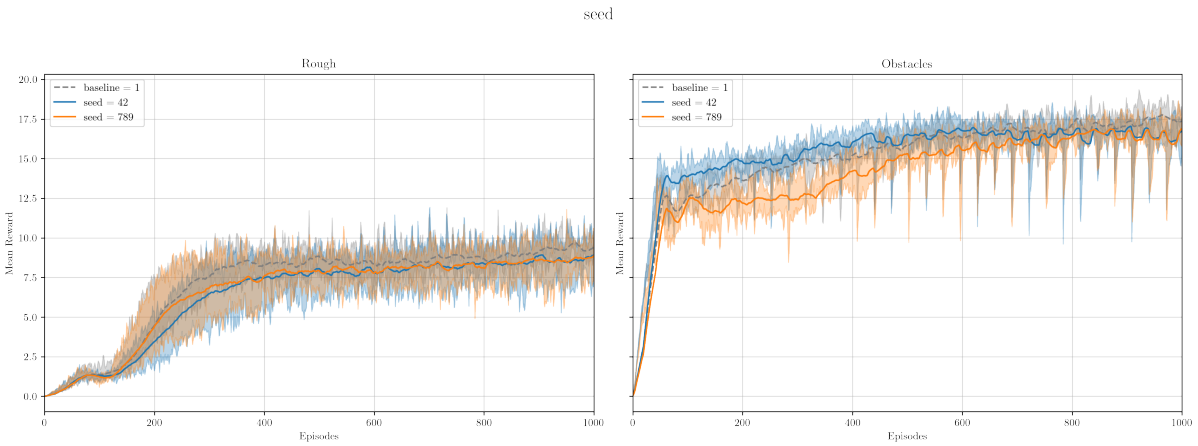


**Figure 4.14:** *Comparison of different seeds with the baseline for the rough and obstacles training stage: The left graph shows* `Rough` *training with the baseline value (1) in a dashed grey line, compared to two new experiments, blue (42) and orange (789). The right graph presents the same comparison for the* `Obstacles` *stage.*

### 4.1.3 Best Training Parameters

After completing the hyperparameter study, Table 4.2 summarizes the global mean reward values for each experiment, highlighting the percentage improvement or decline compared to the baseline:

| Symbol | Baseline* | Experiment 1 | | | Experiment 2 | | |
|---|---|---|---|---|---|---|---|
| | | Value | Rough (%) | Obstacles (%) | Value | Rough (%) | Obstacles (%) |
| $n_{\text{epochs}}$ | 5 | 3 | -30.24 | -9.52 | 10 | +1.25 | -6.25 |
| $n_{\text{batches}}$ | 4 | 2 | -17.34 | -1.10 | 8 | -1.42 | -0.43 |
| $\epsilon$ | 0.2 | 0.1 | -54.27 | -29.88 | 0.3 | -57.13 | -44.70 |
| $\gamma$ | 0.99 | **0.95** | **+10.99** | +0.80 | 0.9997 | -97.67 | -94.02 |
| $\lambda$ | 0.95 | 0.9 | -3.49 | -3.37 | 0.99 | -12.67 | -9.19 |
| $c_{\text{value}}$ | 1.0 | 0.5 | -2.52 | -6.30 | 0.75 | -5.93 | -4.97 |
| $c_{\text{entropy}}$ | 0.01 | **0.005** | **+8.79** | +3.79 | **0.0075** | +0.63 | **+10.26** |
| $\alpha$ | 1e-3 | 5e-4 | +0.66 | -0.54 | **2e-3** | -5.08 | **+7.94** |
| $\|g\|_{\max}$ | 1.0 | 0.5 | -10.99 | -2.00 | 0.75 | -2.18 | -0.17 |
| $\text{KL}_{\text{desired}}$ | 0.01 | 0.005 | -56.64 | -40.83 | 0.02 | -32.82 | -36.72 |
| seed | 1 | 42 | -9.11 | +1.58 | 789 | -6.72 | -6.94 |

**Table 4.2:** *Hyperparameters summary: percentage improvement or decline compared to the baseline for each experiment. All values that show an improvement in the mean are highlighted in green, with those exceeding a 5% improvement emphasized in bold. *Baseline values come from [84].*

As previously mentioned and shown in the Table 4.2, reducing $\gamma$ from 0.99 to 0.95 improves the overall average reward by 11% over the course of 1000 iterations in the first phase (`Rough`). For the `Obstacles` phase, the improvement is more modest at 0.8%. Additionally, lowering the entropy coefficient ($c_{\text{entropy}}$) from 0.01 to 0.005 increases the overall mean reward by 8.79% for `Rough` and 3.79% for `Obstacles`. However, a smaller reduction in $c_{\text{entropy}}$, from 0.01 to 0.0075, produces mixed results: it increases the global mean by 10.26% for `Obstacles`, while only yielding a 0.63% improvement for `Rough` compared to the Baseline.

It is important to consider these values in conjunction with the corresponding graphs. While the average reward may be higher, what ultimately matters is the trend, particularly the behavior in the final iterations and the maximum reward levels reached. In this context, $\gamma$ has the most significant impact, followed by the entropy coefficient ($c_{\text{entropy}}$). However, further validation through a longer study is recommended, ideally 7000 iterations for `Rough` and 2000 for `Obstacles`, to confirm these findings.

After analyzing the impact of hyperparameters on training, the next step is to verify whether the new values indeed enhance performance. To do this, two simulations were conducted (as can be seen in Figure 4.15): one with the baseline hyperparameters (4.15.a) and the other with the newly optimized settings ($\gamma = 0.95$, $c_{\text{entropy}} = 0.005$) (4.15.b), both using 6500 iterations for the first stage (`Rough`) and 1500 iterations for the second stage (`Obstacles`). Each simulation was repeated five times to assess variability and ensure the results are generalizable.

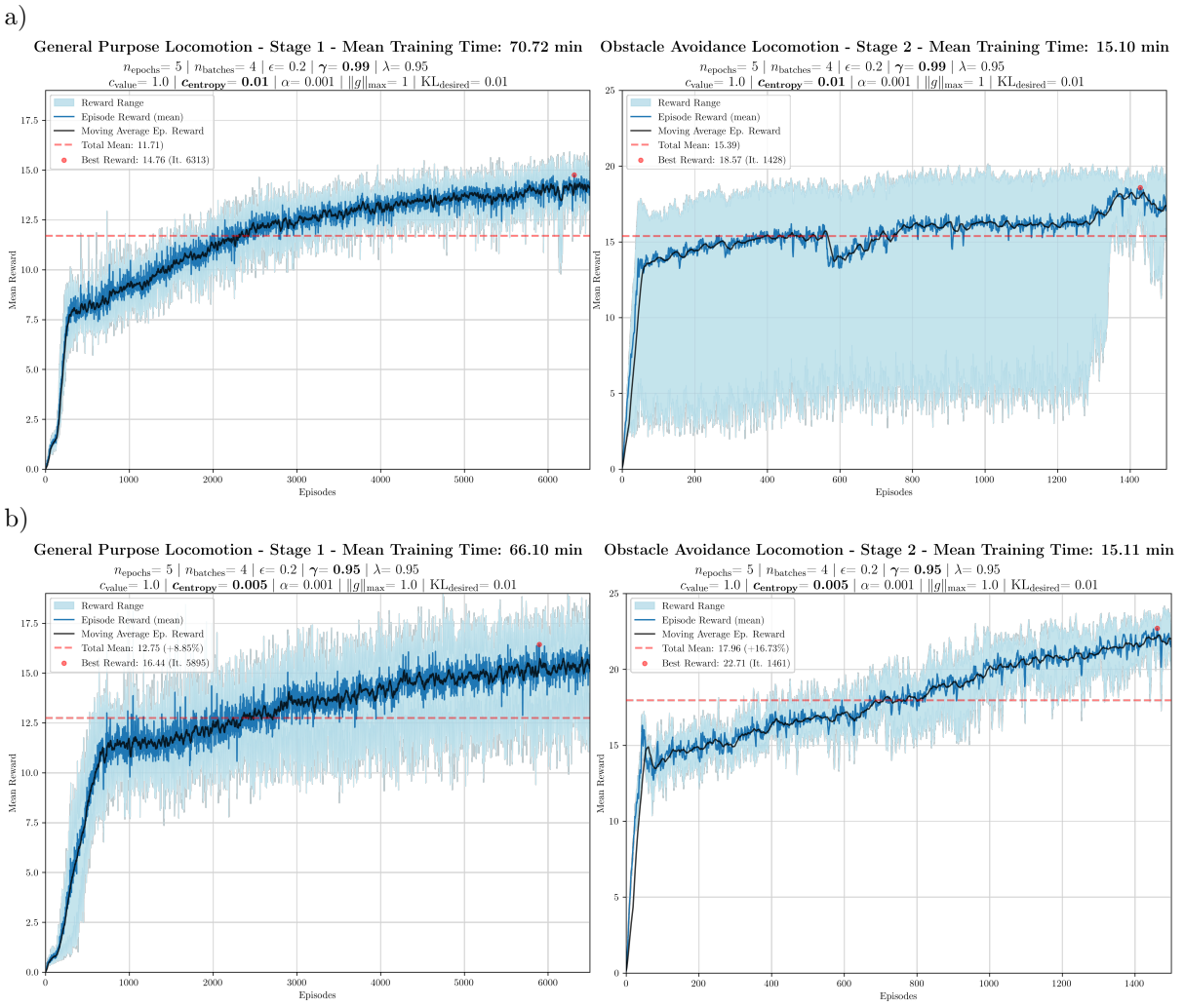


**Figure 4.15:** *Comparison of training results: (a) Performance with baseline method for* `Rough` *and* `Obstacles`*, and (b) Enhanced performance with optimized parameters (highlighted in bold)*

In Figure 4.15, the global average for each stage of the training is plotted. Specifically, in Figure 4.15.b, the increase relative to the overall average of the baseline is calculated, **showing an improvement of +8.85% for `Rough` and +16.73% for `Obstacles`.**

Overall, the training processes are similar, except in the case of `Obstacles`, where the Baseline exhibited significant variability in the training outcomes.

To more effectively visualize the comparison between the baseline training and the newly proposed optimized parameters, a combined graph has been created from Figures 4.15.a and 4.15.b. Figure 4.16 illustrates a consistent improvement throughout most of the training for both `Rough` and `Obstacles`. This indicates that the newly proposed values lead to better training outcomes for the Aliengo quadruped robot in the Isaac Gym training environment.

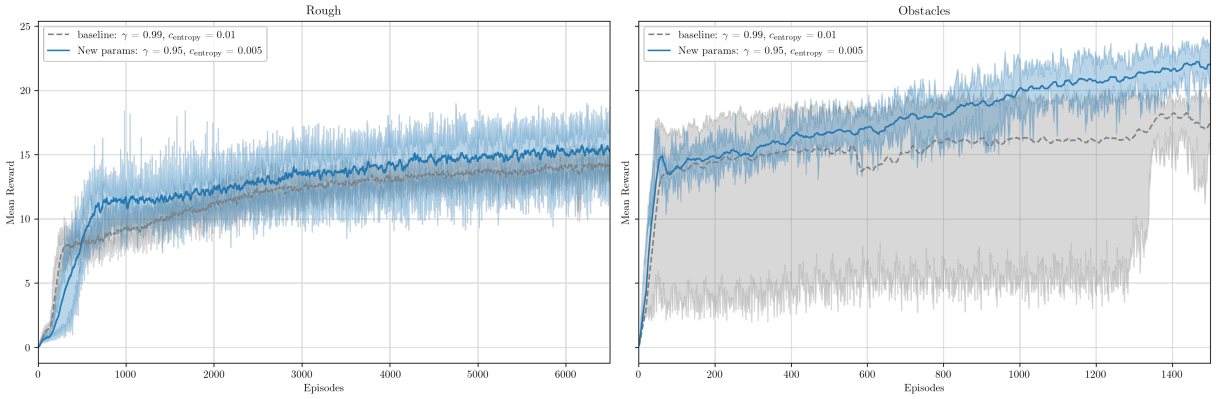


**Figure 4.16:** *Comparison of baseline training and optimized parameters for (a)* `Rough` *and (b)* `Obstacles` *stages, showing consistent improvement due to new values for discount rate ($\gamma$) and entropy coefficient ($c_{entropy}$) hyperparameters.*

### 4.1.4 Outstanding Tasks and Potential Issues

#### *Hyperparameter Study*

As previously mentioned, it would be valuable to reconsider the study of hyperparameters with a higher number of iterations. Increasing the iteration count from 1000 to 6000 for the first phase would generally be necessary, though extending this number to 7000 might prove even more beneficial. For the second phase, a similar extension from 1000 to 2000 iterations could be advantageous. Additionally, while the current study examines hyperparameters individually, it would be worthwhile to explore combinations of pairs or trios of hyperparameters, particularly those likely to be correlated. It would be beneficial to evaluate the accuracy of these conclusions in the context of robots with different morphologies, as the training process is significantly influenced by the specific characteristics of the robot.

Moreover, it is important to verify whether the learning depicted in the graphs (such as the evolution of reward over episodes) accurately reflects the desired behavior of the robots. To illustrate this point, consider a comparison between two simulations with $\gamma = 0.95$ and $\alpha = 0.005$. Although the latter generally exhibits more robust behavior, the graphical reward for the $\gamma = 0.95$ simulation is significantly higher. This discrepancy may indicate that the reward function could benefit from further refinement, as it might not fully capture the desired behaviors. As a result, policies that maximize the reward

may not exhibit the most desirable behaviors, while suboptimal policies could potentially align better with the intended outcomes despite lower rewards.

While we do not believe that the reward function designed in [7] is necessarily inadequate, it would be prudent to remain aware of this possibility. It is essential to ensure that the graphs truly optimize the reward and that the actual behavior observed in simulations aligns with the graphical representations.

***New Architectures***

As discussed in Section 3.3, it remains crucial to replicate the work of the Legged Robotics group at ETH. Specifically, the approach detailed by [6], where a teacher model is trained on diverse terrains of increasing difficulty and then distilled into a student model via supervised learning and using a belief encoder/decoder, offers a promising foundation for further research. These new architectures will be primarily designed for general-purpose locomotion, enabling the robot to operate in post-disaster environments with enhanced robustness and reliability.

***Sim-to-Real***

It will be essential to conduct a thorough study of the types and levels of noise the real robot will encounter, particularly concerning the height map. Following this, the training process must be adjusted to enhance the robot's robustness. This will involve fine-tuning the noise added to the observations, as well as incorporating other randomizations, such as variations in mass, external pushes, and base accelerations.

***Testing***

Many of the issues discussed in this section remain unresolved. While we have outlined a potential general architecture for implementing and validating the policy in ROS, it is still subject to numerous changes and refinements. Further work is needed to fully develop this framework, starting with the implementation of the controller and its integration with the rest of the architecture. To support this effort, our repository provides an initial advancement in this direction, which we believe will facilitate future and faster implementations. The repository includes detailed manuals for installing the core components of the architecture, along with a preliminary version of the controller.

We have briefly discussed some potential issues that may arise. One concern is the manual positioning of the cameras on the front of the Aliengo robot, which could introduce uncertainties that affect the robot's final performance. Additionally, the overall architecture still requires full implementation and validation, leaving several unknowns yet to be addressed. Regarding the Gazebo maps, more challenging alternatives have been proposed (see Footnote 11), which we believe should be sufficient for validating the controller's operation in ROS. The next step, once validated in ROS, would be to deploy the system on the real robot.

## 4.2 Default Controller Performance on Test Circuits

This final section discusses the tests conducted with the real robot on the circuit. It concludes with an outline of future tasks and potential challenges.

### 4.2.1 Baseline for Central Gait Pattern Generator

As a use case, we tested the Central Gait Pattern Generator (CGPG) (see Section A) that Unitree robots come with by default, along with the speed control. This provided a qualitative reference for the robot's performance in navigating the constructed circuit and overcoming its obstacles. The tests were conducted as follows: as shown in Figure 4.17, an operator teleoperated the robot using the manufacturer's joystick (1). The robot was fitted with a harness (2) attached to a thick metal bar (3), which was held by two technicians walking alongside the robot on either side of the circuit. This approach successfully prevented the robot from being damaged on two occasions when it failed to overcome obstacles.

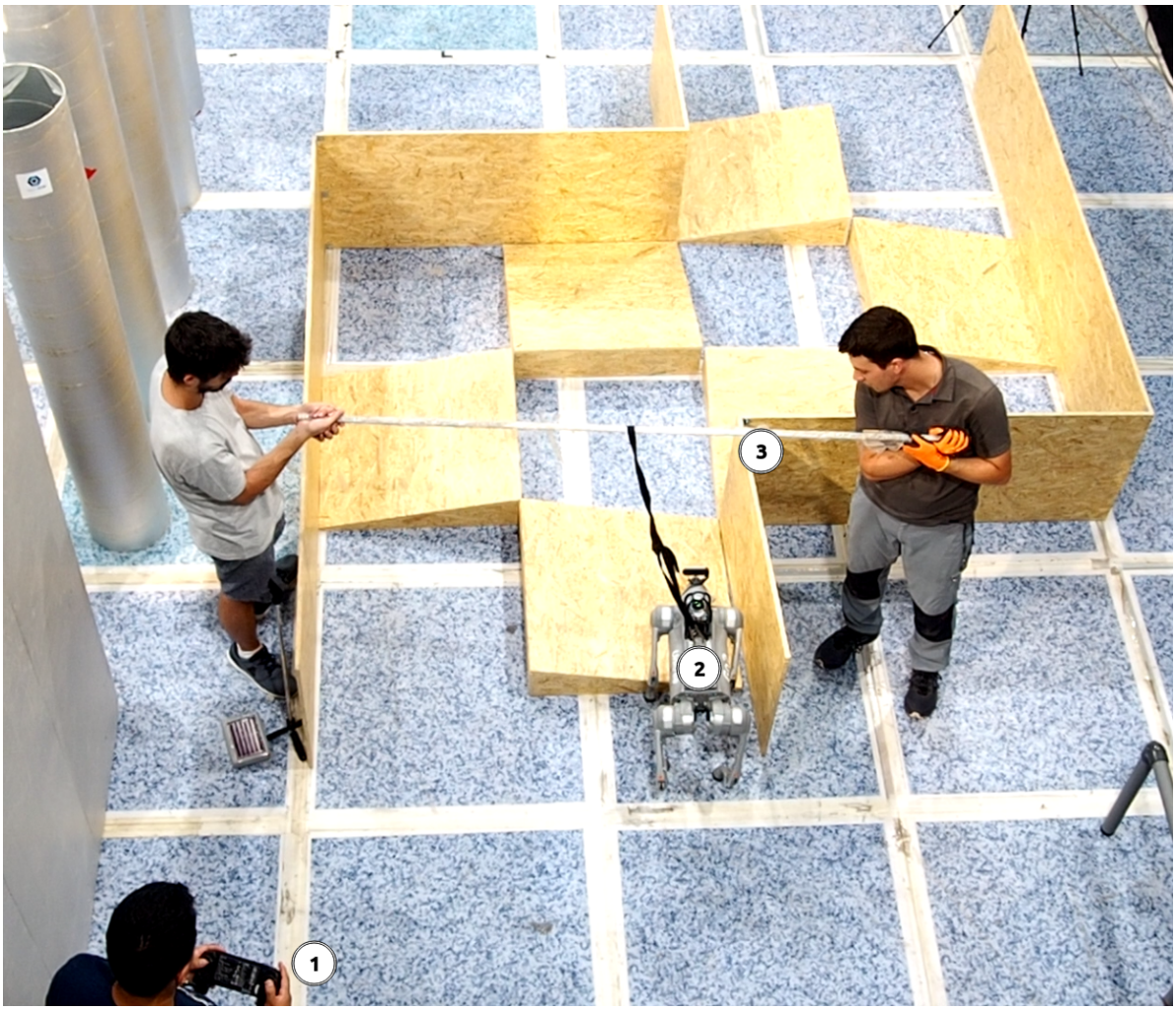


**Figure 4.17:** *Methodology for hardware testing. The robot was teleoperated using the manufacturer's joystick (1), equipped with a harness (2) attached to a metal bar (3), and guided by two technicians. This setup prevented damage during obstacle navigation.*

Firstly, we conducted tests with the Aliengo robot. Figure 4.18 presents snapshots of the robot's navigation through each of the circuits.

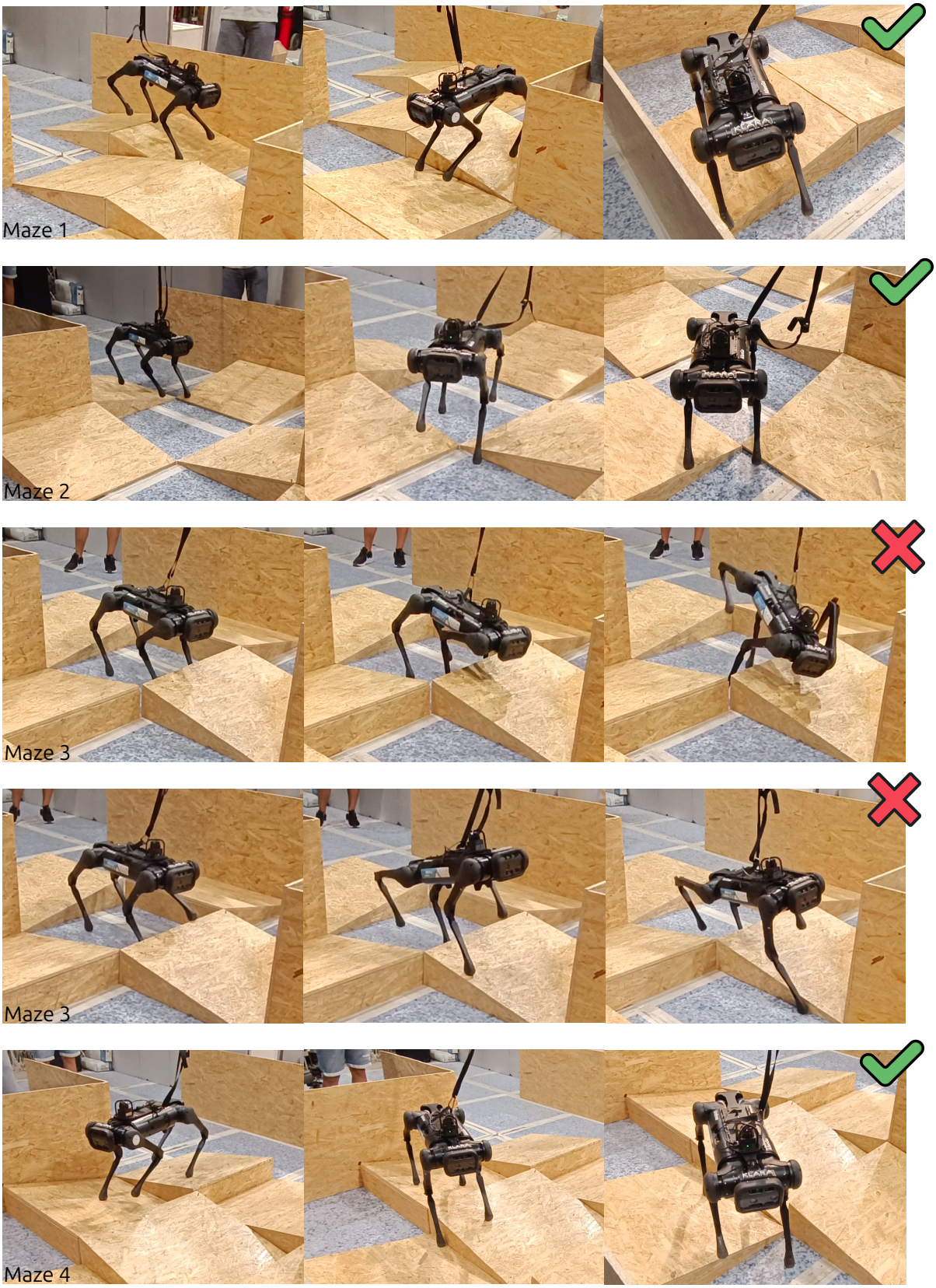


**Figure 4.18:** *Tests with Aliengo robot: snapshots of navigation through each circuit.*

As shown in Figure 4.18, circuits 1 and 2 are the easiest, requiring only a simple gait pattern (assuming the planner is adequate). Circuit 3, however, is clearly the most challenging. Two tests were conducted on this circuit using different gait modes: the normal mode and the stair-climbing mode. The normal mode fails to produce high enough strides to overcome obstacles directly in front of the robot. The stair-climbing mode, while offering higher strides, struggles when descending obstacles. Finally, the fourth circuit presents a medium level of difficulty, which was successfully navigated through careful teleoperation.

Obtaining quantitative metrics based on this study proves challenging, as teleoperated navigation introduces complexities beyond simple path-following tasks. Specifically, when teleoperating delicate hardware, additional factors come into play, such as predicting the robot's movements and adjusting its pace to match the stride rhythm when attempting to climb obstacles or repositioning to gain a better view of the situation. As a result, metrics like path error, time to complete the maze, distance traveled, and the number of collisions with obstacles can be obscured and thus misleading. A more rigorous quantification of this baseline, particularly regarding the Central Gait Pattern Generator for the constructed circuit, remains an open task.

Overall, the gait patterns are a simple and fragile approach. They require constant, careful teleoperation and oversight by additional operators to prevent incidents, like the two mentioned earlier. These walking patterns lack closed-loop control, making an intelligent global planner essential to adjust the robot's trajectory based on terrain difficulty. Moreover, the manufacturer's capabilities are significantly constrained by the available controllers. Further work is needed to test more robust controllers, such as MPC or RL-based controllers, to achieve higher performance levels in these more complex circuits.

In addition to the Aliengo robot, the Go2 was also tested on the circuit, as shown in Figure 4.19. Due to its short strides and poor stability, it was decided not to continue with the rest of the tests.

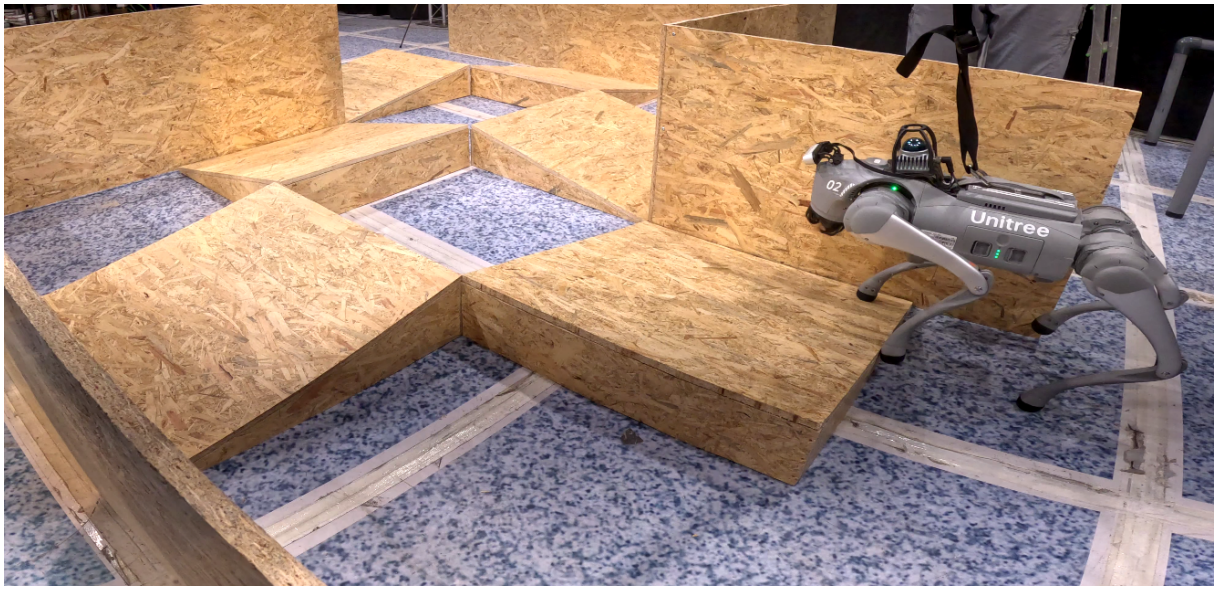


**Figure 4.19:** *Testing the Go2 robot on the obstacle course.*

### 4.2.2 Outstanding Tasks and Potential Issues

The designed and constructed circuit can be easily expanded with new ramps or blocks featuring different configurations and geometries. Additionally, the existing walls and blocks offer endless possibilities for creating circuits with other objects or elements that can further challenge and evaluate the robots' locomotion and navigation capabilities.

For the deployment tests, a semi-automatic system is also being considered to prevent the robot from falling when it fails to overcome obstacles or exceeds its operational limits. A suspended cable with a pulley and a self-locking system could allow tests to be conducted with fewer personnel while enhancing safety. On the other hand, a more precise quantification methodology for the CGPG should be developed to accurately assess locomotion performance in the constructed circuit.

It is possible that initial deployment tests may not be successful, even on a very simple level. This may require revisiting the testing phase to verify sensor communication, message consistency, or potentially retraining the model.

# 5 CONCLUSIONS AND FUTURE WORK

This chapter concludes the report by presenting the key findings and outlining future research directions for advancing locomotion in quadruped robots.

## 5.1 Conclusions

In response to the growing demand for robotic systems in search and rescue missions, this work has focused on quadruped locomotion and its enhancement through Reinforcement Learning (RL) techniques. As a first step, **the evolution and current status of quadruped robots have been documented** from a bibliometric and historical perspective, extending to a state-of-the-art review of locomotion studies. Special care has been taken to ensure the accessibility of this information for any interested reader, particularly for those who may wish to build on this research in the future. Additionally, **a condensed introduction to RL has been prepared**, providing the necessary mathematical foundations (Chapter 2).

Furthermore, **a comprehensive framework has been developed that integrates the training of locomotion policies, testing via a ROS architecture in simulation, and deployment on the real robot using a custom-built test circuit**. A ROS architecture has been proposed, with initial iterations conducted in Gazebo and RViz to set up the robot, integrate the two depth cameras, fuse the point clouds, and position the robot relative to the created circuits (Chapter 3).

Regarding training, state-of-the-art work was replicated, followed by a hyperparameter study of PPO training in Isaac Gym for the Aliengo robot. **New values for the discount rate ($\gamma = 0.95$) and the entropy coefficient ($c_{\mathbf{entropy}} = 0.005$) resulted in overall average improvements of $+8.85\%$ in the first training phase and $+16.73\%$ in the second phase**. Additionally, **the Aliengo's gait patterns were tested** on four different versions of the constructed circuit, **establishing a baseline performance for the robot in this new environment** (Chapter 4).

## 5.2 Future Work and Social Aspects

This work has proposed a framework for implementing locomotion policies in quadruped robots. Despite the progress made, several important tasks remain to achieve a complete iteration from training to real-world robot locomotion. The next steps are broadly outlined as follows:

- **Implement the architecture in ROS**. A new package must be developed to define the neural controller architecture, import the trained weights, and handle network inference. All packages should be connected in ROS through the necessary publishers and subscribers. The goal is to enable robot control via a teleoperation package (using a keyboard or joystick) in Gazebo, both in an empty world and a

test circuit. A potential extension would involve defining a trajectory within the circuit for a local planner to guide the locomotion controller.

- **Deploy the ROS architecture on the real robot** and test its performance across various terrains.
- **Expand the test circuit** by constructing additional ramps (up to six more) or other types of blocks. Additionally, it would be beneficial to implement a more advanced safety system. This could involve a suspended cable, pulley, and self-locking system to facilitate easier and safer robot deployment.

In terms of social and ethical considerations, quadruped robots hold significant potential for future search and rescue missions in a variety of disaster scenarios. Advancements in locomotion systems could make the difference between life and death, enhancing the ability to save lives in critical situations. Moreover, regarding the Sustainable Development Goals (SDG), this project aligns with SDG 9 (Industry, Innovation, and Infrastructure) by advancing robotics and AI for industrial and disaster applications. It supports SDG 11 (Sustainable Cities and Communities) through the development of robots for search and rescue, enhancing urban resilience. Lastly, it contributes to SDG 13 (Climate Action) by aiding disaster recovery efforts in climate-affected areas.

# REFERENCES


[1] J. Jennings, G. Whelan, and W. Evans, "Cooperative search and rescue with a team of mobile robots," in *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, 1997, pp. 193–200. DOI: 10.1109/ICAR.1997.620182.

[2] R. Ventura and P. U. Lima, "Search and rescue robots: The civil protection teams of the future," in *2012 Third International Conference on Emerging Security Technologies*, 2012, pp. 12–19. DOI: 10.1109/EST.2012.40.

[3] C. D. Bellicoso et al., "Advances in real-world applications for legged robots," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1311–1326, 2018. DOI: https://doi.org/10.1002/rob.21839. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21839. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21839.

[4] C. Mastalli et al., "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2536–2542. DOI: 10.1109/ICRA40945.2020.9196673.

[5] E. Arcari et al., "Bayesian multi-task learning mpc for robotic mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3222–3229, 2023. DOI: 10.1109/LRA.2023.3264758.

[6] T. Miki et al., "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, Jan. 2022, ISSN: 2470-9476. DOI: 10.1126/scirobotics.abk2822. [Online]. Available: http://dx.doi.org/10.1126/scirobotics.abk2822.

[7] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, *Learning to walk in minutes using massively parallel deep reinforcement learning*, 2022. arXiv: 2109.11978 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2109.11978.

[8] S. Kareer, N. Yokoyama, D. Batra, S. Ha, and J. Truong, *Vinl: Visual navigation and locomotion over obstacles*, 2023. arXiv: 2210.14791 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2210.14791.

[9] J.-P. Sleiman, F. Farshidian, and M. Hutter, "Versatile multicontact planning and control for legged loco-manipulation," *Science Robotics*, vol. 8, no. 81, eadg5014, 2023. DOI: 10.1126/scirobotics.adg5014. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.adg5014. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.adg5014.

[10] P. M. Wensing et al., "Proprioceptive actuator design in the mit cheetah: Impact mitigation and high-bandwidth physical interaction for dynamic legged robots," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 509–522, 2017. DOI: 10.1109/TRO.2016.2640183.

[11] Robotic Systems Lab: Legged Robotics at ETH Zurich, *Anymal unleashed: Revolutionizing search-and-rescue with legged robots (dtc: Deep tracking control)*, https://youtu.be/_R29DrAx0Xs?feature=shared, Accessed: 2024-08-02, Feb. 2023.

[12] G. News, *New dog-like robot from boston dynamics can open doors*, Feb. 2018. [Online]. Available: https://www.youtube.com/watch?v=wXxrmussq4E.

[13] C. R. Marcius, "See 'spot' save: Robot dogs join the new york fire department," *The New York Times*, Mar. 2022, https://www.nytimes.com/2022/03/17/nyregion/fdny-robot-dogs.html.

[14] B. Dynamics, *Boston Dynamics*, 2021. [Online]. Available: https://bostondynamics.com/.

[15] H. Chai et al., "A survey of the development of quadruped robots: Joint configuration, dynamic locomotion control method and mobile manipulation approach," *Biomimetic Intelligence and Robotics*, vol. 2, no. 1, p. 100 029, 2022, ISSN: 2667-3797. DOI: https://doi.org/10.1016/j.birob.2021.100029. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667379721000292.

[16] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, eabc5986, Oct. 2020. DOI: 10.1126/scirobotics.abc5986. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abc5986.

[17] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019. DOI: 10.1109/LRA.2019.2891991.

[18] A. Davids, "Urban search and rescue robots: From tragedy to technology," *IEEE Intelligent Systems*, vol. 17, no. 2, pp. 81–83, 2002. DOI: 10.1109/MIS.2002.999224.

[19] E. Technologies, *Multi-Mission Modular Inspection crawlers*, 2024. [Online]. Available: https://www.eddyfi.com/en/product/versatrax-inspection-crawlers.

[20] Emilyrobot, *EMILY Rescue Robot*, 2020. [Online]. Available: https://www.emilyrobot.com/.

[21] M. Kwong, *Nepal earthquake: Drones used by Canadian relief team*, Apr. 215. [Online]. Available: https://www.cbc.ca/news/world/nepal-earthquake-drones-used-by-canadian-relief-team-1.3051106.

[22] E. Cunningham, *Why spacex bought a robotic dog*, Accessed: 2024-05-29, Apr. 2023. [Online]. Available: https://primalnebula.com/why-spacex-bought-a-robotic-dog/.

[23] Clarivate, *Web of Science*, 2024. [Online]. Available: https://www.webofscience.com/wos/woscc/basic-search.

[24] Elsevier, *Scopus Search*, 2024. [Online]. Available: https://www.scopus.com/search/form.uri?display=basic#basic.

[25] V. Gomez-Jauregui, C. Gomez-Jauregui, C. Manchado, and C. Otero, "Information management and improvement of citation indices," *International Journal of Information Management*, vol. 34, no. 2, pp. 257–271, 2014, Cited by: 45; All Open Access, Green Open Access. DOI: 10.1016/j.ijinfomgt.2014.01.002. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84893458584&doi=10.1016%2fj.ijinfomgt.2014.01.002&partnerID=40&md5=b5d0c5147089f9f52a0ee12a8bded6ab.

[26] E. S. Vieira and J. A. N. F. Gomes, "A comparison of scopus and web of science for a typical university," *Scientometrics*, vol. 81, no. 2, pp. 587–600,

2009, Cited by: 335. DOI: 10.1007/s11192-009-2178-0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77956614648&doi=10.1007%2fs11192-009-2178-0&partnerID=40&md5=785f6067a35c20fbf9a0c1c5c5323f83.

[27] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[28] M. Aria and C. Cuccurullo, "Bibliometrix: An r-tool for comprehensive science mapping analysis," *Journal of Infometrics*, vol. 11, no. 4, pp. 959–975, 2017. [Online]. Available: https://doi.org/10.1016/j.joi.2017.08.007.

[29] V. Makoviychuk et al., *Isaac gym: High performance gpu-based physics simulation for robot learning*, 2021. arXiv: 2108.10470 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2108.10470.

[30] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model algorithmic control of industrial processes," *IFAC Proceedings Volumes*, vol. 10, no. 16, pp. 103–120, 1977, Preprints of the 5th IFAC/IFIP International Conference on Digital Computer Applications to Process Control, The Hague, The Netherlands, 14-17 June, 1977, ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)69513-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017695132.

[31] WIPO, *Search international and national patent collections (patentscope)*, Apr. 2024. [Online]. Available: https://www.wipo.int/patentscope/es/.

[32] D. Robotics, *DEEP Robotics - Global Quadruped Robot Leader*. [Online]. Available: https://www.deeprobotics.cn/en.

[33] M. H. Raibert and E. R. Tello, "Legged robots that balance," *IEEE Expert*, vol. 1, no. 4, pp. 89–89, 1986. DOI: 10.1109/MEX.1986.4307016.

[34] P. G. de Santos, E. Garcia, and J. Estremera, *Quadrupedal Locomotion: An Introduction to the Control of Four-legged Robots*, 1st ed. London: Springer London, 2006, pp. XIV, 268, ISBN: 978-1-84628-306-2. DOI: 10.1007/1-84628-307-8.

[35] R. Mosher, "Test and evaluation of a versatile walking truck," in *Proceedings of Off-Road Mobility Research Symposium, Washington DC, 1968*, 1968, pp. 359–379. [Online]. Available: https://cir.nii.ac.jp/crid/1570291225044499840.

[36] S. Hirose and K. Kato, "Study on quadruped walking robot in tokyo institute of technology - past, present and future," Cited by: 55, vol. 1, 2000, pp. 414–419. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0033727412&partnerID=40&md5=2e5fe82cd8df76a08d2484457713b9cc.

[37] H. Taheri and N. Mozayani, "A study on quadruped mobile robots," *Mechanism and Machine Theory*, vol. 190, p. 105 448, 2023. DOI: 10.1016/j.mechmachtheory.2023.105448.

[38] U. Robotics, *Aliengo urdf description*, https://github.com/unitreerobotics/unitree_ros/tree/master/robots/aliengo_description/urdf, Accessed: 2024-08-13, 2024.

[39] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, *Extreme parkour with legged robots*, 2023. arXiv: 2309.14341 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2309.14341.

[40] N. Rudin, D. Hoeller, M. Bjelonic, and M. Hutter, *Advanced skills by learning locomotion and local navigation end-to-end*, 2022. arXiv: 2209.12827 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2209.12827.

[41] T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997, vol. 1.

[42] OpenAI et al., *Learning dexterous in-hand manipulation*, 2019. arXiv: 1808.00177 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1808.00177.

[43] S. L. Brunton, *Machine learning meets control theory*, Video, Reinforcement learning, Cassyni, 2021. DOI: 10.52843/cassyni.x2t0sp. [Online]. Available: https://doi.org/10.52843/cassyni.x2t0sp (visited on 08/18/2024).

[44] Intuitive Robots, *Buy spot - intuitive robots*, https://www.intuitive-robots.com/buy-spot/, Accessed: 2024-08-18, 2024.

[45] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961. DOI: 10.1109/JRPROC.1961.287775.

[46] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html.

[47] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1707.06347.

[48] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2017. arXiv: 1502.05477 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1502.05477.

[49] V. Mnih et al., *Asynchronous methods for deep reinforcement learning*, 2016. arXiv: 1602.01783 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1602.01783.

[50] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation*, 2018. arXiv: 1506.02438 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1506.02438.

[51] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[52] N. Heess et al., *Emergence of locomotion behaviours in rich environments*, 2017. arXiv: 1707.02286 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1707.02286.

[53] F. Jenelten, R. Grandia, F. Farshidian, and M. Hutter, "Tamols: Terrain-aware motion optimization for legged systems," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3395–3413, 2022. DOI: 10.1109/TRO.2022.3186804.

[54] A. Herdt, N. Perrin, and P.-B. Wieber, "Walking without thinking about it," in *IEEE IROS*, Taipei, Taiwan, Oct. 2010, pp. 190–195. DOI: 10.1109/IROS.2010.5654429. [Online]. Available: https://ieeexplore.ieee.org/document/5654429.

[55] S. Kajita et al., "Biped walking pattern generation by using preview control of zero-moment point," in *IEEE ICRA*, vol. 2, Taipei, Taiwan, Sep. 2003, pp. 1620–1626. DOI: 10.1109/ROBOT.2003.1241826. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1241826.

[56] M. H. Raibert, *Legged Robots That Balance*. MIT press, 1986.

[57] W. Bosworth, J. Whitney, S. Kim, and N. Hogan, "Robot locomotion on hard and soft ground: Measuring stability and ground properties in-situ," in *IEEE ICRA*, Stockholm, Sweden, May 2016, pp. 3582–3589. DOI: 10.1109/ICRA.2016.7487541. [Online]. Available: https://ieeexplore.ieee.org/document/7487541.

[58] S. Fahmi et al., "Stance: Locomotion adaptation over soft terrain," *IEEE J T-RO*, vol. 36, no. 2, pp. 443–457, Apr. 2020. DOI: 10.1109/TRO.2019.2954670. [Online]. Available: https://ieeexplore.ieee.org/document/8957061.

[59] S. Kuindersma et al., "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Journal of Autonomous Robots (J-AUTON)*, vol. 40, no. 3, pp. 429–455, Jul. 2015. DOI: 10.1007/s10514-015-9479-3. [Online]. Available: https://link.springer.com/article/10.1007/s10514-015-9479-3.

[60] H.-W. Park, P. M. Wensing, and S. Kim, "High-speed bounding with the mit cheetah 2: Control design and experiments," *International Journal of Robotics Research (IJRR)*, vol. 36, no. 2, pp. 167–192, Mar. 2017. DOI: 10.1177/0278364917694244. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/0278364917694244.

[61] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "Bigdog, the rough-terrain quadruped robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10 822–10 825, Mar. 2008. DOI: 10.3182/20080706-5-KR-1001.01833. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016407020.

[62] L. Righetti and S. Schaal, "Quadratic programming for inverse dynamics with optimal distribution of contact forces," in *IEEE Humanoids*, Osaka, Japan, Nov. 2012, pp. 538–543. DOI: 10.1109/HUMANOIDS.2012.6651572. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6651572/.

[63] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *IEEE Humanoids*, Madrid, Spain, Nov. 2014, pp. 295–302. DOI: 10.1109/HUMANOIDS.2014.7041375. [Online]. Available: https://ieeexplore.ieee.org/document/7041375.

[64] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv*, 2019. DOI: 10.48550/arXiv.1909.06586. [Online]. Available: https://arxiv.org/abs/1909.06586.

[65] G. Bledt et al., "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *IEEE IROS*, Madrid, Spain, Oct. 2018, pp. 2245–2252. DOI: 10.1109/IROS.2018.8593885. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8593885.

[66] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model predictive control," *IEEE Transactions on Robotics*, 2023-05. DOI: 10.3929/ethz-b-000564782.

[67] A. Loquercio, A. Kumar, and J. Malik, "Learning visual locomotion with cross-modal supervision," in *arXiv*, 2022.

[68] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," *IEEE Transactions on Robotics*, pp. 1–20, 2022.

[69] D. Coyle and L. Hampton, "21st century progress in computing," *Telecommunications Policy*, vol. 48, no. 1, p. 102 649, 2024, ISSN: 0308-5961. DOI: https:

//doi.org/10.1016/j.telpol.2023.102649. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030859612300160X.

[70] J. Tan et al., "Sim-to-real: Learning agile locomotion for quadruped robots," in *Conference on Robot Science and Systems (C-RSS)*, Pittsburgh, Pennsylvania, USA, Jun. 2018, pp. 1–9. DOI: 10.15607/RSS.2018.XIV.010. [Online]. Available: https://arxiv.org/abs/1804.10332.

[71] J. Hwangbo et al., "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, Jan. 2019. DOI: 10.1126/scirobotics.aau5872. [Online]. Available: https://www.science.org/doi/full/10.1126/scirobotics.aau5872.

[72] Z. Xie et al., "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conference on Robot Learning (C-CORL)*, Osaka, Japan, Nov. 2020, pp. 1–13. DOI: 10.48550/arXiv.1903.09537. [Online]. Available: https://proceedings.mlr.press/v100/xie20a.html.

[73] Z. Fu, A. Kumar, J. Malik, and D. Pathak, "Minimizing energy consumption leads to the emergence of gaits in legged robots," in *Conference on Robot Learning (C-CORL)*, London, UK, Nov. 2021, pp. 928–937. DOI: 10.48550/arXiv.2111.01674. [Online]. Available: https://arxiv.org/abs/2111.01674.

[74] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," in *Conference on Robot Science and Systems (C-RSS)*, Virtual, Jul. 2021. DOI: 10.48550/arXiv.2107.04034. [Online]. Available: https://arxiv.org/abs/2107.04034.

[75] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion," *IEEE Robotics and Automation Letters (RAL)*, vol. 7, no. 2, pp. 4630–4637, Apr. 2022. DOI: 10.1109/LRA.2022.3151396. [Online]. Available: https://ieeexplore.ieee.org/document/9714001.

[76] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, *Rapid locomotion via reinforcement learning*, 2022. arXiv: 2205.02824 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2205.02824.

[77] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, "Anymal parkour: Learning agile navigation for quadrupedal robots," *Science Robotics*, vol. 9, no. 88, eadi7566, 2024. DOI: 10.1126/scirobotics.adi7566. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.adi7566. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.adi7566.

[78] Z. Zhuang et al., *Robot parkour learning*, 2023. arXiv: 2309.05665 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2309.05665.

[79] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, *Legged locomotion in challenging terrains using egocentric vision*, 2022. arXiv: 2211.07638 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2211.07638.

[80] X. Cheng, A. Kumar, and D. Pathak, *Legs as manipulator: Pushing quadrupedal agility beyond locomotion*, 2023. arXiv: 2303.11330 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2303.11330.

[81] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Published as COINS Technical Report 84-2, Ph.D. dissertation, University of Massachusetts, Amherst, MA, 1984. [Online]. Available: http://www.incompleteideas.net/papers/Sutton-PhD-thesis.pdf.

[82] R. E. Bellman and R. E. Kalaba, *Dynamic Programming and Feedback Control*. Santa Monica, CA: RAND Corporation, 1959.

[83] L. Robotics, *Ppo algorithm implementation in rsl_rl*, Accessed: 2024-08-27, 2023. [Online]. Available: https://github.com/leggedrobotics/rsl_rl/blob/master/rsl_rl/algorithms/ppo.py.

[84] L. Robotics, *Legged_gym: Legged robot configurations*, Accessed: 2024-08-27, 2024. [Online]. Available: https://github.com/leggedrobotics/legged_gym/blob/17847702f90d8227cd31cce9c920aa53a739a09a/legged_gym/envs/base/legged_robot_config.py%5C#L215.

[85] AurelianTactics, *Ppo hyperparameters and ranges*, Medium, https://docs.google.com/spreadsheets/d/1fNVfqgAifDWnTq-4izPPW_CVAUu9FXl3dWkqWIXz04o/edit?usp=sharing, Jul. 2018. [Online]. Available: https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe.

[86] V. V. Kindratenko et al., "Gpu clusters for high-performance computing," in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–8. DOI: 10.1109/CLUSTR.2009.5289128.

[87] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, *Learning by cheating*, 2019. arXiv: 1912.12294 [cs.RO]. [Online]. Available: https://arxiv.org/abs/1912.12294.

[88] M. Savva et al., "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[89] R. Haschke, W. Woodall, D. Gossow, D. Hershberger, and J. Faust, *Rviz: 3d visualization tool for ros*, http://wiki.ros.org/rviz, 2024.

[90] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[91] E. Vollenweider et al., *Advanced skills through multiple adversarial motion priors in reinforcement learning*, 2022. arXiv: 2203.14912 [cs.RO].

[92] F. Jenelten, J. He, F. Farshidian, and M. Hutter, "Dtc: Deep tracking control," *Science Robotics*, vol. 9, no. 86, Jan. 2024, ISSN: 2470-9476. DOI: 10.1126/scirobotics.adh5401. [Online]. Available: http://dx.doi.org/10.1126/scirobotics.adh5401.

[93] P. Arm, M. Mittal, H. Kolvenbach, and M. Hutter, *Pedipulate: Enabling manipulation skills using a quadruped robot's leg*, 2024. arXiv: 2402.10837 [cs.RO].

[94] S. A. I. L. et al., *Robotic operating system*, version ROS Noetic Ninjemys - Ubuntu 20.04, May 23, 2020. [Online]. Available: https://www.ros.org.

[95] U. Robotics, *Unitree_ros: Ros package for unitree robotics robots*, Accessed: 2024-09-02, 2024. [Online]. Available: https://github.com/unitreerobotics/unitree_ros.

[96] macc-n, *Ros_unitree: Ros integration for unitree robots with simulation and navigation support on ros noetic*, Accessed: 2024-09-02, 2024. [Online]. Available: https://github.com/macc-n/ros_unitree.

[97] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 3019–3026, 2018. DOI: 10.1109/LRA.2018.2849506.

[98] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[99] I. RealSense, *Realsense-ros: Intel realsense ros wrapper for d400 series, sr300 camera and t265 tracking module*, Accessed: 2024-09-02, 2024. [Online]. Available: https://github.com/IntelRealSense/realsense-ros.

[100] M. Tartari, *Realsense_gazebo_description: Realsense camera models and urdfs for gazebo simulation*, Accessed: 2024-09-02, 2024. [Online]. Available: https://github.com/m-tartari/realsense_gazebo_description.

[101] C. Cruz, J. del Cerro, and A. Barrientos, "Perception sensor integration for improved environmental reconstruction in quadruped robotics," *Jornadas de Automática*, vol. 45, 2024. [Online]. Available: https://doi.org/10.17979/jacea.2024.45.10830.

[102] J. Jeon and H. Moon, *Icra 2023 quadruped robot challenges: Simulation map environment*, https://github.com/rise-lab-skku/ICRA2023_Quadruped_Robot_Challenges, version 2.0.1, Maintained by RISE Lab, Sungkyunkwan University. Accessed: 2024-09-03, 2023.

[103] P. Henderson et al., *Deep reinforcement learning that matters*, 2019. arXiv: 1709.06560 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1709.06560.

[104] A. Juliani et al., *Unity: A general platform for intelligent agents*, 2020. arXiv: 1809.02627 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1809.02627.

[105] R. Yuste, J. N. MacLean, J. Smith, and A. Lansner, "The cortex as a central pattern generator," *Nature Reviews Neuroscience*, vol. 6, no. 6, pp. 477–483, 2005.

[106] F. Delcomyn, "Neural basis of rhythmic behavior in animals," *Science*, vol. 210, no. 4469, pp. 492–498, 1980.

[107] J. Wang, C. Hu, and Y. Zhu, "CPG-based hierarchical locomotion control for modular quadrupedal robots using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7193–7200, 2021.

[108] G. Bellegarda and A. Ijspeert, "CPG-RL: Learning central pattern generators for quadruped locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12 547–12 554, 2022, conference Name: IEEE Robotics and Automation Letters.

[109] M. Thor and P. Manoonpong, "A fast online frequency adaptation mechanism for CPG-based robot motion control," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3324–3331, 2019.

[110] Konery, *Panorama del mercado energético en junio 2024*, Accessed: 2024-09-05, 2024. [Online]. Available: https://konery.com/panorama-del-mercado-energetico-en-junio-2024/.

# ANNEXES

## A Additional Material

| Parameter | Value | Units | Description |
|---|---|---|---|
| **Trunk** | | | |
| Mass (Trunk) | 9.041 | kg | Mass of the robot's trunk |
| Inertia (Trunk) $I_{xx}$ | 0.033260231 | kg·m$^2$ | Moment of inertia about the x-axis |
| Inertia (Trunk) $I_{yy}$ | 0.16117211 | kg·m$^2$ | Moment of inertia about the y-axis |
| Inertia (Trunk) $I_{zz}$ | 0.17460442 | kg·m$^2$ | Moment of inertia about the z-axis |
| Body Length | 0.647 | m | Length of the trunk |
| Body Width | 0.15 | m | Width of the trunk |
| Body Height | 0.112 | m | Height of the trunk |
| Friction Coefficient (Trunk) $\mu1$, $\mu2$ | 0.2 | | Coefficient of friction for the trunk |
| Damping Coefficient (Trunk) $k_d$ | 1.0 | | Damping coefficient for the trunk |
| Stiffness Coefficient (Trunk) $k_p$ | 1,000,000 | | Stiffness coefficient for the trunk |
| **Legs** | | | |
| Leg Length (Thigh) | 0.25 | m | Length of the thigh link |
| Leg Length (Calf) | 0.25 | m | Length of the calf link |
| Mass (Hip) | 1.993 | kg | Mass of the hip link |
| Inertia (Hip) $I_{xx}$ | 0.002903894 | kg·m$^2$ | Moment of inertia about the x-axis of the hip |
| Inertia (Hip) $I_{yy}$ | 0.004907517 | kg·m$^2$ | Moment of inertia about the y-axis of the hip |
| Inertia (Hip) $I_{zz}$ | 0.005586944 | kg·m$^2$ | Moment of inertia about the z-axis of the hip |
| Mass (Thigh) | 0.639 | kg | Mass of the thigh link |
| Inertia (Thigh) $I_{xx}$ | 0.005666803 | kg·m$^2$ | Moment of inertia about the x-axis of the thigh |
| Inertia (Thigh) $I_{yy}$ | 0.005847229 | kg·m$^2$ | Moment of inertia about the y-axis of the thigh |
| Inertia (Thigh) $I_{zz}$ | 0.000369811 | kg·m$^2$ | Moment of inertia about the z-axis of the thigh |
| Mass (Calf) | 0.207 | kg | Mass of the calf link |
| Inertia (Calf) $I_{xx}$ | 0.006341369 | kg·m$^2$ | Moment of inertia about the x-axis of the calf |
| Inertia (Calf) $I_{yy}$ | 0.006355157 | kg·m$^2$ | Moment of inertia about the y-axis of the calf |
| Inertia (Calf) $I_{zz}$ | 0.000039188 | kg·m$^2$ | Moment of inertia about the z-axis of the calf |
| Mass (Foot) | 0.06 | kg | Mass of the foot link |
| Inertia (Foot) $I_{xx}$, $I_{yy}$, $I_{zz}$ | 0.000016854 | kg·m$^2$ | Moment of inertia for the foot link (all axes) |
| Friction Coefficient (Foot) $\mu1$, $\mu2$ | 0.6 | | Coefficient of friction for the feet |
| Joint Limit (Effort) - Hip | 44 | Nm | Torque limit for hip joint |
| Joint Limit (Effort) - Thigh | 44 | Nm | Torque limit for thigh joint |
| Joint Limit (Effort) - Calf | 55 | Nm | Torque limit for calf joint |
| Joint Limit (Velocity) - Hip | 20 | rad/s | Velocity limit for hip joint |
| Joint Limit (Velocity) - Thigh | 20 | rad/s | Velocity limit for thigh joint |
| Joint Limit (Velocity) - Calf | 16 | rad/s | Velocity limit for calf joint |
| **Sensors and Control** | | | |
| IMU Mass | 0.001 | kg | Mass of the IMU |
| IMU Inertia $I_{xx}$, $I_{yy}$, $I_{zz}$ | 0.0001 | kg·m$^2$ | Moment of inertia for the IMU (all axes) |
| IMU Update Rate | 1000 | Hz | Update rate for the IMU sensor |
| Foot Contact Sensor Update Rate | 100 | Hz | Update rate for foot contact sensors |

**Table A.1:** *Main relevant data for the Aliengo robot. Based on URDF file [38].*

| Parameter | Value | Units |
|---|---|---|
| **Environment Configuration** | | |
| num_envs | 1024 | - |
| **Initial State** | | |
| pos | [0.0, 0.0, 0.38] | m |
| FL_hip_joint | 0.1 | rad |
| RL_hip_joint | 0.1 | rad |
| FR_hip_joint | -0.1 | rad |
| RR_hip_joint | -0.1 | rad |
| FL_thigh_joint | 0.8 | rad |
| RL_thigh_joint | 1.0 | rad |
| FR_thigh_joint | 0.8 | rad |
| RR_thigh_joint | 1.0 | rad |
| FL_calf_joint | -1.5 | rad |
| RL_calf_joint | -1.5 | rad |
| FR_calf_joint | -1.5 | rad |
| RR_calf_joint | -1.5 | rad |
| **PD Control** | | |
| stiffness (joint) | 40.0 | N*m/rad |
| damping (joint) | 2.0 | N*m*s/rad |
| action_scale* | 0.25 | - |

**Table A.2:** *General training configurations. *The action scale refers to the gain: target_angle = actionScale · action + default_angle.*

### *Central Gait Pattern Generator*

Central Gait Pattern Generator (CGPG) is a control mechanism inspired by neural circuits in animals that generate rhythmic movements like walking [105, 106]. It produces rhythmic joint signals for quadrupedal locomotion without needing detailed models [107, 108, 109], offering simplicity and efficiency, but with limited adaptability to changing environments due to its open-loop nature.

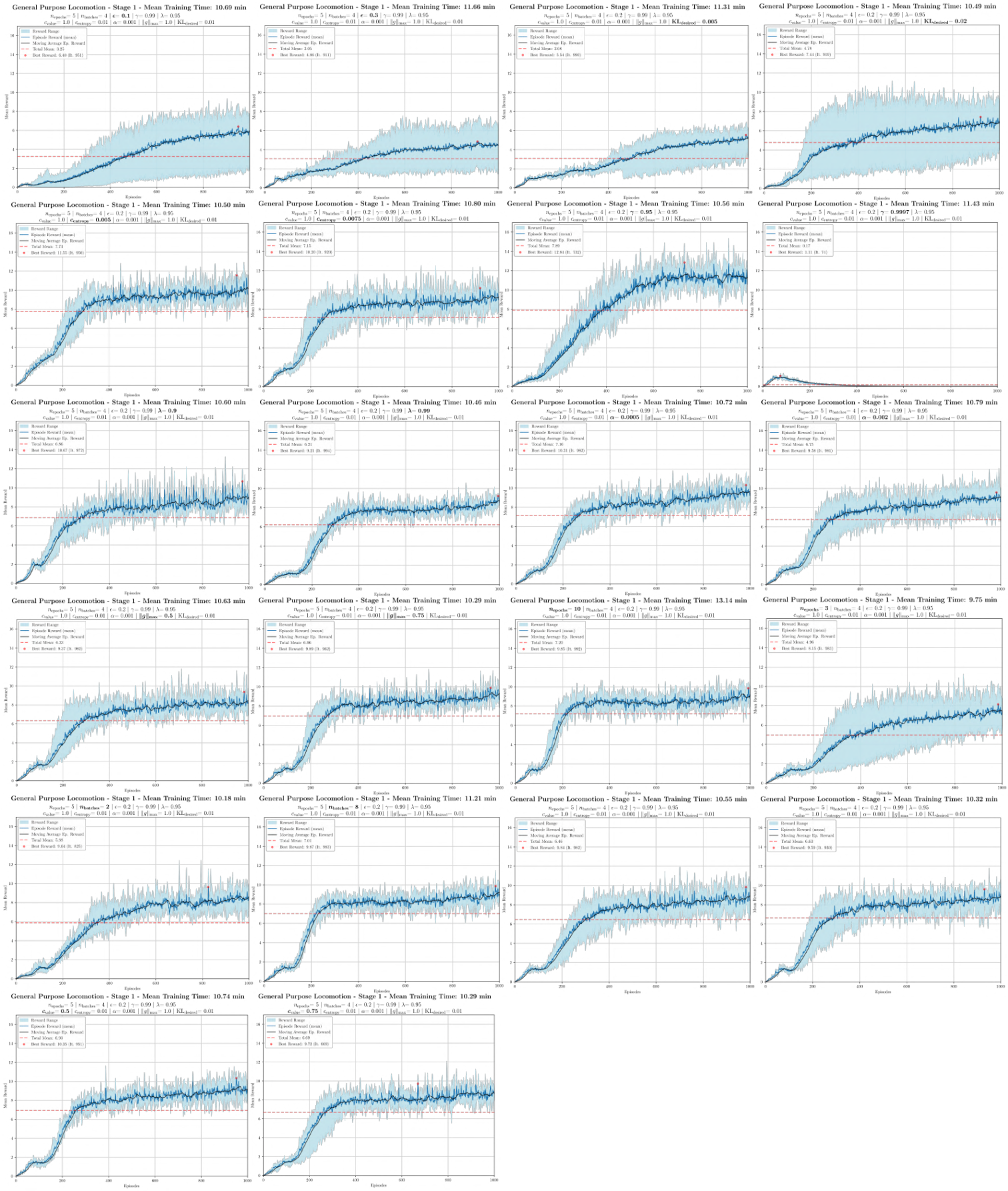


**Figure A.1:** *Grid of plots for the first training stage. Hyperparameter study.*

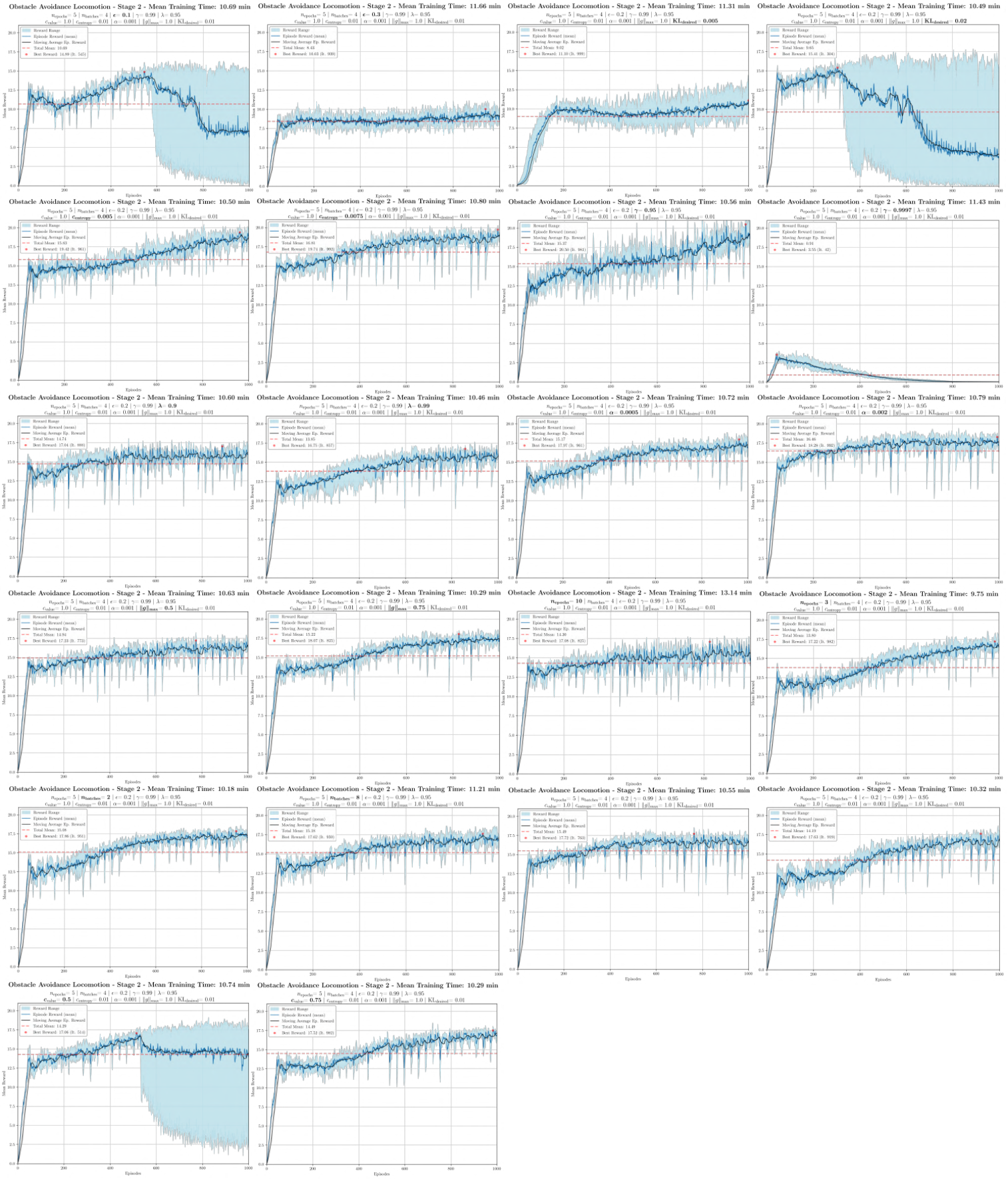


**Figure A.2:** *Grid of plots for the second training stage. Hyperparameter study.*

# B Work Breakdown Structure

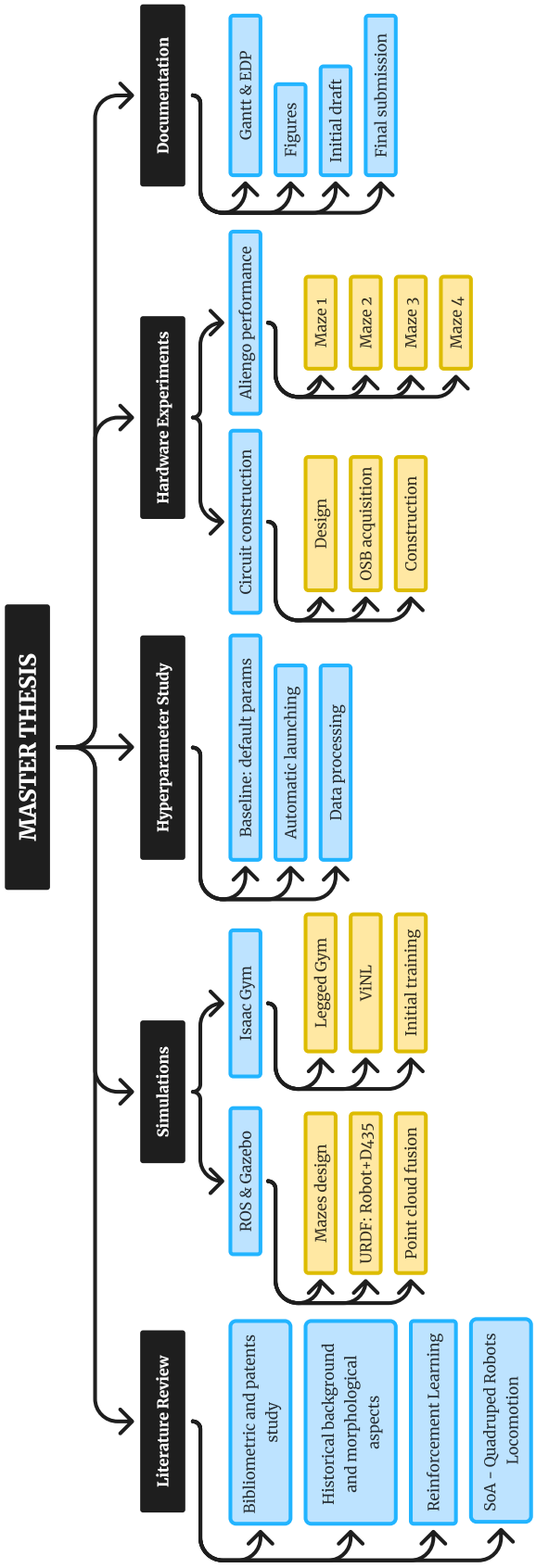


**Figure B.3:** *Work Breakdown Structure (WBS). Hierarchical project breakdown, for the visualization of project components and deliverables, facilitating their orderly execution.*

# C Time Planning

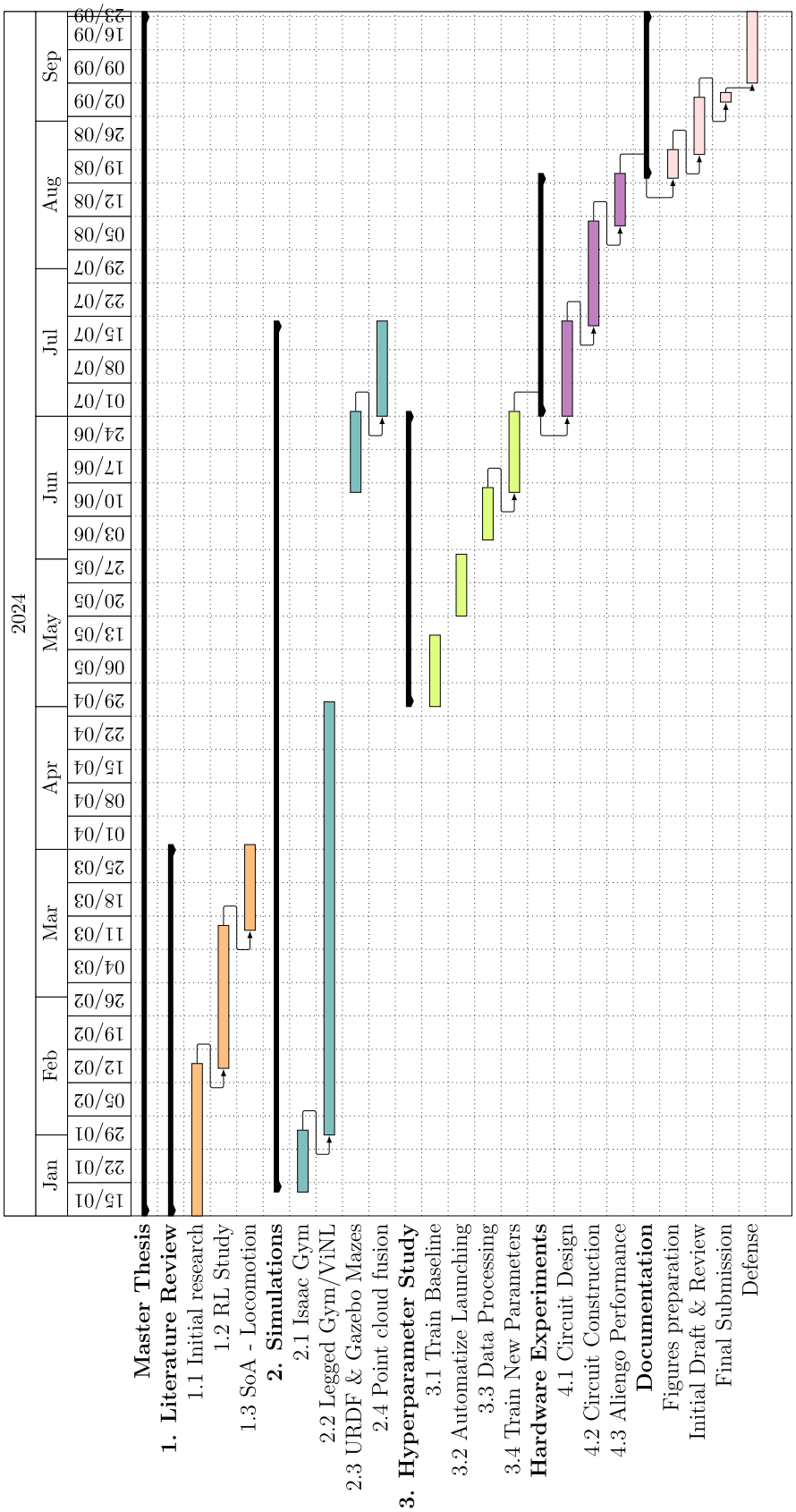


**Figure C.4:** *Gantt Diagram: Planning of the project.*

# D Budgeting

The budget is categorized into three main types of resources:

### Energy Resources

Accurately estimating the electricity costs for the simulations is difficult. As a solution, we calculate an average expense to provide a general estimate for the major simulations.

| Name | Number of Simulations | Duration (h) | Consumption (kW)[(1)] | €/kWh[(2)] | Cost (€) |
|---|---|---|---|---|---|
| Initial simulations | 20 | 1.00 | 0.15 | 0.04323 | 0.1297 |
| Baseline | 20 | 2.00 | 0.15 | 0.04323 | 0.2594 |
| Hyperparameter study | 220 | 1.0 | 0.15 | 0.04323 | 1.4267 |
| Final comparison | 20 | 2 | 0.15 | 0.04323 | 0.2594 |
| | | | | **Total energy resources:** | **2.0752** |

**Table D.3:** *Energy Resources - GPU Simulations*

*(1) Calculated using the average consumption for the NVIDIA RTX 4060.*
*(2) Computed using the average electricity price (including VAT) for the month of May and June 2024 [110].*

### Human Resources

This section includes the total hours worked by the student Josep María Barberá Civera, calculated at a rate of 10 €/hour, alongside the hours contributed by the three supervisors, each valued at 50 €/hour. The total hours for Josep María have been determined as follows:

- During the Spring 2024 Robotics Master's program, which spanned five months from January to May 2024, work was conducted 2 days a week for 5 hours each day. This amounts to a total of 20 weeks, resulting in 200 hours.
- In the summer of 2024, he worked for three months, from June to August, with 5 days of work per week for 4 hours each day. This adds up to 12 weeks, accounting for 240 hours.
- Furthermore, during the first week and a half of September, an intensive workload was undertaken, consisting of ten days at eight hours per day, totaling 80 hours.

In total, this amounts to 520 hours.

| **Name** | **Hours** | **€/hour** | **Cost (€)** |
|---|---|---|---|
| Josep María | 520 | 10.00 | 5,200.00 |
| Supervisors | 100 | 50.00 | 5,000.00 |
| **Total human resources:** | | | **10,200.00** |

**Table D.4:** *Allocation of Human Resources.*

### Computer Resources

Lastly, the software utilized for the project is listed in this resource section. All software incurred zero cost, as it was accessed through various licensing agreements, including academic licenses, OpenSource software, trial versions, or licenses provided by the Technical University of Madrid (Universidad Politécnica de Madrid).

| **Program** | **License** | **Price (€)** |
|---|---|---|
| **Writing and Editing** | | |
| VSCode | OpenSource | 0 |
| Inkscape | OpenSource | 0 |
| Microsoft Excel (from Microsoft 365) | UPM | 0 |
| Microsoft Word (from Microsoft 365) | UPM | 0 |
| **Simulation and Programming** | | |
| GitHub | OpenSource | 0 |
| Visual Studio Code | OpenSource | 0 |
| Jupyter Notebooks | OpenSource | 0 |
| Isaac Gym | OpenSource | 0 |
| Gazebo | OpenSource | 0 |
| **Total computer resources:** | | 0 |

**Table D.5:** *Overview of Computer Resources Utilized*

### Total Budget

The total budget for this project has been **10,200.0752 €**.