POLITECNICO DI TORINO

Master's degree programme
Computer Engineering

Master Thesis

Learning Adaptive Locomotion for a Reconfigurable Hexapod Robot From Simulation to Real World Deployment

Supervisors
Prof. Marcello Chiaberge
Prof.ssa Olga Napolitano
Prof. Valerio Modugno

Candidate
Luca Pellicciotti

Academic year 2025-2026

Ai miei amici
A Cristina
Alla mia famiglia

Contents

List of Tables 7

List of Figures 8

1 Introduction 10

1.1 Objectives . . . 10

1.2 Thesis Structure . . . 11

2 Background and Related Work 13

2.1 Legged Robotics . . . 13

2.1.1 Main Legged Robot Morphologies . . . 13

2.1.2 Hexapod Robots . . . 14

2.2 Reinforcement Learning for Robot Locomotion . . . 16

2.2.1 RL for Legged Locomotion . . . 17

2.2.2 Model-Free and Structured Reinforcement Learning Approaches . . . 17

2.3 Sim-to-Real Transfer . . . 18

2.3.1 Reality Gap in Robotics . . . 19

2.3.2 Approaches for Reducing the Gap . . . 20

3 Problem Context and Motivation 23

3.1 Problem Definition . . . 23

3.2 Existing Approaches . . . 24

3.3 Proposed Approach . . . 25

3.4 Application Context . . . 26

4 System Overview 27
4.1 System Requirements and Design . . . 27
4.2 Reconfigurable Hexapod Platform . . . 28
4.2.1 Mechanical Structure . . . 28
4.2.2 Leg Detachment Mechanism . . . 29
4.2.3 Adaptable Standing Configuration . . . 31
4.3 Hardware Architecture . . . 32
4.3.1 Computing Units and Low-Level Controllers . . . 33
4.3.2 Actuation System . . . 34
4.3.3 Sensors and State Estimation Hardware . . . 34
4.3.4 Power Supply and Electrical Integration . . . 35
4.4 ROS2-Based Control Architecture . . . 35
4.4.1 Sensing and State Estimation Pipeline . . . 36
4.4.2 Motor Control and Feedback . . . 37
4.4.3 Leg Connection Interface . . . 38
4.4.4 Policy Deployment Interface . . . 39

5 Simulation Environment and Reinforcement Learning Formulation 41
5.1 Robot Modelling in Isaac Lab . . . 41
5.2 Morphological Configurations . . . 43
5.2.1 Configuration Encoding . . . 43
5.2.2 Reset-Time Amputation Procedure . . . 43
5.2.3 Adapted Standing Posture . . . 45
5.3 Terrain and Domain Randomization . . . 45
5.3.1 Terrain Setup . . . 46
5.3.2 Domain Randomization . . . 46
5.3.3 Curriculum Strategy . . . 47
5.4 Training Algorithm and Policy Structure . . . 48
5.4.1 PPO Training Setup . . . 48
5.4.2 Policy Network Architecture . . . 49
5.4.3 Training Parameters . . . 50
5.5 Markov Decision Process . . . 51

5.5.1 Observation Space . . . 51
5.5.2 Action Space . . . 52
5.6 CPG-Based Action Interface . . . 52
5.6.1 Oscillator Formulation . . . 53
5.6.2 Policy Action Decoding . . . 55
5.6.3 Joint Target Generation . . . 55
5.6.4 Morphology Dependent Mapping . . . 57
5.7 Reward Design . . . 58
5.7.1 Main Reward Terms . . . 59
5.7.2 Reward Terms and Weights . . . 59
5.8 Termination Conditions . . . 61

6 Policy Deployment and Sim-to-Real Integration 63
6.1 Policy Application . . . 63
6.1.1 Checkpoint Loading and Policy Inference . . . 64
6.1.2 CPG Application and Joint Remapping . . . 64
6.2 Real-Robot Observation Reconstruction . . . 64
6.2.1 Observation Noise and Velocity Estimation Issues . . . 65
6.3 Real-Time Execution and Actuator Interface . . . 65
6.4 Iterative Sim-to-Real Adaptations . . . 66
6.5 Deployment Debugging Procedure . . . 67

7 Experimental Evaluation 69
7.1 Simulation Evaluation . . . 69
7.2 Deployment Experimental Setup . . . 72
7.3 Real-World Policy Execution . . . 73
7.4 Motor Tracking Analysis . . . 74

8 Conclusions and Future Work 79
8.1 Limitations . . . 80
8.2 Future Work . . . 80

Acknowledgements 87

List of Tables

5.1 Main parameters of the simulated hexapod model in Isaac Lab. 42

5.2 Binary leg configurations considered in the training setup 44

5.3 Domain randomization parameters used in the training environment. . . . 47

5.4 Policy architecture and PPO hyperparameters used during training. 51

5.5 Morphology dependent oscillator assignment lookup table 58

5.6 The reported weights are the baseline values used and were adjusted experimentally. 60

5.7 Termination conditions used during training. 61

7.1 Summary of the final simulation performance for the six-legged policy on uniform terrain. 72

7.2 Observation configurations used during real-world policy execution. 73

7.3 Comparison of policy outputs for the two velocity-observation configurations. 74

List of Figures

2.1 Examples of legged robot morphologies: (a) bipedal humanoid robot, (b) quadrupedal robot, (c) hexapod robot, and (d) hybrid wheeled-legged platform. Images adapted from Wikimedia Commons for (a) and (b), from the robotic platform used in this work for (c), and from the M4 Morphobot work for (d) [3]. 15

2.2 Basic reinforcement learning interaction loop. The agent receives an observation from the environment, selects an action, and receives a reward signal used to improve its behavior over time. 16

2.3 Sim-to-real transfer setting for the hexapod platform: (a) robot model in the simulation environment and (b) corresponding physical robot used for real-world deployment. 19

3.1 Comparison between the basilisk locomotion on water and its standard quadrupedal posture on land. 24

4.1 Overview of the reconfigurable hexapod platform: (a) CAD model showing the mechanical layout and onboard components, and (b) corresponding physical prototype used for real world locomotion experiments. 28

4.2 Initial hexapod configuration considered during the early development phase. 29

4.3 CAD model of the leg detachment mechanism. 30

4.4 Electrical contact interface used for the detachable leg modules: (a) contact connector integrated into the leg-side interface and (b) magnetic pogo pin connector used as reference for the contact-based electrical connection. . . 31

4.5 Examples of morphology-dependent standing postures 32

4.6 Hardware architecture of the reconfigurable hexapod platform. 33

4.7 Top view of the physical hardware integration on the hexapod robot. . . . 33

4.8 ROS2-based control architecture of the reconfigurable hexapod platform . . 36

5.1 Examples of simulated robot morphologies used during training 45

5.2 Examples of rough terrain configurations used in simulation. The first terrain introduces irregular height variations, while the second terrain contains small blocks 46
5.3 Actor-critic training loop used by PPO. 49
5.4 Actor network architecture used to generate the policy action. 50
5.5 Overview of the CPG-based action interface used during training 53
6.1 The divergence between the estimated velocity and the commanded values highlights the drift affecting the IMU-based velocity estimation. 65
7.1 Mean reward during training for the six-legged policy on uniform terrain. . 70
7.2 Mean episode length during training. The final policy reaches the timeout limit of 1000 steps. 70
7.3 Evolution of the main reward terms during training. 71
7.4 Planar and yaw velocity tracking errors during training. 72
7.5 Qualitative trajectories of the robot during the real-world experiments. The dashed lines indicate the approximate paths followed by the robot over time. 74
7.6 Motor tracking example for the first joint of leg 1. The plot compares the desired joint angle with the measured angle returned by the DYNAMIXEL feedback. 75
7.7 Motor tracking example for the second joint of leg 1. The plot compares the desired joint angle with the measured angle returned by the DYNAMIXEL feedback. 76
7.8 Motor tracking example for the third joint of leg 1. The plot compares the desired joint angle with the measured angle returned by the DYNAMIXEL feedback. 76

Chapter 1

Introduction

In the last years a great effort has been put in order to develop autonomous robots able to work in real complex environments. Applications as search and rescue, building inspection and exploration of hard accessible and hard replicable environments requires robots capable to move in a reliable way on both structured and unstructured terrain.

In those contexts traditional wheel robots are usually limited, as their mobility is strongly influenced by terrain characteristics. Robots with articulated locomotion are a promising solution, thanks to their capacity to adapt himself to complex surfaces as we do as humans.

In particular hexapod robots give a good balance between stability and flexibility, as they can keep the contact with more than one leg at the same time, ensuring more robustness then configuration with a lower number of legs.

Despite these advantages, the problem of locomotion in real environment is still open. The coordinated locomotion control of multiple degrees of freedom in real environments, combined with the presence of contacts with the terrain that are difficult to model with accuracy, makes complex the design of robust and efficient control. This scenario motivates research approaches that want to face challenges in a more flexible way and that they are able to manage the complexity of the locomotion in real environments.

1.1 Objectives

The main objective of this thesis is the development of a locomotion system for a real legged robot based on Reinforcement Learning. In particular, the idea is to train a policy in a simulated environment, defining fundamental components during the training process, and then integrate the policy into a physical robot system.

A key aspect is the implementation of a policy that can be effectively applied on a real robot, with the goal to obtain a working system that is able to do a locomotion task in a stable manner. In this context, the simulated environment is used as a tool for the training while the real system represents the final validation of the approach.

Additionally, the proposed method is designed to allow a certain degree of adaptability of the locomotion behaviour with respect to the robot configuration, enabling the system to mantain functionality under different conditions.

In the end, the thesis aims to analyze the behavior of the final system bringing the attention on the capability of the robot to perform the locomotion task in a stable manner as terrain conditions and robot configurations changes.

1.2 Thesis Structure

The thesis is organized as follows. At the beginning there will be a background about legged robots and reinforcement learning introducing the main problems of transfer the learning from the simulation to the reality.

Afterwards, the platform and the adopted software architecture are introduced. Then is presented the simulation environment used for the policy training, describing reinforcement learning and training methods.

In the following chapters the aspects of system integration and policy deployment on the real robot are addressed. Finally, the experimental results are presented, discussing about limitations of the work and possible future developments.

Chapter 2

Background and Related Work

This chapter provides the main concepts about legged robotics, reinforcement learning and sim-to-real transfer. These topics constitute the main theoretical foundations of this work and provide the necessary context to understand the design choices adopted.

In particular characteristics of articulated locomotion robot systems, the main problems of control methods, approaches based on learning and the challenges that you have to face for a sim-to-real problem are presented .

2.1 Legged Robotics

Legged robots are robotic systems designed to achieve locomotion tasks through articulated limbs inspired by the movement of humans and animals. Unlike wheeled robots, legged platforms rely on discrete contact points with the ground, allowing them to move across uneven terrains, obstacles and discontinuous surfaces. For this reason legged robots are relevant in environments where traditional locomotion systems are limited by the need for regular contact with the ground. [1]

Despite these advantages, legged robots are difficult to design and control as their locomotion depends on non linear dynamics and interaction between the robot and the environment. As a consequence different morphologies have been developed, each one introducing different trade-offs between stability, mobility and complexity. [2]

2.1.1 Main Legged Robot Morphologies

Legged robots can be classified according to the number of their limbs. Bipedal robots are designed to mimic human locomotion by walking on two legs while maintaining an upright posture. This configuration allows these robots to operate in environments specifically designed for humans such as indoor spaces and structured environments.

One of the main characteristics of bipedal systems is the requirement of balance and stability control. During locomotion, only one leg is usually in contact with the ground

while the other is swinging, making the system inherently unstable. As a result, bipedal robots require advanced control strategies to keep the body in balance and ensure a reliable motion. Ensuring stable locomotion in real-world environments continues to be an open problem, particularly when dealing with uneven terrain or external disturbances.

Quadruped robots are characterized by a four legged structure, typically with a left-right symmetric structure. This design provides a balance between stability and mobility, making quadrupedal systems one of the most widely studied configurations in legged robotics.

An advantage of quadrupedal robots is their ability to adapt to a wide range of terrains. Thanks to the coordination of their four limbs, these systems can maintain stable locomotion while going through uneven surfaces, slopes and obstacles. Additionally, quadrupeds can exploit various gait patterns, such as walking, trotting, and galloping, which allow them to achieve both stable and dynamic locomotion depending on the task requirements.

Robots with more than four legs, such as hexapod or octopod systems, further increase static stability and redundancy. These platforms can maintain multiple simultaneous contacts with the ground, which simplifies balance control and improves robustness. However, increasing the number of legs also increases the degrees of freedom, the control and mechanical complexity and the coordination among limbs.

Hybrid configurations, such as wheeled legged robots, combine the efficiency of wheeled locomotion on flat surfaces with the adaptability of legged motion in complex environments. Although these configurations are less common, they highlight the variety of possible designs in mobile robotics.
Examples of different morphologies are shown in Fig. 2.1

2.1.2 Hexapod Robots

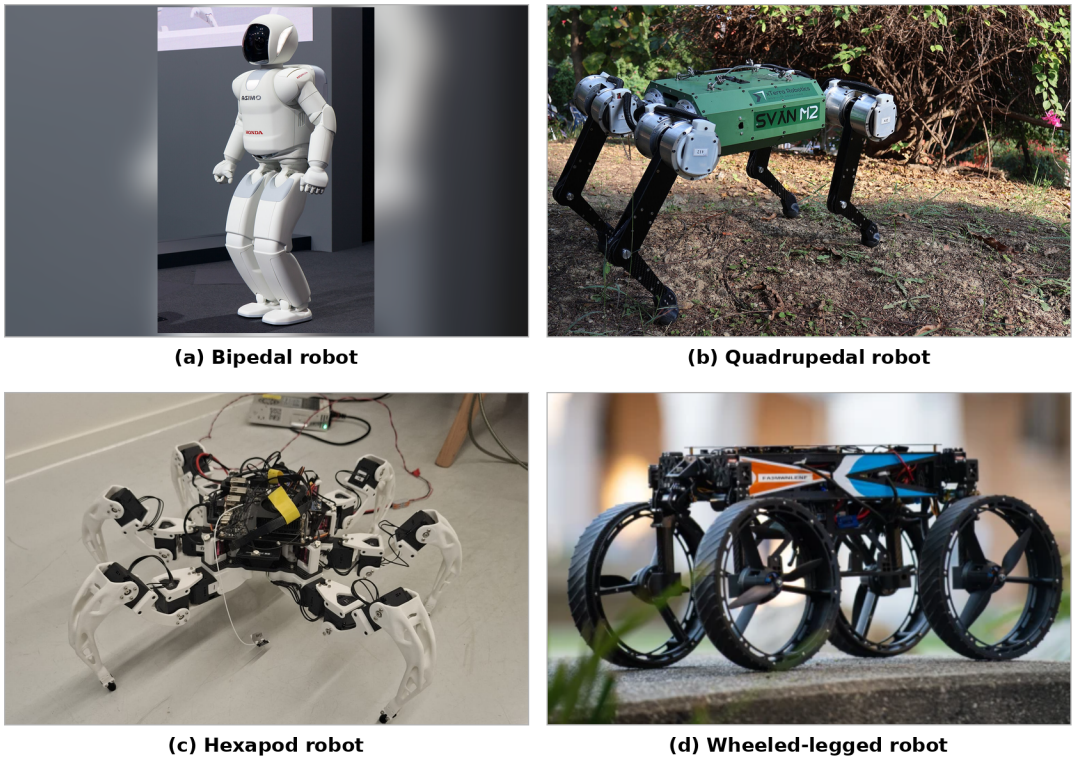
Hexapod robots are characterized by a six legged structure. These robots are inspired by biological systems such as insects and they are designed to achieve locomotion through coordinated movements of multiple limbs.[5] Compared with bipedal and quadrupedal systems, hexapods have a higher number of degrees of freedom(DoFs) which influences both their capabilities and their control complexity.[6]

One of the main advantages of hexapod robots lies in their inherent stability. Thanks to the presence of six legs, it is possible to have multiple points of contact with the ground at all times. This property enables hexapod robots to use simple and robust gait patterns, such as the tripod gait, where three legs are in contact with the ground while the other three are in the swing phase. This capacity makes hexapod robots suitable for locomotion on uneven and unstructured terrains.[4]

In addition to stability, hexapod robots provide a high level of redundancy. The presence of multiple limbs allows the system to modify its locomotion strategy depending on terrain conditions, obstacles and external disturbances. This redundancy also improves robustness in the presence of failures or changes in morphology, such as the loss of one or more limbs.[8]

However, these advantages come at the cost of increased control complexity. The most difficult task is the coordination of multiple legs managing a large number of degrees of freedom and the synchronization of different limbs. Moreover the interaction with environment introduces non linear dynamics making it difficult to design control strategies both efficient and generalizable.

From an application point of view, hexapod robots are well suited for tasks in complex and hazardous environments, such as search and rescue, space missions and operations in difficult terrains. Their ability to maintain stability and adapt their locomotion behavior makes them a valuable solution in scenarios where reliability is critical. Due to these characteristics, hexapod robots represent an interesting platform for studying strategies in systems where both adaptability and control complexity are important.



(a) Bipedal robot **(b) Quadrupedal robot**

(c) Hexapod robot **(d) Wheeled-legged robot**

Figure 2.1: Examples of legged robot morphologies: (a) bipedal humanoid robot, (b) quadrupedal robot, (c) hexapod robot, and (d) hybrid wheeled-legged platform. Images adapted from Wikimedia Commons for (a) and (b), from the robotic platform used in this work for (c), and from the M4 Morphobot work for (d) [3].

2.2 Reinforcement Learning for Robot Locomotion

Reinforcement learning has emerged as a powerful framework for solving complex control problems in robots, particularly in scenarios where explicit modeling of the system dynamics is difficult. An agent learns to perform a task through interaction with the environment, improving its behavior based on a reward signal that evaluates its actions. [9]

In general, Reinforcement Learning relies on a trial and error process in which the agent observes the current state, selects an action, and receives feedback from the environment in the form of a reward. By iteratively interacting with the system, the agent learns a policy that maps states to actions with the goal of maximizing the cumulative reward over time. The basic interaction loop between the agent and the environment is shown in Fig. 2.2. At each step, the agent receives an observation, selects an action, and receives feedback from the environment in the form of a reward.

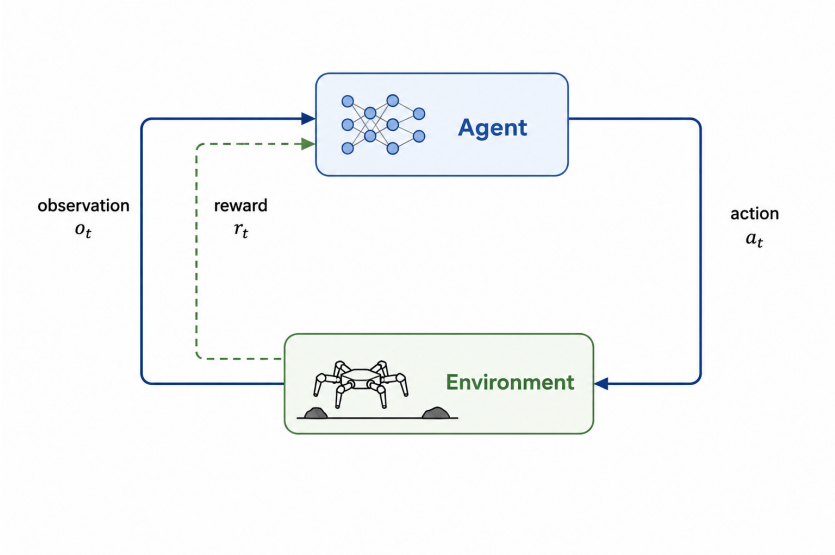


Figure 2.2: Basic reinforcement learning interaction loop. The agent receives an observation from the environment, selects an action, and receives a reward signal used to improve its behavior over time.

In recent years the integration of deep learning techniques has led to the development of the deep reinforcement learning, enabling the use of neural networks to approximate control policies directly from high dimensional sensory inputs. This allows robotic systems to learn complex behaviors by mapping perception to action without relying on precise analytical models.

2.2.1 RL for Legged Locomotion

In the context of legged reinforcement learning provides an innovative approach to address the challenges associated with locomotion in complex environments. Traditional model based methods often require accurate models of robot dynamics and of the surrounding environment. The interactions between robot and terrain are difficult to model precisely because they depend on terrain properties, impacts, friction and external disturbances. For this reason learning based approaches allow robots to learn a control policy directly through the interaction with the environments and adapt their behavior to different operating conditions [10, 7].

In locomotion tasks the objective of the learning process is to generate stable and efficient movement patterns. These behaviors come from the optimization of a reward function that encodes desired properties such as stability, energy efficiency and tracking performance. Through iterative training, the robot learns to coordinate its joints and adapt its motion strategy to different environmental conditions.

One of the main advantages of reinforcement learning in this context is its ability to handle high dimensional control problems and to discover complex behaviors that are difficult to design manually. This has enabled the development of robots capable of performing a wide range of locomotion tasks, including walking, running, jumping, and navigating complex terrains. Representative works have shown that reinforcement learning can be used to train agile locomotion policies in simulation and transfer them to real legged robot. [11]

However, applying reinforcement learning to legged locomotion also introduces several challenges. The learning process is computationally expensive and requires a large number of interactions with the environment. In addition, as in many data driven approaches, the quality of the data used during training is very important determining the effectiveness of the learned policy. In practice, this data is often generated in simulated environments, which may not always capture all the complexities of the reality. Moreover, the design of the reward functions is critical, as it strongly influences the resulting behavior.

These considerations have led to the development of different learning paradigms and control strategies, aimed at improving performance, robustness, and generalization.

2.2.2 Model-Free and Structured Reinforcement Learning Approaches

Model-free Reinforcement Learning approaches represent one of the most widely adopted paradigms for legged locomotion, as they allow the direct learning of control policies through interaction with the environment, without requiring an explicit model of the robot dynamics. In these methods, the policy is typically learned by optimizing a reward function that encodes the desired locomotion behavior, such as stability, energy efficiency, or velocity tracking.

A common approach within this category is end-to-end learning, where the policy directly maps observations to control actions. This formulation enables the system to learn complex and highly nonlinear behaviors, making it particularly suitable for high dimensional

robotic systems.
Among model-free methods, policy-gradient and actor-critic algorithms are widely used for continuous robotic control. Proximal Policy Optimization is one of the most common algorithms in this context, as it provides a balance between training stability, implementation simplicity and performance in large scale environments. For this reason, PPO has been widely used in learning based locomotion pipelines. [12]

In addition, model-free approaches can be combined with imitation learning techniques to improve training efficiency and stability. In this setting, reference motions are used to initialize or guide the learning process while reinforcement learning is used to refine the policy. This can help produce more natural locomotion, although it requires suitable demonstration or reference trajectories.

In contrast to purely model-free approaches, structured and model-based methods incorporate prior knowledge into the learning process. Model-based reinforcement learning methods learn or use an approximate model of the system dynamics, allowing the agent to predict future states and optimize its behavior more efficiently. This can reduce the number of required interactions with the environment, but the performance of these methods depends on the accuracy of the learned or predefined model.

Another important class of methods is represented by hierarchical reinforcement learning, where the control policy is structured into multiple levels of abstraction. In this framework, high-level policies define motion objectives or task-level decisions, while lower-level controllers are responsible for generating the corresponding motor commands. This decomposition can improve scalability and make complex locomotion behaviors easier to learn.

Bio inspired control strategies can also be combined with reinforcement learning. Central Pattern Generators (CPGs) are models inspired by biological neural circuits that generate rhythmic motion patterns [13]. In legged locomotion, CPGs can provide structured periodic signals for gait generation, while reinforcement learning can be used to adapt their parameters according to the environment [14]. In this way hybrid approaches aim to balance the adaptability of learning based methods with the stability and structure of traditional controllers.

Despite these advantages, both model-free and structured approaches present limitations. Model-free methods are flexible and can discover complex behaviors but they often require large amounts of training data and careful reward design. Structure, model-based and hybrid approaches can improve interpretability and sample efficiency but they require domain knowledge and careful integration. For this reason the choice of the learning paradigm depends on the task, the robot morphology and the desired tradeoffs between flexibility, robustness and complexity.

2.3 Sim-to-Real Transfer

In the context of reinforcement learning for robotics, simulation environments are widely used as a primary tool for training control policies. This is mainly due to the high

cost, time requirements, and safety constraints associated with collecting data directly on real robotic systems. Simulation allows the generation of large amounts of data in a controlled and scalable setting, enabling efficient training of complex policies through repeated interactions with the environment. [15].

In addition, simulation provides a safe environment in which robots can explore different behaviors without the risk of damaging hardware or operating in unsafe conditions. This is particularly important for reinforcement learning methods, which rely on trial and error exploration and may require a large number of interactions before converging to a stable policy.

However, in order to deploy the learned policies on real robotic platforms, it is necessary to transfer the knowledge acquired in simulation to the physical system. This process is commonly known as sim-to-real transfer, and it aims to enable policies trained in simulated environments to perform effectively in real world scenarios, where conditions may differ from those encountered during training. Fig. 2.3 shows a physical hexapod robot and its model in the simulation environment.

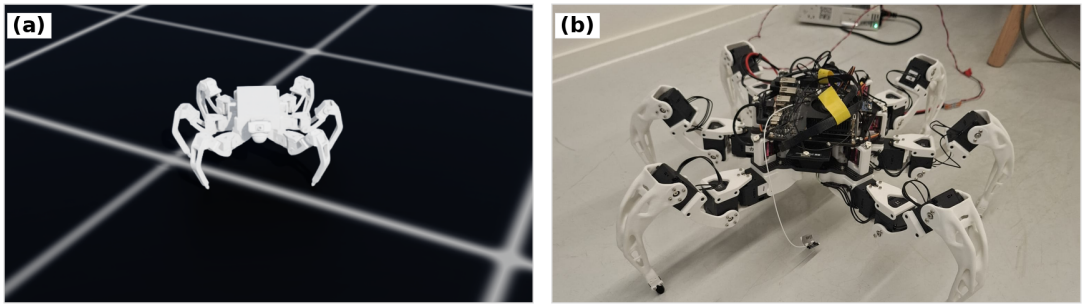


Figure 2.3: Sim-to-real transfer setting for the hexapod platform: (a) robot model in the simulation environment and (b) corresponding physical robot used for real-world deployment.

2.3.1 Reality Gap in Robotics

Despite the advantages of training in simulations, transferring learned policies to real robotic systems is a significant challenge. This difficulty is commonly known as the reality gap, which refers to the differences between simulated environments and real world conditions. [16]

The reality gap came from multiple sources, primarily related to differences in system dynamics, perception, and actuation. Simulators often rely on simplified models of physical interactions, such as contact forces, friction, and rigid-body assumptions, which may not accurately reflect real world behavior. As a result, policies trained in simulation may exploit modeling inaccuracies that do not exist in real environments, leading to bad performance when deployed on physical systems.

In addition, discrepancies in perception and sensing can affect the transfer process. Simulated sensors typically provide idealized observations, while real world measurements are affected by noise, limited resolution, and environmental variability. These differences can lead to a mismatch between the data distributions encountered during training and those observed during deployment, making it difficult for the learned policy to generalize effectively.

Finally, the actuation and control interface introduce further challenges. Real robotic systems are subject to delays, nonlinearities, and hardware constraints that are often simplified in simulation. These factors can cause discrepancies between the expected and actual effects of control actions, further increasing the difficulty of transferring learned behaviors to real world platforms.

These combined factors make the reality gap a central problem in robot learning, as even policies that achieve high performance in simulation may fail when applied to real systems. Understanding and addressing these discrepancies is therefore essential for enabling a reliable deployment of learning based control strategies.

2.3.2 Approaches for Reducing the Gap

To address the reality gap, many approaches have been proposed in the literature, targeting differences between simulation and real world environments. These methods can be categorized into techniques that aim to improve the fidelity of the simulation and those that enhance the robustness and adaptability of the learned policies.

A widely adopted strategy is domain randomization, which consists of introducing variability in the simulation by randomly perturbing visual and physical parameters during training. The main idea is to expose the agent to experience a diverse set of simulated conditions, so that the real world appears as one possible variation of the training distribution. [17] In locomotion tasks this may include randomizing masses, friction coefficients, contact properties, sensor noise and actuator parameters. [18]

Another class of methods focuses on domain adaptation, where the objective is to reduce the discrepancy between simulated and real data by aligning their feature distributions. These approaches are particularly relevant in perception driven tasks, where differences in sensor observations can significantly affect performance.

In addition, system identification techniques aim to improve the accuracy of the simulation by estimating physical parameters of the real system and incorporating them into the simulator. This reduces the mismatch in robot dynamics and allows for more reliable policy transfer.

In legged robots, actuator modeling is particularly important because the behavior of the motors can strongly influence the resulting locomotion. Real actuators introduce torque limits, saturation, delays and possible differences between commanded and executed motions. Modeling these effects in simulation can reduce the mismatch between the expected and actual behavior of the robot during deployment.

In addition to improve simulation fidelity, another strategy is to design policies that

can adapt online to change in the environment or in the robot dynamics. Rapid Motor Adaptation is an example of this approach: it combines a base policy with an adaptation module that estimates information about the current environment and robot dynamics, allowing the robot to adapt in real time to unseen conditions such as different terrains and payloads. [19].

More recent approaches explore learning based strategies to enhance adaptability, such as meta-learning and robust reinforcement learning, which aim to enable policies to adjust to variations not encountered during training. In addition, hybrid pipelines that combine simulation and real world data through iterative refinement can improve policy performance across domains.

These methods highlight the diversity of strategies developed to cover the gap between simulation and reality, reflecting the ongoing effort to achieve reliable deployment of learning based control systems on real robotic platforms.

Chapter 3

Problem Context and Motivation

This chapter focuses on the definition of the problem addressed in this work. While the previous chapter introduced the main theoretical backgrounds related to legged robotics, reinforcement learning, and sim-to-real transfer, the objective here is to describe the challenges associated with locomotion in real world scenarios.

In particular, the chapter outlines the key difficulties of the problem, discusses the limitations of existing approaches, and motivates the need for locomotion strategies that can adapt to both environmental conditions and changes in robot morphology.

3.1 Problem Definition

Legged locomotion in real world environments is a complex problem that requires robots to operate in unpredictable conditions. In nature, many animals show adaptability in their locomotion strategies, changing their gait or modifying the use of their limbs depending on the situation. For example the basilisk lizard is able to switch from a quadrupedal locomotion on solid ground to a bipedal motion when running on water [20], exploiting different locomotion strategies to adapt to the environment as it is shown in fig. 3.1.

(a) Common basilisk running on water. Image source: BBC Science Focus Magazine, image credit: NaturePL.

(b) Plumed basilisk on land. Image source: HowStuffWorks, image credit: KenCanning/Getty Images.

Figure 3.1: Comparison between the basilisk locomotion on water and its standard quadrupedal posture on land.

Similarly, insects can continue to walk even after losing one or more limbs [21], dynamically modifying their gait to compensate for structural changes.

Inspired by these behaviors, robotic locomotion systems should not only achieve stable movement but also to adapt to changes in both the environment and their own physical structure. Traditional approaches typically focus on designing controllers capable of generating stable motion patterns for a specific robot configuration, often assuming a fixed morphology and well defined system dynamics.

However robotic systems may experience changes in their physical structure, either due to damage, hardware limitations, or design choices aimed at increasing flexibility. Under these conditions, locomotion cannot be treated as a problem that depends only on the external environment, but must also account for the relationship between the control policy and the robot morphology itself.

Therefore, the objective is not only to generate stable and efficient motion, but also to enable the system to adapt its behavior according to its current morphology and its interaction with the surrounding environment. The robot is treated as an agent whose locomotion depends on the coupling between control, morphology and environmental conditions.

This highlights the need for locomotion strategies that can be generalized to different conditions and configurations, representing a key challenge in the development of adaptable and resilient robotic systems.

3.2 Existing Approaches

Existing approaches in legged locomotion have achieved significant progress in robots operation in complex and unstructured environments. Recent works have focused on improving robustness and adaptability, allowing robotic systems to maintain locomotion under disturbances or under partial damage conditions. [8] These approaches demonstrate

the effectiveness of both model based and learning based methods in handling unexpected events.

However, many of these methods are developed under the assumption of fixed robot morphology, [23] where the physical structure of the system remains unchanged during operation. In this context, adaptation is mainly obtained by compensating for the external disturbances through the control policy, rather than explicitly considering variation in the physical configuration of the robot.

Morphological changes are usually treated as exceptional conditions, such as failures or degradations, rather than as an intrinsic dimension of the locomotion problem. This limits the ability of existing approaches to generalize across different configurations, where the number of limbs or the structural properties of the system may vary. [24]

Overall while current methods provide robustness within a given configuration, the role of morphology is often not explicitly considered as a central element in the locomotion problem.

3.3 Proposed Approach

The considerations discussed in the previous sections highlight the need for locomotion strategies that do not rely exclusively on a fixed robot configuration but explicitly account for the interaction between the robot and the environment. The objective is to allow the system to adapt its behavior according to its physical configuration and the surrounding conditions.

This motivates the development of approaches in which locomotion is treated as an adaptive process. Locomotion behavior emerges from the interaction between the control policy, robot morphology, and the environment. [22] In particular learning based methods such as reinforcement learning provide a suitable framework to address this problem, since they allow control policies to be optimized through direct interaction with the environment.

Reinforcement learning enables the system to learn a control policy directly through interaction with the environment. This is particularly advantageous if the goal is to capture the coupling between morphology and the environment conditions. When the robot configuration varies, the learned behavior should be able to adapt to both external changes and structural variations of the platform.

Within this vision, the focus of this work is on the development of locomotion strategies that can operate under different conditions, providing a foundation for future extensions toward more advanced adaptive and morphology aware systems.

The ability to adapt to both structural variations and environmental interactions is essential for enabling robotic systems to operate in a wide range of real world scenarios.

3.4 Application Context

The ability to adapt locomotion strategies to both environmental conditions and morphology changes is particularly relevant in applications where robustness and flexibility is very important.

For example in search and rescue operations robots are required to navigate in highly unstructured environments such as debris and uneven terrain. In these contexts the ability to maintain mobility despite changes in contact conditions is critical for ensuring reliable operation.

In maintenance and inspection tasks, robots are often deployed in constrained environments like underground pipelines, ventilation systems or air ducts. These scenarios impose strict geometric constraints, requiring locomotion strategies that can adapt his morphology to fit in these type of environments.

Another relevant application domain is space exploration, where robots must operate in extreme and highly variable conditions, often without the possibility of human intervention. In these scenarios, the ability to maintain locomotion capabilities under unforeseen interactions becomes a key requirement for autonomous robots.

In the end, the development of adaptable robotic systems is essential for enabling to operate reliably across different scenarios, where surrounding configurations may vary over time.

Chapter 4

System Overview

This chapter presents the physical system developed to support the experimental study of adaptive locomotion on a reconfigurable hexapod. After introducing the motivation for adaptive locomotion, it is necessary to describe the robotic platform on which this concept is implemented and tested. The system is based on a reconfigurable hexapod robot, designed to combine the stability of multi-legged locomotion with the possibility of modifying the physical configuration of the robot.

The main design requirements that guided the development of the platform will be first introduced. Then, the reconfigurable hexapod structure is described, with particular attention to the general mechanical layout and the leg detachment mechanism. The hardware architecture is then presented, including the onboard computing units, the actuation system, the sensing components and the power supply. Finally, the ROS2-based control architecture is introduced showing how different components are integrated.

4.1 System Requirements and Design

The system was designed and developed in order to provide an experimental platform for studying adaptive locomotion on a reconfigurable legged robot. Since this work is not limited to simulation, the physical design must consider the requirements imposed by locomotion and policy deployment.

A central requirement of the platform is the reconfigurability. Since the problem addressed in this work is locomotion under different physical configurations, the robot must be able to perform controlled changes in its morphology. The leg detachment is not only considered as a failure condition, but also as an intentional structure variation that can be used to study locomotion strategies adapted to different situations.

A multi legged structure is particularly suitable for this objective, as it provides redundancy and multiple contact points with the ground. The hexapod was chosen among different multi legged configurations because it offers a good compromise between stability, thanks to the redundancy, and complexity, allowing the robot to maintain locomotion

even when the configurations of active limbs changes.

The system requires a modular architecture. The sensing pipeline, policy inference, actuator communication and leg connection mechanism must be separated into independent components in order to test, modify and integrate each unit without changing the entire system. This modularity is important during experimental development where hardware constraints must be handled and debugged on the real robot.

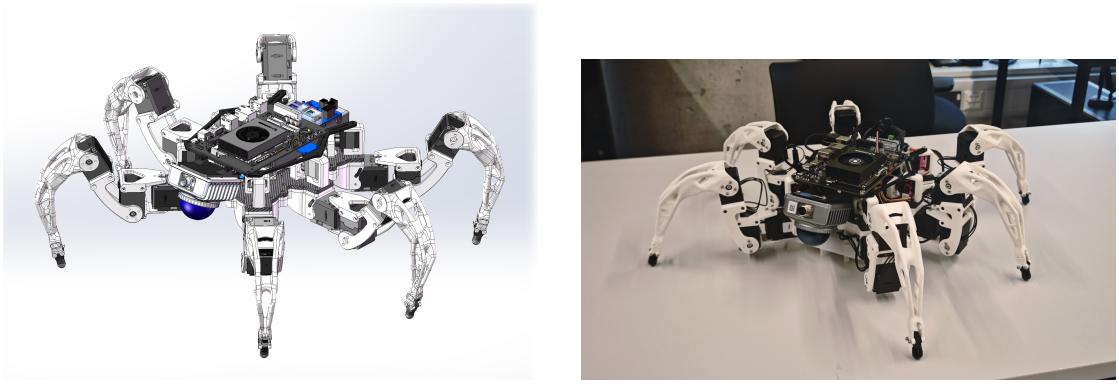
Overall, the system requirements are driven by the need to combine mechanical reconfigurability, stable locomotion, and real-time deployment. This is the foundation for the hardware and software architecture described in the following sections.

4.2 Reconfigurable Hexapod Platform

The robotic platform used in this work is a reconfigurable hexapod. Unlike conventional hexapods, where the six legs are assumed to remain permanently attached and functional, the physical configuration of this platform can be intentionally modified.

The platform combines a central body, six detachable leg modules, onboard electronics, sensing components and a dedicated connection mechanism. In this section, the general mechanical structure of the robot is first described, followed by the leg detachment mechanism that enables the reconfigurable nature of the platform.

As shown in Fig. 4.1, the platform was developed from a CAD model and implemented as a physical prototype.



(a) CAD model. (b) Physical prototype.

Figure 4.1: Overview of the reconfigurable hexapod platform: (a) CAD model showing the mechanical layout and onboard components, and (b) corresponding physical prototype used for real world locomotion experiments.

4.2.1 Mechanical Structure

The mechanical structure of the robot is based on a hexapod layout composed of a central body and six modular legs, arranged symmetrically on the left and right sides of the body.

This configuration provides multiple contact points with the ground and offers a stable basis for locomotion experiments.

Each leg is designed as an independent module and uses a three joints kinematic chain. This configuration allows the foot position to be controlled in three dimensional space while keeping the mechanical and control complexity within a manageable range. The modular nature of the legs is essential, since each limb can be considered as an independent unit.

During the initial development phase, a hexapod configuration with four joints per leg was also considered and tested. This design increased the theoretical mobility of each limb, but it also introduced an additional level of complexity in the kinematic chain, actuator coordination and control pipeline. This additional degree of freedom did not provide a proportional advantage for the final task of stable locomotion. Therefore, a simpler three joint leg configuration was preferred, reducing mechanical and control complexity while keeping the required capabilities for the experimental study. The initial four joint leg configuration considered during early development is shown in Fig. 4.2.

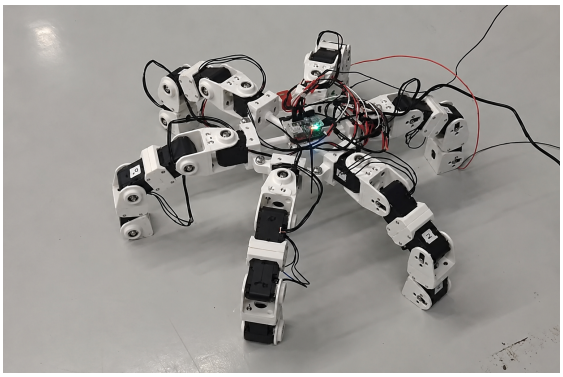


Figure 4.2: Initial hexapod configuration considered during the early development phase.

The resulting structure is a compromise between stability, modularity and controllability. It provides enough flexibility without introducing complexity that would make real world deployment and policy transfer more difficult.

Since the platform needs to support intentional changes in morphology, the connection between each leg and the body is a key element of the system. The mechanism used to attach and detach the leg modules is described in the following subsection.

4.2.2 Leg Detachment Mechanism

The leg detachment mechanism is one of the key components of this hexapod platform. Each leg is connected to the central body through a dedicated mechanical interface, designed to provide a stable connection during locomotion while allowing controlled release when a change in morphology is required. The mechanism must satisfy two opposite requirements: it must provide a stiff structural connection during walking, but it must also be releasable when the robot configuration has to be modified.

The interface must be sufficiently rigid to withstand the loads generated by the body and interaction between the leg and the ground. At the same time the mechanism must also remain compact and lightweight, as it is replicated for all six legs and it contributes to the overall mass and size of the robot.

The design is inspired by robotic tool changing mechanisms, where a mechanical locking interface is used to attach and release modules in a repeatable way. However, the industrial pneumatic actuation commonly used in tool changers is not suitable for this platform, due to the limited space and weight constraints of a compact hexapod body. For this reason, a mechanically actuated solution was adopted.

The mechanism consists of a small servo motor that drives a cam which converts a rotational motion into the linear displacement of a locking slider. when the slider moves forward it pushes a set of locking balls outward into corresponding grooves on the leg-side interface. This allows the leg module to be locked during locomotion through the mechanical constraints between body-side and leg-side plates. When the servo drives the cam in the opposite direction, thanks to the same mechanism, the locking balls are released, allowing the leg detachment from the body. In order to reduce power consumption and avoid unnecessary heating of the locking actuator during normal walking the contact force between the cam and the slider is arranged so that the mechanism can remain locked without requiring continuous torque from the servo. The leg detachment mechanism is shown in Fig. 4.3.

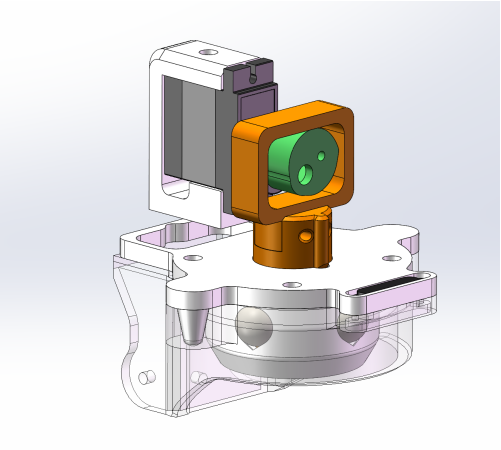
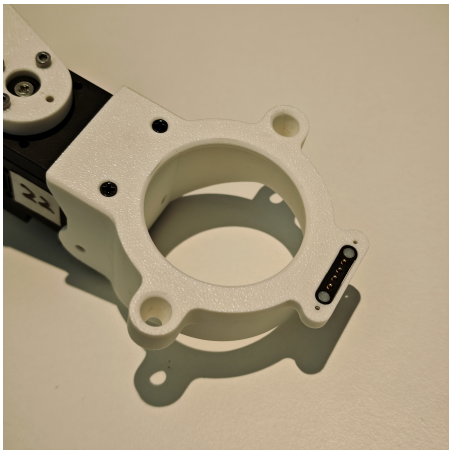


Figure 4.3: CAD model of the leg detachment mechanism.

In addition to the mechanical connection, shown in Fig. 4.4, it is important to preserve electrical continuity when the leg is attached, allowing power and control signals to be transmitted from the body. A contact based connector is used so that power and communication signal can be transmitted when the leg is attached, allowing the module to separate during detachment.



(a) Integrated contact interface.

(b) Magnetic pogo pin connector.

Figure 4.4: Electrical contact interface used for the detachable leg modules: (a) contact connector integrated into the leg-side interface and (b) magnetic pogo pin connector used as reference for the contact-based electrical connection.

From the perspective of this work, the detachment mechanism is not treated only as a response to an accidental failure but it is part of the robot design and enable intentional reconfiguration. This provides the physical basis for studying different behaviors depending on the set of active limbs.

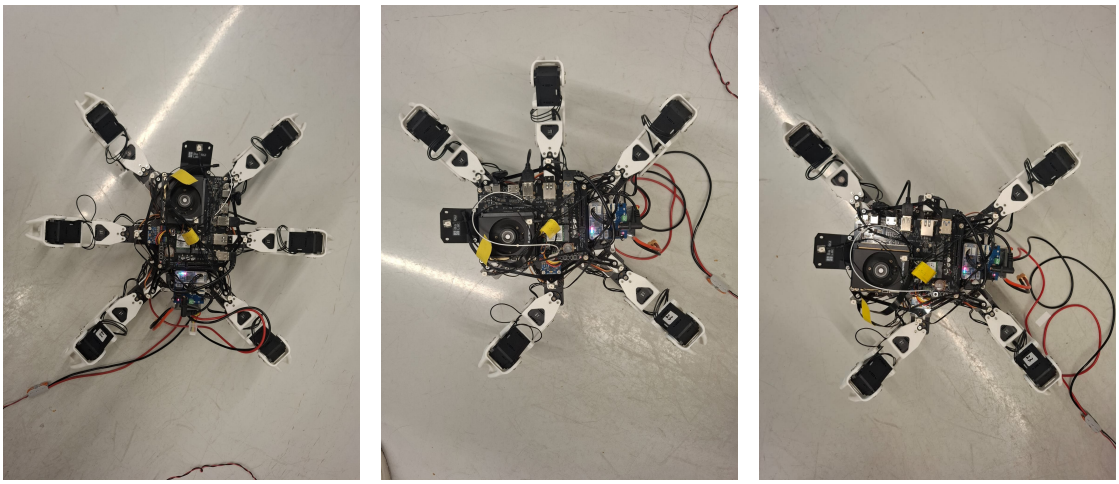
4.2.3 Adaptable Standing Configuration

The adaptable standing configuration was not part of the initial design of the system. In the first tests, it was used the same standing posture adopted for the complete six legged robot. However, both simulation and real world experiments showed this posture was not always suitable when one or more legs were removed.

In simulation, the original standing posture was less stable during locomotion for reduced leg configurations, especially when the missing legs were the external legs of each side. Some of these configurations were highly unstable and even small movements could lead to a loss of balance and falls. Similar issues were observed during hardware tests, where motors on the side of a missing leg sometimes had difficulty supporting the robot weight causing a motor shutdown.

For this reason the nominal standing configuration was adapted according to the active leg configuration. When an external leg is removed, the middle leg on the same side is rotated by 45° toward the missing leg. This rule is applied independently on the left and the right sides of the robot. This modification allows the middle legs to partially compensate for missing legs, improving the distribution of the body weight over the available contact points.
At the same time, the set of valid configurations was limited to avoid excessive load on the remaining actuators. In particular the robot was required to keep at least two active legs

on each side of the body. Configurations that violated this constraint were avoided, since they increased the risk of motor shutdown during real world experiments due to motor overload. An example of the morphology-dependent standing postures used for different active-leg configurations is shown in Fig. 4.5.



(a) Nominal posture with six active legs.

(b) Adapted posture with five active legs.

(c) Adapted posture with four active legs.

Figure 4.5: Examples of morphology-dependent standing postures

This adjustment highlights that morphological reconfiguration does not only affect the number of active limbs, but also changes the reference configuration around which locomotion is generated. The standing posture is not a fixed property of the hexapod model but changes depending on morphology configuration.

4.3 Hardware Architecture

The reconfigurable mechanical structure of the robot requires hardware that can support onboard computation, sensing, actuation and power distribution. Since the platform is intended for real world locomotion experiments, the hardware must provide reliable communication with the actuators and sufficient computational resources for running the control pipeline.

The hardware architecture consists of a main onboard computer, low-level controllers, joint actuators, sensing units and a power supply system. Each component plays a specific role in the overall platform, while their integration must satisfy the constraints imposed by real world locomotion.

The overall hardware architecture is shown in Fig. 4.6, while the physical integration on the robot is shown in Fig. 4.7..

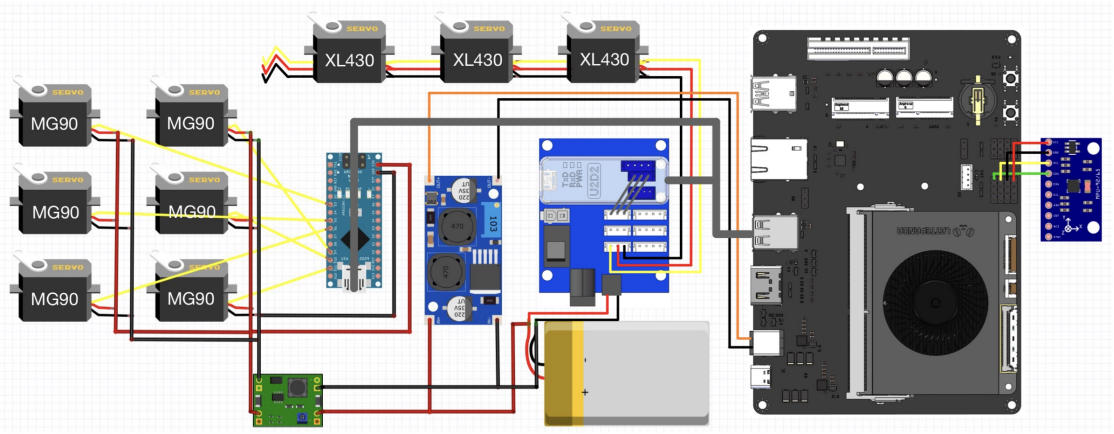


Figure 4.6: Hardware architecture of the reconfigurable hexapod platform.

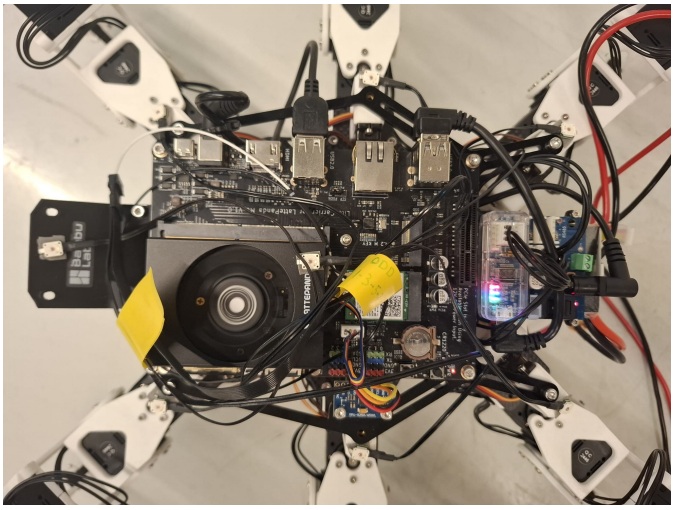


Figure 4.7: Top view of the physical hardware integration on the hexapod robot.

4.3.1 Computing Units and Low-Level Controllers

The computing architecture is divided between a main onboard computer and a dedicated microcontroller. This separation improves modularity and allows different robot components to be managed independently. The main computing unit is responsible for running the high level control pipeline, including policy inference, state processing and communication with the actuators. A LattePanda Mu with an Intel Core i3-N305 processor was selected because it provides a higher computational capability than a typical microcontroller, making it suitable for running real time locomotion and policy inference.

Low level auxiliary tasks that do not require a lot of computational resources are handled by an Arduino Nano. In particular, it is used to control the leg locking mechanism by

generating the PWM signals required by the servos. This microcontroller was selected because of its small size and simple programming workflow.

This separation between onboard computation and low level auxiliary unit reduces the workload of the main computer and improves the modularity of the system.

4.3.2 Actuation System

The actuation system is divided into two main parts: the joint motors used for locomotion and the servos used for the leg connection mechanism. The first group directly generate the movements of the robot during walking, while the second group is responsible only for the reconfiguration task.

The joint actuators are DYNAMIXEL XL430-W250 bus servos. Each leg is actuated by three motors: coxa, femur and tibia joints, for a total of eighteen actuators. These motors were selected because they have a compact design and provide serial bus communication, position and velocity feedback and configurable control parameters, which are important for coordinated multi-joint locomotion.

The bus based architecture reduces wiring complexity and allows multiple motors to share the same communication line. This is particularly important when a large number of actuators must be controlled within limited space. The availability of joint feedback also makes it possible to have information about motor states and integrate them into the control pipeline.

One important constraints is the half-duplex nature of the DYNAMIXEL communication bus imposes timing constraints. Motor command and feedback data cannot be transmitted simultaneously on the same data line and they must be coordinated within the control loop. [32, 33] This aspect is particularly relevant for real time locomotion since irregular timing can affect the stability of the generated gait.

In addition to locomotion actuators, the platform uses small MG90 servos for the leg locking mechanism. These servos only move the locking mechanism between locked and unlocked states. Their compact size and sufficient torque make them suitable for this auxiliary function without adding excessive weight to the robot.

4.3.3 Sensors and State Estimation Hardware

The robot integrates onboard sensing hardware to provide the state information required by the locomotion control pipeline. The sensing system should rely primarily on onboard sensors rather than external measurement as the platform is intended for real world applications.

The main proprioceptive sensor is an MPU9250 inertial measurement unit, which provides linear acceleration, angular velocity and magnetometer measurements. These signals are used to estimate information related to the base motion of the robot. These signals are particularly important for constructing the policy observation in order to maintain awareness of the robot body state during locomotion. In addition, an onboard LiDAR

sensor is used to provide information about the surrounding environment. This sensor is included as an additional sensing component, allowing the platform to support future experiments that involves environmental perception. In the context of this work, the LiDAR is an additional sensor that can be used to enrich the policy input and investigate how environmental information affects locomotion behavior.

During laboratory experiments, an external PhaseSpace motion capture system was used as an additional sensing source. This system provides accurate measurements of the robot pose and velocity in a controlled environment. It is useful for validation, debugging and comparison with onboard estimation.

4.3.4 Power Supply and Electrical Integration

The robot is powered by a 3S LiPo battery with a nominal voltage of 11.1 V. This voltage level is suitable for directly supplying the DYNAMIXEL actuators, avoiding an additional voltage regulator on the main motor power line. This is important because the joints can require high current peaks during locomotion, and an undersized regulator could introduce instability in the motor supply.

Dedicated voltage converters are used since different components require different supply voltages. A step-down converter provides a 5 V supply for the Arduino Nano and the MG90 servos, while a boost converter provides the voltage required by the LattePanda Mu. A common ground is maintained between the battery, actuators, onboard computer, microcontroller, sensors and voltage conversion modules, ensuring a stable reference for communication between different components.

In the end since the robot frame includes conductive carbon fiber components, electronic modules on the structure require appropriate insulation to reduce the risk of short circuits. These aspects are particularly relevant where power stability and communication reliability directly affect the behavior of the robot.

4.4 ROS2-Based Control Architecture

The control system of the robot is implemented using ROS2, which provides a modular architecture for the integration of the hardware and software components described in the previous sections. This organization is useful for the development since each component can be tested, modified and debugged independently while remaining part of the complete pipeline.

The architecture is organized in two main data flows. The first one collects sensor measurements and motor feedback and provides the state information required by the locomotion controller. The second one receives high level joint commands generated by the controller and converts them into actuator commands. In parallel, a dedicated interface manages the leg connection mechanism, in order to control the reconfigurable structure from the software side.

In addition to the learned policy controller, this architecture supports auxiliary control

modes used for testing and debugging. Individual parts of the system must be validated before running the entire learned locomotion pipeline.

The ROS2 control architecture is shown in Fig. 4.8.

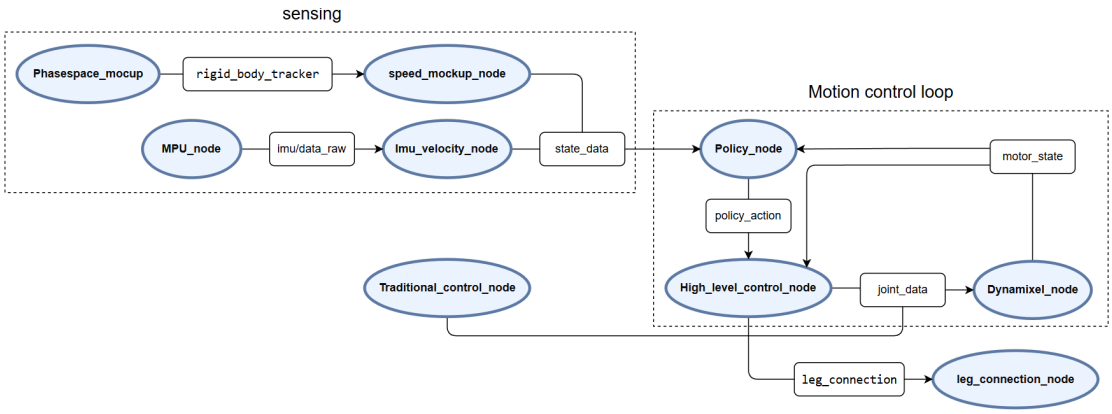


Figure 4.8: ROS2-based control architecture of the reconfigurable hexapod platform

4.4.1 Sensing and State Estimation Pipeline

The sensing and state estimation pipeline provides the state quantities related to the robot base required by the locomotion controller. Two sensing paths are implemented: an onboard IMU-based path, intended for real world autonomous operation, and an external PhaseSpace motion capture path, used during laboratory experiments as a reference source for validation and debugging.

In the onboard sensing path, an IMU node reads accelerometer and gyroscope data and publishes them on the `/imu/data_raw` topic. The accelerometer readings are converted into linear acceleration in $\mathrm{m/s^2}$, while the gyroscope readings are converted into angular velocity in rad/s. During startup, the robot is kept stationary for a short calibration phase. The mean gyroscope value measured during this interval is treated as the gyroscope bias and subtracted from subsequent measurements:

$$\boldsymbol{\omega}_b = \boldsymbol{\omega}_{raw} - \boldsymbol{b}_\omega$$

where $\boldsymbol{\omega}_b$ is the calibrated angular velocity expressed in the robot base frame. This signal contains the angular velocity of the base around the three body-frame axes, corresponding to roll, pitch, and yaw rates. It is published on `/angular_velocity` and corresponds to the `base_ang_vel` observation used during training.

The `imu_velocity_node` also estimates the orientation of the robot base by integrating the calibrated gyroscope measurements using a quaternion representation. From the estimated body-to-world orientation, the gravity direction is projected into the robot base frame. This projected gravity vector is computed as:

$$\boldsymbol{g}_b = R(q_{bw})^T \boldsymbol{g}_w$$

where q_{bw} represents the estimated body-to-world orientation of the robot base, $R(q_{bw})$ is the corresponding rotation matrix, and $\boldsymbol{g}_w = [0,0,-1]^T$ is the gravity direction in the world frame. The resulting vector is published on `/projected_gravity` and corresponds to the projected gravity observation used during training.

The same node estimates the base linear velocity from the accelerometer measurements. After the initial calibration phase, the accelerometer bias is subtracted from the measured acceleration, and the gravity component is removed. Since direct integration of accelerometer measurements is highly sensitive to noise and drift, the linear acceleration is filtered before being integrated. In the implementation, a low-pass filter is applied:

$$\boldsymbol{a}^f_{lin,b,k} = \alpha \boldsymbol{a}^f_{lin,b,k-1} + (1-\alpha)\boldsymbol{a}_{lin,b,k}$$

with:

$$\alpha = e^{-2\pi f_c \Delta t}$$

where $f_c = 5\,\mathrm{Hz}$ is the cut-off frequency. A small deadband is also applied to suppress low amplitude acceleration components caused by sensor noise. In the code, components whose absolute value is below $0.05\,\mathrm{m/s^2}$ are set to zero.

The filtered acceleration is then integrated over time to estimate the base velocity:

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \boldsymbol{a}^f_k \Delta t$$

The resulting velocity estimate is published on `/base_velocity`. To limit error accumulation, the implementation also includes a simple stationary detection condition: when the angular velocity or the filtered linear acceleration falls below a predefined threshold, the estimated velocity is reset to zero.

Since velocity estimation is based only on IMU integration is very noisy an external motion capture path was used for laboratory experiments. The PhaseSpace system provides the pose of the robot rigid body, including position and orientation. The `speed_mockup_node` computes the world-frame linear velocity by finite difference:

$$\boldsymbol{v}_{w,k} = \frac{\boldsymbol{p}_{w,k} - \boldsymbol{p}_{w,k-1}}{\Delta t}$$

where $\boldsymbol{p}_{w,k}$ is the measured body position at time step k. The measured orientation is then used to express this velocity in the robot base frame, and the resulting value is published on `/base_velocity_mocup`. This path provides a cleaner velocity estimate for controlled experiments and allows comparison with the onboard IMU-based state estimation.

The state estimation pipeline provides the policy with three main quantities: base linear velocity, base angular velocity, and projected gravity. These signals allow the deployed controller to receive information consistent with the simulation observation space.

4.4.2 Motor Control and Feedback

The motor control and feedback interface is implemented in the `/dynamixel_node`, which is the bridge between the ROS2 control pipeline and the DYNAMIXEL actuators. Its

role is to receive high level commands, send them to the motors and publish the measured joint state during execution.

The locomotion commands are expressed as desired joint positions for each of the actuators. The `/dynamixel_node` subscribes to the `/dynamixel_control` topic, where target positions are provided by the high level control interface. These commands may have different origins depending on which experiment is chosen.

Before sending the command to the motors, the desired position are converted from robot joint convention to the DYNAMIXEL position convention. This conversion is needed because the reference frame used by the servomotors does not directly correspond to the one used in the robot kinematic model. Joint limits are also applied before transmission, in order to prevent dangerous or unfeasible configurations.

After this phase of conversion and validation, the `/dynamixel_node` sends the desired position to the active actuators through the DYNAMIXEL communication bus. The implementation uses synchronous write operations, in order to transmit multiple commands in a single cycle, preserving the timing of the gait and supporting the coordination of multiple joints.

In parallel, `/dynamixel_node` reads feedback from the motors using synchronous read operations. The measured joint positions and velocities are published on the `/motor_state` topic. This feedback is used to provide joint state information required by the policy observation pipeline.

In the end, the `/dynamixel_node` provides the low level actuation interface. It ensures that the commands generated by the high level controller are translated into motor references, while publishing feedback from the actuators.

4.4.3 Leg Connection Interface

The leg connection module manages the mechanical locking and release mechanism. In the ROS2 architecture, this functionality is handled by the `/leg_connection_node`, which provides a communication layer between the high level ROS2 system and the Arduino Nano responsible for controlling the locking servos.

The `/leg_connection_node` subscribes to the `/leg_connection` topic, where it receives commands specifying which leg should be locked or unlocked. Each command contains the target leg and the desired connection state. It is possible to address a command to a single leg or broadcast the command to all legs at the same time.

After receiving a command, the `/leg_connection_node` converts it into a serial message and sends it to the Arduino Nano. The microcontroller then generates the signals required to move the MG90 servos associated with the selected locking mechanisms. In this way the low-level control of the locking system is delegated to the microcontroller for the connection mechanism.

This separation simplifies the integration of the reconfiguration mechanism in the control architecture. From the high level side the leg attachment or detachment is a discrete command, while the Arduino handles the complete mechanism. This makes the leg connection

system modular and independent from the locomotion control loop.

Overall, the `/leg_connection_node` provides the software interface required to reconfigure the structure of the robot. It allows the physical morphology to be modified through ROS2 commands, enabling locomotion experiments with different number of active legs while keeping separate the auxiliary locking mechanism.

4.4.4 Policy Deployment Interface

The policy deployment interface connects the trained locomotion policy to the real robot architecture. Its role is to reconstruct the policy observation from real sensor and motor data, execute the trained neural network, and send the resulting command to the low-level actuation pipeline.

In the implemented architecture, the `/policy_node` is responsible for executing the trained policy during experiments. The node collects the quantities required by the policy observation, including base linear velocity, base angular velocity, projected gravity, joint positions, joint velocities, previous actions, commanded velocity and leg configuration. These signals are assembled into an observation vector consistent with the one used during training.

Furthermore during the startup of the system, the `/policy_node` loads a trained PyTorch policy and once the observation is built it performs inference at runtime. The output of the policy represents the high level action produced by the learned controller. At this stage the internal conversion from policy output to joint level references is not detailed, since they depend on the policy formulation described in the following chapters.

The communication between `/policy_node` and `/high_level_control_node` is performed through the `/policy_action` topic. This topic is read by the `/high_level_control_node` which acts as an intermediate layer between policy execution and low level interface. This node receives the desired joint command, formats it into the expected structure by `/dynamixel_node` and publishes the motor command on the `/dynamixel_control` topic.

This separation makes the deployment architecture modular. The layer described can be tested independently. This structure allows an alternative command source, such as trajectory playback or debugging controllers, to use the same pipeline without using `/dynamixel_node`. In particular the playback mode can be used to test joint targets generated in simulation directly on the physical robot, in order to test separately the policy input and the policy output.

The described interface allows the integration of the trained policy into the physical control system. The focus is on how the policy is connected to sensing and actuation, while the details of the policy application are addressed in the following chapters.

Chapter 5

Simulation Environment and Reinforcement Learning Formulation

The simulation environment plays an important role in this work, since training directly on the physical robot is impractical due to safety risks, hardware constraints and time requirements. Isaac Lab is used to define the locomotion task and train the policy through parallelized environments. The robot can be exposed to different configurations in a safe and repeatable way. [25]

The chapter first describes the robot model and the leg configurations representation during training. Then the terrain setup and domain randomization strategy used to improve robustness are introduced. Finally, the reinforcement learning formulation is presented, including the observation space, action space, CPG-based action interface, reward function, and training setup.

The objective of this chapter is to describe the learning process to obtain the locomotion controller.

5.1 Robot Modelling in Isaac Lab

The simulated robot model was developed in Isaac Lab starting from the mechanical design of the physical hexapod. The robot was initially designed on SolidWorks and exported as a URDF model. The URDF description was then converted into a USD file and imported into Isaac Lab as an articulated system. In this way the simulator preserves the kinematic structure of the real platform while making the robot compatible with Isaac Lab.

The model corresponds to the physical platform described in the previous chapters. The naming convention used for the revolute joints of each leg is: `LxJ1`, `LxJ2` and `LxJ3` where

`x` identifies the leg number. Each foot is represented by a terminal link, named *Tx*, connected to the last link through a fixed joint.

In Isaac Lab the robot is configured as an articulated system with contact sensor enabled to detect interactions between the feet and the terrain, this information is useful during policy training. The initial state of the robot is defined by setting the initial base position and assigning a nominal standing posture. This posture was chosen because it is stable and distributes the load across the actuators.
The joints are modeled using implicit PD actuators. The joints are divided into three groups, corresponding to the first, the second and the third joint of each leg. These groups are associated with the hip, knee and ankle joints. Each group is assigned effort limits, velocity limits, stiffness and damping values. This model provides a simple representation of the real DYNAMIXELs, without reproducing every low level effects but providing a controllable approximation for the reinforcement learning training. The parameters used in the simulated robot model are summarized in Table 5.1.

A specific foot contact model was introduced to improve contacts simulation. A simple cylindrical collision shape was added at the tip of each foot. This additional primitive helps in the simulation of contacts between the ground and the robot, improves stability and consistency of contacts compared with using the detailed visual mesh directly.

This described model provides the basis for training locomotion policies in simulation, trying to capture the most relevant aspects of the real robot before the transfer of the policy on the real robot.

Table 5.1: Main parameters of the simulated hexapod model in Isaac Lab.

Category	**Parameter**	**Value**
Robot structure	Number of legs	6
	Actuated joints per leg	3
	Total actuated joints	18
	Joint naming convention	`LxJ1`, `LxJ2`, `LxJ3`
Initial state	Base position	(0.0, 0.0, 0.2) m
	Joint 1 initial position	0.0 rad
	Joint 2 initial position	-0.8306 rad
	Joint 3 initial position	-0.2726 rad
Actuator model	Actuator type	Implicit PD actuator
	Actuator groups	Hip, knee, ankle
	Effort limit	2.5 Nm
	Velocity limit	10.0 rad/s
	Stiffness	40.0
	Damping	3.0
Foot contact model	Cylinder radius	0.003 m
	Cylinder length	0.0005 m

5.2 Morphological Configurations

Since the objective of the training environment is to obtain a single policy capable of handling different robot morphologies, the simulation must support configurations in which one or more legs are not available. In this work, morphological variation is applied at the beginning of each episode, rather than being simulated as a continuous detachment event during locomotion. Therefore each environment is initialized with a specific leg configuration.

This section describes how these configurations are represented, how they are applied during reset and how the initial standing posture is adapted.

5.2.1 Configuration Encoding

The simulation environment must be able to represent both the complete hexapod morphology and reduced morphologies. In the training environment, each morphology is encoded through a binary vector called `leg_alive_mask`, composed of six elements, one for each leg of the robot.

Each element of the vector indicates whether the corresponding leg is active or inactive, `1` indicates an active legs and `0` indicates an inactive legs. For example, the complete six legged configuration is represented as:

$$[1,\ 1,\ 1,\ 1,\ 1,\ 1]$$

while a configuration in which the first and sixth legs are removed is represented as:

$$[0,\ 1,\ 1,\ 1,\ 1,\ 0]$$

This representation is used both to modify the simulated morphology and to condition the policy on the current robot configuration. The policy is not forced to infer the morphology only from the observed dynamics, but receives explicit information about which legs are available.

Not all possible combinations are used during training. The set of admissible configurations is selected according to mechanical and stability constraints, avoiding morphologies that would be excessively unbalanced or unrealistic for the physical robot. In particular, configurations that would leave one side of the robot with insufficient support are avoided, since they would lead to poor load distribution and a high risk of instability as already explained in the previous chapter. The complete set of binary leg configurations considered in the training setup is reported in Table 5.2.

5.2.2 Reset-Time Amputation Procedure

Morphological changes are applied at reset time. At the beginning of each episode, a leg configuration is selected from a predefined set of masks and assigned to the corresponding simulated environment. This approach allows different environments in the parallel

Table 5.2: Binary leg configurations considered in the training setup

Configuration	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**
1	1	1	1	1	1	1
2	0	1	1	1	1	1
3	1	0	1	1	1	1
4	1	1	0	1	1	1
5	1	1	1	0	1	1
6	1	1	1	1	0	1
7	1	1	1	1	1	0
8	0	1	1	0	1	1
9	0	1	1	1	0	1
10	0	1	1	1	1	0
11	1	0	1	0	1	1
12	1	0	1	1	0	1
13	1	0	1	1	1	0
14	1	1	0	0	1	1
15	1	1	0	1	0	1
16	1	1	0	1	1	0

training setup to run with different morphologies without requiring a separate robot asset for each possible configuration.

Before applying a new configuration, the robot is first restored to its complete state. This step is necessary because without restoration, residual modifications from the previous configuration could remain active, such as hidden links, disabled collisions, modified mass and inertia values, or zeroed actuator effort limits. Restoring the full robot before applying the next mask ensures that each reset starts from a clean model.

After restoration, the selected mask is applied. The legs marked as inactive are removed from the simulated model through a reset time amputation procedure. This procedure modifies the robot model at the beginning of the episode so that the selected legs no longer affect the simulation.

For each inactive leg, the visual geometry is hidden and the collision geometry is disabled. In addition, the mass and inertia of the corresponding links are set to very small values to avoid `NVIDIA PhysX` errors, so that the removed leg does not contribute significantly to the dynamics of the robot. At the actuation level, the effort limits of the joints belonging to the inactive leg are set to zero, preventing the removed limb from generating torque. The implementation also provides a restore function that re-enables the physical properties for the reasons previously explained.

This procedure makes the simulated amputation more consistent with the focus of the experiments than simply ignoring the action commands of the inactive legs. A removed leg does not produce torque and does not collide with the environment. As a result, the policy is trained on a robot whose physical morphology is actually modified in simulation, rather than only being masked at the control level.

5.2.3 Adapted Standing Posture

Changing the set of active legs also changes the support geometry of the robot and the same initial posture is not suitable for all the morphological configurations. In particular, when a leg is removed, the support points on that side are less distributed, and the robot may start from an unbalanced configuration that limits the process of learning.

To delete this effect, the initial standing position is adapted according to the selected `leg_alive_mask`. The six leg posture is used as a baseline and , as previously explained, when an external leg is inactive the middle leg on the same side is rotated toward the missing support.

In practice, if `L1` or `L3` is inactive, the joint `L2J1` is rotated toward the missing leg. Similarly, if `L4` or `L6` is inactive, the joint `L5J1` is adjusted toward the missing support location. If the middle leg itself is inactive, no compensation is applied on that side.

After the computation of the adapted posture, the new configuration is written in the simulated robot state. As a result, the generated locomotion is centered around the new standing posture. Moreover the generated motion is centered around the morphology dependent standing posture.

In this way the standing posture becomes part of each episode as the robot is initialized around a referenced configuration that is more consistent with the selected morphology. Fig. 5.1 shows examples of the complete morphology and reduced-leg configurations used in simulation.

(a) Complete morphology (b) One missing leg (c) Two missing legs

Figure 5.1: Examples of simulated robot morphologies used during training

5.3 Terrain and Domain Randomization

In addition to the robot structure, it is important to define the terrain conditions and the physical variations experienced during training.

This section describes the terrain configurations used in simulation, the randomized parameters and the curriculum strategy adopted during training.

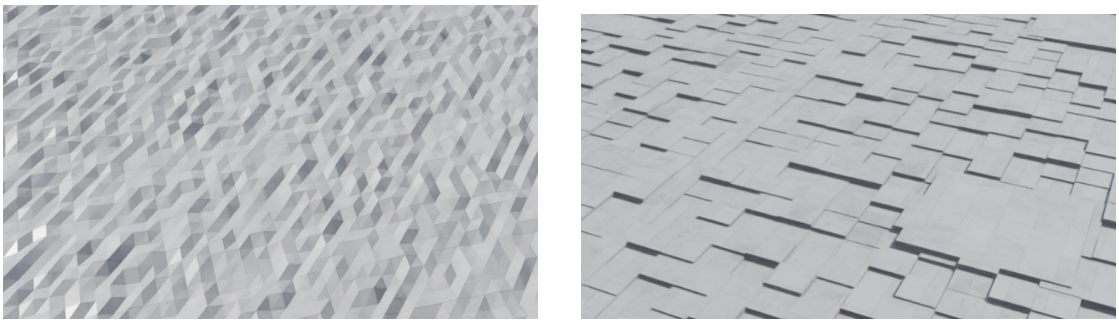
5.3.1 Terrain Setup

The training setup includes two terrain configurations: flat terrain and rough terrain. The flat terrain represents a controlled locomotion environment in which the robot is on a planar surface. This environment is used because it provides a simple setting for validating the action interface and training basic locomotion behaviors, since the policy can focus on gait generation, velocity and stability.

The second configuration is a generated rough terrain environment. The terrain is generated using the Isaac Lab terrain generator and is used to expose the robot to more variable contact conditions. The rough terrain is restricted to two types of terrain primitives: random boxes and random rough terrain. These terrain types introduce irregular foot contact with the ground, and require the policy to adapt its motion. More complex terrains, such as stairs, are not used because they would impose excessive loads on the real actuators during physical deployment.

The flat environment provides a clean reference condition for locomotion, while the rough environment increases the diversity of physical interaction experienced by the robot. This combination allows the policy to be trained and evaluated under different conditions.

Fig. 5.2 shows the rough terrain configurations used to expose the policy to irregular foot-ground interactions.



(a) Random rough terrain (b) Random boxes terrain

Figure 5.2: Examples of rough terrain configurations used in simulation. The first terrain introduces irregular height variations, while the second terrain contains small blocks

5.3.2 Domain Randomization

Domain randomization is applied at reset time to vary physical properties of the robot and the environment. The objective is to avoid training the policy on a single idealized simulation setup and to expose it to a range of possible dynamic conditions.

In the implemented environment, terrain friction is randomized by sampling the static and dynamic friction coefficients from predefined ranges. In addition to terrain friction several robot parameters are randomized. The robot mass is scaled within a limited range, the center of mass of the base is shifted and actuator parameters, such as damping and

stiffness, are varied, in order to get a better approximation of the real motors. Joint parameters are also randomized to introduce variability in the simulated actuation.

The initial state of the robot is randomized during the reset. The base position and yaw orientation, together with initial linear and angular velocities are sampled within predefined ranges. This prevents the policy from relying on a single fixed starting condition and improves robustness to small variations in the initial state of each episode.

The randomization ranges used during training are summarized in Table 5.3.

Table 5.3: Domain randomization parameters used in the training environment.

Category	**Parameter**	**Range**
Terrain contact	Static friction	$0.6 - 1.0$
	Dynamic friction	$0.4 - 0.9$
Robot dynamics	Mass scaling	$0.95 - 1.05$
	Base CoM shift x	$-0.05 - 0.05$ m
	Base CoM shift y	$-0.05 - 0.05$ m
	Base CoM shift z	$-0.01 - 0.01$ m
Actuation	Stiffness scaling	$0.9 - 1.1$
	Damping scaling	$0.9 - 1.1$
Joint dynamics	Joint friction scaling	$0.9 - 1.1$
	Armature scaling	$0.9 - 1.1$
Initial state	Base position x	$-0.5 - 0.5$ m
	Base position y	$-0.5 - 0.5$ m
	Base yaw	$-3.14 - 3.14$ rad
	Linear velocity x, y, z	$-0.5 - 0.5$ m/s
	Angular velocity roll, pitch, yaw	$-0.5 - 0.5$ rad/s

5.3.3 Curriculum Strategy

The training environment includes curriculum mechanisms to adjust the difficulty experienced by the policy during learning. The goal is to avoid exposing the agent immediately to the most difficult conditions, in order to first develop a stable locomotion under simpler settings.

A terrain level curriculum is used for rough environments. The terrain generator assigns each environment to a terrain level, which is updated according to the robot performance. The terrain difficulty is adapted according to the locomotion capability of the policy. If the robot is able to follow the commanded velocity, the terrain difficulty is increased in later episodes, otherwise the environment remains at a lower terrain level. The update is done when the ratio between base velocity and commanded velocity is above or below a predefined threshold for a sufficient number of updates.

The implementation also supports curriculum mechanisms for selected domain randomization parameters. These mechanisms modify the randomization ranges according to the policy performance. When the ratio between the actual base velocity and the commanded

velocity remains above predefined threshold for a sufficient number of updates, the corresponding randomization range is widened. This strategy is applied to robot mass, actuator stiffness, actuator damping and terrain friction, not only changing geometric difficulty of the terrain but also increasing the variability of the dynamics. In the flat environment the terrain level is disabled because the ground is a plane and the other curriculum can be disabled through a configuration file.

5.4 Training Algorithm and Policy Structure

The policy is trained with Proximal Policy Optimization (PPO) in Isaac Lab using parallel environments. During training, experience is collected from different terrain conditions and morphological configurations within the same learning process. This section explains the learning algorithm and policy architecture used.

5.4.1 PPO Training Setup

The locomotion policy is trained using Proximal Policy Optimization (PPO) through RSL-RL integrated in Isaac Lab. PPO is an on-policy actor-critic algorithm commonly used for continuous control locomotion tasks, especially when many parallel environments can be used to collect experience efficiently.

The use of parallel environments is very important as each environment can be initialized with a different terrain condition and with a different leg configuration. This makes it possible to consider two possible training strategies.[26] The first strategy consists of learning all admissible configurations in parallel. The problem is treated as a multi task learning in which the same policy is optimized across different morphologies at the same time. The second strategy consist in first learning the six legged locomotion and then introducing reduced leg configurations through curriculum.

Both approaches were considered during development. However the final implementation treats the problem as a multi task learning problem. This choice was made because locomotion with six legs is not just an easier version of locomotion with five or four legs. Different morphologies require different mappings between the policy output and the generated motion. Training all configurations within the same distribution encourages the policy to learn a behavior that remains effective across different morphologies. The detailed mapping between policy output and joint motion is described in the following sections.
Fig. 5.3 summarizes the actor-critic training loop used by PPO.

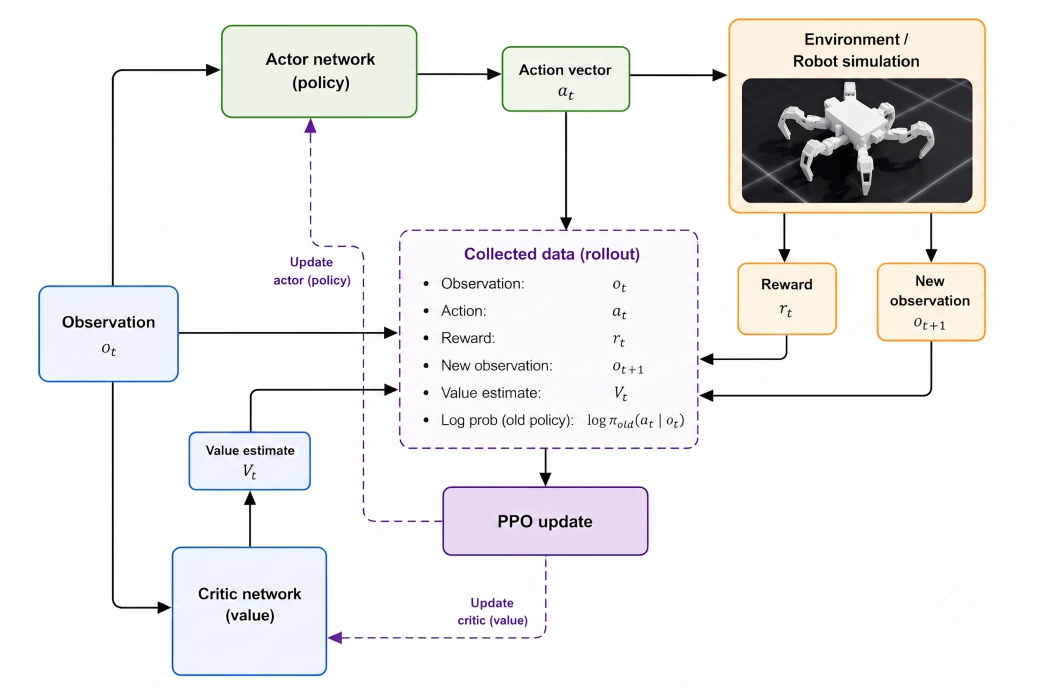


Figure 5.3: Actor-critic training loop used by PPO.

5.4.2 Policy Network Architecture

The locomotion controller is implemented as a custom actor-critic network. The network follows the classic actor-critic structure required by PPO, with an actor used to generate actions and a critic used to estimate the value function.

The actor input is not fixed but is computed by concatenating all observation terms of the policy group, while the critic input is obtained in the same way from the critic observation group. This makes the architecture flexible with respect to changes in the observation space.

The actor is composed of a shared multilayer perceptron trunk followed by one or two output heads. The trunk extracts a compact feature representation from the observation vector. The base output head produces 15 action parameters, which correspond to the standard CPG based action parametrization. When a larger action space is required an additional output head is used to generate the remaining parameters.

This design makes the policy architecture extensible. The base locomotion output can be preserved while introducing additional action parameters to achieve more complex locomotion. This is useful when more mobility is required, since the network can be extended without changing the basic structure of the actor. The structure of the actor network used to generate the policy action is shown in Fig. 5.4.

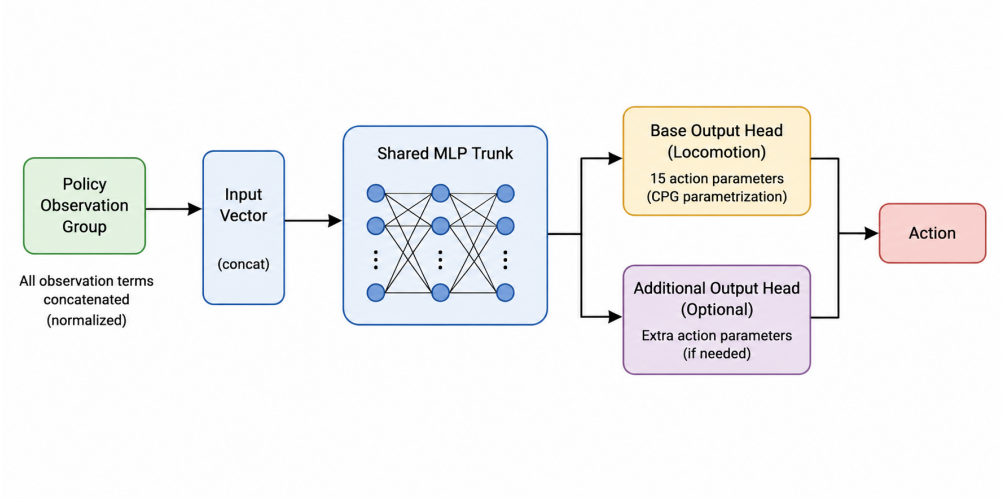


Figure 5.4: Actor network architecture used to generate the policy action.

During the training, is used a diagonal Gaussian action distribution. The network output represents the mean of the distribution, while the standard deviation is represented by trainable parameters. Actions are sampled from this distribution during training in order to encourage exploration. During inference the deterministic mean is used.

The critic is implemented as a separate multilayer perceptron that maps the critic observation to a scalar value. Both observations are normalized, which helps to stabilize the learning.

5.4.3 Training Parameters

The PPO training setup is defined by the described architecture and by the optimization hyperparameters used during the policy update. The policy and PPO parameters are summarized in the following Table 5.4.

Table 5.4: Policy architecture and PPO hyperparameters used during training.

Category	Parameter	Value
Policy	Architecture	Actor-critic
	Actor hidden dimensions	[512, 256, 128]
	Critic hidden dimensions	[512, 256, 128]
	Activation function	ReLU
	Initial action noise std.	0.25
	Actor observation normalization	Enabled
	Critic observation normalization	Enabled
PPO	Value loss coefficient	0.6
	Clipping parameter	0.2
	Entropy coefficient	0.003
	Learning epochs	5
	Mini-batches	16
	Learning rate	3.0×10^{-4}
	Learning rate schedule	Adaptive
	Discount factor γ	0.99
	GAE parameter λ	0.95
	Desired KL divergence	0.01
	Maximum gradient norm	1.0

5.5 Markov Decision Process

The locomotion task is formulated as a Markov Decision Process (MDP). At each control step the policy receives the observation of the current state of the robot and the output is an action that is applied to the robot. The environment, then evolves according to the robot dynamics and returns a reward used to update the policy during training.

The policy receives an observation composed of different types of information extracted from the simulation and designed to be reproducible on the physical robot. The action produced by the policy is a high level control action that is then converted into joint targets by a CPG action interface.

The following subsections describe the command space, observation space, and action space used. The reward function is not described at this stage, since it is more useful to discuss it after defining how the policy output is converted into joint motion.

5.5.1 Observation Space

The most important characteristic of the observation space is that it can be reproduced in a real world environment. The policy does not receive the complete simulator state, but an observation vector composed of quantities required for locomotion control and obtainable on the physical robot. The information included in the observation vector consists of: proprioceptive data, current command, previous action and morphology encoding.

The proprioceptive terms include linear and angular velocity of the robot base, the projected gravity and the joint positions and velocities. These terms describe the physical state of the robot within the environment.

The velocity command is in the observation and is defined as:

$$\mathbf{c}_t = \begin{bmatrix} v_x^{cmd} & v_y^{cmd} & \omega_z^{cmd} \end{bmatrix},$$

where v_x^{cmd} and v_y^{cmd} are the desired linear velocities and $\omega_z^{\text{cmd}}$is the desired yaw rate. This command allows the policy to generate the different locomotion behaviors.

The observation also contains the previous action, which gives information about the most recent output. Finally the morphology is represented through a binary vector which indicates the active legs. This information allows the policy to adapt its behavior according to the morphology of the robot.

5.5.2 Action Space

The action is not directly used as a joint position target, but it represents high level parameters that modulate the CPG locomotion interface.

Two action parametrizations are supported. In the standard version of the controller, the policy output is a 15-dimensional vector. In the extended version, the action vector is increased to 39 dimensions, allowing a richer modulation of the locomotion pattern.

The detailed interpretation of these parameters depends on the CPG mapping and on the robot morphology.

5.6 CPG-Based Action Interface

As described in the previous section the action is not directly applied to the robot joints but it is used by a Central Pattern Generator (CPG) based action interface. The goal of this unit is to convert the policy output into a rhythmic joint position targets. This design is inspired by CPG-RL approaches, where reinforcement learning is used together with rhythmic generator to improve adaptability and reduce the risk of unstable gaits. [28]

The CPG interface is adapted to the reconfigurable setting. The policy modulates the parameters of the oscillator network and of the joint mapping, while the final joint targets are generated through the rhythmic controller. This influences the gait without having an arbitrary control of the joints at every control step. Moreover, the interface can be modified according to the configuration of the active legs.
Fig. 5.5 summarizes the pipeline from the policy observation to the final joint mapping.

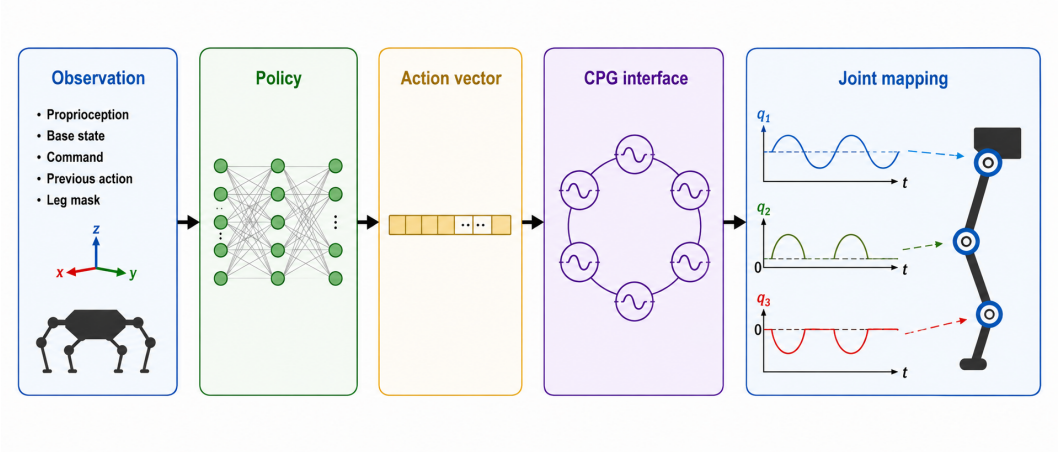


Figure 5.5: Overview of the CPG-based action interface used during training

5.6.1 Oscillator Formulation

The CPG network used is based on Hopf oscillators. A Hopf oscillators are commonly used for locomotion control because it naturally generates a stable periodic signal and animal movements are often characterized by approximately rhythmic pattern during walking. The Hopf oscillator is a compact way to reproduce this natural behavior as, after a transient phase, the oscillator converges to a limit cycle, meaning that its output continues to oscillate with a regular amplitude and frequency. This provides a simple way to obtain a rhythmic signal without requiring the policy to produce a complete joint trajectory at every control step.

More specifically, the oscillator produces two periodic variables: x_i and y_i. The variable y_i is used as the main signal for the motion of the leg and $\dot{y}_i$ is used to identify the phase of the gait that can be the stance phase or the swing phase. Therefore the oscillator does not directly command the motors but is a temporal signal from which the joint targets are generated.[27]

The CPG interface supports two modes: a standard mode and an extended mode. The first one uses two coupled Hopf oscillators, producing two signals in opposite phase. The extended mode increases the number or oscillators to provide a more expressive gait representation.

The dynamics of each oscillator in the standard mode can be written as:

$$\begin{pmatrix} \dot{x}_i \\ \dot{y}_i \end{pmatrix} = \begin{pmatrix} \alpha(\mu - r_i^2) & -\omega_i \\ \omega_i & \alpha(\mu - r_i^2) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \delta \begin{pmatrix} 0 \\ \Delta_{ji} \end{pmatrix}$$

with

$$r_i^2 = x_i^2 + y_i^2$$

and

$$\Delta_{ji} = y_j \cos(\theta_{ji}) - x_j \sin(\theta_{ji})$$

To make the implementation clearer, the same dynamics can be equivalently written as

$$\dot{x}_i = \alpha(\mu - r_i^2)x_i - \omega_i y_i$$

$$\dot{y}_i = \omega_i x_i + \alpha(\mu - r_i^2)y_i + \delta\Delta_{ji}$$

Here, α controls how quickly the oscillator converges to the limit cycle, μ is the squared radius of the oscillation, ω_i is the angular frequency, δ is the coupling strength, and θ_{ji} is the desired offset between the two oscillators. This last term is set to π in order to have two signals in opposition of phase.

The frequency of the oscillator is controlled by the policy. The corresponding action component is first clipped to the interval $[-1,1]$ and then mapped to the physical frequency range $[0.5, 2.0]$ Hz. The result is then converted into angular frequency through

$$\omega_i = 2\pi f_i.$$

In the standard mode both oscillators use the same controlled frequency, in this way the policy can decide the overall rhythm of the gait but the two oscillators remain coupled. [28]

In the extended mode the oscillator network is expanded to six Hopf oscillators, one for each leg. The oscillators are not coupled through the term $\delta\Delta_{ji}$ and each oscillator follows the non-coupled Hopf dynamics:

$$\dot{x}_i = \alpha(\mu - r_i^2)x_i - \omega_i y_i$$

$$\dot{y}_i = \omega_i x_i + \alpha(\mu - r_i^2)y_i$$

In this case, each leg has its own oscillator. The policy still controls a common base frequency but it also introduces a frequency offset and a leg offset for each leg. Therefore the frequency of each oscillator and the corresponding angular frequency are:

$$f_j = f_{\text{base}} + \Delta f_j$$

$$\omega_j = 2\pi f_j$$

The phase of each oscillator is first computed from its internal state. The controlled phase offset is then added to this phase and the result is used to compute the final signal. This procedure can be written as:

$$\phi_j = \text{atan2}(y_j, x_j), \quad \phi_j^{\text{eff}} = \phi_j + \phi_j^{\text{off}},$$

$$y_j^{\text{eff}} = \cos(\phi_j^{\text{eff}}), \qquad \dot{y}_j^{\text{eff}} = -\omega_j \sin(\phi_j^{\text{eff}}).$$

This extended version provides a richer representation, since the legs are not coupled and the controller decides how to modulate each signal. This makes this mode more expressive, but it increases the dimensionality of the action space and the complexity of the learning problem.

5.6.2 Policy Action Decoding

The action vector produced by the policy is decoded into the parameters used by the CPG interface. In the standard mode the action vector has 15 parameters [28], while in the extended version the parameters are 39.

In the standard mode, the first two components define the hip modulation vector A_1, shared across the two tripods of the robot. The next six parameters are the knee amplitudes A_2, one for each leg, while the next six components define the ankle amplitudes A_3. The last component controls the oscillator frequency. The final action vector is:

$$\mathbf{a}_t = \begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} & \mathbf{A_3} & \omega \end{bmatrix}$$

where

$$\mathbf{A_1} \in \mathbb{R}^2, \qquad \mathbf{A_2} \in \mathbb{R}^6, \qquad \mathbf{A_3} \in \mathbb{R}^6, \qquad \omega \in \mathbb{R}.$$

In the extended mode the additional parameters include leg frequency offsets, phase offsets, stance-threshold and hip offset terms. These parameters allow the policy to have modulate each leg more independently. The extended action vector is:

$$\mathbf{a}_t = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{A}_3 & f & \Delta\mathbf{f} & \boldsymbol{\phi}^{\text{off}} & \mathbf{s} & \Delta\mathbf{A}_1 \end{bmatrix}$$

where

$$\mathbf{A}_1 \in \mathbb{R}^2, \qquad \mathbf{A}_2 \in \mathbb{R}^6, \qquad \mathbf{A}_3 \in \mathbb{R}^6, \qquad f \in \mathbb{R}$$

$$\Delta\mathbf{f} \in \mathbb{R}^6, \qquad \boldsymbol{\phi}^{\text{off}} \in \mathbb{R}^6, \qquad \mathbf{s} \in \mathbb{R}^6, \qquad \Delta\mathbf{A}_1 \in \mathbb{R}^6.$$

After decoding the action components are mapped to physical parameter ranges and passed to the CPG oscillator and then mapped to joint positions.

5.6.3 Joint Target Generation

As described in the previous sections, the policy output does not directly command the joint position. Instead the action vector is used to modulate the CPG interface, which converts the oscillator signals into joint angle offsets for each leg. More precisely it is produced a an offset for each joint that is added to the default standing posture of the robot. The generated gait is a rhythmic modulation around a nominal value.

For each leg the mapping uses the oscillator output y_i and its derivative $\dot{y}_i$. The offset of the first joint is generated directly from the oscillator signal:

$$\phi_{1,j} = A_{1,j} y_j$$

In the standard configuration, the hip joints are controlled by two A_1 parameters. These values correspond to the two alternating oscillator groups generated by the CPG. A_{11} modulates the hip of the first group, while A_{12} modulates the other group of hips. The groups change according to the current morphology of the robot. This design is used to maintain coordination between the two groups, and the policy is encouraged to generate a more regular gait.

In the extended configuration this constraint is relaxed by introducing an additional A_1 offset for each leg. These offsets allow the policy to modify the hip modulation of each leg independently, increasing the expressiveness of the controller.

The second joint angle is associated with the current phase of the robot. The two phases are: swing and stance. A continuous swing mask is computed from $\dot{y}_j$ using a sigmoid function, in order to have a smooth transition between stance and swing instead of using a binary value. This mask can be written as

$$m_{\text{swing},j} = \sigma\left(\frac{\dot{y}_j - \tau_j}{w_j}\right)$$

where τ_j is the stance threshold and w_j controls the width of the transition. In the standard mode, the transaction is centered around $\dot{y}_j = 0$, while in the extended mode the threshold of each leg is decided by the policy. This additional modulation of the extended version allows the extended version to provide each leg more independence over the transition between swing and stance phases. The knee angle is then computed as

$$\phi_{2,j} = m_{\text{swing},j} A_{2,j}\left(1 - |y_j|^2\right)$$

The third joint angle is coupled to the knee motion:

$$\phi_{3,j} = -A_{3,j}\phi_{2,j}$$

Therefore A_{1j},A_{2j} and A_{3j} control the amplitude of the hip, knee and ankle. The resulting offsets are then added to the default joint position.

$$q^{\text{target}} = q^{\text{default}} + \phi$$

During the early experiments, the movements were too abrupt while applying directly these target positions to the legs, especially when the policy changed the parameters

rapidly between two consecutive steps. For this reason it was introduced a moving average filter in order to generate smoother angle offsets. The filtered target is computed as

$$\bar{\phi}_t = \frac{1}{N} \sum_{k=0}^{N-1} \phi_{t-k}$$

where N is the window size and the value used is $N = 5$. This step reduces the sudden changes and produces a more continuous leg motion, while preserving the rhythmic structure of the CPG.

5.6.4 Morphology Dependent Mapping

As explained in the previous sections, in the standard mode the CPG produces two rhythmic signals. These signals must be assigned to the legs before the joint mapping can be applied. The full leg assignment is built in order to create a tripod gait. The first oscillator is associated with legs L_1, L_3 and L_5, the second oscillator is associated with the legs L_2, L_4 and L_6. In this way the oscillator outputs define the two coordinated groups.

When the morphology of the robot changes, the described assignment is not always suitable. Keeping the complete robot assignment for reduced leg configurations can lead to unbalanced motion with some configuration. For this reason a morphology dependent lookup table is defined based on the current `leg_alive_mask`.

The `leg_alive_mask` is encoded as a binary key that represents the current leg configuration. The key is used to access a predefined lookup table, which returns a six-dimensional vector. Each element of this vectors corresponds to one leg and can have three possible values: 0, 1 or -1. A value of 0 means that the leg is associated with the first oscillator, a value of 1 means that it follows the second oscillator, and a value of -1 indicates an inactive leg.

For example, the lookup table assignment of the complete morphology is:

$$[0,\ 1,\ 0,\ 1,\ 0,\ 1]$$

When only one leg is missing the same alternating structure is preserved. One oscillator controls two active legs, while the other oscillator controls the remaining three legs of the opposite group.

For the four legged morphologies, the lookup table is designed to preserve a diagonal coordination, which fits well with the two oscillator structure. The idea is to pair diagonally opposite legs with the same oscillator, in order to have the front leg of one side coordinated with the rear leg of the opposite side, and vice versa. Considering legs L_1, L_2 and L_3 on the right side, with L_1 as the front leg and legs L_4, L_5 and L_6 on the left side, with L_6 being the front leg, the assignment is constructed by pairing the most forward available leg on one side with the most rearward available leg on the opposite side. The remaining active legs are assigned to the other oscillator group. This provides a simple rule for maintaining a balanced rhythmic structure when only four legs are active.

Although this rule could be computed at runtime, it was implemented as a lookup table because during training, the computation of the oscillator assignment at every reset introduced too much overhead. Using a predefined lookup table makes the assignment immediate.

In the extended mode the lookup table is not required as each leg is already associated with its own oscillator. The morphology dependent assignment is only used in the standard mode, where two signals must be distributed across the active legs.

The complete oscillator assignment lookup table is reported in Table 5.5.

Table 5.5: Morphology dependent oscillator assignment lookup table

Configuration	**`leg_alive_mask`**	**Oscillator assignment**
1	1 1 1 1 1 1	0 1 0 1 0 1
2	0 1 1 1 1 1	−1 1 0 1 0 1
3	1 0 1 1 1 1	0 −1 0 1 0 1
4	1 1 0 1 1 1	0 1 −1 1 0 1
5	1 1 1 0 1 1	0 1 0 −1 0 1
6	1 1 1 1 0 1	0 1 0 1 −1 1
7	1 1 1 1 1 0	0 1 0 1 0 −1
8	0 1 1 0 1 1	−1 0 1 −1 0 1
9	0 1 1 1 0 1	−1 0 1 0 −1 1
10	0 1 1 1 1 0	−1 0 1 0 1 −1
11	1 0 1 0 1 1	0 −1 1 −1 0 1
12	1 0 1 1 0 1	0 −1 1 0 −1 1
13	1 0 1 1 1 0	0 −1 1 0 1 −1
14	1 1 0 0 1 1	0 1 −1 −1 0 1
15	1 1 0 1 0 1	0 1 −1 0 −1 1
16	1 1 0 1 1 0	0 1 −1 0 1 −1

5.7 Reward Design

The reward function is one of the most important elements as it guides the policy during training. In this work the reward is mainly designed to incentivize velocity tracking, stable motion, smooth policy actions and an appropriate foot-ground interaction.

Several reward components were tested during development and some of them were disabled or replaced by more effective variants. The following paragraphs focus on the rewards that were most useful in order to obtain stable locomotion.

5.7.1 Main Reward Terms

The most important terms can be divided into five groups: velocity tracking rewards, stability penalties, motion regularization penalties, contact related rewards and gait terms. These groups represent the objectives that must be balanced during training.

The first group contains the tracking rewards, which define the main task. The policy is trained to follow a desired planar linear velocity and a desired yaw angular velocity. The linear velocity tracking term compares the desired velocity with the actual base velocity of the robot. This allows the policy to learn forward, lateral and diagonal motion commands. The yaw tracking term applies the same idea to the angular velocity around the vertical axis, encouraging turning motions.

The second group includes stability penalties. These terms penalize behaviors that produce unstable body motion. In particular the vertical velocity of the base is penalized to reduce bouncing, while roll and pitch angular velocities are penalized to limit body oscillations. In order to discourage excessive body tilting it is included a flat orientation penalty. These terms are important because the CPG interface helps to generate rhythmic leg motion, but does not guarantee a stable body dynamics.

The third group contains motion regularization terms. Large or rapidly changing actions can produce abrupt changes in the generated gait, usually unfeasible for real motors and physical feasibility. For this reason penalties on the action magnitude and action rate are used to encourage smoother changes in the policy parameters. In addition a joint regularization terms penalize large torques, excessive joint acceleration and large deviations from the nominal standing posture. These terms improve the physical deployment of the learned locomotion.

The fourth group is related to contact rewards and penalties. These terms improve the quality of the interaction between the feet and the terrain. The foot sliding penalty discourages motion of the feet while they are in contact with the ground. The swing height reward encourages the feet to do a proper step by reaching a desired height during swing phase. Undesired contacts are strongly penalized when non foot links touch the ground, preventing unrealistic support strategies.

The final group includes gait terms. During development more explicit gait rewards were implemented like tripod gait rewards, foot air time rewards, leg usage balancing and symmetry terms. These terms are useful for experimentation when training the six legged morphology but imposing a specific pattern is too restrictive when the robot is trained across different configurations. For this reason the final reward design relies mainly on the first four groups described and more explicit gait terms are implemented but kept optional.

5.7.2 Reward Terms and Weights

Each reward term is multiplied by a scalar weight before being added to the total reward. Desired behaviors have positive weights, while discouraged behaviors have negative weights. Terms with zero weight are implemented and tested but disabled in most training

runs. The values reported in the Table 5.6 are the baseline weights used in most training setup.

Table 5.6: The reported weights are the baseline values used and were adjusted experimentally.

Reward term	**Description**	**Weight**
`track_lin_vel_xy_exp`	Rewards tracking of the commanded planar linear velocity	18.0
`track_ang_vel_z_exp`	Rewards tracking of the commanded yaw angular velocity	1.75
`standstill_penalty`	Penalizes the robot when it remains stationary despite a non zero velocity command.	0.0
`lin_vel_z_l2`	Penalizes vertical velocity of the base.	-1.0
`ang_vel_xy_l2`	Penalizes roll and pitch angular velocity	-0.1
`flat_orientation_l2`	Penalizes deviations from a flat body orientation	-0.2
`base_height`	Penalizes deviations from a target base height	0.0
`feet_not_under_body_during_swing`	Penalizes swing feet that move too far toward the center of the robot body	-0.2
`dof_torques_l2`	Penalizes large joint torques	-0.001
`dof_acc_l2`	Penalizes large joint accelerations	-1e-7
`action_rate_l2`	Penalizes rapid changes in the policy action	-0.005
`action_l2`	Penalizes large action magnitudes	-0.005
`joint_excursion`	Penalizes excessive joint displacement from the CPG default standing posture	-0.05
`dof_pos_limits`	Penalizes joint positions close to their limits	-1e-4
`full_leg_symmetry`	Encourages symmetric leg usage in the complete morphology	0.0
`feet_usage_balance`	Encourages balanced use of the feet during locomotion	0.0
`A1_l2`	Penalizes large hip-amplitude CPG parameters	0.0
`tripod_gait`	Encourages an alternating tripod-like contact pattern	0.0
`feet_air_time`	Rewards feet that spend sufficient time in the air	0.0
`feet_slide`	Penalizes foot motion while the foot is in contact with the terrain	-0.1
`undesired_contacts`	Penalizes contacts involving non-foot leg links	-10.0

Reward term	Description	Weight
`feet_swing_height_reward`	Rewards swing feet that reach a desired height	0.2

5.8 Termination Conditions

Termination conditions are used to stop an episode when the robot reaches states that are no longer useful for the training. In particular the episodes terminate when the robot leaves the valid terrain area, when the base height falls below a threshold or when the body tilt exceeds a maximum allowed value or when there is an undesired contact indicating a failed locomotion attempt.

In the reported configuration, the maximum episode duration is set to 20s. The minimum base height is 0.08m, while the base angle limit is 0.9rad. Contact termination was not enabled in this flat terrain configuration, because undesired contacts were used in the reward function. However, this condition is useful in rough terrain experiments, where it is harder to define a base height in a rough terrain as different points of the base can have different height values.

Table 5.7: Termination conditions used during training.

Termination	Condition	Value
Timeout	Maximum episode duration	20 s
Out of bounds	Robot outside valid terrain area	Enabled
Base too low	Base height below threshold	0.08 m
Bad orientation	Base tilt above limit	0.9 rad
Contact-based termination	Undesired body contact	Optional

Chapter 6

Policy Deployment and Sim-to-Real Integration

After defining the training environment and the used platform, the next step is to analyze how they are connected and transfer the policy to the physical platform. This step is not just limited to exporting the neural network, since several parts of the control are handled internally by Isaac Lab. During the deployment, the complete application pipeline must be reproduced explicitly on the real robot.[30]

This chapter describes how the trained policy is integrated into the control architecture of the hexapod and how the main sim-to-real issues are addressed, giving particular attention to the differences between simulated control loop and physical system.

The deployment process of the system was iteratively. Problems that were not critical in simulation became relevant on the real robot. Some of the issues were addressed by modifying the real world pipeline, while others required changes in the simulation setup, reward design or training configuration.

This chapter focuses on the practical integration of the policy and on the adaptations required by the sim-to-real gap.

6.1 Policy Application

In simulation the policy is part of the control loop. Once the policy is deployed on the robot, this execution must be implemented as an independent component.[31] The deployment code must reproduce only the parts required for inference and action application.

The controller starts from the checkpoint produced during training and reconstructs the actor network used to generate actions. The output is then passed to the same CPG interface used in the simulation.

The most important requirement is the consistency between training pipeline and deployment pipeline. Even small inconsistencies can lead to different behaviors between

simulation and reality.

6.1.1 Checkpoint Loading and Policy Inference

The first step of the deployment is loading the policy checkpoint produced during training. During deployment only the components required for inference are used. In particular, the actor network is reconstructed from the checkpoint, while the critic and training related components, such as rewards, are not used in the real robot.

The observation normalization statistics used during training are also restored, in order to use the same input distribution seen by the network in the simulation. Therefore each observation is first reconstructed, then normalized using the saved mean and standard deviation, and finally passed to the actor.

During inference the policy uses the deterministic output of the actor and the resulting action is a high level vector that is processed by the CPG interface to generate joint position targets.

6.1.2 CPG Application and Joint Remapping

After the policy produces the action vector, the next step is to replicate the same action interface used during training. As in the simulation the action is not directly sent to the motors but is first decoded by the CPG interface, which updates the oscillator parameters and generates the joint position targets.

This operation is no longer handled internally by Isaac Lab. On the real robot the logic has to produce commands compatible with the ROS2 motor interface. A critical point is the ordering of the joint targets. The ROS2 motor interface uses a leg-major order, where the three joints of each leg are grouped together but the CPG implementation uses a degree of freedom major order, where the joints of the same type are grouped together. Therefore a remapping is required before publishing the command.

This remapping is essential as it ensures that each target joint position is applied to the correct physical joint. Without this conversion the generated command would be assigned to the wrong motors, producing unstable movements.

6.2 Real-Robot Observation Reconstruction

The deployed policy requires an observation vector with the same structure used during training. In simulation, these quantities are directly available from Isaac Lab, while on the physical robot they must be reconstructed from different sources.

The robot observation is built by combining motor feedback, inertial measurements, command information and internal controller states. Joint positions and velocities are read from the DYNAMIXEL actuators, while angular velocity, linear velocity and projected gravity are provided by the IMU state estimation pipeline. The velocity command and previous action are inside the deployment node, and the active leg configuration is included when reduced morphologies are used.

6.2.1 Observation Noise and Velocity Estimation Issues

A critical aspect of the reconstruction is that the measurements must remain within the distribution observed during training. The policy was trained using normalized observations, where state quantities are affected only by noise explicitly introduced in the environment. On the real robot sensor measurements can contain noise, bias and delays not always reproducible in the simulator. [34]

The issue was particularly relevant for quantities from the IMU. Angular velocity and projected gravity are affected by sensor noise and IMU bias. To improve robustness against this mismatch, noise was increased in the simulated observations during training.

The most problematic quantity was the base linear velocity. This value is estimated from the inertial measurements, removing bias and gravity from the accelerometer signal and then integrating the acceleration over time. Even with calibration and filtering the estimation drifts after a few seconds, producing meaningless values for the policy. Fig. 6.1 shows an example of the noisy and drifty base linear velocity observation.

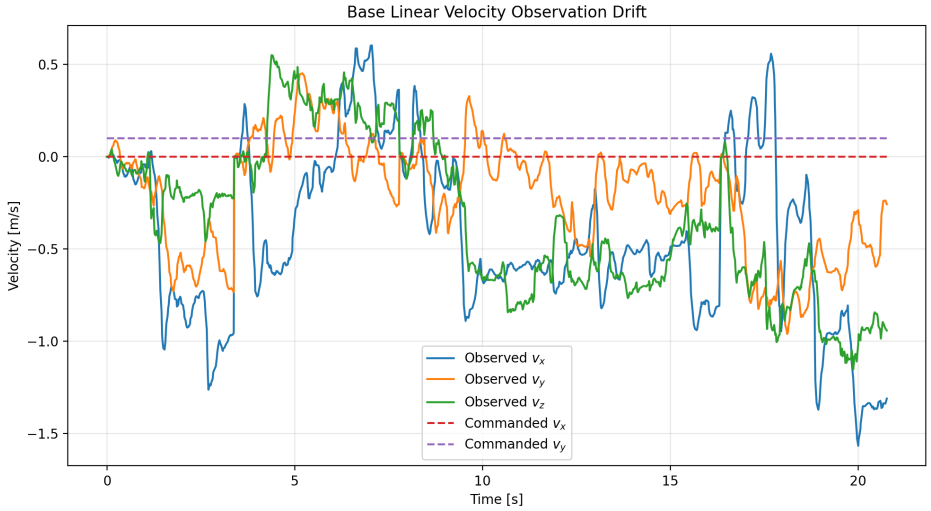


Figure 6.1: The divergence between the estimated velocity and the commanded values highlights the drift affecting the IMU-based velocity estimation.

This was a significant deployment issue: when the policy receives a value far outside the experienced range during training, its output became unstable. For this reason the velocity estimation from the IMU was not used extensively during final experiments.

6.3 Real-Time Execution and Actuator Interface

In simulation the computed action is applied with a regular timing. The simulator advances with a fixed time step and the desired joint positions are applied at the expected

control frequency. On the real robot the execution can have delays depending on ROS2 timers, sensor updates and communication with the motors.

It is important to maintain the same control frequency on both simulation and on the real robot. If in the real robot the policy is applied more slowly, the evolution of the CPG and the feedback received by the policy no longer correspond to the conditions experienced during training. This leads to unstable or unexpected behaviors, even if the same policy is correct in the simulation.

One of the main limitation is introduced by the communication with the DYNAMIXEL motors. Since the bus is half-duplex, it is not possible to read the motor state and send new commands at the same time. In the first deployment version the policy execution was linked to the motor feedback. The idea was to read the motor state at $50Hz$ and then apply the policy, in order to have consistent joint states. However the time required to read the DYNAMIXEL states was not constant, and this issue introduced irregularities in the application frequency. This issue also explained why some trajectories generated from the simulation and executed through the CSV playback were smoother than online policy execution, as the playback did not depend on the motor state.

An attempt was also made to increase the application frequency to `100 Hz`, in order to reduce abrupt changes between consecutive commands. However this solution was not possible in the real platform. The DYNAMIXEL communication bus became the main bottleneck, since reading feedback and writing commands at this rate it was not possible. As a result, increasing the control frequency did not solve the vibration problem and introduced huge time irregularities.

For the reasons explained the final deployment used a dedicated policy timer at 50Hz. In this configuration, the policy is applied at a fixed frequency independently from the motor feedback. This choice made the control loop more regular, even if in some cases the policy has to use the latest available motor state. The goal is to preserve the policy application frequency instead of slowing down policy inference because of the delays of the feedback.

Another required fix was the tuning of the DYNAMIXEL P gains. With low gains the motors followed the target position too slowly, causing a delay between the commanded gait and the leg motion. Increasing the P gains improved the tracking but produced overshoot and vibrations. Therefore the final tuning was a compromise between target tracking without excessive overshoot, adjusting also the D gains of the DYNAMIXELs. This tuning was necessary because the simulated actuator cannot reproduce the response of the real motors.

6.4 Iterative Sim-to-Real Adaptations

The deployment of the policy was not an immediate transfer. Policies that achieved stable locomotion in simulation showed problematic behaviors during the execution on the real platform. The causes of these problems were sensing, actuator response, timing, contact interaction and discrepancies between simulated and real dynamics. For this reason the sim to real process was iteratively developed: the policy was first trained in the simulation,

then tested on the physical robot and in the end the observed failures or problem were used to modify either the deployment pipeline or the simulation setup.

The first adaptations were introduced in the deployment pipeline to make the execution more consistent with the training conditions. Initial tests showed an application rate lower than $50Hz$, mainly because of the delays of the DYNAMIXEL communication loop. The baud rate was increased from $1Mbps$ to $4Mbps$ and the communication delays of the USB and motor configuration were minimized, but these optimizations were not sufficient to make the feedback fully regular and stable. For this reason the final implementation used a dedicated policy timer, applying the policy at a fixed rate with the latest available motor state. Other changes addressed the quality of the movements of the robot: the DYNAMIXEL gains were tuned to improve tracking, and a moving average window was added to smooth the movements of the robot. In the end sensor calibration and filtering was introduced and tested for the IMU based state estimation, partially reducing the effect of noise.

Other issues required changes in the simulation and training formulation. The early policies generated motions that were too wide and fast for the physical robot. In these cases the robot could fall or the actuators could shut down due to excessive load. To reduce these behaviors regularization terms were used during training. The joint excursion penalty discouraged excessive deviations from the nominal posture, while action magnitude and action rate penalties encouraged smoother changes in the CPG parameters.

Real world tests also showed problems related to the interaction between the feet and the ground. In some cases, the feet moved too close to the body during swing or slid on the ground instead of doing a proper step. To address these problems, two penalty terms were introduced that penalized the feet moving under the body and foot sliding during contact. It was introduced also a reward to encourage sufficient step height during the step.

In the end the experiments with different morphologies revealed some mechanical feasibility issues that cannot be evaluated in the simulation. Removing legs changes the load distribution on the remaining actuators causing in some cases the shutdown of the motors. For this reason it was introduced a limitation of the leg configurations and a morphology dependent standing posture.

6.5 Deployment Debugging Procedure

The real robot was tested progressively, in order to simplify the debugging of the code and the identification of unstable behaviors.

The first tests were performed with the robot lifted from the ground. In this condition, the policy could be executed without additional complexity introduced by the contact with the ground. These tests were useful to verify if the policy was loaded and working correctly and if the joint mapping was consistent with the physical robot layout. They also allowed the motor gains to be tuned without risks of unstable and dangerous movement on the ground.

A second debugging tool was the CSV playback. Instead of generating the command

online from the real observation, joint targets generated in simulation, could be replayed on the real robot with the same interface used by the policy. This helped to separate the problem of command execution from the problem of policy application and it was very useful especially during very noisy observation experiments.

After these steps, the robot was tested on the ground. This introduced issues with the body weight, foot-ground interaction and actuator load. These tests were necessary to evaluate the generated gait and to reveal issues not visible during lifted tests, like foot sliding, small steps and instability during reduced leg configurations.

During the debugging process several quantities were logged to analyze the behavior of the system. The real observations, policy inputs and outputs, desired joint targets, actual motor positions, tracking errors, motor velocities, and load-related values were recorded during experiments. The real observations, policy inputs and outputs, desired joint targets, actual motor positions, tracking errors, motor velocities, and load-related values were logged during experiments. This information was used to compare the commanded motion with the physical motion of the robot in order to understand possible issues and fix the pipeline.

Overall, testing the system first in simpler conditions and then adding complexity of the real locomotion, made possible to isolate the main causes of failure and guided the modifications described in the previous sections.

Chapter 7

Experimental Evaluation

This chapter presents the results obtained during the experiment in the simulation and on the real robot. This evaluation focuses on the complete six legged morphology on uniform terrain, which represents the first configuration used to validate the entire pipeline.

The goal is not to provide a complete evaluation of all possible morphologies and terrain conditions, but to check if the trained policy can generate a stable locomotion in both simulation and real world. For this reason, the chapter first analyzes the simulation performance and reports real world test on the physical robot.

At this stage the reduced leg morphologies and rough terrain policies are still under training and refinement and they are not reported as final experimental results in this chapter.

7.1 Simulation Evaluation

The first evaluation is about simulation performance, focusing on the complete morphology on uniform terrain. During training the policy improved its ability to generate a stable locomotion. As shown in Fig. 7.1, the mean reward increased from an initial value of approximately 5 to a value close to 380. This indicates that the policy learned to satisfy the main objectives encoded in the reward function, namely velocity tracking, body stability, smooth control actions and a good foot-ground interaction.

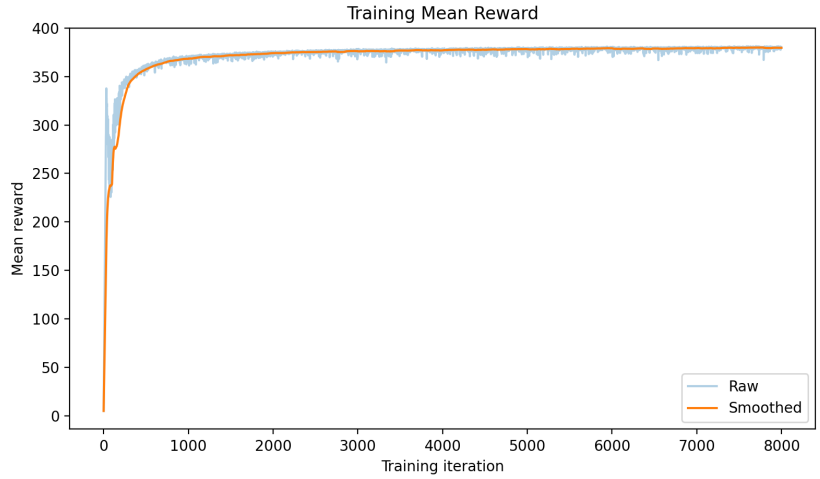


Figure 7.1: Mean reward during training for the six-legged policy on uniform terrain.

An improvement can also be observed in locomotion stability in the episode length during training. At the beginning the episodes terminated after a small number of steps because the robot was not yet able to maintain a stable gait. During the training the mean episode length increased until reaching the timeout limit of 1000 steps, as reported in Fig. 7.2. This limit was reached early during the training as the redundancy of the hexapod design helps the robot remain stable even when the generated movements are not yet optimized.

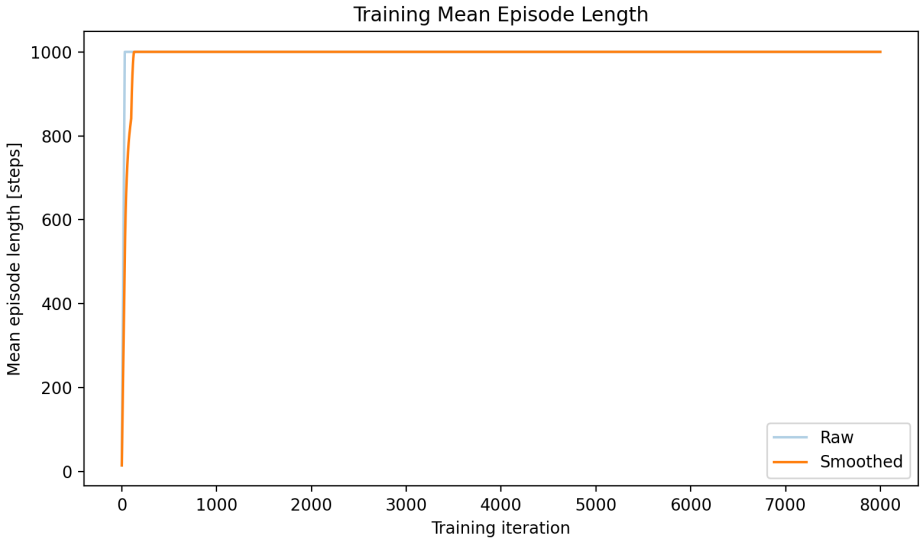


Figure 7.2: Mean episode length during training. The final policy reaches the timeout limit of 1000 steps.

The evolution of the main reward components is shown in Fig. 7.3. The velocity tracking

rewards increase during training, because velocity tracking has the highest positive weight, showing that the policy learns to follow the command. At the same time the penalty terms used for the regularization of the locomotion, remain bounded indicating that the policy does not use aggressive actions or unstable joint motions while following the command.

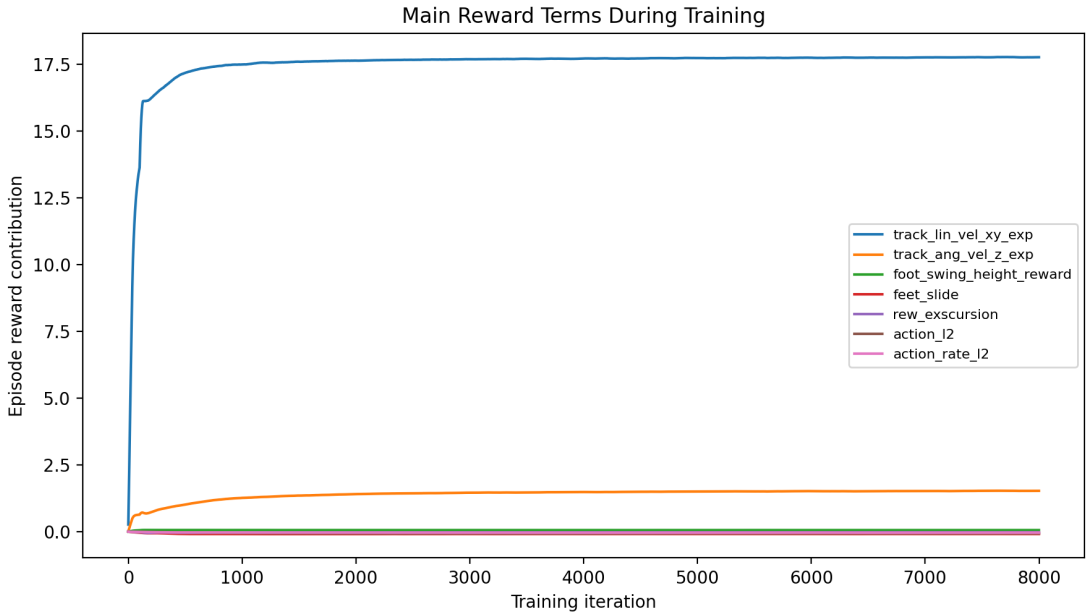


Figure 7.3: Evolution of the main reward terms during training.

The velocity tracking errors are reported in Fig. 7.4. In the final part of the training the mean planar velocity error is approximately 0.095 m/s and the yaw velocity tracking error is approximately $0.31 rad/s$. These values show that the policy can learn a stable locomotion, reflecting the weights of the rewards as the velocity tracking is considered weighted more than the yaw tracking.

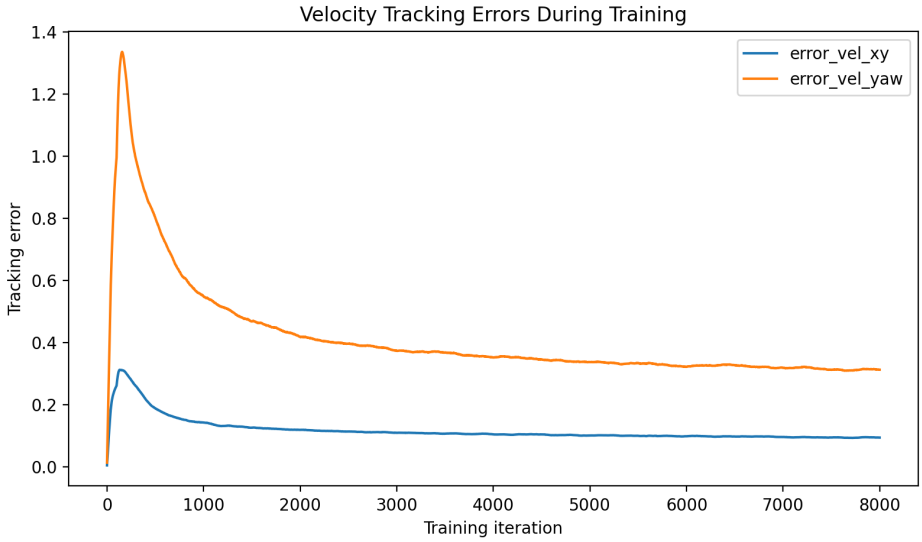


Figure 7.4: Planar and yaw velocity tracking errors during training.

The final simulation metrics are summarized in Table 7.1, indicating that the policy is able to generate stable locomotion.

Table 7.1: Summary of the final simulation performance for the six-legged policy on uniform terrain.

Metric	**Final value**
Mean reward	≈ 380
Mean episode length	1000 steps
Planar velocity tracking error	$\approx 0.095\,\mathrm{m/s}$
Yaw velocity tracking error	$\approx 0.31\,\mathrm{rad/s}$
Dominant termination condition	Timeout

These results provide the baseline for the deployment experiments described in the following sections.

7.2 Deployment Experimental Setup

After the simulation evaluation of the simulation the trained policy was deployed on the physical robot. The real world experiments were performed with the complete six legged morphology on a flat terrain. This configuration was chosen because it is the most stable morphology and because the first deployment was useful to validate the full control pipeline.

The policy was executed on the onboard LattePanda at a frequency of $50Hz$. The output of the policy was processed by the CPG interface and converted into joint position targets

as already described in the previous sections.

The tests were performed using a fixed velocity command. This choice was made to use a simple version of the command rather than using commands that change during the execution. In the reported experiments the command was set to:

$$v_x^{cmd} = 0, \qquad v_y^{cmd} = 0.1\,\mathrm{m/s}, \qquad \omega_z^{cmd} = 0\,\mathrm{rad/s}.$$

A critical issue during the deployment was the estimation of the base linear velocity. As already discussed in the previous chapter the IMU velocity estimation was noisy and experiments that used the estimated velocity produced completely unstable movements after a few seconds of execution. For this reason two fixed velocity observations were tested. In the first configuration, the base linear velocity observation was forced to zero:

$$\mathbf{v}_{base} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}.$$

while in the second configuration, the base linear velocity observation was forced to match the command:

$$\mathbf{v}_{base} = \begin{bmatrix} 0 & 0.1 & 0 \end{bmatrix}.$$

7.3 Real-World Policy Execution

The policy was evaluated by analyzing two runs performed with the same commanded velocity but with different base linear observations. The goal was to understand the behavior of the policy when the unreliable base velocity estimate is replaced by fixed observation values. During experiments also other commands were tested but they are not reported in detail.

In this scenario the commanded velocity was:

$$v_x^{cmd} = 0, \qquad v_y^{cmd} = 0.1\,\mathrm{m/s}, \qquad \omega_z^{cmd} = 0\,\mathrm{rad/s}.$$

and the two observation configurations are summarized in Table 7.2.

Table 7.2: Observation configurations used during real-world policy execution.

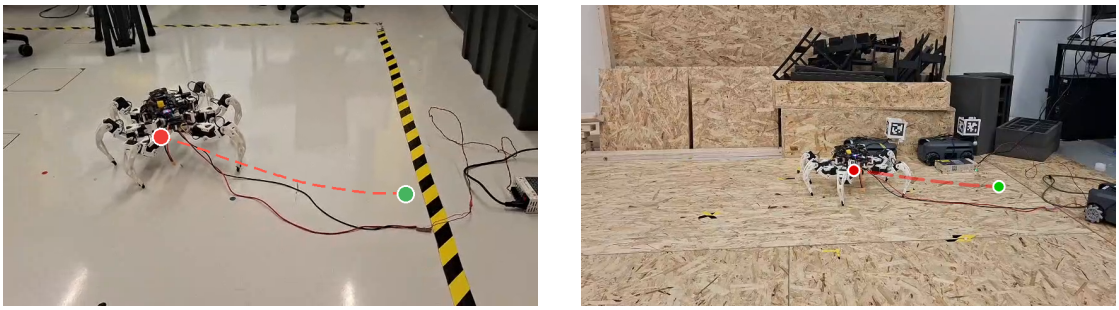
Run	**Command**	**Base velocity observation**
Zero velocity feedback	(0, 0.1, 0)	(0, 0, 0)
Commanded velocity feedback	(0, 0.1, 0)	(0, 0.1, 0)

In both configurations the fixed base linear velocity prevents the use of noisy velocity estimation while providing incomplete feedback. The policy outputs generated in the two configurations show similar behavior and successful locomotion, however the second configuration produced more variable actions. This can be observed from the mean absolute value of the raw action and from the range of the generated joint targets summarized in Table 7.3.

Table 7.3: Comparison of policy outputs for the two velocity-observation configurations.

Metric	Zero velocity feedback	Commanded velocity feedback
Mean absolute raw action	≈ 0.337	≈ 0.392
Joint target range	$[-1.716,\ 0.182]$ rad	$[-1.748,\ 0.478]$ rad
Base velocity observation	$(0,\ 0,\ 0)$	$(0,\ 0.1,\ 0)$

The higher variability in the second experiment suggests that the policy reacts to the base linear velocity, since the velocity tracking is one of the main objectives of the learned controller. However the fixed velocity observations did not produce large qualitative differences in the observed locomotion behavior, as long as the input remained within the range experienced during training.
Fig. 7.5 shows the qualitative trajectories extracted from recorded videos during the experiments. The two experiments are on different types of terrains showing the adaptability of the trained policy to different terrains.



(a) Splippery surface

(b) Wooden surface

Figure 7.5: Qualitative trajectories of the robot during the real-world experiments. The dashed lines indicate the approximate paths followed by the robot over time.

The experiments show that the trained policy can be executed on the physical robot and can generate joint targets that produce locomotion. At the same time, they highlight that a reliable base velocity estimation remains a critical requirement even though the tested velocity observations did not produce qualitative differences in the observed locomotion behavior.

7.4 Motor Tracking Analysis

In addition the experiments were analyzed at the actuator level. This is important because the actual motion of the robot depends on how well the DYNAMIXEL motors can track the targets produced by the CPG interface.

The actuator tracking behavior was evaluated comparing the desired joint positions with measured joint positions returned by the DYNAMIXEL feedback. This comparison identifies delays, tracking errors and inconsistencies between the sent commands and the real

motion.

The tracking error for each joint was computed as:

$$e_i(t) = q_i^{des}(t) - q_i^{meas}(t)$$

where $q_i^{des}(t)$ is the desired position of joint i at time t, and $q_i^{meas}(t)$ is the corresponding measured position returned by the actuator feedback.

Examples of motor tracking for the three joints of one leg are shown in Fig. 7.6, Fig. 7.7, and Fig. 7.8. The measured positions generally follow the desired trajectories, showing that the low level actuator interface is able to reproduce the generated gait. However a small delay can be observed between the generated target and the measured position. This delay is expected since the command must pass through all the described pipeline before being applied by the real motors.

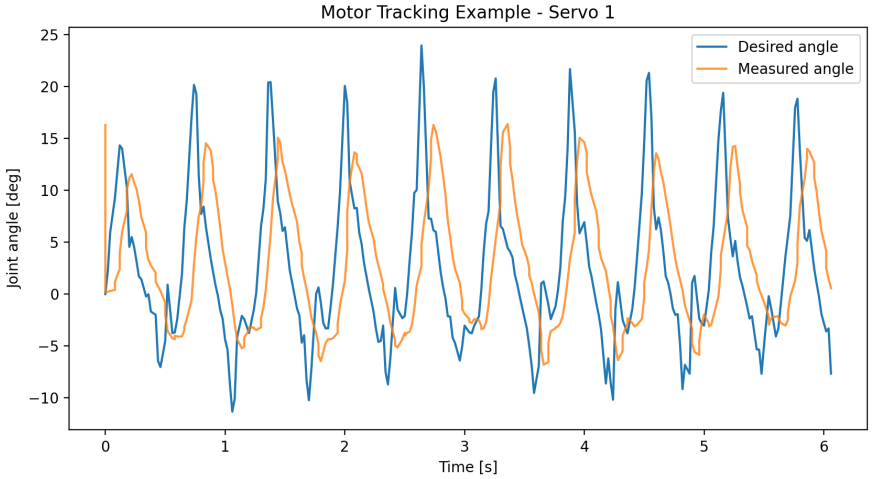


Figure 7.6: Motor tracking example for the first joint of leg 1. The plot compares the desired joint angle with the measured angle returned by the DYNAMIXEL feedback.

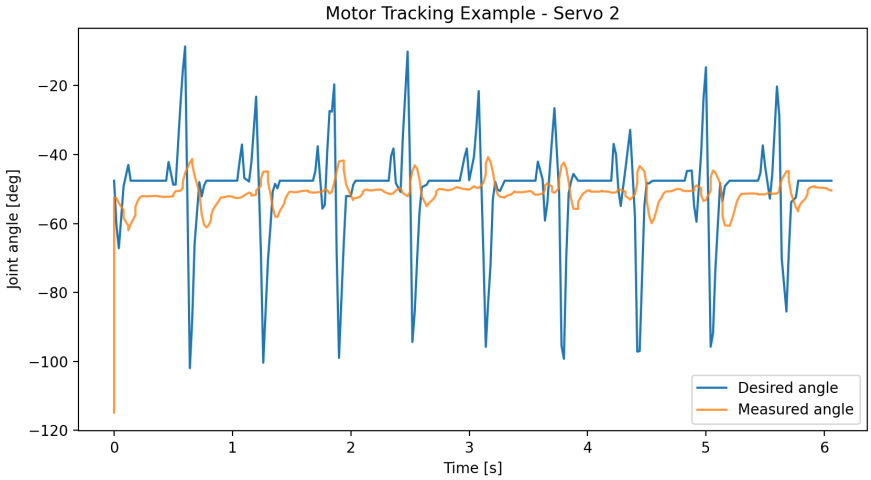


Figure 7.7: Motor tracking example for the second joint of leg 1. The plot compares the desired joint angle with the measured angle returned by the DYNAMIXEL feedback.

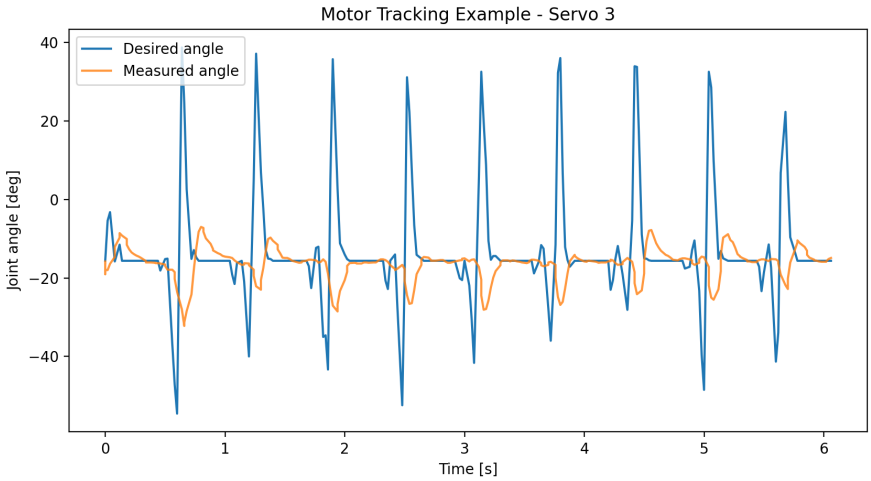


Figure 7.8: Motor tracking example for the third joint of leg 1. The plot compares the desired joint angle with the measured angle returned by the DYNAMIXEL feedback.

Another important aspect is the presence of peaks in some parts of the desired trajectories. These peaks are produced by the learned policy when the action changes rapidly between consecutive steps. Similar behaviors are commonly observed in reinforcement learning robotic control, where policies can generate jerky command signals if smoothness is not sufficiently constrained [35, 36].

In this work these peaks do not lead to equivalent peaks in the physical joint motion. The reason is that policy output passes through the CPG interface that process the raw

actions and then the processed actions are applied to the motors. Moreover the physical actuators have a finite bandwidth and cannot follow very abrupt target changes. As a result the measured joint motion is smoother than the raw desired command and the resulting locomotion remains coherent despite the presence of local errors.

Observing the tracking performance of the motors, the mean absolute tracking error is in the order of a few degrees, with values around 4.3°. Larger peaks can occur during transient phases but the average behavior shows that the actuators can reproduce an accurate locomotion.

Overall the motor tracking analysis confirms that the low level interface can follow the joint targets produced by the policy to obtain locomotion on the physical robot. At the same time it shows that a well designed low level controller is important in order not to avoid applying raw policy actions that can produce unstable behaviors. These results validate the execution of the learned controller on the physical robot for the complete six legged morphology on uniform terrain, while the main limitations are discussed in the following chapter.

Chapter 8

Conclusions and Future Work

This thesis presented the development and deployment of a learning based locomotion system for a reconfigurable hexapod robot. The work covered the complete pipeline from the design of the platform to the training of a reinforcement learning policy and the integration of the policy on the physical robot.

A central aspect was the combination of reinforcement learning with the Central Pattern Generator structure, instead of commanding the robot joints directly, the policy modulates the rhythmic parameters of the CPG that are then converted into joint targets through a structured locomotion layer.

The simulation environment was developed in Isaac Lab and included the hexapod model, the morphology representation and removal procedure, terrain generation and reward design. The policy was trained with PPO and evaluated first in simulation and then deployed on the real robot.

The deployment was performed through a ROS2-based control architecture. This architecture was built in order to reconstruct the observation vector from the real sensors, loads the trained actor and normalization statistics, applying the CPG interface and sending the commands to the DYNAMIXEL motors. The experiments showed that the policy can be executed on the physical platform and can generate joint targets that produce locomotion.

During the deployment several issues of the sim to real transfer were addressed. In particular real time execution, communication delays, sensor noise that influences directly the behavior of the robot. These issues highlights that transferring a policy to the physical robot is not only a matter of exporting a neural network but requires integration and tuning of the entire system and pipeline.

Overall the work validates the complete deployment of the learned controller and provide the foundation for future extensions. The main limitations of the current implementation and the possible future developments are discusses in the following sections.

8.1 Limitations

The presented implementation has several limitations that should be considered. The first limitation is related to the real world validation. Although the simulation environment supports different leg configurations and terrain conditions, the most reliable physical deployment was obtained with the complete morphology on uniform terrain. Reduced morphologies and rough terrain policies were implemented and tested but they were not validated enough to be reported as final experimental results.

A second limitation is related to the DYNAMIXEL XL430-W250 motors as their torque margin was limited for some locomotion conditions. This became particularly relevant when legs were removed as the load on the remaining actuators increased significantly. For this reason the used configurations had to be limited and the locomotion experiments were focused on configurations that were feasible with the available actuators.

Another important limitation is related to the state estimation. The policy observation includes the base linear velocity, which is difficult to estimate on the physical robot using only the IMU. The estimated velocity was affected by noise and produced values outside of the range experienced during training. The fixed velocity observation used during the experiments allowed the deployment pipeline to be tested but it does not represent a complete autonomous solution.

In the end the experiments were conducted under controlled conditions, tested on a flat indoor surface without external disturbances or complex terrain interaction. The reported results validate the execution of the learned controller but they do not demonstrate a robust locomotion across a wide range of real environments.

8.2 Future Work

Future developments can address both the current limitations and the objective of making the robot more adaptive in real environments.

A first direction concerns the actuation system. The experiments showed that the available DYNAMIXEL XL430-W250 [33] |motors were sufficient for the complete morphology but they are limited in more demanding configurations. A possible improvement would be the use of more powerful motors with a similar form for the most stressed joints, such as DYNAMIXEL XH430-W350-T [37]. Increasing the available torque would make the reduced morphologies more feasible and allow the use of a larger set of configurations, including configurations excluded in the current implementation because of the excessive load produced on the remaining actuators.

A second direction is related to the state estimation. The current estimated base linear velocity was one of the main limitations during deployment, for this reason an Extended Kalman Filter could be used to fuse IMU data with additional measurements such as leg odometry in order to obtain a more robust state estimation pipeline. A more reliable estimation of the base velocity would allow the policy to use real sensory feedback and

allow the robot to be validated on rough terrain and outdoor environments testing on more realistic operating conditions. [38]

Another important extension is the experimental validation of the reduced leg morphologies. Future work should test the trained policy with fewer active legs on the physical robot and how the gait changes depending on the missing legs. Furthermore with more powerful motors this evaluation can be extended to more challenging configuration not feasible with the current actuation system.

Future extension of this work can go toward continual learning [40] and antifragile locomotion. The robot should not only be able to move with different morphologies, but also decide when and how to reconfigure itself according to the environment, learning to exploit reconfiguration as an active strategy. The interaction with the environment would become a source of information and adaptation, allowing the robot to improve and adapt its behavior over time.

This direction is related to the idea that the robot would not only remain functional under disturbances but would use them to discover better behaviors. In this case reinforcement learning could be extended with online adaptation allowing the robot to update its locomotion from experience [19]. This would be relevant for environments that cannot be well represented during training where the robot must learn from real interaction rather than use only the knowledge learned into the simulation.

Another possible development is the exploit of the reconfigurable nature of the robot, treating each detachable leg as a separate operational unit. In this way a detached leg could become a small independent module able to move, sense and communicate information back to the main robot. This would transform reconfiguration from a mechanical adaptation problem into a cooperative multi unit behavior. For example in a search and rescue scenario, a detached leg could inspect narrow passages and collect information where the complete robot could not move. The main body and the other modules could then cooperate by sharing information about the environment. [39]

Finally, the platform could be evaluated in real application scenarios where an important characteristic is the ability to adapt. Possible domains include search and rescue, inspection of tunnels, industrial maintenance and space robotics. In these contexts the adaptation of the robot morphology could become an advantage, especially when the environment is unknown or difficult to model in advance.

Overall, future work should move toward a more autonomous and adaptive use of the reconfigurable nature of the robot. Improving actuation, state estimation and continuous adaptation would make the platform a stronger basis for studying reconfigurable locomotion in complex environments.

Bibliography

[1] Z. Wu, K. Zheng, Z. Ding, and H. Gao, "A survey on legged robots: Advances, technologies and applications," *Engineering Applications of Artificial Intelligence*, vol. 138, 109418, 2024, doi: 10.1016/j.engappai.2024.109418.

[2] S. S. Kotha, N. Akter, S. H. Abhi, S. K. Das, M. A. A. R. Khan, M. A. Hasan, and P. Das, "Next generation legged robot locomotion: A review on control techniques," *Heliyon*, vol. 10, no. 18, e37237, 2024, doi: 10.1016/j.heliyon.2024.e37237.

[3] Sihite E., Kalantari A., Nemovi R. et al. Multi-Modal Mobility Morphobot (M4) with appendage repurposing for locomotion plasticity enhancement. Nat Commun 14, 3323 (2023). https://doi.org/10.1038/s41467-023-39018-y

[4] X. Ding, F. Yang, Y. Cao, and C. Wang, "Fault-Tolerant Tripod Gait Planning and Verification of a Hexapod Robot," *Applied Sciences*, vol. 10, no. 8, 2959, 2020, doi: 10.3390/app10082959.

[5] F. Delcomyn and M. E. Nelson, "Architectures for a Biomimetic Hexapod Robot," *Robotics and Autonomous Systems*, vol. 30, no. 1, pp. 5–15, 2000, doi: 10.1016/S0921-8890(99)00062-7.

[6] U. Saranli, M. Buehler, and D. E. Koditschek, "RHex: A Simple and Highly Mobile Hexapod Robot," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 616–631, 2001, doi: 10.1177/02783640122067570.

[7] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning Quadrupedal Locomotion over Challenging Terrain," *Science Robotics*, vol. 5, no. 47, eabc5986, 2020, doi: 10.1126/scirobotics.abc5986.

[8] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015, doi: 10.1038/nature14422.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[10] S. Ha, J. Lee, M. van de Panne, Z. Xie, W. Yu, and M. Khadiv, "Learning-based legged locomotion: State of the art and future perspectives," *arXiv preprint arXiv:2406.01152*, 2024.

[11] J. Hwangbo et al., "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, 2019, doi: 10.1126/scirobotics.aau5872.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] A. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008, doi: 10.1016/j.neunet.2008.03.014.

[14] G. Bellegarda and A. Ijspeert, "CPG-RL: Learning Central Pattern Generators for Quadruped Locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12547–12554, 2022, doi: 10.1109/LRA.2022.3218167.

[15] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020, doi: 10.1109/SSCI47803.2020.9308468.

[16] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics," in *Advances in Artificial Life*, F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, Eds. Berlin, Heidelberg: Springer, 1995, pp. 704–720, doi: 10.1007/3-540-59496-5_337.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, 2017, pp. 23-30, doi: 10.1109/IROS.2017.8202133.

[18] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3803–3810, 2018, doi: 10.1109/ICRA.2018.8460528.

[19] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid Motor Adaptation for Legged Robots," *Robotics: Science and Systems XVII*, 2021, doi: 10.15607/RSS.2021.XVII.011.

[20] S.T. Hsieh, G.V. Lauder, "Running on water: Three-dimensional force generation by basilisk lizards," *Proceedings of the National Academy of Sciences*, vol. 101, no. 48, pp. 16784–16788, 2004, doi: 10.1073/pnas.0405736101.

[21] M. Zeng, C. Meng, B. Han, et al., "Gait Characteristics and Adaptation Strategies of Ants with Missing Legs," *Journal of Bionic Engineering*, vol. 21, pp. 2409–2423, 2024, doi: 10.1007/s42235-024-00572-w.

[22] R. Pfeifer, et al., "Self-organization, embodiment, and biologically inspired robotics," *Science*, vol. 318, no. 5853, pp. 1088–1093, 2007, doi: 10.1126/science.1145803.

[23] N. Cheney, J. Bongard, V. SunSpiral, and H. Lipson, "Scalable co-optimization of morphology and control in embodied machines," *Journal of The Royal Society Interface*, vol. 15, no. 143, 20170937, 2018, doi: 10.1098/rsif.2017.0937.

[24] S. Kriegman, S. Walker, D. S. Shah, M. Levin, R. Kramer-Bottiglio, and J. Bongard, "Automated shapeshifting for function recovery in damaged robots," *Robotics: Science and Systems XV*, 2019, doi: 10.15607/RSS.2019.XV.028.

[25] M. Mittal et al., "Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning," arXiv preprint arXiv:2511.04831, 2025.

[26] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning," arXiv preprint arXiv:2109.11978, 2021.

[27] L. Righetti and A. J. Ijspeert, "Pattern generators with sensory feedback for the control of quadruped locomotion," *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, 2008, pp. 819-824, doi: 10.1109/ROBOT.2008.4543306.

[28] S. Huang, Z. Xiao, M. Zheng, and W. Shi, "Hierarchical Reinforcement Learning for Enhancing Stability and Adaptability of Hexapod Robots in Complex Terrains," *Biomimetic Intelligence and Robotics*, vol. 5, article 100231, 2025, doi: 10.1016/j.birob.2025.100231.

[29] Q. Yang, Y. Gao, and S. Li, "Terrain-Adaptive Central Pattern Generators with Reinforcement Learning for Hexapod Locomotion," arXiv preprint arXiv:2310.07744, 2023.

[30] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion for Quadruped Robots," in *Robotics: Science and Systems XIV*, 2018.

[31] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, Architecture, and Uses in the Wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022, doi: 10.1126/scirobotics.abm6074.

[32] ROBOTIS, "DYNAMIXEL Protocol 2.0," 2026. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/protocol2/

[33] ROBOTIS, "XL430-W250-T e-Manual," 2026. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/

[34] D. H. Titterton and J. L. Weston, *Strapdown Inertial Navigation Technology*, 2nd ed. London, U.K.: Institution of Electrical Engineers, 2004.

[35] I. Lee, H.-G. Cao, C.-T. Dao, Y.-C. Chen, and I.-C. Wu, "Gradient-based Regularization for Action Smoothness in Robotic Control with Reinforcement Learning," *arXiv preprint arXiv:2407.04315*, 2024.

[36] S. Gangapurwala, L. Campanaro, and I. Havoutis, “Learning Low-Frequency Motion Control for Robust and Dynamic Robot Locomotion,” *arXiv preprint arXiv:2209.14887*, 2022.

[37] ROBOTIS, “XH430-W350-T/R e-Manual,” 2026. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xh430-w350/

[38] P. Agarwal; S. Kumar; J. Ryde; J. Corso; V. Krovi; N. Ahmed; J. Schoenberg; M. Campbell; M. Bloesch; M. Hutter et al., "State Estimation for Legged Robots: Consistent Fusion of Leg Kinematics and IMU," in Robotics: Science and Systems VIII , MIT Press, 2013, pp.17-24.

[39] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, “Modular Self-Reconfigurable Robot Systems,” *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007, doi: 10.1109/MRA.2007.339623.

[40] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, “Continual Lifelong Learning with Neural Networks: A Review,” *Neural Networks*, vol. 113, pp. 54–71, 2019, doi: 10.1016/j.neunet.2019.01.012.

Acknowledgements

Piccola premessa prima di iniziare, adesso è l'una di notte del giorno in cui devo consegnare la tesi e non so quando finirò di scrivere questi ringraziamenti. All'inizio non sapevo se volerli scrivere o meno, li prendevo un po' come una formalità ma poi mi sono ricordato le parole di mia mamma che mi dice sempre: "Grazie è la parola più potente del mondo" ed effettivamente non ha tutti i torti. Quindi eccoci qui, sperando di non dimenticare nessuno, sono pronto a pronunciare infinite volte questa parola magica.

Innanzitutto, vorrei ringraziare il professor Chiaberge per la sua disponibilità e il supporto dimostrati nei miei confronti. In particolare, lo ringrazio per essere stato sempre disponibile al confronto ogni volta che gli ho chiesto una chiamata o un consiglio, anche se non stessi lavorando direttamente a un suo progetto.

Ci tengo a ringraziare anche i miei supervisori, e ormai amici, di Londra Olga e Valerio. Grazie per avermi accolto nel vostro progetto anche se non avevo la più pallida idea di cosa realmente stessi andando a fare. Da voi ho imparato moltissime cose sia dal punto di vista accademico che dal punto di vista umano. Vi ringrazio ancora per tutte le esperienze che mi avete proposto anche se non ne avevo assolutamente le competenze e vi ringrazio anche per essere stati al mio fianco quando non funzionava proprio nulla. Spero di poter continuare a lavorare con voi e poter assimilare quanta più conoscenza, umanità e semplicità da voi.

Bene, ora sarà complicato ricordarsi tutti quanti ma dai su ci proviamo.

First of all I would like to say thanks to my London friends: Richard, Teresa, Yuri, Valentina, Matilde, Chiara, Bernardo, Guio, Matalia, Mateo, Francesco, Cecilia C., Gianluca, Cecilia T., Matteo and Anna Maria. Thank you to everybody as you accepted me and you loved me with all my limitations. In London it was a tough period at the beginning but you taught me that you can be happy even when you are far away from you friends and in a complete different context, and that the most important thing is even if I see everything black around me, there is always Someone that is always calling me and that I just have to say my yes.

Volevo ringraziare Edo, Bambi, Paul, Dani, Carlo, Gio, Abel, Scamu, Tere ovvero il COVO per avermi dato quei momenti di spensieratezza e un luogo in cui poter soffrire e gioire nell'arco di novanta minuti e soprattutto in compagnia.

Ringrazio anche Paolo, Marco, Ale, Carlo A., Jojo, Carlo P., Prospe per le partitelle a

basket, anche se spesso erano difficili da organizzare date le mille cose da fare.

Ci tengo a ringraziare anche i termolesi Nico, Giorgio, Marco, Giorgia, Luca S., Andrea, Mariagrazia, Giusto e Daniele. Grazie per essere stati un luogo sicuro in cui tornare durante le vacanze e soprattutto grazie per avermi accompagnato nella mia vita fino ad ora.

Volevo ringraziare la famiglia Stanchi e la famiglia Rondinelli che mi hanno accolto nelle loro case facendomi sentire voluto bene incodizionatamente e facendomi sentire quelle mura come un luogo dove so di poter sempre tornare e dove sarò sempre accolto.

Ringrazio anche la famiglia Annese, con voi sono cresciuto molto, anche perché ormai sono quasi otto anni che cammino insieme a voi. Vi ringrazio per tutti i momenti di spensieratezza dalle leggi di Ohm fino alle infinite pizze mangiate a casa vostra. Con voi sono passato dall'essere terrorizzato(si Gerardo all'inizio avevo molta paura di te, e anche Eugenio non scherzava) a sentirmi pienamente a casa quando sono con voi.

Inoltre volevo ringraziare Jim che mi ha accolto e accompagnato quando ero in Cina. Non mi hai accompagnato solo parlando al posto mio ma sei stato un compagno di viaggi e avventure in un luogo lontanissimo da casa. Ti ringrazio anche per la tua curiosità e per la tua serietà che hai messo nel nostro rapporto. Un momento che mi è rimasso impresso è che non scorderò mai è quando mi hai chiesto come funzionasse il cristianesimo, e quando mi hai parlato dei piani per il tuo futuro. Vedere come prendevi sul serio certe cose mi ha insegnato a prendermi più sul serio anche a me.

Se oggi sono diventato la persona che sono lo devo soprattutto al CLU in tutte le sue sfaccettature che proverò a ripercorrere tutto.

Per iniziare volevo ringraziare Cocca, Sofi, Zanna, Eli e Cri senza di voi non sarei qui a scrivere. Grazie per avermi ospitato a casa vostra durante il mio primo anno a Torino e avermi fatto vedere quella scintilla che poi mi ha cambiato totalmente la vita. Qui vorrei anche ringraziare Cossu, Djubo e Ale. All'inizio mi stavate un pochino antipatici, in particolare Cossu, ma poi mi avete fatto scoprire che la bellezza che si trovava in quella casa era un qualcosa di più grande e di ancora più bello.

Sicuramente uno dei posti in cui ho potuto vedere chiaramente questa bellezza è il Poli. Comincio nel ringraziare a partire dai vecchi: Djubo, Ale, Cossu, Zanna, Eleonora, Pino, Tumnus, Jojo, Gatto, Diba, Cinci, Tommy, Tere, Chicca, Save ed Elena che mi hanno insegnato tecniche anti-matricole e il fantastico gioco della Peppa e facendomi vedere che al Poli si studia ma non proprio sempre sempre. Continuo ringraziando i giovani del Poli, o meglio quelli che ci sono ancora(ora sicuro mi dimentico qualcuno): Paolo, Dani, Steve, GP, Franca, Dome, Sere, Cate, Cri, Nicola, Chiara, Michele, Michela, Carlo, Bubu, Settete, Agnu, Agne, Bambi, Bea, Coco, Fulmine, Gio, Ema, Mattia, Jack, Gap, Laura, Lidia, Marghe, MGB, Mari, Marvel, Mati, Cheloz, Mauro, Andrea, Nicolò, Pia, Paul, Vito e Simona. Vi ringrazio per avermi accompagnato in questa avventura nello studio per più o meno tempo, vi auguro di poter trovare la bellezza in quello che studiate e potervi appassionare come per grazia è successo a me.

Ringrazio tutto il coro per avermi fatto scoprire un luogo dove poter affidare le mie fatiche

e un nuovo modo di pregare insieme. Il coro è stata una delle cose che mi ha aiutato di più in questi anni, all'inizio era solo una questione di mi piace cantare, ma poi è diventato improvvisamente e misteriosamente un vero e proprio accenno di Paradiso.

Come non ringraziare i CP, siete stati una parte molto importante della mia vita, anche se a volte faticosa. Grazie ai CP ho potuto fare un'esperienza rara che mi ha fatto approfondire cosa vuol dire vivere l'università e prendersi sul serio fino in fondo. Siete stati in un luogo in cui sono cresciuto, capendo un po' di più cosa vuol dire rischiare e prendere sul serio tutto quello che mi circonda, a partire dalle riunioni in dipartimento fino ad arrivare alle tragedie che succedono oggi nel mondo, senza tralasciare nulla e andando a fondo di tutte le cose.

Poi volevo ringraziare i miei coinquilini. Coccettino, Carlo e Dani per avermi fatto conoscere cosa vuol dire vivere una vita insieme fatta di pura semplicità. Tommy, Save, Esteban e Bambi per avermi accolto dopo il mio periodo in Cina ed esservi messi in gioco con me. Tumnus e Pino per avermi fatto riscoprire la bellezza e la gioia del vivere una vita comune, che avevo perso nella catapecchia in cui vivevo a Londra.

Andando verso la conclusione voglio dire un grande grazie a Djubo ed Ale, se sono entrato in questa grande famiglia è soprattutto grazie a voi. Le cene e le chiacchierate infinite su discorsi più o meno seri, le nottate ad Oropa facendo le rendicontazioni. Mi avete fatto conoscere una compagnia che ha ribaltato completamente la mia vita. Per me siete stato un esempio vivente di cosa vuol dire la felicità, senza fare cose straordinarie ma semplicemente camminando al mio fianco nelle cose quotidiane. Ricordo perfettamente le chiacchierate in macchina con Djubo oppure il primo bando con Ale e mi stupisco ancora di come da cose così semplici sia potuto nascere un rapporto così bello e intenso, fondato sulla semplicità.

Ci tengo a ringraziare, in particolare, Chicca, MGB, Bambi e Tommy che sono stati dei compagni di viaggio, specialmente negli ultimi anni. Sono molto grato di come nel rapporto con voi io sia totalmente libero e di come i dialoghi con voi mi abbiano aiutato a prendermi seriamente, iniziando a capire cosa vuol dire diventare grandi. Da voi mi sono sempre sentito accompagnato tra le varie chiacchierate, pizze e giochi da tavola. Ho capito veramente cosa vuol dire un'amicizia costante e quotidiana nella mia vita. Anche quando non ero fisicamente a Torino sapevo che un messaggio o una chiamata con voi mi avrebbe sempre aiutato a risollevermi. Grazie per essere stati questi fantastici compagni di vita.

Poi volevo ringraziare il piccolo Dani e Paolo, siete stati una presenza importante nella mia vita. Dal cantare cori al giocare a basket, mi avete aiutato essendo presenti nella mia vita. Con la vostra semplicità avete aiutato ad essere semplice anche me, soprattutto quando mi facevo le mega pippe mentali per l'associazione o per il mio futuro. Mi avete insegnato che le difficoltà della vita vanno affrontate insieme. Vi ringrazio per come mi avete trattato sul serio, parlandomi a muso duro e riprendendomi quando stavo prendendo la tangente. Spero che la distanza fisica che ci sarà, per forza di cose, nei prossimi mesi e anni non faccia scemare questo rapporto che ho con voi.

Dai forse abbiamo quasi finito, non posso concludere senza ringraziare la mia famiglia.

Sicuramente non riuscirò a trovare le parole giuste per ringraziarvi ma farò un tentativo che sicuramente non racchiude tutti i ringraziamenti che ci sarebbero da farvi.

Grazie a Christian, nonostante tu sia una grandissima testa di cazzo, ti voglio bene e ti ringrazio per i momenti di spensieratezza che abbiamo avuto, per i discorsi filosofici avuti a casa, per i litigi per cose più o meno serie e soprattutto per le chiamate in questo ultimo periodo che sono stato a Londra. Ricordo con felicità quella videochiamata in cui ci siamo messi a piangere entrambi come due rincoglioniti, ma rimane tra me e te. La cosa di cui sono più felice è che posso chiamarti non solo fratello ma amico, ovvio non vengo ad ubriacarmi con te ma solo perchè non mi piace la birra, però con te io mi sento totalmente libero e posso essere me stesso, nel bene e nel male. Conlcudo facendoti un augurio, ti auguro che tu riesca a trovare la tua strada per la felicità come io ho trovato la mia e ti auguro di tenere sempre alta la testa e non rinchiuderti a riccio nei momenti di difficoltà. Ti voglio bene.

Grazie a mamma e papà per avermi sostenuto in questi anni, soprattutto moralmente. È vero non vi chiamo tutti i giorni ma siete una presenza costante nella mia vita. Io so che in ogni ora di ogni giorno voi ci siete stati e ci sarete sempre, per qualsiasi motivo. Dal semplice farmi compagnia durante i miei viaggi in motorino fino al capire che strada prendere nel mio futuro. Se sono la persona che sono adesso lo devo soprattutto grazie a voi, mi avete insegnato ad essere sempre semplice e corretto in ogni occasione. Mi avete insegnato a crederci sempre fino in fondo in tutte le cose che faccio senza mollare di un millimetro perché se non ci credo io fino in fondo nessuno lo farà per me e mi avete insegnato che nella vita essere se stessi e onesti è cento volte più importante che ottenere un risultato, che il mio valore non è dato da una performance ma dalla somma di tutta la mia persona(anche se mamma a volte se la dimentica questa cosa). Queste sono solo piccole parole di ringraziamento anche perché esprimere tutto il bene che vi voglio e quanto sono grato con delle parole. Grazie!

Infine voglio ringraziare Cristina, già nominata un paio di volte ma che merita un paragrafetto a parte. Non so oggettivamente da dove partire, la maggior parte delle cose che ho fatto nella mia vita è stato soprattutto grazie a te. Sei stata catapultata nella mia vita quasi per caso e sei stato il carburante che ogni volta mi spingeva ad andare avanti e fare cose che non avrei mai immaginato. La cosa che mi stupisce di più del nostro rapporto è che siamo cresciuti insieme in questi anni. Non siamo più gli stessi di otto anni fa(e menomale) ma il nostro rapporto è cresciuto parallelamente alla nostra persona individuale. La cosa più bella è che in questo rapporto è racchiuso tutto ed è grazie a te che sei stata sempre la mia fan numero uno che io sono stato spinto e sono diventato la persona che sono adesso. Grazie a te e alle cene nei tuoi vari appartamenti che ho conosciuto questa fantastica compagnia, grazie a te se sono andato dall'altra parte del mondo, grazie a te ho fatto la tesi a Londra ed ora sto scrivendo questi ringraziamenti. Ti ringrazio per aver messo il bene per me prima ancora delle tue sofferenze, so che non è stato facile stare per quasi un anno separati e non poter vivere la quotidianità a cui siamo sempre stati abituati, e grazie per non avermelo fatto mai pesare e anzi essere sempre stata vera con me, trattandomi come se fossi sempre stato lì al tuo fianco e non a centinaia di chilometri di distanza. Grazie per questi ultimi giorni in cui sono stato molto pesante e stressato

e grazie per avermi sempre sopportato oltra a supportato. Mi fa molto ridere che dopo aver riletto la mia tesi tu sei ora sul divano ad aspettare il momento in cui finisco di scrivere questi ringraziamenti per te. Anche qui non esistono parole adatte per descrivere quanto sono grato del nostro rapporto e ci vorrebbero altre cento pagine di tesi per poter esprimere decentemente tutto. Concludo solo con un ultima riflessione, senza di te al mio fianco i ringraziamenti si feremerebbero probabilmente al terzo paragrafo e anzi questa tesi non esisterebbe neanche. Questo solo per dirti quanto sei stata, sei e spero che sarai impattante e importante nella mia vita. Grazie Cristina!

Infine, ora la vera fine perché ho fame e voglio andare a mangiare. un ringraziamento al buon Dio che mi ha fatto vedere ed essere parte di tutta questa bellezza e che mi ha fatto un miliardo di regali di cui sono veramente grato e che sentivo e sento ancora di non meritarmeli tutti.

Grazie a tutti e spero che mi continuerete ad aiutare a dipingere questo dipinto della mia vita per sempre, che spero che diventerà sempre di più un capolavoro.

Vi voglio bene!

Luca