Controlling the locomotion of quadruped robots with learning methods

Michel Aractingi

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse

---

Présentée et soutenue par

Michel ARACTINGI

Le 6 décembre 2023

Contrôle de la locomotion des robots quadrupèdes à partir de méthodes d'apprentissage

---

Ecole doctorale : **SYSTEMES**

Spécialité : **Informatique et Robotique**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Philippe SOUERES

Jury

M. Jean-Baptiste MOURET, Rapporteur
M. David FILLIAT, Rapporteur
M. Nicolas MANSARD, Examinateur
Mme Madiha NADRI-WOLF, Examinatrice
M. Philippe SOUERES, Directeur de thèse

Acknowledgements

I would like to express my gratitude to my supervisors, Philippe and Tomi. I have learned a great deal from them about conducting meaningful research with high integrity. Our numerous hours of discussion were invaluable, and they were always available whenever I needed assistance and advice. Much of my understanding of robotics and its connection to machine learning and AI has been shaped by their insights. I would also like to extend my thanks to Pierre-Alexandre and Thomas, with whom I worked closely, learning many things about real robots and hardware.

Special thanks go to Julien and Chris, with whom I consistently had educational interactions. Naver Labs Europe and its members provided an excellent working environment that made this research journey very enjoyable. I appreciate the discussions I had with Nicholas and Olivier from Gepetto, along with other team members, whom I only wish I had more time to interact with.

I am grateful to the many friends who accompanied me throughout this journey, from the time I left Syria in 2013 to Lebanon and then France. While there are too many friends to mention individually, many of them have had a profound effect on me personally and on my path.

Lastly, my deepest appreciation goes to my parents and two brothers, who are the driving force behind everything I will accomplish in my life. Every endeavor is an attempt to represent them well and to give back a small part of what they have given me. I am especially grateful to Perla, who shared this journey with me from the beginning. Our story began in 2007, and I hope we will share many more chapters in the future. I couldn't have achieved this without her support; she is my main motivator.

This thesis is dedicated to the loving memory of my uncle, Dr. Mazen Jabali. I consider myself extremely lucky to have had him as a role model growing up. I only hope to be a fraction of the man and person he was.

Contents

Chapter 1

Introduction

1.1 Motivation

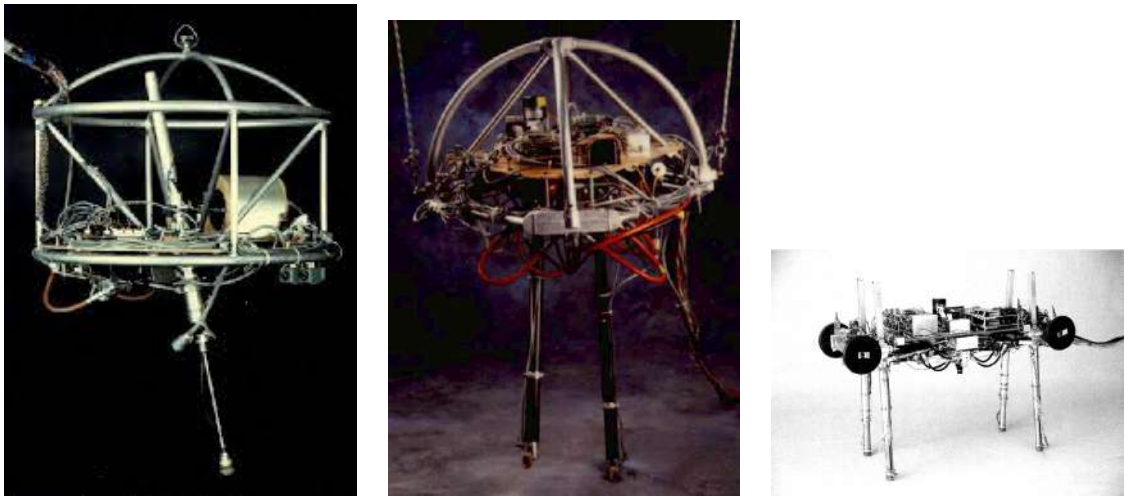
Legged creatures have superior navigation ability in wild environments. Legs enable terrestrial creatures to have better mobility and adaptability on different terrains and conditions. Moreover, we have designed our artificial environments in a way that necessitates the use of legs. Therefore, if we want to design robots for the purpose of aiding humans in everyday tasks then they would need to have legs because they would need to go up stairs, hike on mountains, access rough areas and inspect unstructured regions. Having legged robots would change many aspects of our lives once achieved.

For these reasons quadrupedal and bipedal locomotion are becoming increasingly popular topics of robotics and artificial intelligence research. While the mechanical design of legged robots is still considered a challenge, several robotic platforms have been developed for the purpose of studying and mastering quadruped locomotion [Hut+16; Gri+20; KCK19; Wan18] and biped locomotion [Sta+17; Kan+19; Kan+04b; Sak+02].

The major challenge in legged robotics lies in developing the necessary intelligence to enable legged robots to move efficiently. This involves addressing complex control and planning problems that arise due to many physical constrains and uncertainty in the dynamics, contacts with the environment and intricate robot kinematics. Many traditional approaches fail to address these issues in a manner that allows the robot to leave the lab and move towards uncontrolled environments.

Recent years have witnessed the rise of deep learning, a paradigm that applies machine learning algorithms to deep neural networks that act as non-linear function approximators [GBC16]. Deep learning has shown tremendous capabilities in fitting very complex functions with high-dimensional data samples. Deep learning has been successfully applied in supervised learning, in the fields of computer vision [LeC+89; LeC+98; KSH12] natural language processing [BCB15; SVL14], and reinforcement learning (RL), in the field of artificial games [Mni+15; Sch+19; Ber+19]. This has prompted many researchers to investigate the possibility of using similar techniques for robot control learning.

The primary goal of this thesis is to investigate the possibility of learning robust controllers for quadruped robots. We address different themes relating to the methodology of how to integrate learning in the robot control loop and the efficiency and efficacy of learned controllers. We validate our work on several real robots that we ran in challenging interior and exterior environments.



(a) 3D One Legged Hopper. (b) 3D Biped. (c) Quadruped Machine

Figure 1.1: Some of the early robots to come out of the Leg Lab at MIT. These robots were some of the first dynamical systems that moved and kept balancing on legs.

1.2 A Brief History of Walking Robots

Work on building legged machines inspired from animals and humans has been around for a long time. In modern times, the work done by the Marc Raibert's Leg Lab at MIT [Lab] created several excellent robots that would inspire a generation of researchers for years and culminate in the creation of Boston Dynamics [Dynb]. In the eighties, the Leg Lab introduced dynamic systems that balance on legs (see Figure 1.1), including the 3D one-legged hopping machine [RHC84], which could balance with one leg, the 3D biped robot, that could perform a back-flip [PR92], and the quadruped machine [RCB86], which could trot, pace, bound and perform transitions between the gaits. The design and control of these robots are relevant to this day and many heuristics used in current works are based on the seminal research done at the Leg Lab.

As the technology advanced around electronics, integrated circuits and sensors, so did the design of legged machines as it allowed robots to have on-board power and computation. Honda's ASIMO (see Figure 1.2a) was one of many bipedal robots to come out of that company. It had 26 degrees of freedom (DOF) and it could run at a speed of 9km/h and perform simple bi-manual manipulation tasks [Sak+02]. In 2005, the Korea Advanced Institute of Science and Technology (KAIST) [KAI] developed the Hubo robot (see Figure 1.2b) which had voice recognition and synthesis capabilities as well as eyes and vision in which its two eyes move independently of one another. The Humanoid Robotic Programme HRP initiative by the National Institute of Advanced Industrial Science and Technology (AIST) [AIS] developed the HRP-2 robot (Figure 1.3) that was the basis for a family future humanoid robots. Lot of optimal control research was done on the robot's that came out of this initiative to make it climb stairs [Car+16] and navigate through tight spaces [Kan+04a].

Along with humanoids, several kinds of legged robots started developing. In this thesis, we will focus on the topic of four legged robots (quadrupeds) since they are the main platform of our research. In the last three decades, Boston Dynamics have achieved big leaps in building legged robots that can move outside the lab and withstand disturbances coming from rough terrains, external pushes, feet slippage and added payloads. Big Dog

(a) ASIMO robot. (b) HUBO robot. (c) Atlas.

Figure 1.2: A later generation of modern robot with on-board computing and power.

Figure 1.3: HRP-2 robot climbing stairs [Car+16].

[Dyna], shown in Figure 1.4a, was the first robot that showed impressive robustness under extreme conditions on flat grounds and mountains which spawned a number of consecutive projects that resulted in the development of Petman [Nel+12], Spot (Figure 1.4e) and Atlas (Figure 1.2c). Earlier versions of these robots were equipped with hydraulic actuators and an internal combustion engine, which produced high power and torque. However, the practicality of these robots were limited due to the noise and difficulty of maintenance. At the Italian Institute of Technology (IIT), Semini et al. [Sem+11] created hydraulics actuated legged robots called HyQ (Figure 1.4b). The robot showed ability to move at 1.3 m/s using its powerful actuators. However, it relied on an external power supply due to its low power efficiency. In addition, it also used an internal combustion engine so it was not applicable for indoor applications. Its successor HyQ2Max [Sem+17] is a smaller version of HyQ and features compact and lightweight hydraulic actuators, thus it can carry an on- board power supply.

The legged robot lab in ETH Zurich developed numerous robots that, compared to the ones mentioned previously, were smaller and easier to deploy by a single person. Their robots were equipped with Series Elastic Actuators (SEA) [PW95] which allowed them to be efficient, robust against impact and provide accurate torque feedback [Geh+13]. This work culminated in the development of the ANYmal robot [Hut+16] (Figure 1.4c) that

(a) Big Dog quadruped.

(b) HyQ quadruped.

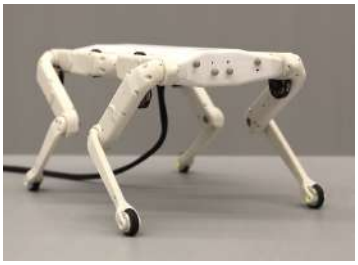
(c) ANYmal C quadruped.

(d) MIT's Cheetah.

(e) SpotMini.

(f) Unitree's Go1.

(g) MIT's Mini-Cheetah.



(h) Solo-8 quadruped.

(i) Raibot quadruped.

Figure 1.4: A later generation of modern robot with on-board computing and power.

has been the main platform for several works that develop different control methods for quadruped locomotion [Hwa+19; Mik+22; Lee+20; Mas+19].

The research done at the MIT's Biomimetic Robotics Lab revolutionized the electric actuator. Seok et al. [Seo+13] proposed design principles for the electric actuator in order to minimize energy losses that are common in locomotion due to heat losses from the actuators, friction losses in the transmission, and the interaction losses. Their design incorporates a low ratio planetary gear box and a large-gap-radius motor which results in high torque density motors and low impedance transmission. This design allowed the Cheetah robot to perform dynamic maneuvers including galloping and jumping over obstacles [PWK15; Hyu+14] (Figure 1.4d). This work later led to the development of the Mini-Cheetah robot [KCK19] (Figure 1.4g) that is able to perform all kinds of gaits at very high velocities and perform back-flips. The Mini-Cheetah's design, that consists of modular actuators that enable high-bandwidth and high-density force control, inspired the Unitree's series of quadruped robots including the Fo1 [Wan] (Figure 1.4f) and the Raibot that came out of the Railab in KAIST [Cho+23] (Figure 1.4i).

Finally, the open-dynamics robot's initiative (ODRI) was initiated by the Max Planck Institute of Tübingen [Cam] and the LAAS-CNRS [LAA] in order to provide low cost and low complexity actuator modules using brushless motors that can be used for different torque-controlled robots with simple 3D printed components. Using this technology, the TriFinger Manipulator Platform has been developed for benchmarking dexterous manipulation tasks and real time testing of RL policies remotely [Wüt+20]. Other projects include the Bolt biped [Bor+21] and the Solo robot [Gri+20] (Figure 1.4h). The Solo-12 and Mini-Cheetah quadrupeds are the main platforms of research used in this thesis. They will be further discussed in section 2.1.

1.3 The Challenges of Locomotion of Legged Robots

For a long time, the significant challenge in legged robotics was associated with mechanical design. Building capable legged systems for real-world applications is a big challenge that is not solved even with the capabilities of current technologies. Commercial robot like Spot [Dynb] and ANYmal [Hut+16] show impressive ability in inspection tasks and ANYmal has been shown to be able to hike on mountains [Mik+22]. Despite the impressive performances displayed by these state-of-the-art robots, doubts remain about their practical applicability. Their intricate design makes them expensive, unreliable, and fragile. Additionally, the unpredictable dynamics of their actuators make control a challenging task.

Quadruped robots are floating base systems, meaning their bodies are not fixed to an initial point which results in regular instability and balance problems. Additionally, quadrupeds are under-actuated systems which makes controlling them harder. Under-actuated means that the number of actuators is less than the number of DoFs, e.g., the base of the quadruped robot adds six degrees of freedom (position and orientation of the body) that are not controlled. This results in redundancy and non-linear dynamics that further complexifies the control. The agile and continuous control required for legged robots forces the controller to operate at a high frequency. The contact model of the robot is complicated due to the numerous feet and the lack of contact sensors on most of the quadrupeds. Contacts are hard to predict especially when the robot is blind and does not observe the surrounding environment.

Methods that rely on modelling have a hard time placing the robot in a new, never before observed environment. The complex dynamics and non-linear relationship between the base posture, joints and the contact with the environment make traditional methods that rely on linearization of the models obsolete. The noisy sensors and actuators of these robots also add to the issue of accurately modeling these robots which makes simulating them hard.

Model-free control approaches use a parameterized control policy. Methods that rely on Central pattern generators (CPGs) have been used in quadruped locomotion to define a nominal periodic motion of the joints [Rut+08; NAA03]. In CPGs, parameters like speed and frequency can be modified to change the nominal behaviour while keeping a basic pattern of motion that is easy to deploy and validate on the real robot. However, designing CPGs requires some knowledge of the robot and environment to design and could lack generality. On the other hand, Reinforcement learning approaches have been proposed to learn a policy from interaction data in the environment. The learning begins with a completely random policy outputting random actions to explore the states and rewards of the environment. The idea is to incrementally improve the agent's performance by

maximizing the cumulative reward signal over time [SB18]. Designing reward for RL is not an easy task and RL algorithms often require huge amounts of data to converge to the right behaviour. For that reason, we need to leverage simulators to generate this data. RL could exploit imperfections in the simulators where the modeling is inaccurate. Learned controllers that rely on simulations for data collection often learn to control a false version of the robot. Thus, when using RL for control, one needs to address the problem of transferring learned policies from the simulation to the real robot, or develop accurate modeling in simulation which is a hard a tedious task.

1.4 Thesis Statement and Organization of the Manuscript

The aim of this thesis is to contribute to the control of the locomotion of quadruped robots by developing machine learning and reinforcement learning based techniques for producing robust controllers for several robots of different sizes and dynamics properties. At first, the Solo-12 robot was the main robotics platform on which we conducted our work. Solo-12 is a lightweight robot that is very suitable for testing various policies that perform locomotion and acrobatic moves. Due to its lightweight nature and simple actuation design, we found it very easy to work with this robot and try learned policies directly on it. At a later stage, the MIT's Mini-Cheetah was acquired by Naver Labs Europe [NLE] which presented the opportunity to work on a heavier robot that exhibits stronger forces. Mini-Cheetah can locomote over harder exterior environments than the small Solo-12 robot. It allowed us to test the extent of the robustness and performance of the learned controller and the limit of our learning algorithms. However, working on a heavier and stronger robot leads to a wider Sim2Real gap which has to be addressed in the proposed methods to ensure an efficient control transfers to the real system.

The sections of each chapter provide the motivation of the work, the related literature, method description and results so that the full context for each idea can be explored and understood. Following this introductory chapter, the remaining part of the manuscript is structured in four chapters. Chapter 2 provides a more detailed presentation of the quadrupeds Solo-12 and Mini-Cheetah. It then presents general theory of the reinforcement learning and introduces related learning techniques that will be referenced and used throughout the thesis. It also makes the connection between the learning based techniques and the more traditional optimal control methods that are usually used to obtain controllers through modeling and planning.

Chapter 3 shows the first project conducted on Solo-12 that used the model-based controller developed by Leziart et al. [Léz+20]. The purpose of the work was to add a learned module in the controller to modify the gait pattern that is otherwise fixed. We show how the gait of the nominal controller is modified depending on the commanded velocity and discuss the significance and drawbacks of the approach.

Chapter 4 then describes the full procedure to learn end-to-end controllers for Solo-12. The chapter contains a complete description of the RL setup, control setup and additional techniques required to make the learned controllers work on the real robot. The experiments contain both simulation runs and runs on the real robot. Later, we explain that the proposed procedure does not work as well for a heavier robot in Mini-Cheetah. We show how to adapt what works on Solo-12 for Mini-Cheetah. We also discuss deploying our RL-based controller on a new custom-made quadruped robot built by our team at LAAS-CNRS. During the course of this thesis, we managed to successfully deploy RL-based controllers on three different quadrupeds. We discuss possible extension of our work to

bipedal robots and show preliminary experiments.

Chapter 5 shows how to expand the learning procedure to obtain policies that have diverse behaviour by exploiting the multiple constraints set in the reward function. We show impressive results on the Mini-Cheetah robot for climbing stairs and steep slopes all doing so without any vision capabilities.

Finally, Chapter 6 summarizes the contributions of the thesis and outlines future directions of research.

1.5 Related Publications

The work carried out in this thesis has led to the writing of the following papers:

- Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. "Learning to Adapt the Trotting Gait of Solo Quadruped". preprint, 2021
- Gianluca Monaci, Michel Aractingi, and Tomi Silander. DiPCAN: Distilling Privileged Information for Crowd-Aware Navigation. In: Robotics: Science and Systems (RSS) XVIII. 2022.
- Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. Controlling the Solo12 quadruped robot with deep reinforcement learning. Scientific Reports 13, 11945 (2023).
- Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. A Hierarchical Scheme for Adapting Learned Quadruped Locomotion. IEEE-RAS Humanoids, Austin, USA, 2023.

Chapter 2

Background

In this chapter, we provide general information about the robotic platforms used in this thesis. We also outline a concise theory of reinforcement learning and the algorithms used throughout the thesis. Finally, we present domain randomization and curriculum learning that are heavily used in this thesis. This chapter constitutes a backbone for the rest of the thesis.

2.1 Quadruped Platforms

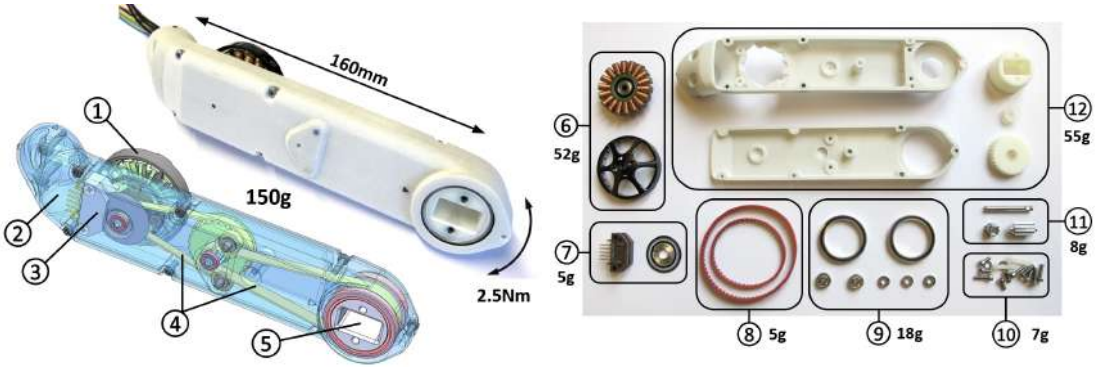
The main robotics platforms used in this thesis are the Solo-12 robot and the Mini-Cheetah. In this section we will introduce details about the mechanical design of both platforms. We will highlight their differences that lead to different choices when developing controllers for each robot. Figure 2.1a shows the Solo-12 robot and Figure 2.1b shows the Mini-Cheetah robot.

Solo

The Solo robot was developed as part of the Open Dynamic Robot Initiative (ODRI) [ODR]. Initially, the goal of this project was to develop a wide range of open-source low cost and low complexity small actuator modules that can be used to build torque-controlled robots with articulated joints. These robots can be assembled mostly from 3D printed and widely available components. We will describe the main actuation module and electronics that make up the robot and then discuss details about the robot's design.

(a) Solo-12 robot. (b) MIT's Mini-Cheetah.

Figure 2.1: The Solo-12 and Mini-Cheetah robots.



(a) Actuator module assembled.

(b) Disassembled components.

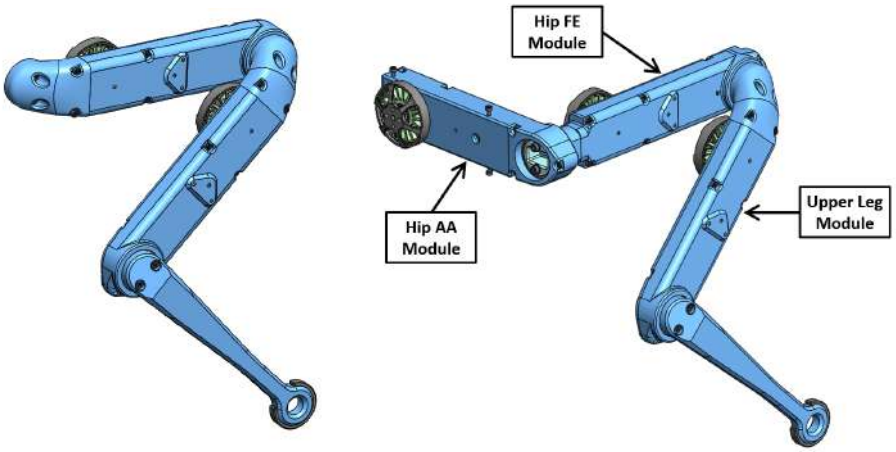
Figure 2.2: Brushless actuator module (a) assembled and (b) disassemled. BLDC motor ①, two-part 3D printed shell structure ②, high resolution encoder ③, timing belts ④, and output shaft ⑤. Brushless motor ⑥, optical encoder ⑦, timing belts ⑧, bearings ⑨, fasteners ⑩, machined parts (motor shaft and pulleys) ⑪ and 3D printed parts ⑫. Figure extracted from [Gri+20].

Actuator module. The actuator module consists of a brushless motor and a 9:1 dual-stage timing belt transmission. The low transmission ratio allows the actuator to output peak torques and high velocity at the joint while ensuring sufficient transparency to enable accurate torque control through motor current measurements alone. The actuator can output 2.7 Nm joint torque at 12 A. The module is also equipped with a high-resolution optical encoder and a 5000 count-per-revolution code wheel mounted directly on the motor shaft.The full actuation module in its assembled form and its individual components are shown in Figure 2.2

Driver boards. The driver boards are custom-made and open-source. They have been developed to execute dual motor torque control with reduced mass and volume compared to commercial driver boards. The driver boards are managed by a single master board which handles wireless and wired communications with the control computer. The drivers can run an onboard impedance controller at 10 kHz and operate at motor voltages up to 40 V.

The Solo robot. Grimminger et al. [Gri+20] presented the Solo-8 robot; an 8 degrees of freedom (DOF) quadruped with two joints per leg (Figure 2.3a). The robot is lightweight at 1.3 Kg. The Solo-8 2-DOF leg is composed of two actuator modules that control the hip flexion extension and the knee. The subsequent version of this robot is the Solo-12 that is 12-DOF since it has an extra joint per leg that allows the abduction-adduction motion of the leg as shown in Figure 2.3b. The electronics remain the same except for four additional motor drivers for the four additional hip joints. The additional hip actuator allows more freedom when controlling the rotation of the body of the robot. The body's dimensions are 45x30x6 [cm] which means it requires little effort to deploy and maintain.

Robot's Sensors. The robot is equipped with an Inertial Measurement Unit (IMU) attached to the body and incremental encoders at each joint. The incremental encoders provide the joint angle measurements $\mathbf{q} \in \mathbb{R}^{12}$ as well as the joint velocities $\dot{q} \in \mathbb{R}^{12}$ through finite differences. The IMU includes an accelerometer and a gyroscope to output linear acceleration, angular velocities and base orientation. Those sensors do not allow to directly measure the linear position and velocity of the base. State estimation techniques based on sensor fusion use forward kinematics with knowledge of the contact state of the feet and linear accelerations to estimate the position and linear velocity of the base



(a) Leg of Solo-8.

(b) Leg of Solo-12.

Figure 2.3: The CAD model of the leg of Solo-8 and Solo-12. The Hip AA joint (AA stands for abduction and adduction) controls the extra degree of freedom in the Solo-12 leg that allow lateral motions for the leg. The Hip FE joint controls the flexion extension motion of the leg. Figure extracted from [ODR].

[Léz22]. These quantities are important for model-based controllers that plan through time the position of the base [Léz+20]. However, in learning-based controllers we can dispose of these state information since they are implicitly present in the sequence of proprioception and IMU measurements as will be discussed in subsequent chapters.

Both the Solo-8 and Solo-12 platforms have been involved in numerous locomotion studies achieving different maneuvers like back flip and walking on two feet [FXP22; Li+22a; Li+22b]. Leziart et al. [Léz+20; Léz+22] developed MPC-based controllers for Solo-12. These controllers will later be discussed and used in Chapter 3 of this thesis.

Mini-Cheetah

Mini-Cheetah, developed by the MIT Biomimetic Robotics Lab [KCK19; Kat18], is more powerful, more reliable and more agile than lightweight quadrupeds like Solo. The morphology is similar to Solo as it is torque controlled and has 12-DOF with three actuated joints per leg. The robot has a computer and a battery so that it can be run autonomously only sending commands to it through a wireless joystick controller. The reliability of Mini-Cheetah allows us to test it in rougher environments and push the performance of the proposed controllers.

The innovation in Mini-Cheetah lies in the design of its actuator. Inspired by the actuator of the MIT Cheetah [Seo+13], the actuator design follows similar principles to optimize torque density, and a custom single-stage planetary gear box. The actuator was used to design agile quadruped robots like the MIT Cheetah 3 [Ble+18]. However, these actuators were very costly and not safe to be used as the robot's sizes were huge. Katz et al. [KCK19; Kat18] developed a small low-cost actuator designed with the same principles as the original MIT Cheetah actuator, with high-torque density motors, low-ratio and backdriveable transmission, which makes high bandwidth torque control possible. The size of Min-Cheetah is 60% smaller than the MIT Cheetah 3 which facilitates its maintenance and deployment.

The Mini-Cheetah has an IMU that provides information on the base's orientation,

angular velocity and linear accelerations. Each joint has a digital encoder for position sensing. We used the MIT Cheetah software that provides a coding interface to the robot's sensor readings and actuators [Di 20]. The built-in PD loop runs at a 40 KHz frequency, four times more than the Solo's PD loop. This low-level loop has a much higher bandwidth than high-level controllers, but it may be useful for tasks that involve high-speed leg trajectory tracking. Mini-Cheetah has been widely used as a test-bench for model-based controllers [Kim+19; Di +18] and learning-based controllers [Ji+22a; Mar+21; Mar+22].

2.2 The Reinforcement Learning Problem

Learning new skills for achieving different tasks is essential to the progress and generalization of embodied agents. Humans, often, resort to interacting with their environment in order to collect the necessary information, test different outcomes and understand their surroundings. This allows us to improve our decision making in order to choose actions that lead to the most favorable outcomes. Quoting the seminal textbook by Sutton & Barto [SB18], *"Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence"*.

Reinforcement learning (RL), a sub-domain of machine learning and artificial intelligence, focuses on how an agent can learn to make decisions by interacting with an environment in order to maximize some notions of cumulative reward. In RL, an agent learns to perform actions in an environment to achieve certain goals. The agent receives feedback in the form of rewards or penalties from the environment based on the actions it takes. The goal of the agent is to learn a policy or a strategy that will lead to the most favorable outcomes over time.

In model-free RL, the agent has no knowledge about its environment and the dynamics that defines it. *"Tabula rasa"* is a Latin phrase that translates to *"blank slate"*. In the context of learning and cognition, it refers to the idea that individuals are born without any innate knowledge and their understanding of the world is formed entirely through experience and interaction with their environment. Model-free RL is both a problem and a class of solutions that are specifically designed to address the problem of an agent learning from scratch from its experience [SB18; Sze10]. The goal is to learn the suitable actions to perform *reactively* in order to maximize the reward without planning with a model of the environment.

We will formalize the RL problem in the context of Markov Decision Processes MDPs, a formalization that will be essential throughout the thesis. After that, we will introduce some RL fundamentals and algorithms that are used in the following chapters.

Formalization

We model the reinforcement learning (RL) environment as a Markov decision process (MDP) with continuous state and action spaces [SB18]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, P_0)$, where $\mathcal{S} \subset \mathbb{R}^{d_S}$ is a set of states, and $\mathcal{A} \subset \mathbb{R}^{d_A}$ is a set of actions. In RL setting, only spaces $\mathcal{S}$ and $\mathcal{A}$ of the MDP are known to the learning agent. The agent starts by observing the initial state $s_0 \in \mathcal{S}$ and it performs actions $a_t \in \mathcal{A}$ in the environment at discrete times indexed by $t \in \mathbb{N}$, after which it receives a stochastic reward $r_{t+1} \in \mathbb{R}$ and observes a new stochastic state s_{t+1}.

The environment dynamics are described by a transition probability distribution $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, such that $\mathcal{T}(s, a, s') = p(s' \mid s, a)$ is the probability (density) that

the next state is s' given that the current state is s and that the action taken is a. P_0 is the initial state probability distribution. Similarly, the stochastic reward $r \in \mathbb{R}$ after taking an action a in a state s and observing a state s' next is governed by the function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \to \mathbb{R}_+$ that defines the probability densities $p(r \mid s, a, s')$. While in general $\mathcal{R}$ is defined as a density, in our simulations the reward function is a deterministic function of a and s'.

To formalize the goal of learning, we define a stochastic policy $\pi_\theta(s, a) = p_\theta(a_t = a \mid s_t = s)$, parameterized by θ, that gives the probability density of taking an action a given a state s. The learning objective is to find the parameters θ of the policy for which the expected discounted sum of rewards,

$$J(\theta) := \mathbb{E}[\sum_{t=1}^{H} \gamma^{t-1} r_t], \tag{2.1}$$

is maximized. In this expression H is the horizon of the episode and $\gamma \in [0, 1]$ is a discount factor. The expectation is taken over the stochastic policy, the initial state distribution, and the stochasticity of rewards and state dynamics.

Value Functions

In order to find the optimal solution, the RL problem has some metrics that relate the state, policy and transitions, to the delayed reward. In other words, the value of a certain state is a measure of how good the accumulated reward will be when starting from that state and following the actions from the policy. We define the state value function as the following expectation:

$$V_\pi(s) = \mathbb{E}_{s_{t+1} \sim \mathcal{T}, a_t \sim \pi}\Big[\sum_{j=0}^{H} \gamma^j r_{t+j} \mid s_t = s\Big]. \tag{2.2}$$

The value function V at state s_t indicates the expected sum of future discounted rewards starting at state s and acting under policy π. The aim is to find the policy that maximizes the value function at all states. The discount factor $\gamma \in [0, 1]$ has important consequences on the practical behaviour of the RL algorithms. Intuitively, when $\gamma < 1$ the future rewards are worth exponentially less than the reward at the first stage, therefore it allows to control the importance of immediate high rewards rather than high rewards in the future [Sze10].

Another important measure is the state-action value function that represents the expected cumulative reward an agent can obtain starting from a state-action pair then following a certain policy. We define the state-action value function, also known as the *Q-function*, as:

$$Q_\pi(s, a) = \mathbb{E}_{s_{t+1} \sim \mathcal{T}, a_{t+1} \sim \pi}\Big[\sum_{j=0}^{H} \gamma^j r_{t+j} \mid s_t = s, a_t = a\Big] \tag{2.3}$$

The Q-functions quantify the expected discounted sum of returns starting at state s and taking action a and acting according to π after that, from the next state. The Q-value is a useful relative measure that can be used to compare the returns of different actions starting from the same state.

To solve a reinforcement learning problem we have to find an optimal policy denoted as π^*. An optimal policy is expected to return the maximum amount of rewards when

run for a long time. The value functions can indicate whether a reward is optimal since the relation between π^* and a policy π is always $V_{\pi^*} \geq V_\pi$. There exists at least one optimal policy in the policy space. An optimal policy is defined by achieving the optimal state-value function V^* and the optimal action-value function Q^*, defined as:

$$V^*(s) = \max_\pi V_\pi(s), \text{and} \tag{2.4}$$

$$Q^*(s,a) = \max_\pi Q_\pi(s,a). \tag{2.5}$$

An optimal policy π^* can be represented by the optimal value functions,

$$\pi^*(s) = \arg\max_\pi V_\pi(s), \tag{2.6}$$

or optimal state-action value function, which is the basis of Q-learning[Wat89]:

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{2.7}$$

A final measure that will be very important in the next section is the advantage function A defined as :

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{2.8}$$

The advantage function measures how much better (or worse) an action is compared to the average action in a given state. A positive advantage indicates that the action is better than the average, while a negative advantage indicates that the action is worse.

The notion of the optimal value functions and policies is often useful in small finite MDP problems with a discrete state space and action space. Classic RL methods can solve these problems with dynamic programming style algorithms in a tabular and recursive manner where the dynamics of the environment are known. These methods can have guarantees of convergence [MR15]. For example, Policy Iteration is a model-based method that directly finds the optimal policy by iterating between two steps: policy evaluation where, for all states, the value function is computed using a random policy for providing the actions, and policy improvement that involves updating the current policy in a greedy manner according to the actions yielding the highest returns [SB18]. In model-free approaches where the transition function is unknown [RN94; Wat89], the idea is to estimate the optimal value function rather than performing a brute force search over all states. For example, temporal difference learning aims to learn the value of states or state-action pairs by bootstrapping from estimated values of subsequent states rather than unrolling the entire trajectory after each state [Sut88]. The value function estimates are updated, based on the Bellman equation, using a combination of the immediate reward and the estimated value of the next state or state-action pair. Approximate methods might have proofs of convergence in expectation only.

Value functions and deep learning. Using RL with neural networks has been explored in much of the seminal work by Sutton et al. [SB81; Sut88]. However, it was only with the recent rise of deep learning that research in that direction started growing. The combination of RL and deep neural networks methods acquired a lot of fame after the *Deep Q-learning* paper was published by Mnih et al. [Mni+15], which surpassed human levels in playing Atari games. The paper presents Deep Q-Networks a deep variant of Q-learning, originally presented in [Wat89]. In that work, the authors use Q-learning with the raw pixels gathered from the last t frames from the Atari game as input. The policy

only takes the greedy action (the argmax of the estimated Q-function). The novelty of the paper was both in the proposed deep Q-learning (DQN) architecture based on convolutional neural networks that map the observation images to the predicted Q-value of each action (control command in the Atari simulator) and the use of separate networks for outputting Q-targets to fit and estimating the current Q-values. This method was able to achieve super-human performance in the majority of the games it was trained on which increased the interest in deep RL.

Testing deep RL algorithms on Atari games is a common practice in the field especially after the success of the DQN method. Video games are simple testbenches with cheap simulation costs and a direct access to a built-in reward function. Other methods were developed to improve the state-of-the-art for Atari games [Mun+16; HGS15], while other methods attempted to extend the approach for problems with continuous actions [Lil+16] and even some early work on using Q-learning to aid in robotics perception problems [LR13]. However, for the task of learning control policies for continuous actions in robotics, DQN-like methods are not preferable. These methods work well for discrete action spaces, but are intractable when the action space is high-dimensional, and they are infamous for suffering from many stability issues that often lead to failures or inconsistencies in the final policies.

Policy Gradients

The policy is a mapping from states to actions. There exist an entire class of RL algorithms, called Policy Search, that directly learns a parameterized policy without consulting a value function. A form of the value function is still involved in the learning process, but it is not used in the action selection. One family of algorithm that fall under Policy Search are Policy Gradients. The idea is to perform gradient descent updates on the parameters that define the policy in the direction that maximizes the objective J (Equation 2.1) [Sut+99].

Performing gradient descent directly on J is not possible as the objective is based on expectations and the underlying functions are unknown. Williams [Wil92] proposed REINFORCE, a method for estimating the gradient of the objective through Monte Carlo Sampling. The general REINFORCE gradient is:

$$\nabla_\theta \mathbb{E}\left[\sum_t^H \gamma^t r(s_t)\right] = \mathbb{E}\left[\sum_{t=0}^H G_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{2.9}$$

where the choice of G_t is generally called a return. Typically the choice of G_t has an effect on the variance of policy gradients. It can be represented by different values:

- the total sum of rewards of the trajectory: $\sum_t^H r(s_t)$.
- a baselined version of the previous formula. The idea is to compare the trajectories with the previous history and see how the return is improving relative to the returns of previous trajectories: $\sum_t^\infty r(s_t) - b_t$.
- the state-action value function: $Q_\pi(s_t, a_t)$.
- the advantage function: $A_\pi(s_t, a_t)$.
- the temporal difference residual: $r(s_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$.

Actor-Critic algorithms constitute a class of reinforcement learning (RL) methods that combine elements of both value-based and policy gradient approaches. They aim to learn both a policy (the actor) and an estimate of the value function (the critic) simultaneously. This combination improves the stability and efficiency of RL algorithms. There exists a large amount of actor-critic algorithms [KT99; Mni+16; Sch+15; Sch+17; Haa+18], but for brevity, we will only mention details related to the ones used in the thesis.

Throughout this thesis, we use proximal policy optimization (PPO) [Sch+17] as the choice of RL algorithm. PPO is a policy gradient algorithm that is implemented in an actor critic style. PPO in on-policy, which means that the collected data at each episode is used to update the policy and then discarded before the next exploration phase. The ideas in PPO are based on a previous optimization algorithm called trust-region policy optimization by Schulman et al. [Sch+15] (that was also based on the seminal work by Kakade & Langford [KL02]), where the goal is to solve the following constraint optimization problem:

$$\begin{aligned} &\max_{\theta} \mathbb{E}_t\left[\frac{\pi_\theta}{\pi_{\theta_{old}}} A_{\theta_{old}}(s_t, a_t)\right], \\ &\text{subject to } \bar{D}_{KL}(\pi_{\theta_{old}}, \pi) \leq \delta, \end{aligned} \tag{2.10}$$

where $\mathbb{E}_t$ indicates the empirical average over a finite batch of samples and $\bar{D}_{KL}$ refers to the KL divergence. The goal of TRPO is to find a local region in which we can update the parameters of the policy with proven theoretical guarantees of improvements and without diverging too far away from the current policy. The constraint on the KL divergence between the new policy and the old policy is used to take steps in a robust way, i.e., within a trust region.

In PPO, the authors choose to solve an unconstrained penalized optimization problem. Therefore, the constraints is added to the objective in the form of the clipping operator. Since the gradient estimation is noisy, then the optimization should take little steps in the direction of maximizing the advantage objective without moving too far from the old policy:

$$\max_{\theta} \mathbb{E}_t\left[\frac{\pi_\theta}{\pi_{\theta_{old}}} A_{\theta_{old}}(s_t, a_t)\right] + \beta \bar{D}_{KL}(\pi_{\theta_{old}}, \pi_\theta), \tag{2.11}$$

where β is a coefficient that balances between objective and constraint. However, it is hard to find the right value for β and fixing it through training results in sub-optimal learning [Sch+17]. The authors of PPO proposed a version based on clipping, which leads to a simpler algorithm. The clipped objective defines the core of the RL algorithms used in the next chapters:

$$\max_{\theta} J_{PPO}(\theta) \triangleq \max_{\theta} \mathbb{E}_t\left[\min\left(\frac{\pi_\theta}{\pi_{\theta_{old}}}\hat{A}_t, \text{clip}(\frac{\pi_\theta}{\pi_{\theta_{old}}}, 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{2.12}$$

where $\hat{A}_t = A_t(s, a)$, and ϵ is the clipping ratio. The clipped term is a bounded version of the unclipped term. We chose PPO as our RL algorithm since it has been proven to work for a variety of problems and is easy to implement and use.

Practical Implementation. In our work, the full learning objective includes maximizing the PPO objective and minimizing the regression objective over the estimation of the

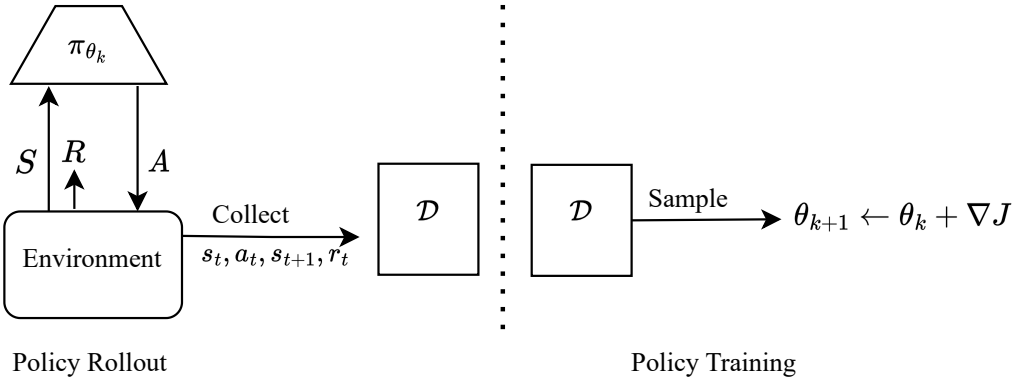


Figure 2.4: Scheme of the on-policy training process. The training happens alternates between collecting data by rolling out the policy in the environment (typically simulated) and improving the current policy based on that data.

value function for the critic network (which helps in reducing the variance of the algorithm) and maximizing the entropy of the policy in order to encourage exploration at the beginning of training. The on-policy buffer collects state, action, reward and done flags to indicate whether the policy failed or not. The returns are computed based on the rewards with the generalized advantage estimate method [Sch+16]. When the action space are continuous, as in Chapter 4 and Chapter 5, they are implemented as a multivariate Gaussian distributions with a diagonal covariance matrix. Each action has its own mean that is learned by the policy network and a standard deviation that is only used in learning to sample random actions around the mean. At deployment, the mean is directly taken as the action. Figure 2.4 shows a simplified scheme of how general on-policy RL algorithms, like PPO, work. The learning alternates between two stages: (1) rolling out the policy and collecting the states, rewards and action encountered and outputted by the current policy in the on-policy data buffer $\mathcal{D}$, and (2) sampling batches of the data in $\mathcal{D}$, after calculating the returns, and updating the parameters θ in the direction that maximizing the RL objective.

2.3 Domain Randomization

The sim-to-real (sim2real) problem is a significant challenge in machine learning, particularly in the domain of robotics and reinforcement learning. It refers to the problem of transferring a model or policy learned in a simulated environment to perform similarly in the real-world environment.

Currently, the most successful deep RL methods require massive amounts of data to converge to proper models and solutions [Mni+15]. Simulated environments provide a controlled and cost-effective way to generate data for training machine learning models for robot control and perception, without the risk of damaging expensive hardware or encountering dangerous situations. However, the synthetic data generated from simulators does not match the real observations which creates a reality gap that prohibits the learned models to work well on data that comes from a source different from that used during training.

Robotics in particular suffer hugely from the reality gap. The gap is triggered by an inconsistency between parameters of the physics engine, like the friction, stiffness,

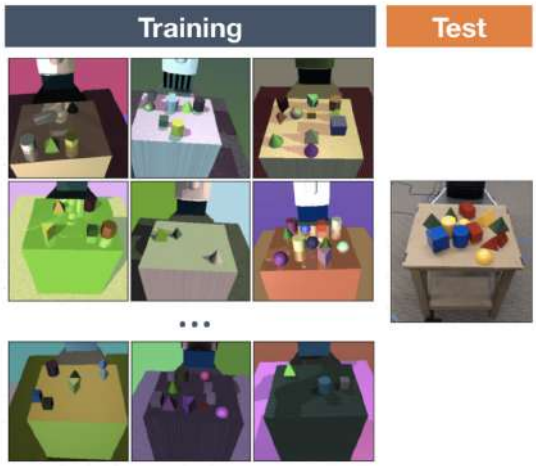


Figure 2.5: Randomizing visual properties of the testing environment as well as the physical location and orientation of the camera and objects. Figure extracted from [Tob+17].

damping, mass and density, and, mismatch in the physical model due to wrong contact and collision models and the difficulty of simulating different material types and soft surfaces [Che+18; And+18; Wen19]. Simulators are very important in model-free RL research as learning from scratch by performing random actions on the robot could be dangerous and break the real system. It would also be very costly as most RL algorithms require a lot of data to converge and learning on the real robot would be too slow. Physics engines use different rigid body dynamics algorithms to generate the states and the transitions when actions on the robot are executed [Fea08]. The accuracy and complexity of these algorithms are the two important factors that determine the usability of these simulators. Therefore, in many simulators, many aspects like friction and contact are approximated to reduce the complexity [HLH18]. These simplifications results in inaccurate models that can be exploited by the RL agent during training to learn unrealistic behaviour that would not work well in the real world.

Researchers have proposed different techniques to address sim2real issues. For example, system identification is the process of building mathematical models of dynamical systems from measured data. To ensure that the models are realistic and match, careful calibration is necessary in each new situation. This calibration can be expensive and inadequate as the properties of dynamical systems can vary widely depending on different environmental elements [BL05; YLT17]. Domain adaptation is another technique for addressing the reality gap. The goal is to change the data distribution from the source (simulation) to match the data of some target distribution (observed on the real system). This mostly relies on regularization techniques and adversarial losses [Goo+14] to obtain a mapping that is often task related. Performing domain adaptation for robotics is very tricky as the data in the target domain is scarce and requires collecting expert data which requires running the robot.

Finally, domain randomization aims at improving the generalization of machine learning models by randomizing different properties of the simulated environment. The goal is to produce models that can adapt to different variations of the same instances [Tob+17]. Domain randomization injects variations within the source domain in order to represent the discrepancies between the source and target domains [Pen+17a].

Suppose we can control a set of N randomization parameters in the source domain e_ϵ

Figure 2.6: Simulating different visual appearances of the same task. Figure extracted from [And+18].

with a configuration ϵ, sampled from some randomization distributions of η.The data collected during training is sampled from the source domain with the applied randomization. The RL objective remains to find the optimal parameters θ^* for the policy π such that it maximizes the expected reward R for trajectory τ (randomized by ϵ) averaged across a distribution of configurations:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\epsilon\sim\eta}\Big[\mathbb{E}_{\pi_\theta,\tau\sim e_\epsilon}[R(\tau)]\Big]. \tag{2.13}$$

Figure 2.5 shows different examples of applying randomization to robotics tasks. Domain randomization has been used in robotics perception tasks to enable the sim2real transfer of vision based manipulation policies by adding noise to synthetic RGB images as well as parameters related to the position and orientation of the arm and the target blocks [Tob+17; SL16]. Peng et al. [Pen+17a] applied randomization to learn a control policy that transfers to the real robot by randomizing the dynamics of the robot in simulation (mass of the links, PD gains, action delay, etc.). Similar techniques have been applied in dexterous manipulation [And+18] (Figure 2.6) and locomotion [Hwa+19; Lee+20]. Most works use fixed uniform or Gaussian noise for the randomization but some propose learning or updating the randomization distribution to improve the generalization of the final policy [Cub+18; RSC19].

In this thesis, we use dynamics randomization in order to facilitate the deployment of RL policies on different real robots. Legged robots are hard to simulate because their actuators are hard to model [Hwa+19]. Additionally, their agile nature results in noisy measurements and instabilities that require accurate collision models and contact detectors to simulate properly. For Solo-12, adding noise to the observations and some dynamics was enough to learn a policy that transferred successfully, However, for Mini-Cheetah we encountered more difficulties that required a more complex transfer method.

2.4 Curriculum Learning

Curriculum learning is a machine learning technique that involves presenting training data to a model in a specific order or sequence, gradually increasing the difficulty of the learning task over time [Ben+09]. The goal of curriculum learning is to improve the

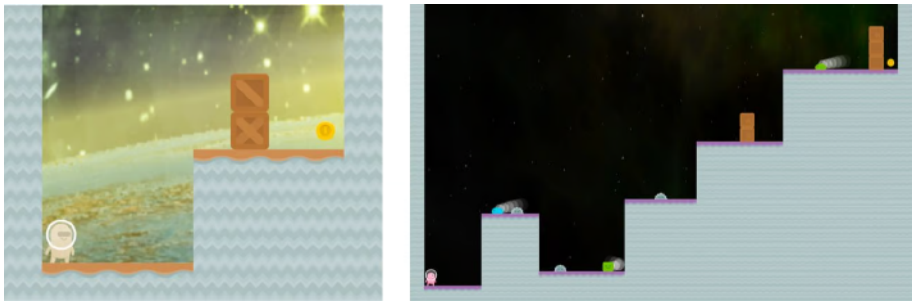


Figure 2.7: Two environments of the same task, generated by procedural generation in CoinRun at different difficulty levels. Figure extracted from [Cob+18].

learning process and overall performance of the model by leveraging the idea that certain tasks are easier to learn than others. This idea is inspired by the way humans learn, starting typically with simple concepts and gradually moving to more complex ones.

Curriculum learning has been shown to improve the sample efficiency and generalization. It is also regarded as a way to help the model escape local minima since it can prevent it from getting stuck in regions with high error early in the training process [WC18]. These methods have shown success in supervised learning [ZS14; STD19] and reinforcement learning [Mat+17; Suk+17; Cza+18; Gra+17].

Curriculum learning techniques are present in the literature of learning to solve games where the environments are generated procedurally. The Procgen benchmark [Cob+19] and CoinRun [Cob+18] provide a way to design diverse game levels of various difficulty (see Figure 2.7). They have been explored for the purpose of learning a policy that can solve the hardest levels by starting from the simplest levels and increasing the complexity of the generated levels as training progresses.

Using a curriculum turned out to be essential in learning locomotion. The locomotion task is generally centered around tracking a command velocity. The space of possible locomotion controllers that can be learned to follow this abstract task is very big. Therefore, one needs to add more constraints on the reward to make the task more specific and guide the behaviour towards a specific style of locomotion. However, adding constraints can hurt the learning process as it becomes too complex and the agent might learn to fulfill the constraints only without performing the actual velocity tracking. Therefore, as with the game environments [Cob+19; Cob+18], a curriculum can be very natural in this setting by letting the policy learn the main task first and gradually adding the constraints in order to refine the learned movement. We will discuss in section 4.3.1 the procedure of applying curriculum on the reward function.

Later, we would ask the robot to move over complex terrain such as slopes and stairs. This task makes the locomotion learning even more difficult. Starting to walk directly on stairs proves to be difficult and requires the design of a curriculum where the robot first starts learning on a flat ground and small stairs and steps terrain before learning to traverse the full complex environment.

Chapter 3

Learning Gait Transitions for Model-based Optimal Control

Contents

3.1 Motivation

Locomotion is defined through different measures and elements. One of the main measures to assess locomotion and differentiate different styles of locomotion is the *gait*. In general, a gait is defined as a periodic pattern of limb movements made during locomotion, both for robots and animals. Animals often have the ability to adapt their gait depending on the environmental conditions and desired characteristics such as speed, stability, maneuverability or energy efficiency.

Hoyt & Taylor [HT81] show that horses use different gaiting patterns for different velocities, which in turn controls their possible achieved velocities and energy consumption. By observing the oxygen consumption as an indicator for energy consumption, the

authors find that there is a natural gait for any speed that would consume the least energy. Figure 3.1 shows the energy consumption for different gaits at different speeds. We see the trade-off between velocity tracking and energy consumption for the different gaits.

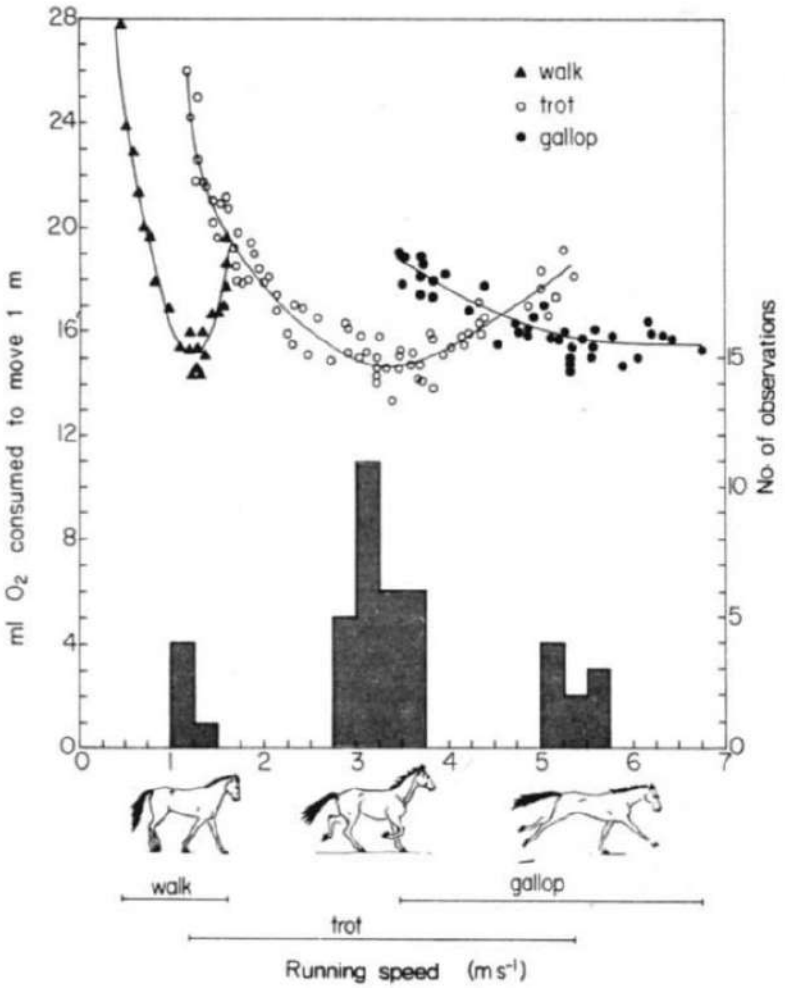


Figure 3.1: The energy consumption of horses at different speeds for different gaits. The figure shows that there is a suitable gait for each velocity that is energy efficient. Figure extracted from Hoyt & Taylor [HT81].

Gaits can be distinguished by the stepping pattern that in turn is determined by the contact phases of the feet with the ground over a certain period of time, i.e., observing the sequence of contact/swing phases for each foot. Figure 3.2 shows different sequence pattern for the four feet that define the walking, trotting and galloping. These gaits can be naturally observed on different animals [Gra22].

We set out to study the possibility of implementing different gaits for quadruped robots. Several model-based controllers, developed for quadrupeds, rely on a mix of trajectory optimization and whole body control [Di +18; Kim+19; Bel+18; Léz+20] in order to achieve different gaits. These controllers generally display efficient locomotion on different platforms like Mini-Cheetah [KCK19] and Solo-12 [Léz+20]. However, they rely on complex control architectures composed of sequences of blocks with hand-tuned parameters that output a solution to one part of the problem. The reliance on hand-tuned parameters makes decision such as gait pattern and frequency hard to adapt in a flexible manner.

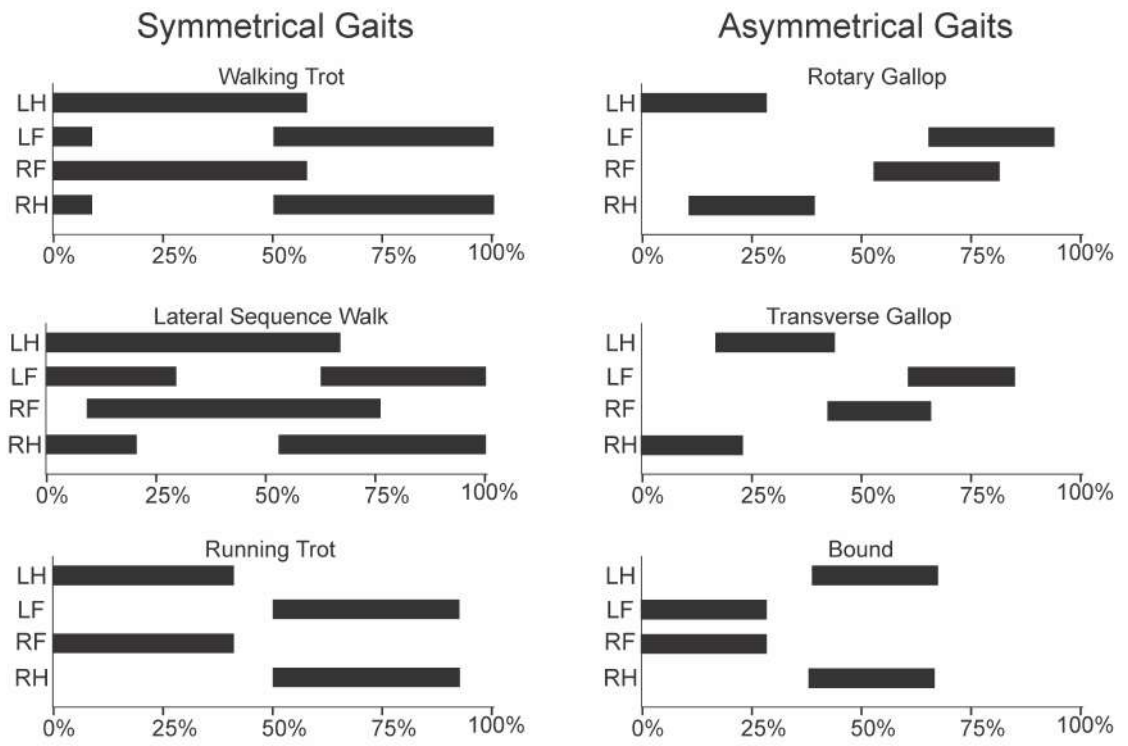


Figure 3.2: Examples of different gaits defined by the contact sequences over the span of a period for quadrupeds. The black area characterizes the time when a foot is in contact with the ground (stance phase) and the white area shows for when a foot is in a swing phase. L and R stand for left and right, F and H stand for front and hind. Figure extracted from [Gra22].

Model-free reinforcement learning (RL) makes it possible to learn controllers without hand-tuning such control blocks. RL methods have shown some success in learning locomotion tasks [Pen+20; PP16; Pen+17b], particularly in adapting to new situations where classic controllers fail to work satisfactorily [Son+20]. The efficiency of model-based control and the ability of deep learning approaches to learn complex non linear mappings present an interesting opportunity to combine the merits of both approaches.

Outline. In this chapter, we study the problem of how to adapt the gait of a quadruped in order to improve its spent energy and velocity tracking. Our motivation comes from the fact that humans and animals can lower their cost of transport by adapting their gait depending on their velocity [HT81]. Our main contribution is a method that uses a model-free deep RL policy to adapt the timings of the contact/swing phases of each foot independently. We use this policy to augment a model-based controller that was previously developed to endow the Solo quadruped with trotting, walking and static gait capabilities [Léz+20]. We show that the proposed method can adapt the nominal trotting in different situations between a walking trot pattern to a rapid trot. This improves the energy consumption and reference velocity tracking of the controller. Another contribution is the policy architecture. We argue that the prediction of events that would impact future decisions can be improved by considering a history of states. To take advantage of this history, we propose using self-attention mechanisms [Vas+17] that are commonly used in language tasks to handle sequences of inputs. The experiments demonstrate improvements in the convergence and performance of the learning. The chapter is organized as follows. Related literature is discussed in Section 3.2. Description of background in-

formation about the controller and gaiting mechanism is provided in Section 3.3. The method and experimental results are outlined in Section 3.4 and Section 3.5 respectively.

3.2 Related Literature

In this section, we briefly present related literature to model-based control for quadrupeds. The focus will be on the methods based on trajectory optimization as it is the main optimal control technique related to our approach. After that, we review approaches with similar ideas of combining model-based control with machine learning techniques.

Among them, the control scheme developed by [Kim+19] for Mini Cheetah combines Model Predictive Control (MPC) and Whole-Body Control (WBC). The MPC is tasked with long horizon planning based on simplified dynamics while the WBC handles low-level control at finer timesteps. This hybrid architecture proved to be stable while achieving a record running speed for this robot. A similar control pipeline was used for Solo in [Léz+20] while simplifying solutions for the computation of the WBC.

Some recent works in the related literature proposed to adapt the contact sequence through a learned process. [Da+20] developed a policy that learns to choose the next contact sequence from a set of predefined contacts. [Lee+20] proposed to learn a swing phase delta-variable, which decides whether or not the foot should be in contact, along with the displacement of the foot position. A method to learn gait transitions by learning the gait schedule as a function of the reference velocity was introduced by [Yan+21]. The work, in this section, focuses on using a more elaborate state and policy representation that considers a history of observations which allows the policy to be more confident in its decisions.

Other methods choose to have an MPC alongside a learned model [CFH20; SS20]. [CFH20] proposed a modification of guided policy search [LK13a] where the policy learning is guided by the MPC and the objective is to optimize the control Hamiltonian. [SS20] proposed to learn high-level decision variables for a low-level MPC to adapt its solution in difficult situations where it normally fails. [Tso+20] used model-based motion planning ideas to train a gait planner and gait executioner policies when moving on non-flat terrains. We propose a simpler approach to adapt a nominal gait to directly improve the spent energy and velocity tracking.

3.3 Model-based Controller

The robotic platform used in this chapter is the Solo12 quadruped, a 12 degrees of freedom (DoFs) version of the Solo8 quadruped, introduced in [Gri+20], with three actuators per leg. The task is to follow a user-defined reference velocity with three components: forward and lateral linear velocities and angular velocity around the vertical axis. The RL approach in this chapter comes as a subpart to the model-based controller proposed by Leziart et al. [Léz+20]. It provides minor modifications to the foot trajectory generator, so that the planned trajectory of the swing feet can be modified and adapted by the RL agent on the fly. In the following section we briefly describe the nominal control architecture we used and its mechanism for defining and executing gaits.

3.3.1 Architecture Overview

The nominal controller (shown in Figure 3.3) is centered around three main control blocks: a footstep planner, a model-predictive controller (MPC) and a whole-body controller (WBC). The footstep planner plans the sequence of future foot steps positions on the ground. The MPC computes a sequence of ground contact force based on the centroidal dynamics model and the location of current and future footholds. Its objective is to have the body track the reference velocity prescribed by the user (with a joystick for example). The WBC takes as inputs the desired contact forces, for the feet in stance phase, and the desired motion of feet in swing phase and outputs the desired torques, positions and velocities for the 12 actuators. To do so, it incorporates the information from the full body dynamics and uses inverse kinematics to find the proper coordinates of the swing leg.

The final torque values sent to the actuators are the WBC torques values to which are added the feedback torques of a PD+ controller based on the difference between desired and current joint positions and velocities and a feedforward torque.

While the WBC solves an instantaneous problem and provides low-level commands at high frequency (500 Hz), the MPC plans over a prediction horizon knowing the future footholds but at lower frequency (50 Hz) due to computational requirements. WBC can only consider one step ahead and operates at a high 500Hz frequency while the MPC can plan over a long prediction horizon and runs with a frequency of 50Hz.

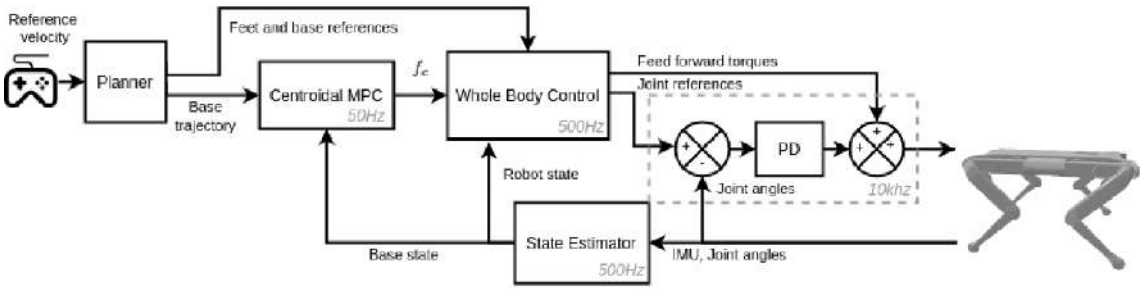


Fig. 2: Reactive walking controller architecture

Figure 3.3: The model-based controller architecture. Figure extracted from Leziart et al. [Léz+20].

The trajectory of a foot during swing phase is planned using polynomial functions to link its current position to a target position on the ground. Those polynomials are constrained by non-slipping conditions, i.e., zero velocity and acceleration when the foot takes off and lands on the ground. The foot trajectory generator outputs a reference position, velocity and acceleration at each time step of the swing phase. These values are used by the WBC as references for the inverse kinematics and torques computation. Based on Raibert's heuristics [Rai90], the footstep planner outputs the target locations of footsteps using heuristics that rely on the gait, the current and desired body velocities to be tracked. The controller has shown successful trials on the real Solo12 platform. The controller works well for gaits where two or more feet are in contact with the ground, i.e., it supports various trotting, walking and static gaits. Further details on the nominal control architecture can be found in the PhD manuscript of Pierre-Alexandre Leziart [Léz22].

3.3.2 Gait Transition Mechanism

Formally, the gait sequence is discretized to fit the discrete control nature of the Solo-12 controller. This means that for each MPC optimization timestep a specific binary indicator

for each foot has to be planned to determine whether that foot will be in stance phase or in swing. Therefore, Solo's trotting sequence is determined at time t by a binary gait matrix $G_t \in \{0,1\}^{M\times 4}$ that describes the planned feet contacts for the incoming M time steps.

$$G_t = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}. \tag{3.1}$$

Column j of the matrix describes the future ground contact states for foot j (the columns are in the order of the feet FL, FR, HL, HR). The i^{th} row of G_t describes the ground contact states of the four feet that the controller should consider for the i-th time step of the MPC prediction horizon, i.e, at time $t + i \times \Delta t_{\text{mpc}}$, where Δt_{mpc} is the length of one MPC time step. The number of rows M matches the number of time steps in the MPC prediction horizon.

A trotting gait is defined by having two diagonally opposite feet in contact with the ground while the other two are in a swing phase. Once one MPC timestep is passed, the executed first row is moved to the end of the matrix while the second row is advanced to first,

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \xrightarrow{t+1} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{t+2} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \cdots \xrightarrow{t+N} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

The policy can change the contact sequences and adapt the gait period and the duration of both stance and swing phases within one period, and therefore the potential overlap of those phases between the feet. These modifications can lead to gait variations that have different properties in terms of speed, energy consumption, reactivity and robustness. The controller's gait matrix is pre-defined to trotting with a period of 0.32s.

3.4 Learning Gait Adaptation Policies

Our first contribution was to propose a method to learn how to adapt the contact patterns for each foot in order to improve the control performance. We chose to formalize the task as a sequential decision problem and solve it with RL techniques. The formalization follows the one in Section 2.2. One can write a trajectory optimization program, based on an optimal control solution, to adapt the contact phases of the gait [Win+18]. However, such a method would require intensive computations at each MPC cycle, whereas with a policy learned with deep RL the decision is made with a single forward pass through the network. Figure 3.4 shows a scheme of the proposed control architecture with the added gait adaptation module that is learned to modify the nominal gait G_t with ΔG_t

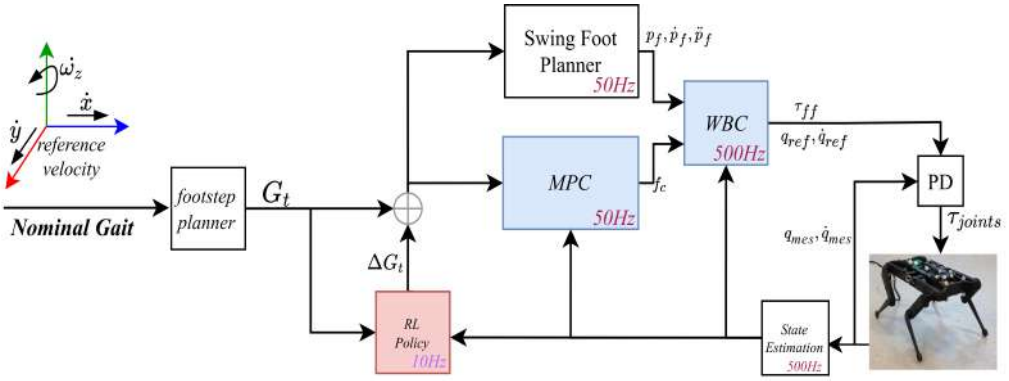


Figure 3.4: Description of the controller. A user commands the reference velocity and gait. The planner outputs a gait matrix G_t which is updated by the RL policy. Considering this updated value, the MPC plans contact forces for the ground feet while the swing feet trajectories are determined by a planner. The information from both blocks are sent to the whole-body control that outputs reference torques and reference joint positions and velocities. An impedance controller finally computes the joint torques based on encoders measurements.

3.4.1 Controlling the Gait Timings

While we could in theory directly control the binary indicators in the gait matrices G_t, this would create an intractable action space with 2^{4M} actions, most of which would not correspond to any relevant locomotion. Since the nature of quadrupedal locomotion is periodic, we propose to view the creation of the gait matrices through parameterized oscillation functions. We define a base oscillation $\bar{f}(t; \tau_0, \tau_1)$ where $\tau_0 < \tau_1$ are the timings for which a change in value occurs:

$$\bar{f}(t; \tau_0, \tau_1, T) = \begin{cases} 0, & \text{if } \tau_0 < t \bmod T < \tau_1, \text{ and} \\ 1, & \text{otherwise.} \end{cases} \tag{3.2}$$

Given $C(j)$, a binary indicator of the current contact state of foot $j \in \{1 \dots 4\}$, the oscillation function $f_j(t) : \mathbb{R}_+ \to \{0, 1\}$ describes the future contact states of foot j as a function of time t. The oscillation is parameterized by two switch timing parameters τ_s^j and τ_c^j, which indicate the beginning of the swing and stance phases of foot j respectively. f_j is then defined as:

$$\begin{aligned} f_j(t; \tau_s^j, \tau_c^j) = \quad & \delta_{1,C(j)} * \bar{f}(t; \tau_s^j, \tau_c^j, T^j) \\ + \quad & \delta_{0,C(j)} * (1 - \bar{f}(t; \tau_c^j, \tau_s^j, T^j)), \end{aligned}$$

where the period length is defined as $T^j = \max(\tau_s^j, \tau_c^j)$ and the $\texttt{mod}$ refers to the modulus operation to indicate that after period T^j the time resets to zero. $\delta_{i,j}$ refers to the Kronecker delta, i.e., $\delta_{i,j} = 1$ if $i = j$ else 0.

In order to control the four oscillation functions we introduce a 4×2-dimensional continuous action space, $\mathcal{A} = \{(a^1, a^2, a^3, a^4)\}$ with $a^j = (\Delta\tau_s^j, \Delta\tau_c^j) \in \mathbb{R}^2$. For each foot j there are two actions that define the displacement with respect to the timings of the nominal trotting gait, $\tau_n^j = (\tau_{s,n}^j, \tau_{c,n}^j)$, that are hardcoded in the controller. Controlling the deltas of the timing values rather than the values directly is the key for the method to work as it reduces the exploration space.

The gait matrix can be created by assigning $G_t[i, j] = f_j(i * \Delta t_{\text{mpc}}; \tau_n^j + a_t^j)$, where $i \in \{1, \dots, M\}$ and $j \in \{1, 2, 3, 4\}$. We will next describe the nature of the state space,

reward function and policy architecture that make the RL agent learn to effectively adapt contact timings.

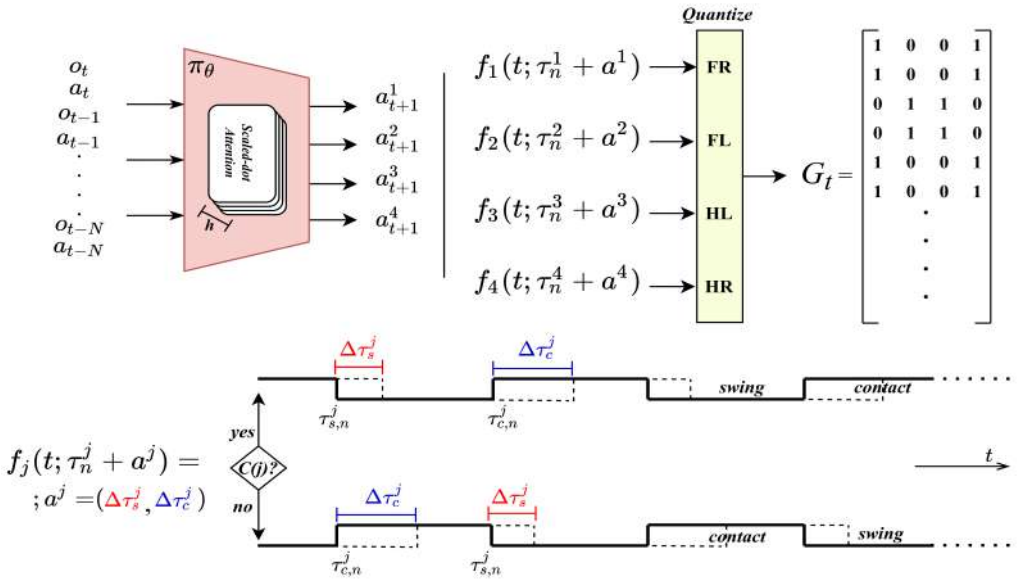


Figure 3.5: Multi-headed self-attention layers, with $h = 8$ heads, are used as the base of the policy. The actions modify the contact sequence given by the oscillations of each leg. The bottom plot shows an example of the nominal sequence of contact/swing in bold. Depending on the current contact state of foot j, the dotted line draws the new oscillation after the shifts in timings given by the policy are taken into account.

3.4.2 MDP Definition

The MDP is defined over fixed discrete timesteps. The RL policy runs at a frequency of 10Hz. We found that this frequency gives the policy enough time for executing an action and receiving a useful learning signal, while keeping its reactivity in adapting gait sequences quickly enough.

State space. We define the observation $O_t \in \mathbb{R}^{d=65}$ to contain proprioceptive information $o_t \in \mathbb{R}^{57}$ about the robot at time t along with the last eight-dimensional command a_{t-1}. The elements composing the observation are the base height and orientation $q_{base} \in \mathbb{R}^4$, base velocity $\dot{q}_{base} \in \mathbb{R}^6$, joint angles $q \in \mathbb{R}^{12}$ and velocities $\dot{q} \in \mathbb{R}^{12}$, feet positions relative to the body frame $p_{feet} \in \mathbb{R}^{12}$, the current and past gait contact sequences (first two rows of G), both four-dimensional binary vectors and the velocity reference command $\dot{q}_{ref} \in \mathbb{R}^3$ which is added to the observation so that the policy is aware of the command. The observation is thus constructed as $O_t = (o_t, a_{t-1})$.

The history of the last $N = 16$ observations are concatenated to construct the state $s_t \in \mathbb{R}^{N \times d}$. We argue that having a history of observations, especially when the period between each RL action is very short, is necessary for decision making in order to detect changes in the environment dynamics.

Reward definition. We designed a basic reward function based on three terms: (1) a positive constant c per timestep to encourage the policy not to commit any actions that would end the episode early, (2) the squared distance between the commanded velocity and the velocity of the robot and (3) an energy penalty term to encourage the policy to learn actions that save energy.

The reward function used in this work is similar to the ones proposed by [Da+20; Yan+21]. However, we propose using the energy instead of the torques magnitudes. As the energy is a function of the torques and joint angles, optimizing the torques magnitudes, while important from a control perspective, does not equate to minimizing the energy under certain joint angles. The energy for joint l at time t is the integration of power P_l spent over the last RL timestep.

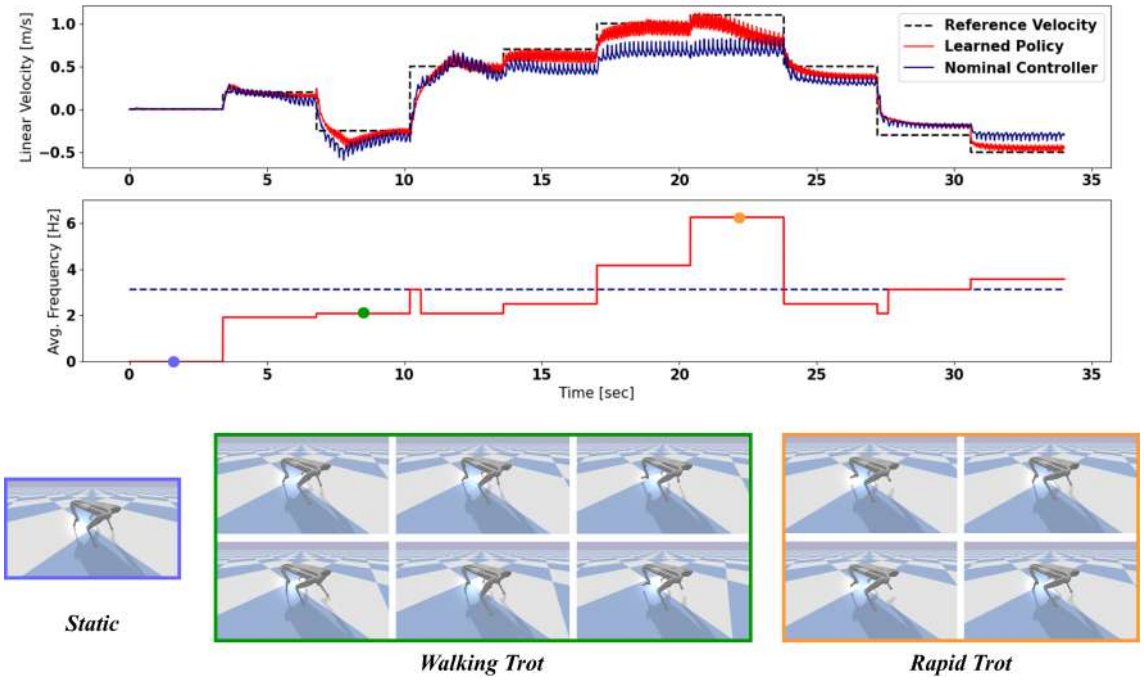


Figure 3.6: Top: achieved velocity of the robot when following a predefined velocity plan (dashed line). The policy is able to get the robot closer to the desired reference in most cases. Middle: the blue dashed line represents the nominal trotting frequency. The red line indicates the average frequency of all legs as adapted by the policy. The frequency of stepping is slowed down and sped-up to accommodate the reference velocity. Bottom: snapshots of the achieved gaits at different levels of the run.

In order to express the energy term, We first defined the instantaneous power of a joint as a function of the motor torque τ_m, joint angular velocity ω and the friction torque τ_f (the additional torque necessary to overcome the friction forces):

$$\tau_f = \tau_u * \text{sign}(\omega) + b * \omega, \tag{3.3}$$

where τ_u is the Coulomb friction term and b is the viscous frequency term. Then the power dissipation due to the friction torque $P_f \in \mathbb{R}^{12}$ is :

$$P_f = \tau_f * \omega, \tag{3.4}$$

and the power loss due to the Joule effect:

$$P_\tau = K * \tau_m^2, \tag{3.5}$$

where K is the motor resistance term. The instantaneous power $P_t \in \mathbb{R}^{12}$ at time t is then obtained as the sum of both terms:

$$P_t = P_f + P_\tau \quad [W]. \tag{3.6}$$

The total power at time t is the sum over all joints: $P_t(\tau_m, \omega) = \sum_{l=1}^{12} P_t(\tau_{m,l}, \omega_l)$. We can calculate the energy for joint l at time t by integrating the power since the start of the episode:

$$E_{l,t} = \int_0^t P_t(\tau_{m,l}, \omega_l) dt \quad [J]. \tag{3.7}$$

The total energy at time t is summed over all joints:

$$E_t = \sum_{l=1}^{12} \int_{t-T_{RL}}^{t} P_{l,\nu} d\nu, \tag{3.8}$$

where T_{RL} is the time period between each RL step and wthe notation $P_{l,t} = P_t(\tau_{m,l}, \omega_l)$ has been introduced for simplification. The reward at time t is then defined to be:

$$R_t = c - \int_{t-T_{RL}}^{t} \|\dot{q}_{ref,\nu} - \dot{q}_{base,\nu}\|^2 d\nu - \lambda E_t, \tag{3.9}$$

where λ is the coefficient that balances the importance of energy conservation in the reward function.

Policy design: In order to take advantage of the sequence of observation-action pairs, we utilize self-attention layers in our policy model [Vas+17]. The state is passed through three linear layers to output a query Q, a key K and a value V. The three outputs are then passed through a multi-headed attention layer where, at each head, they are processed according to the scale-dot product attention formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V. \tag{3.10}$$

The scaling d_k is the dimension of the key. Multiple attentions are applied at $h = 8$ heads. Vaswani et al. [Vas+17] argued that using multiple parallel attentions allows the model to find a variety of information at different representation subspaces of the input. Whereas, using a single attention where the attention is averaged in the softmax inhibits this diversity. The multi-headed attention is followed by dropouts and layer norms. In this work, we found that setting the dropout probability to zero improves the overall performance of the RL training procedure.

Self-attention has been very successfully used in language tasks where the input is sequential. We argue that with self-attention the model can focus on parts of the state that indicate changes in the dynamics. Moreover, it also facilitates coordination of the different legs by contextualizing inputs from one another. In Subsection 3.5.4, we show that using a self-attention model yields greater rewards with less samples than a standard stack of fully-connected layers.

3.5 Results

In this section, we present the experimental results of training a policy to adapt the trotting gait of the Solo12 quadruped with the proposed action space. The main questions we answer are: (1) *can the policy learn to adapt the trot so that the robot tracks the reference velocity while optimizing the energy consumption?* and (2) *What is the effect of using a self-attention mechanism in the policy network?*

Environment. The training process takes place in a synthetic environment. The simulation is based on PyBullet [CB21] that uses the Bullet physics engine for simulating rigid body dynamics and detecting collisions. The Pinocchio library [Car+19] is used for low-level dynamics and kinematics, e.g., to get the positions of the feet in the body frame of the robot.

Implementation details. As mentioned before, a self-attention mechanism is used in the architecture of our policy-network. We use the encoder layer of the transformer architecture as our base model [Vas+17]. The output of the encoder is propagated through a multi-layer perceptron that outputs the actions. The actions are quantized into a Multi-discrete space. As the contact patterns are quantized over the timestep of the MPC, the shift in the timings are multiples of the MPC timestep. Therefore, the action space is implicitly discrete. The two switch timings for each leg has six delta possible values (-0.12,-0.08,0.04, 0.0, 0.04, 0.08). The neural network outputs a log-probability for each possible delta for the two switch timings of each leg. This design avoids the combinatorial explosion we get if we were to designed the action space as purely a discrete one with $6^8 = 1679616$ possible actions.

The encoder layer of the transformer [Vas+17] is the base model. The input to the encoder is a batch of $B \times 16 \times 65$, where $B = 512$ is the batch size used in training. The output of the encoder is flattened and fed to a multi-layer perceptron with two hidden layers each with 512 neurons with hyperbolic tangents '`tanh`(.)' as the activation function.

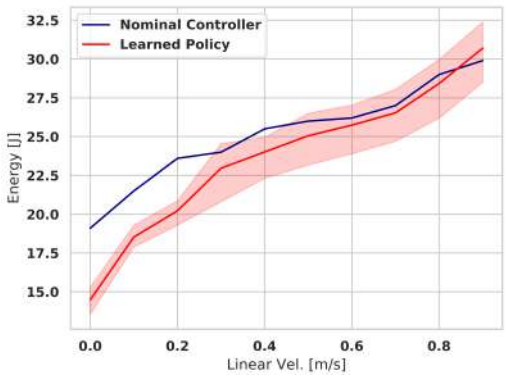
We use the Proximal Policy Optimization algorithm (PPO) [Sch+17] for learning the optimal policy (see section 2.2). We found that pretraining the policy representation on a torque prediction task improves the overall performance of the RL. The Adam optimizer is used with an initial learning rate of 2.5×10^{-4}. The PPO clip ratio is 0.1. An entropy term to encourage exploration is added to the PPO loss with a weighting on 0.01. The generalized advantage method is used to calculate the advantages of the samples. The model is designed with an actor-critic setup. Therefore, along with the policy action layers, a single output to predict the value of the samples is added.

Throughout the experiments, to calculate the energy terms in the reward function (Equation 3.8), we relied on the following parameters: $\lambda = 10$, $c = 1.0$, $K = 4.81 Nm.s$, $\tau_u = 0.0477$[Nm] and $b = 0.000135$[Nm.s]

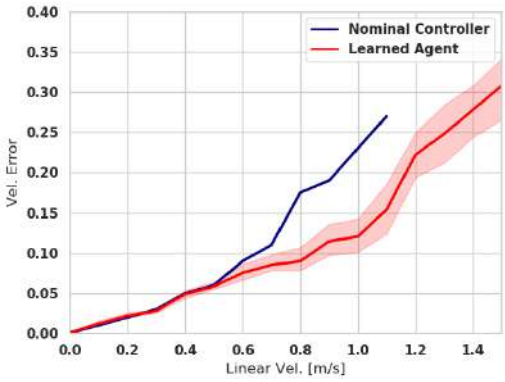
3.5.1 Adapted Gaits and Velocity Tracking

Our simulation results demonstrate that we can learn a policy that adapts the nominal trotting gait for different reference velocities. At zero velocity, we obtain an optimal energy saving policy with a static gait where all feet are in contact with the ground. As the commanded velocity increases the gait evolves into a walking trot at low velocities. At high velocities the policy adapts the timings to output a fast rapid trotting which is more costly in terms of energy, but necessary to follow the velocity command with low error. Examples of the resulting gaits, due to the policy adaption of the nominal trotting, are shown in Figure 3.6.

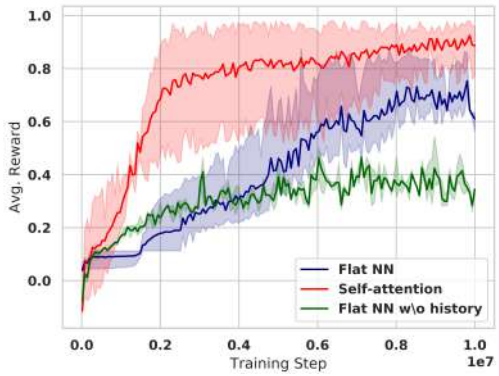
The learned policy shows higher fidelity in tracking the commanded velocity at each moment. In the middle figure, we observe a clear decrease in the frequency, making the stepping slower, for lower velocities. For high velocities the frequency is increased, thereby making the stepping faster which helps stabilize the base and follow the reference velocity.



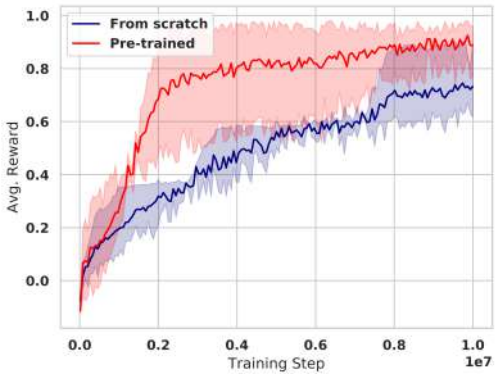
(a) Energy consumption.



(b) Velocity error



(c) Policy architecture.



(d) Pre-training.

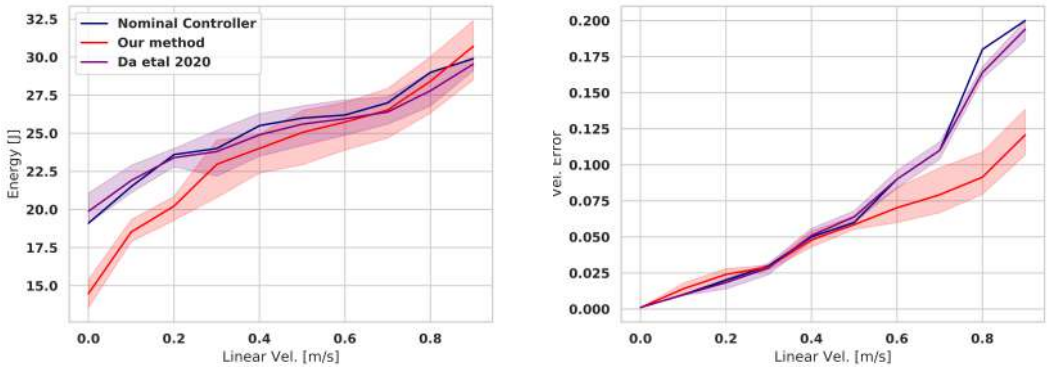
Figure 3.7: Figures (a, b) are comparisons between the learned policy and the nominal controller at test time over an increasing forward velocity reference. (a) Plot of the average energy consumption per episode. (b) Plot of velocity error. The nominal trotting fails after 1.1 m/s. The training curves in (c,d) are averaged over five random seeds. (c) Average reward for different policy architectures. (d) Effect of pre-training on a torque prediction task.

Maintaining Stability at Higher Speeds

We found that the controller with the nominal gait can reach the forward velocity up to 1.1 m/s before failing and falling (see Figure 3.7b). The proposed learned policy was able to break that limit and achieve velocities up to 2.5m/s. The experiments were conducted by gradually increasing the reference velocity over 2.0m/s. This implements a simple linear curriculum over the difficulty of the task. The result is a very rapid trot with period around 0.12s-0.16s.

3.5.2 Energy Efficiency

We run the learned policy five times with different random seeds in a setup where the reference velocity is gradually increased starting from 0 m/s. Figures (3.7a, 3.7b) show the trade-off between the average episodic energy consumption and velocity error for the learned policy vs. the nominal controller with fixed gait. We observe the improvement in energy consumption particularly at low velocities.At higher velocities, the velocity error is lower for the learned agent while still being comparable in energy efficiency. The lower velocity error is due to the policy learning a rapid trot that tracks higher velocities better.



(a) Energy consumption.

(b) Velocity error

Figure 3.8: Plot of the episodic energy consumption and average velocity tracking error over the commanded linear velocity. The comparison is between the nominal controller, our proposed method and the method by Da et al. [Da+20]. Their policy is limited by having the decision over only the next contact phase and its output is similar to that of the nominal trotting. Our method is more suited to slow the gait down (and make it static at low velocities) as it can adapt the entire stepping pattern.

At high velocities the energy consumption of the nominal controller and of the learned policy appear to be almost equal. However, it is important to note that Figure 3.7a is plotted as a function of the reference commanded velocity and not the actual velocity of the robot. As shown in Figure 3.7b, the nominal controller exhibits higher velocity error when the energy consumption is similar to the policy. Figure 3.6 (top) confirms that at high velocities the robot controlled by the nominal controller is slower than when controlled by the learned policy. Therefore, while the energy consumption is comparable for high reference velocities, for the actual realized velocities the robot controlled by the learned policy consumes less energy than the one controlled by the nominal controller.

3.5.3 Comparison with Related Work

The paper by Da et al. [Da+20] proposes using RL to choose the next contact sequence to be executed by the controller. This is in contrast to our approach where we proposed to adapt the entire gait by modifying the flying/contact phases of each foot independently. The authors propose an action space consisting of nine discrete actions, where each action is a four-bit binary vector representation of the ground contact of each foot [Da+20].

We design a method close to their proposal, though not the same as we have a different controller. We employ the same setup in terms of state definition and reward function both for the Da et al. [Da+20] baseline and our experiments. The only difference is that we use the energy as the reward penalty while they use the torques. The action space is discrete with nine actions indicating the choice of contacts for the next four MPC timesteps. The nine actions, as proposed in Da et al. [Da+20] correspond to:

$$A = \{\text{static} : [1,1,1,1], \text{walk1} : [1,1,1,0], \text{walk2} : [1,1,0,1], \\ \text{walk3} : [1,0,1,1], \text{walk4} : [0,1,1,1], \text{pace1} : [1,0,1,0], \\ \text{pace2} : [0,1,0,1], \text{trot1} : [1,0,0,1], \text{trot2} : [0,1,1,0]\}.$$

Figure 3.8 shows the trade-off between the average episodic energy consumption and velocity error for this comparison study. We see that the baseline [Da+20] does not offer

much improvement over the nominal controller. Running the learned policy for 20 test trials each with 100 steps, the average frequency of each action is 42% trot1, 43% trot2, 8% walk1 and 7% walk2. Overall, the method learns to output the nominal trot of the controller. We argue that our method is more general and the policy has the ability to adapt the entire stepping pattern and frequency of movement and not just the next contact sequence. Further, using the method by Da et al., there was no benefit in terms of energy consumption nor velocity tracking with respect to the baseline method.

3.5.4 Ablation Studies

In this section, we present ablations of the policy architecture and of the method of pretraining the representation using torque prediction. We illustrate the advantages of using our proposed method over standard approaches. Throughout this section, we use a task where the agent is simply expected to learn how to stand still and balanced. The agent is rewarded for completing the task successfully without falling and it is penalized for its energy consumption and velocity tracking error.

Policy architecture. We train a policy using three different setups: (1) The proposed self-attention based policy, (2) a feed-forward neural network policy where the input is a sequence of $N + 1$ last states, and (3) a feed-forward neural network policy without history of the last N states. Figure 3.7c shows the expected cumulative reward during the training process for each setup. By using self-attention we attain higher rewards with less training steps than using a standard neural network. For the zero reference velocity, the proposed policy yields a static gait with all feet staying in contact with the ground thus conserving energy. The other policies are not able to reach such a solution in the given number of training iterations, but they settle for a dynamic gait. This explains their overall lower cumulative reward.

Pretraining for torque predictions. Pretraining neural networks to improve the overall sample efficiency and performance of the model is a common practice in computer vision and NLP. Therefore, we propose pretraining our model to predict the torques using a fixed gait from the standard controller. We then use the pretrained model to initialize the policy network for learning the gait timing control task. Figure 3.7d illustrates the improvement in sample efficiency when using pretrained initialization. The overall asymptotic performance, in terms of average reward, is also higher. Without the pretrained representation many experiments with different seeds failed to converge. The pretrained representation stabilizes the training process and decreases the number of failed training runs.

Studying the role of history in state. In order to produce robust policies that can detect changes in the environment and react accordingly, a history of observations should be considered. Instantaneous proprioception is not rich in information about changes that happens between small timesteps. Therefore, we consider a sequence of observation-action pairs in the state of the MDP.

To study this hypothesis, we propose to train a neural network on predicting the change in the slope of the ground as a function of the state. This way we can explicitly test the ability of the network to learn to predict environment events relevant for robust control. We experiment with three setups: (1) using only the instantaneous proprioceptive observation as input, (2) using the last five observations, i.e., at the last five timesteps and (3) using the last ten observations. To make the prediction task challenging, we remove the under-actuated state of the body of the robot (height, roll, pitch, yaw) to see if it is

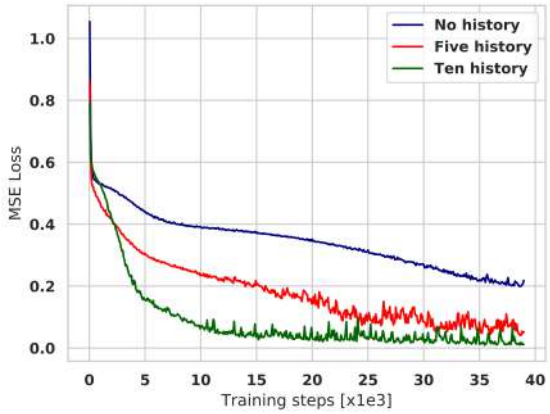


Figure 3.9: Average MSE Loss when training to predict the slope of the ground. The curves use different length of history of the proprioceptive readings. Looking farther back in history allows the network to predict the change in the ground floor better. The curves are averaged over five seeds for each experiment.

still able to predict the slope using only states of the joints and feet. A dataset of states/-ground slope tuples is collected by running the model-based controller, with the nominal trotting gait on a tilted floor with a random slope angle. The dataset contains around 11K interactions. The size of the features of each sample depends on the length of history used.

The models are trained for a regression task with the mean-squared error loss. Results are shown in Figure 3.9. The curves show the mean training loss averaged over five seeds for each experiment. We see a clear improvement in minimizing the prediction loss when using an increasing length of history over using no history at all. Having a wider bandwidth of history of proprioception readings allows for better understanding of environment circumstances.

Discussion and Limitations

This project studied the possibility of modifying the gait of a quadruped to achieve patterns that resemble the ones seen in animals such as walking, trotting and galloping [HT81]. The result was a policy that was able to adapt the period of the trotting gait to be suitable with magnitude of the velocity command. We were able to quantify, in our simulations, that the gaits adapted by the policy do improve the overall performance by changing the frequency of the gait. However, the policy was not able to output diverse and rich gaiting patterns (other that trotting) that we were interested in observing. The final results were underwhelming since the adaptation of the frequency could be achieved witha a simple linear function of the velocity command. We did not find the results interesting enough to deploy on the real robot. We believe the approach for this problem was not suitable for emergence of different gaits.

The model-based optimal controller proposed in Leziart et al. [Léz+20] consists of many parameters most parts of which are hand-tuned together to output an optimal trotting gait. Therefore, modifying only the gait parameter yields suboptimal performance. Thus, the learning was always reverting back to the trotting gait as any other pattern would not be executed well by the model-based controller.

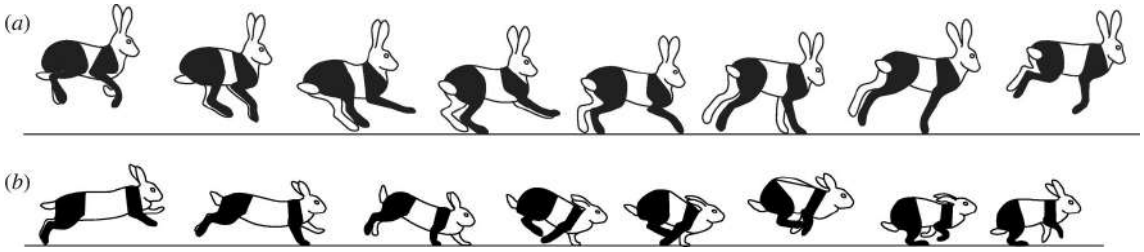


Figure 3.10: Illustration of galloping gait and trotting gait of a rabbit. Notice the difference of extension and flexion of the body of the rabbit for the two gaits. Figure extracted from [BG08].

Another problem with the approach is the gait definition. The gait is not only the pattern of feet contact with the ground. Bertram & Gutman [BG08] show that the galloping and trotting of rabbits and dogs are complex mechanical procedures that require specific extension/flexion of the torso of the animal along with its feet (see Figure 3.10). In our approach, the gait definition is poor and the 12-dof robot's design is not as complex as it is for animals.

3.6 Conclusion

The first part of the thesis work focused around studying a model-free RL method that adapts the nominal trot pattern of a model-based controller designed for the locomotion of the Solo quadruped. The proposed action space consists of displacements of the contact/swing timings for each leg independently. We propose the use of a self-attention mechanism and pretraining the network to improve overall performance and sample efficiency. Our simulation results demonstrate that the learned policy is able to adapt the gait according to the reference body velocity by changing its period. The resulting behaviour conserves energy better than the nominal controller at low velocities and tracks the commanded speed better at high velocities. With the learned policy our robot is able to reach higher speeds than with the original controller.

Unfortunately, we observed that the margin of improvement of the proposed approach was still small and limited. In the proposed approach the capabilities of the learned policy were limited to modify the gait pattern. Therefore, we decided that for the next project to explore developing RL policies that directly control the joint angles of the quadruped robot in order to explore how well a learned policy can perform without the presence of an optimal controller.

Chapter 4

End-to-End Learning of Locomotion

Contents

4.1 Introduction

In the previous chapter, we found that the combination of learning the gait pattern in the model-based controller produced limiting results that prevented us from getting diverse gaits that are executed optimally by the optimal controller. To that end, we decided to move more towards end-to-end learning for locomotion. The main motivation was to attempt to learn a reactive policy from scratch to study the resulting behaviour with limited constraints. We later found, as in previous work [Hwa+19; Lee+20; Mik+22], that the RL process still requires the presence of many of the constraints and the costs that are present in model-based control [Kim+19] to output meaningful motions that can work in the real-world. However, the constraints are present as reward terms and do not require

online optimization as in the optimal control approach. Learning allows us to include the constraints to regularize the behaviour towards what is desirable, while the policy still maintains the ability to violate the constraints for the sake of robustness in certain situations.

Outline. In this chapter, we present our effort in designing and studying controllers that are based on deep RL. We show an approach for learning robust controllers on the Solo12 robot [Léz+20]. We outline the full process that allowed us to reach a stable baseline that outputs a robust joint controller for Solo-12. We use similar RL techniques for learning locomotion while introducing curriculum processes at different levels and randomization schemes for zero-shot transfer to the real robot. We detail our procedure for setting up the MDP components, i.e., state space, action space and reward function, along with the additional techniques required to make the learning converge and transfer to the real robot. Additionally, we introduce and investigate a realistic energy loss penalty, incorporating actuator friction and Joules losses identification, enhancing the policy learning process. We conducted extensive testing of the learned locomotion techniques on the real Solo12 quadruped, both indoors and outdoors, validating their effectiveness. Subsequently, our focus shifts to presenting our research on acquiring robust control policies for MIT's Mini-Cheetah robot. We discuss the contrasting aspects between the Mini-Cheetah and Solo-12, particularly regarding the robot's mass and the significantly amplified torques it produces. We then proceed to illustrate the methods employed to adjust the learning process initially designed for Solo-12, allowing the learning of robust controllers that can transition to the Mini-Cheetah.

In the next section, we review the literature related to quadruped locomotion and then show how the RL approach was developed and adapted for real world experiments on two quadrupeds.

4.2 Learning Approaches for Locomotion: A Literature Review

In contrast to model-based optimization methods, data-driven methods that are based on learning can be used for designing controllers. Specifically, reinforcement learning (RL) is an alternative approach for obtaining highly performant agents that act in their environment in which the dynamics and transitions are modeled as a Markov decision process (MDP) [SB18]. There are many early examples of applying learning based methods to robotic tasks such as manipulation [Gul95; PS08; Kob+10; Kal+12]. In locomotion, Benbrahim's thesis [BF97] in 1997 explored the use of reinforcement learning in bipedal locomotion by designing a model-free approach coupled with central pattern generators (CPGs) which implements walking motions on a real physical biped. Kohl and Stone [KS04] proposed to optimize trotting gait of the Sony Aibo ERS-210A quadruped with policy gradients methods. The gait was a hand-designed sequence of joint angle positions in each leg and and the RL policy was used to offset those joint angles to produce a faster walk.

Learning robust end-to-end policies. However, RL used to be hard to scale and was often limited to solving small sub-problems in the control pipeline in which most of the components were hand-designed. With increased computing power and recent evolution

of deep learning methods that use large scale neural networks, we can now solve problems requiring high-dimensional data [LeC+98; KSH12; LBH15]. Deep RL combines neural networks with RL algorithms to learn value function approximations [Mni+13; Mni+15; Kou+13] and/or, directly, policies [LK13b; Sch+15; Sch+17]. In the recent literature, deep RL has been used for quadrupeds [Hwa+19] and bipeds [Li+21] for the purpose of learning full end-to-end controllers. The seminal work conducted in Hwangbo et al. [Hwa+19] was the first to outline a general RL procedure for learning joint angle controllers from the base and joint states of the robot. The paper outlines a clear plan to learn a direct policy with TRPO [Sch+15] by defining the state, action and the reward structure that would inspire later works. The authors also propose learning a model of the actuation dynamics of ANYmal [Hut+16] from real-data that can then be deployed in simulation, thus enabling the learned policies to transfer to the real-world. The actuator model was essential for ANYmal since it has parallel elastic joints [Hut+16] that are hard to simulate. On our work, we did not find the need to add the actuator model to the simulation, but we took inspiration from the end-to-end RL procedure.

Lee et al. [Lee+20] proposed learning a policy that modifies the phase and shift of CPG functions that determine the foot trajectories which are fed to model-based controller to produce joint angle control. The method also utilize exteroceptive input from a LIDAR to reconstruct the surrounding environment around the robot. The resulting locomotion has impressive robustness and can traverse various terrains of stairs to steep mountains because of the added vision. Miki et al.[Mik+22], deployed a similar learning scheme and augmented the action space with a CPG layer that produces a baseline walking gait pattern for the feet. This work also employed a belief-based strategy that couples a LIDAR with the proprioception to infer the reconstruction of the environment. The policy then learns to manipulate the CPG phase and joint angles to modify the gait. The resulting controller can perform long hikes on mountain terrains in a very robust manner. While the action space in both papers differs from the one originally proposed in Hwangbo et al. [Hwa+19], they are still considered end-to-end learning approaches as they can manipulate the joint actions indirectly. Rudin et al. [Rud+21] summarized the end-to-end learning methods in the legged-gym repository and open-sourced the implementation which is based on the IsaacGym simulator [Mak+21]. They employed a massive number of parallel agents that makes learning locomotion policies possible in record time.

On the Mini-Cheetah robot, Ji et al.[Ji+22a] proposed concurrent learning of a control policy through RL and a state estimation network with supervised learning that tries to predict state variables that are not measured on the real robot but are available in simulation and provide vital information for learning robust policies, e.g., feet contact states and linear velocity of the base. The results are agile policies that can produce a trotting gait with forward base velocity of up to 3.7 m/s, which at the time of the publication was the record for Mini-Cheetah. However, the work by Margolis et al. [Mar+22] outlined a curriculum learning strategy that produced much higher velocities for Mini-Cheetah (around 3.9 m/s for forward velocity and 5.7 rad/s for angular yaw velocity). The authors did not use the same state estimation strategy as in [Ji+22a] but instead used rapid motor adaptation (RMA) [Kum+21]. In our work on Mini-Cheetah, we later discuss how we chose to use RMA for the transfer of policies.

These previous works [Hwa+19; Lee+20; Mik+22; Ji+22a] mostly rely on similar domain randomization techniques that add noise to the sensory input of the policy and to the dynamics of the simulation in order to learn policies that transfer to the real system. Rapid motor adaptation (RMA) presents an alternative method for transfer by adding an adapta-

tion network to the training architecture [Kum+21]. The original policy is initially trained in different simulated conditions by varying ground friction, payload, motor strength, etc. In this first learning phase the algorithm also constructs a compact latent descriptor of the relevant aspects of these different conditions. In the second phase of learning the system learns an adaptation network that estimates this latent condition descriptor using only the history of measurements available in the real robot. This ability to adapt to different conditions also compensates for discrepancy between simulated and real settings [Kum+21].

Learning approaches on Solo-12. For the Solo-12 robot, we were the first and so far only ones that explored employing and end-to-end learning strategy to learn velocity tracking controllers for this locomotion. Recent work proposed to learn different skills for the Solo-8 robot [Gri+20] through imitation learning [Li+22a; Li+23a; FXP22]. These strategies rely on generating trajectories of different skills and motions through a trajectory optimization approach and utilize supervised learning techniques to mimic the generated sequences. In our approach [Ara+23b], we focus on using RL to learn robust end-to-end controllers from scratch for the Solo12 robot.

Coupling vision and locomotion. Other works focus on adding vision to locomotion to improve the robustness and performance on complex structured terrains such as stairs, steps, rough and slopes. Margolis et al. [Mar+21], proposed Depth-based Impulse Control (DIC), a method that combines model-free learning with explicit model-based optimization of ground reaction forces from depth input. The method maps the robot state and vision to a whole body trajectory that is then tracked by a model-based trajectory tracking. Agrawal et al. [Aga+22] learned locomotion with added input from heightmap scans that are available in simulation and proposed a distillation approach to predict the latent representation of the heightmap from a sequence of depth images taken from an egocentric camera aboard the robot. Loquercio et al. [LKM22] proposed to learn visual representation of the terrain from real RGB images recorded by running the robot in the real-world. Both approaches show that the robot can adapt its locomotion purposefully to cross a variety of terrains.

Quadruped tasks. In general the locomotion task is to follow a commanded base velocity in a robust manner. Recently, more work emerged around using quadruped robots to perform different tasks. Some papers proposed the task of learning dribbling skills of a football with a quadruped [Ji+22b; JMA23].In Ji et al. [JMA23], instead of following the command velocity the quadruped is tasked to track the football and dribble it around by perceiving it using a body-mounted camera. The resulting skills are a mix of locomotion and manipulation. Cheng et al. [CKP23] propose to use the feet of the robot as direct manipulators to push specific buttons or reach certain positions on the wall. Fu et al. [FCP22] added a three DOFs arm to the Unitree Go1 quadruped to perform mobile manipulation tasks. They proposed to split the action space of the arm and legs while also splitting the reward function advantages in the PPO objective to separate terms. The result is a policy that can perform whole body control of both legs and the arm simultaneously. On the other side, recent work attempts to couple the velocity tracking locomotion with image input to learn dedicated locomotion over complex structured terrains.

4.3 Learning to Control Solo-12

4.3.1 Designing an RL-Based Solution for Locomotion

In this section, we will elaborate on the process of creating a learning method for controllers that can be effectively implemented on the real Solo-12 robot. This will involve tracing its evolution from the initial unsuccessful attempt to the ultimate successful method. The general RL problem follows the same formalization that was presented in Section 2.2.

First attempt: learning direct torque control

As a first attempt to learn a policy from scratch, we devised a simple MDP with a basic reward based on the velocity tracking and energy expenditure. We used PyBullet [CB21] as the physics engine and experimented with different RL algorithms such as PPO [Sch+17] and TD3 [FHM18]. The policy π_θ was a multi-layer perceptron parameterized by θ. The purpose was to learn parameters that can map the states to the proper torque commands τ:

$$\tau_t = \pi_\theta(s_t), \tag{4.1}$$

to achieve the locomotion. The experiments were conducted on Solo-8 [Gri+20] so the the torque vector here is 8-dimensional.

In this work, We attempted to take a learning perspective around locomotion and focus more on the high-level task rather than the control with three different policies as a way to explore basic deep RL techniques on quadrupeds. We setup three different tasks:

1. **Stand still:** Basic task to stand in place at zero velocity, the reward function was based on maintaining a certain base height with a joint angle pose penalty and torque penalty. The base height reward would be a binary indicator related to whether the height of the body of the robot exceeds a certain threshold δ_z or not:

$$r_{base} = \begin{cases} 1, & \text{if } z_{base} > \delta_z \\ 0, & \text{otherwise} \end{cases}, \tag{4.2}$$

   and the total reward would be:

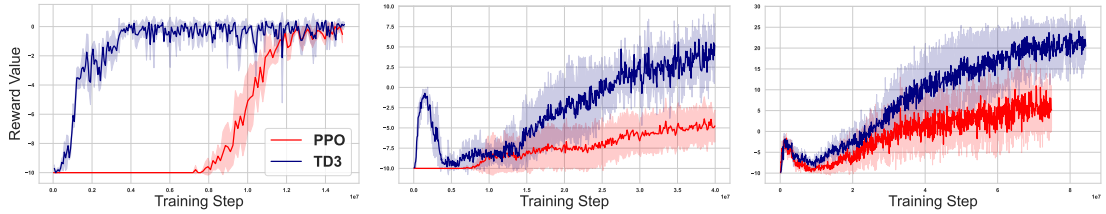
$$r_t^{stand} = r_t^{base} - 0.1\|q_t\|_2^2 - 0.0001\|\tau_t\|_2^2. \tag{4.3}$$

   The purpose of the second term is to penalize the joint pose q_t when the legs are not straight.

2. **Walk forward as fast as possible:** This second task involves walking forward with the highest possible forward velocity v_x. This task reward was built on top of the previous standing task reward by adding a progress reward on the forward velocity of the base along x:

$$r_t^{walk} = r_t^{stand} + \text{sign}(v_x)v_x^2. \tag{4.4}$$

   The aim is to learn policies to move forward at high speed while still maintaining an upright pose.



(a) Stand in place. (b) Walking. (c) Navigate towards a point.

Figure 4.1: Training curves for the three tasks that Solo-8 was trained to complete. The blue curves are simulation experiments trained with TD3 and the red curves are simulationexperiments trained with PPO.

3. **Navigate to a certain X,Y coordinate:** The purpose of this third task was to make the robot move around in different directions without specifying the desired velocity. This task is defined by specifying a point goal $G = (x_g, y_g)$ in the world's coordinates, that the robot should reach, and given its own coordinates $B = (x_{base}, y_{base})$. As before we build on the standing reward task by adding a progress reward based on how close the robot is to the goal point:

$$r_t^{progress} = - \mid \|G, B_t\|_{l_2} - \|G, B_{t-1}\|_{l_2} \mid, \tag{4.5}$$

$$r_t^{pointgoal} = r_t^{stand} + r_t^{progress}, \tag{4.6}$$

where l_2 is the squared Euclidean distance between two vectors.

The tasks and rewards are devised in a very simple manner. The state is composed of the vector of joint angles and their velocities along with the IMU state which consists of the orientation and 6D velocity of the base.

Results. The resulting policies were able to perform the tasks and maximize the rewards. Figure 4.1 shows plots of the mean rewards of the three tasks, when trained with PPO (in red) and when trained with TD3 (in blue), as a function of the training timesteps. The curves are averaged over five-seeds of these experiments in order to show the consistency for the different training runs. First, we notice that for all tasks, the two algorithms seem to improve on the initial random policy as the average reward rises with the training step before plateauing at some point. We also notice that the TD3 algorithm seems to learn much faster than PPO, for the standing task in particular. This is not surprising as off-policy algorithms are expected to have better sample efficiency [Mni+15; Wil92]. Therefore, we conclude that we are able to device algorithms to control the Solo robot in a way that maximizes the simple proposed rewards.

When running the policies in simulation in PyBullet we observe that the behaviour of Solo-8 is not natural in the sense that the resulting motion looks counter intuitive and not optimal in terms of energy saving. Figure 4.2 shows a rollout of the policy that learns the standing task. A simple task like standing in place should be learned quickly as there is only one motion involved. However, the robot struggles to keep its feet fixed on the ground and keeps moving in place. Looking at the training curves in Figure 4.1, it appears that the reward is maximized, but the performance on the robot lacks stability and consistency. Figure 4.3 also shows rollout of the walking policy, we do not observe a regular pattern in the motion as one can observe on animals [HT81]. We see strange joint angles and different flexion/extension for each leg. Finally, Figure 4.4 shows two trajectories for two different policies of the point goal navigation task. The red area is

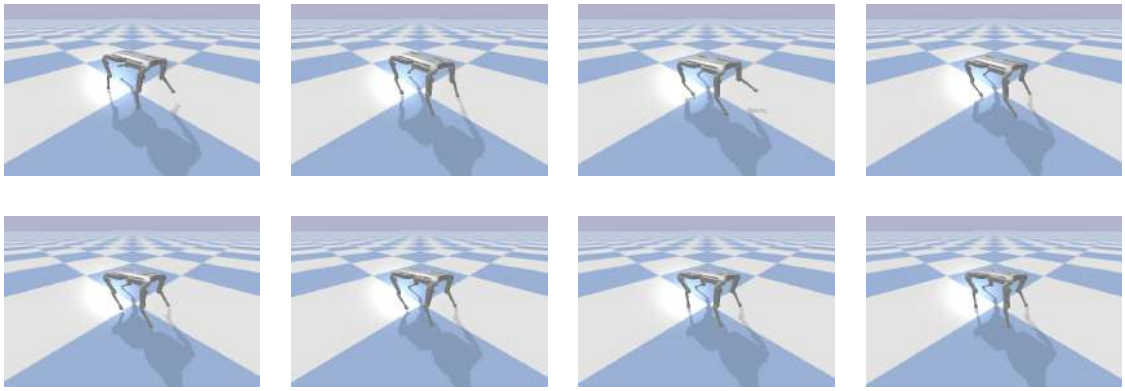


Figure 4.2: Snapshots of a standing task learned by a policy on Solo-8. We see that the robot's leg are consistently moving and never fully fixed on the ground.

the location that the robot has to reach. The first trajectory shows the robot attempting to perform a backflip and jump towards the goal point. Whereas the second trajectory shows that the robot attempts some locomotion towards the goal point.

This contradiction between having bad performance while achieving good rewards has to be attributed to a few main points:

1. The action space: Looking at Figure 4.2, the robot fails to stay in place even though the task is simple, and therefore, the action is expected to be simple. However, since we have direct torque control, a zero action or *stay in place action* is not possible as zero torques means the motors stop working and the robot collapses. This causes the policy to learn a complex function for a simple task.

2. The reward function: Intuitively, if the reward function is maximized and the performance is still not satisfying, then the reward function is badly defined or missing some elements [NHR99].

3. The learning algorithm: At a first glance, it seems that TD3 is the better algorithm to use, however, we found that while it achieves better performance that PPO, the resulting policies from different runs can be quite different. The two trajectories shown in Figure 4.4 are from two different policies trained with the same TD3 implementation (the two approaches for learning to solve the point goal task are vastly different). Therefore, for the rest of the thesis we decided to work with PPO since it always gave consistent output between random seeds.

We observe that the policies are able to reach a good performance from the perspective of reward maximization. However, it seems that the learning reaches a performance that only suits simulations and does not produce efficient and regular patterns that might be safely transferred to the real robot. It is therefore important to shape the reward function and proposed training process to output periodic and more regular locomotion behaviours.

On the difficulty of learning torque control.

In addition to the reward function and the constraints in the control, the action space plays a big part in the learning problem. Peng et al. [PP16] showed how the choice of the action space effects the ability and the efficiency of learning a control problem

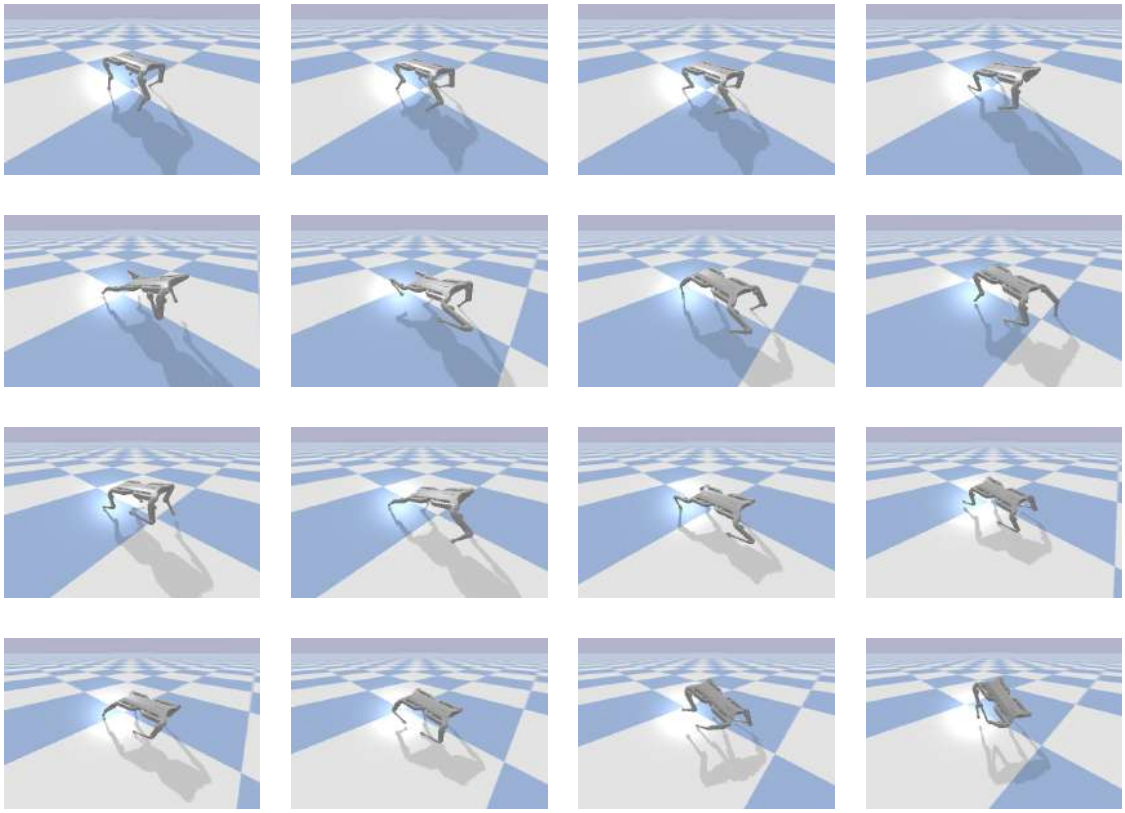


Figure 4.3: Example of the walking task on Solo-8. Notice the inconsistent joint angles between different legs and for different periods of the run. It is hard to deduce a pattern and periodicity as one can do for animals. The observed locomotion also extends the legs to poses that are not safe.

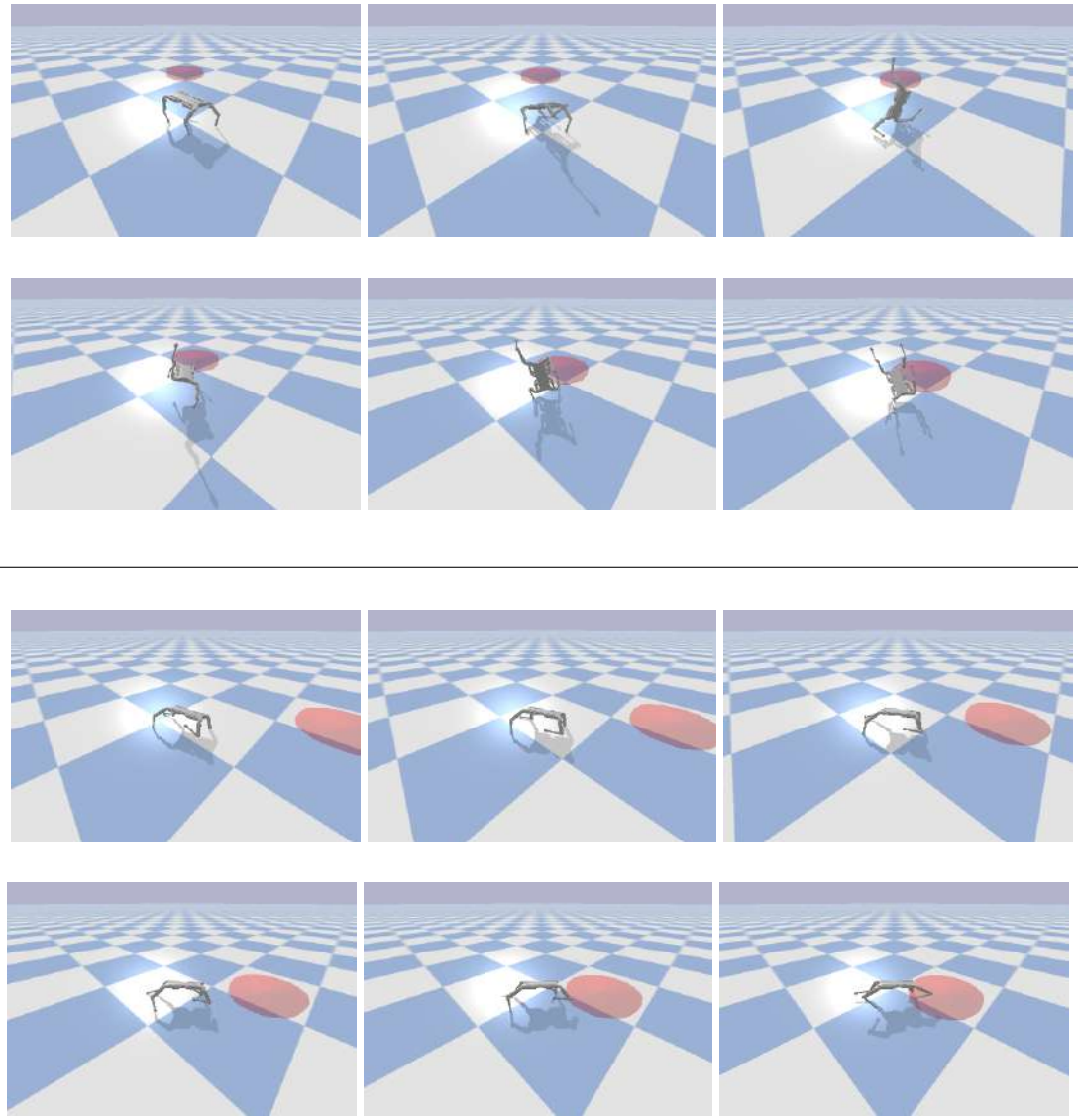


Figure 4.4: Snapshots of two examples of the point goal task showing that learning the same task can be solved in different manners. The first policy (top) learns to jump and throw the body of the robot toward the desired goal, while the second policy (down) learns to walk toward that point. In both cases the performance seems sub-optimal and dangerous.

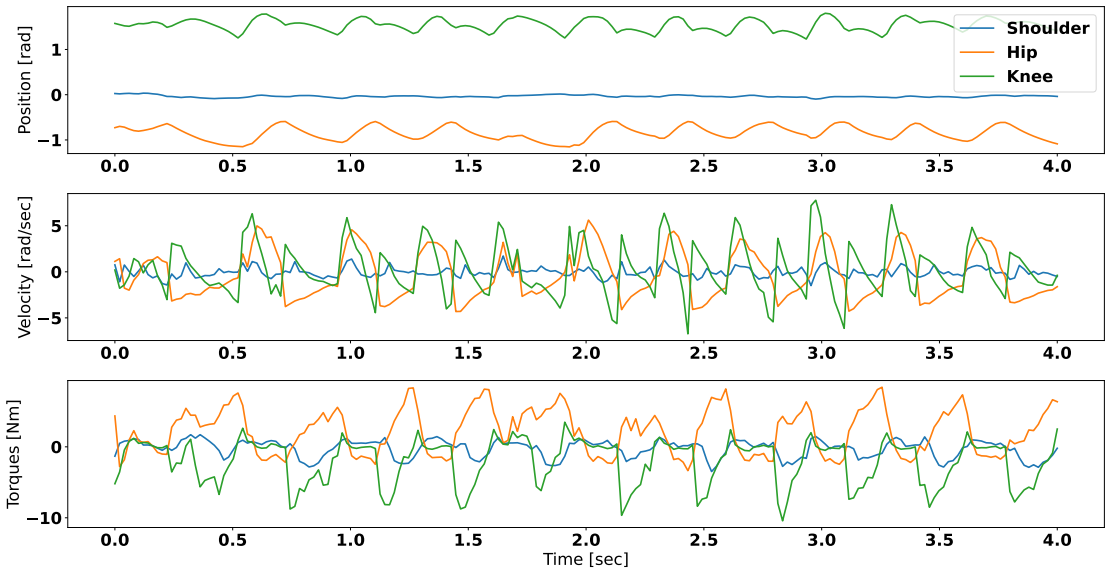


Figure 4.5: Snapshots of sequences of the learned policies for each task.

and the performance of the final policy. Figure 4.5 plots the proprioception data of one leg (three joints) of the mini-cheetah robot when it is moving forward at 1.0 [m/s]. The first row plots the joint angles, the second row plots the joint velocities and the third row plots the torques. We can see how the joint angles of the shoulder, hip and knee joints resemble a periodic, almost sinisoudal, wave, whereas the velocity and torques are more complex and appear to be harder to fit. This is one intuition of why learning torque could be hard. Another reason is the frequency the torque needs to be applied to the motor. Quadruped robots are quite agile and require high-frequency commands. If we control the joint angle control we can lower the control frequency of the policy and leave a low-level proportional-derivative (PD) controller to output torques at higher frequencies. Learning directly the torques would require to perform learning in high-frequency loops where learning might struggle to observe a clear difference in the state transitions.

The next section will discuss the reward shaping and other learning techniques that were required to get efficient policies that can transfer to the robot.

Learning joint angle control

In this section we descrive the MDP elements and the proposed training process that were designed to produce policies that output the joint angle targets for the PD controller that regulates the torque commands of the low-level control. The chosen elements address the shortcomings of the first attempt of learning directly the torque commands.
Our goal is to define an RL method that can learn to control a Solo12 robot to follow a user-defined velocity command. We will describe the design of our state space, action space and reward function in the following paragraphs.

In general, our control policy is implemented as a neural network that takes the state as input, and outputs the actions. The actions that define joint angle targets are then fed to a PD feedback controller in order to get the desired torques for commanding the robot joints. Figure 4.6 shows a summary of the control scheme in terms of the inputs/outputs of the control network and how it is deployed on the real robot. The estimation network in Figure 4.6 is trained with supervised learning to predict the linear velocity of the base.

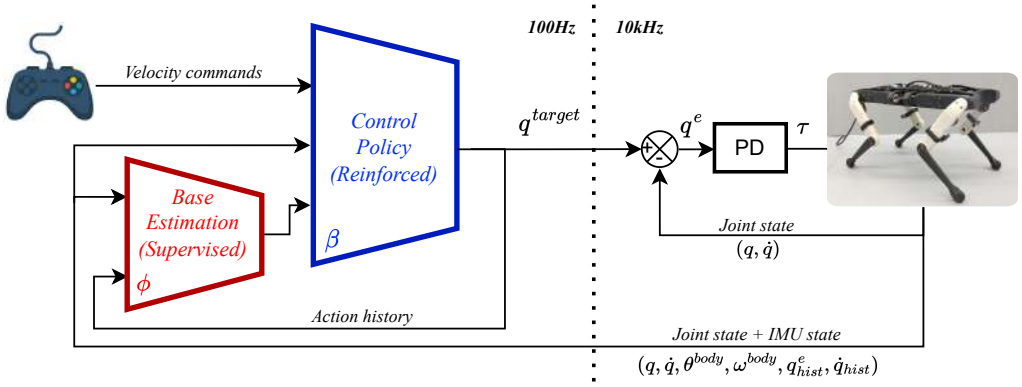


Figure 4.6: Description of the control scheme that includes the policy network, the base estimation network and the low level control setup on the real robot The control policy receives a desired 3D velocity to follow. Using the linear velocity prediction from the base estimation network and other state values measured by the robot's sensors, the control policy outputs joint angle displacements with respect to a nominal joint angle configuration.

The control policy parameters are optimized using the proximal policy gradients objective (PPO) [Sch+17].

State space

The state space of the MDP is constructed from the proprioception of the robot, i.e., the sensory readings from the joint encoders, and the inertial measurement unit (IMU). The state at time t includes the base state and the joint state. The base state consists of the orientation $\theta_t^{\text{body}} \in \mathbb{R}^3$, linear velocity $v_t^{\text{body}} \in \mathbb{R}^3$ and angular velocity $\omega_t^{\text{body}} \in \mathbb{R}^3$ of the body. The joint state consists of the joint angles $q_t \in \mathbb{R}^{12}$, the joint velocities $\dot{q}_t \in \mathbb{R}^{12}$ along with the history of the joint target errors $q^e_{hist,t} = \{q^e_{t-j} \in \mathbb{R}^{12}\}^{j=1...N}$ (explained below) and the joint velocities $\dot{q}_{hist,t} = \{\dot{q}_{t-j} \in \mathbb{R}^{12}\}^{j=1...N}$. In our work $N = 3$, i.e., the velocities and joint target errors from the last three policy steps are stored and added to the state. We also include to the state s_t the last two actions $\{a_{t-j} \in \mathbb{R}^{12}\}^{j=1...(N-1)}$. Finally, the 3D velocity command is also given as an input to the policy neural network.

The orientation and the angular velocity of the base can be provided by an IMU on-board the robot, which internally uses an Extended Kalman Filter (EKF) to estimate the angular orientation from raw gyroscope and accelerometer sensor data. At each joint an optical encoder measures the joint angles, from which one can then compute the joint velocities. The joint target errors are the differences between the target joint angles conveyed to the PD controller and the measured joint angles, i.e., $q^e_t = q^{target}_{t-1} - q_t$. The error q^e_t implicitly provides rich information about the environment such as the contact state of the feet with the ground. The target errors also vary depending on the terrain as the vertical foot position shifts if the terrain is not flat, which changes the resulting joint angles. Therefore, it is also crucial to add the last two actions of the policy to the state so that the learning can observe the change of the joint target errors for similar actions which indicates a change in the terrain.

The on-board IMU does not directly measure the linear velocity, and estimating this quantity from accelerations is not reliable as the integration process often diverges over time due to sensor bias. Like Ji et al. [Ji+22a], we propose training a separate state estimation network for estimating the base linear velocity from the IMU and joint encoder

measurements. The state estimation network is trained, from data collected in simulation, through supervised learning and it receives as input the base orientation and angular velocity along with the joint angles, joint velocity, history of the past joint angle errors, joint velocities and actions. The output is a three-dimensional vector that estimates the linear velocity in the x, y, z directions. Implementation details can be found in Subsection 4.3.4.

Action space

The design of the action space can make a difference on the learning speed and policy quality. As we discussed previously, direct torque control is harder to learn than joint position control in RL-based systems [PP16]. Similar observations were made in the literature on learning quadruped robots' locomotion [Hwa+19; Kum+21]. We also argue that torque control policies are harder to transfer than joint angle control policies, due to the fact that joint angle control is inherently stable after choosing appropriate impedance gains K_p and K_d. While direct torque control can result in diverging motion, especially during the flying phases of the legs where the apparent joint inertia is low, position-based impedance control forces the joints to behave like a spring damper system.

For these reasons, we propose learning a policy π that outputs displacements of the reference joint angles with respect to the nominal pose of the robot, i.e., $\pi_\theta(s_t) = \Delta q_t^\theta$, where π is implemented by the policy neural network parameterized by θ, and s_t is the state input to the policy at time t. The target joint angles can then be computed as:

$$q_t^{target} = q_{init} + \lambda_q \Delta q_t^\theta, \tag{4.7}$$

where q_{init} is the robot's nominal joint configuration around which the policy actions are centered. We define λ_q as a constant that scales the output of the network before adding to q_{init}. Given q_t^{target}, we use a PD controller to compute the torques:

$$\tau_t = K_p(q_t^{target} - q_t) - K_d \dot{q}_t, \tag{4.8}$$

with the proportional and derivative gains K_p and K_d. It is important to note that using such a joint controller does not imply having a rigid position control. The reference angle q_t^{target} should not be interpreted as positions to be reached, but rather as intermediate control variables. The resulting system is analog to elastic strings that pull the joint angles toward q_t^{target}.

Reward function

The reward function defines the task. The main task here is to follow a given reference velocity. In order to get natural locomotion that can be deployed on the robot, some constraints related to the robot's pose, joint torques, joint velocities, etc, needs to be satisfied. After each action a_t, the robot receives a reward r_{t+1}. We split our reward r into one main positive term that rewards the tracking of the commanded velocity and several weighted penalty terms that act as negative costs in the reward. The values of the weights are listed in Table 4.3. The reward terms and state variables stated below are implicitly indexed by the time step index t but we only include this index when necessary for clarity.

Command velocity tracking.

The reward r_{vel} used for tracking the command velocity is based on the squared Euclidean distance between the current 3D vector V_{x,y,w_z} consisting of the forward, lateral

and yaw velocities of the body and the 3D velocity command V^{cmd}, i.e.,

$$r_{vel} = c_{vel} e^{-||V^{cmd} - V_{x,y,w_z}||^2}, \tag{4.9}$$

where coefficient c_{vel} is used to scale the reward.

Foot clearance penalty.

To encourage the robot to lift its feet high even when training on a flat surface, we use the foot clearance objective proposed by Ji et al.[Ji+22a]. Denoting the height of the i-th foot by $p_{z,i}$, we set a constant foot height target p_z^{max} and define the foot clearance penalty as

$$r_{clear} = c_{clear} \sum_{i=1}^{4} (p_{z,i} - p_z^{max})^2 ||\dot{p}_{xy,i}||^{0.5}, \tag{4.10}$$

where $\dot{p}_{xy,i}$ stands for the velocity of the foot i in the x, y direction so that the target is not active during the ground contact and it is approximately maximal in the middle of the swing phase. The scalar c_{clear} is a weight for this penalty.

Foot slip penalty.

When a foot comes in contact with the ground, the x, y components of its velocity $\dot{p}_{xy}$ should be zero in order to avoid slipping. We define a foot slip penalty as

$$r_{slip} = c_{slip} \sum_{i=1}^{4} C_i ||\dot{p}_{xy,i}||^2, \tag{4.11}$$

where C_i is a binary indicator of the ground contact of the i-th foot, and c_{slip} is the penalty weight.

Base orientation and velocity penalties.

The base pitch, roll and velocity in the z direction should all be near zero to produce stable motion. With scalar weights c_{orn} and c_{vz}, we define this penalty as

$$r_{base} = c_{orn}(roll^2 + pitch^2) + c_{vz} V_z^2. \tag{4.12}$$

Joint pose penalty.

We add a penalty on the joint angles in order to learn to avoid large joint displacement. We define this penalty as the deviation from the nominal joint angles values at the initial state q_{init}, as

$$r_{joint} = c_q ||q_t - q_{init}||^2 \tag{4.13}$$

with weight c_q.

Power loss penalty.

For safety reasons and for saving energy, we would usually prefer to minimize the overall power consumption of the robot. The power loss term encapsulates the relationship between the torque and velocity at the joint level. We use the model proposed and identified by Fadini et al. [Fad+21] which includes the heating by Joules loss in the motors P_J as well as the losses by friction P_f.

We denote by τ_f the torque necessary to overcome the joint friction :

$$\tau_f = \tau_u \text{sign}(\dot{q}) + b\dot{q}, \tag{4.14}$$

where q, $\dot{q}$ are respectively the joint position and velocity. The identified model parameters on Solo-12 are the Coulomb friction $\tau_u = 0.0477$[Nm] and the viscous friction coefficient $b = 0.000135$[Nm·s].

The two sources of power losses can then be expressed as

$$P_f = \tau_f \dot{q} \quad [W], \text{ and } P_J = K^{-1}(\tau + \tau_f)^2 \quad [W],$$

where τ is the joint output torque and $K = 4.81$[Nm·s] is linked to the motor coil resistance and motor constant.

The total power over joints and the penalty term used in the reward is taken as the sum over all joints:

$$r_E = c_E \sum_{j=1}^{12} P_{f,j} + P_{J,j} \tag{4.15}$$

with the weight c_E.

Action smoothness penalties.

To generate joint trajectories without vibrations and jitter, we define a penalty on the first and second order differences in the joint angle values:

$$r_{smooth} = c_{a1}||q_t^{target} - q_{t-1}^{target}||^2 + c_{a2}||q_t^{target} - 2q_{t-1}^{target} + q_{t-2}^{target}||^2$$

with weights c_{a1} and c_{a2}.

Total reward.

The final reward is a weighted sum of the positive velocity tracking reward minus a sum r_{pen} of all the penalties explained above:

$$r_{total} = r_{vel} - r_{pen}. \tag{4.16}$$

4.3.2 Domain and Dynamics Randomization

In order to learn policies that transfer to the real robot, we have to identify and bridge the sim-to-real gap. We decided to use domain randomization techniques by adding noise to the state and randomizing some aspects of the simulator dynamics. Table 4.1 shows the noise models used for each element in the state and dynamics. For the dynamics, we found that for Solo12 it was enough to randomize the gains of the PD controller in order to learn

policies that adapt to some stochasticity in the low level control that can come from many factors. This is in contrast to previous work on ANYmal and Mini-Cheetah where more randomization was needed for the center of mass, mass of the body and links, positions of the joints and motor friction [Ji+22a; Hwa+19; Lee+20; Mik+22]. Randomizing the state is essential in order to overcome sensory noise. The experimental work we conducted led us to the conclusion that one can learn a transferable policy on Solo12 using this simple randomization strategy.

Observation Noise	
θ_{body}	$U^3(-0.05, 0.05)$
ω_{body}	$U^3(-0.10, 0.10)$
v_{body}	$U^3(-0.10, 0.10)$
q	$U^{12}(-0.05, 0.05)$
$\dot{q}$	$U^{12}(-0.50, 0.50)$
Dynamics Noise	
K_p	$U(-1.0, 3.0)$
K_d	$U(-0.1, 0.1)$

Table 4.1: Uniform noise for each of the state observations and PD controller gains.

4.3.3 Curriculum Learning

Reward curriculum.

Due to the elaborate penalty terms of the reward function, we observed that the agent may learn to neglect the positive reinforcement signal for tracking the command velocity and learn to stand still, since this behaviour optimizes several penalty terms in the reward. In order to bypass this problem, we introduce a linear curriculum on the reward. Curriculum learning is a popular method that introduces easier tasks to learn at the start of training and gradually increases the level of difficulty as training progresses [Ben+09]. Like Hwangbo et al.[Hwa+19], we multiply the cost terms of the reward function by a curriculum factor $k_c \in [0, 1]$ that is equal to zero at the start of the training and slowly increases up to one through the training iterations. The reward function becomes $r_{total} = r_{vel} - k_c r_{pen}$. This way we first train the agent to follow the command velocity in any manner before emphasizing the cost terms in the reward in order to refine locomotion.

Noise curriculum.

We also introduced a curriculum on the injected noise for randomizing the state and the dynamics. We found that decoupling the curriculum of the reward and the randomization works better. Therefore, the sampled noise in Table 4.1 is multiplied by another curriculum factor $k_{c,noise} \in [0.0, 1.0]$ that is increased at a slower pace than k_c.

Terrain curriculum.

Finally, curriculum is also used to adapt progressively the locomotion to the terrain. We introduced rough terrains at the end of the training to learn from more complex interac-

PPO parameters	Flat terrain	Non-flat terrain
Clip ratio	0.200	0.050
Gradient norm clip	0.500	0.300
Entropy coefficient	0.010	0.000
Learning rate	0.005	0.001

Table 4.2: PPO parameters when training on flat terrain and non-flat terrain.

c_{vel}	c_{clear}	c_{slip}	c_{orn}	c_{vz}	c_q	c_E	c_{a1}	c_{a2}
6.0	20.0	0.07	3.0	1.2	0.5	2.0	2.5	1.5

Table 4.3: Weights of the reward function.

tions when the ground is not flat. This helps in refining the robot's locomotion in terms of lifting all feet equally in order to keep balance. At the last 1000th training iteration, we start sampling random heightmaps at the start of the episodes. We also lower some PPO parameters to perform more conservative updates to the policy, in order to avoid catastrophic forgetting [Fre99] of locomotion on flat terrain, once the rough terrains are introduced and the training data distribution changes. The PPO parameter values before and after introducing the rough terrains are listed in Table 4.2, we refer to Schulman et al. [Sch+17] for a description of these parameters.

4.3.4 Experimental Results on Solo-12

In this section, we analyze the locomotion produced by our learned control policies. We test both symmetric ($\overline{><}$) and non-symmetric ($\overline{<<}$) poses of the legs with the policy being able to learn both successfully. We display results about velocity tracking and energy consumption of the learned controller. Successful real robot transfer experiments are conducted and discussed in the following sections.

Implementation details

The control policy is implemented as a multi-layer perceptron [MP69] with three hidden layers of sizes 256, 128 and 32 with Leaky ReLU activations between each layer [MHN13]. The control policy runs at a frequency of 100Hz. We used the Raisim simulator [HLH18] for training. The simulator frequency is set at 1kHz which means that the PD control between each RL step is executed ten times. On the real-robot we have a low-level loop at 10kHz for communicating with the actuators, but the policy network is still queried every 0.01 seconds (see Figure 4.6). In simulation, 300 different versions of the robot are run in parallel processes in order to collect diverse data faster. The value of the PD control gains are $K_p = 3$ and $K_d = 0.2$ respectively. These values are taken from the one used in the model-based controller that satisfy the second order stability of the PD equation [Léz+20] . On the robot, the computation of actions from states only takes $10\mu s$ on a Raspberry Pi 4 which makes this approach particularly appealing due to its simple setup and high computational speed.

The state estimation network is also a multi-layer perceptron with two hidden layers of sizes 256 and 128 with Leaky ReLU activations and a three-dimensional output corresponding to the linear velocity. To train the state estimation network, we run the learned

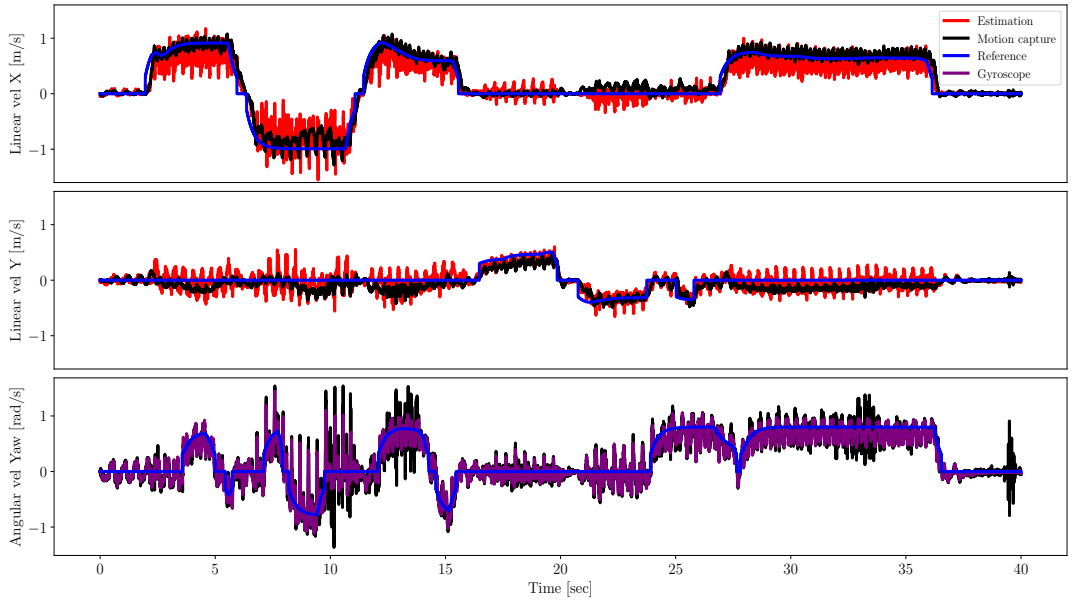


Figure 4.7: Plot of the 3D velocity command controlled by a gamepad to command the real robot in blue. The red curve plots the output of the state estimation. The black plot is the ground-truth base velocity as measured by the motion capture system on the real solo12. The purple plot is the yaw velocity estimate from the gyroscope in the IMU. The x-axis shows time in seconds.

policy in simulation to collect a dataset of states, without linear velocity, that are the input to the state estimation network and the linear velocities that will be its output. We found that a dataset of 50,000 samples (policy steps) is enough to train the estimation network to a good accuracy. In Ji et al. [Ji+22a], the authors proposed to learn both networks (estimation and control) simultaneously. In our experiments, we did not observe any advantage when training both networks together. So we decided to train the estimation network after the control policy in order to not slow down the RL training due to the overhead from performing supervised learning every few RL iterations. The data is collected with the random noise added to the observations and PD gains along with randomizing the terrains between rough and flat. We trained on a supervised cost to minimize the mean squared error loss using the Adam optimization algorithm [KB15].

We used the PPO objective from [Sch+17] to train the policy network. In each training episode, the policy is run for 100 steps (= 1 second of real-time) to collect data for optimizing the objective. The episode ends if the body of the robot comes in contact with the ground. Even though locomotion is not an episodic task with a natural endpoint and the episode is not reset between each training epoch, we chose to introduce random resets at the beginning of some episodes since this appears to stabilize training. At the start of each episode, a random velocity command is sampled and then scaled by the noise curriculum factor so that the network starts learning gradually from one low velocity towards higher ones. The initial state at the start of each episode is set at the nominal joint pose q_{init} with zero joint velocity. We used the `stable-baselines`[Raf+21] open source implementation of the PPO algorithm.

As mentioned before, at the beginning of training the ground is flat, but in order to learn more robust policies, we gradually introduce some non-flat terrains by sampling random height values for points in a regular grid. At the last 1000th training iteration, 80% of the parallel processes start sampling non-flat terrains. We found that we need around 10,000 training iterations which equates to 300 million collected samples with

300 parallel processes.

Table 4.3 shows the coefficient values that are used to scale each term in the reward function. Along with choosing the right values of the weights, we choose the desired maximum foot height in the foot clearance reward to be $p_z^{max} = 6cm$. We scale the output of the policy network with scalar $\lambda_q = 0.3$ before integrating towards the target joint angles.

Velocity tracking

We first judge the quality of the learned controller by its ability to follow the reference velocity in the forward, lateral and yaw directions. During training, we randomly sample the velocity vector based on the following uniform distributions: $V_x \sim U(-1.5, 1.5)$, $V_y \sim U(-1, 1)$ and $W_z \sim U(-1, 1)$. As mentioned before, these values are scaled by k_{noise} in order to start learning with low velocities before gradually increasing the range of sampled velocities.

Figure 4.7 shows the velocity plots of a random walk recorded while guiding the robot with the gamepad across the room. The blue lines plot the reference velocity command in three directions. The black lines represent the robot's body velocity estimation from motion capture data. The red lines in the first two plots are the state estimation network's velocity estimates in the x and y directions. From the plots, we see that the real robot is able to follow the commanded velocity well, as indicated by the alignment between the motion capture plots – which provides ground-truth values – and the reference command plots. The velocity predictions from the state estimation network are similar to the ones from motion capture, while being more noisy. The noise in the prediction, that is given as an input to the control network, does not appear to downgrade the performance of the controller. Indeed, this robustness to noisy estimation is expected as noise is added to the linear velocity input during training.

Figure 4.8 shows the plot of the hind right joint angle target vs. the measured joint angles for the same random run. We observe that the target joint angles are not reached. The difference between the command and the achieved angles showcases the nature of the soft impedance control, which resembles elastic strings where the desired joint velocity is zero. Similar behaviour can be observed for the other legs.

Energy consumption

In order to verify the usefulness of the proposed power loss penalty in the reward function, we run several experiments while varying the power loss weight c_E in the reward and observe its effect on the learned policy. We run the policies in simulation for five seconds for the maximum forward velocity command of $1.5[m/s]$. This test focuses on a rapid and dynamic task that would require most energy.

Table 4.4 lists the effect of c_E on the average power consumption, velocity error and base height during the test task. We first observe that the increase of c_E decreases the power loss. This confirms that the power term on the learned policy makes intuitive sense and that it can be tuned to learn locomotion with different power profiles. We found that for higher values of c_E, where $c_E > 10$, the reward is ill-defined and training fails.

Increasing the weight c_E makes the policy prioritize optimizing the power loss rather than other rewards such as velocity tracking. We observe this effect in the table as the velocity error increases when using policies that have learned to consume less power due

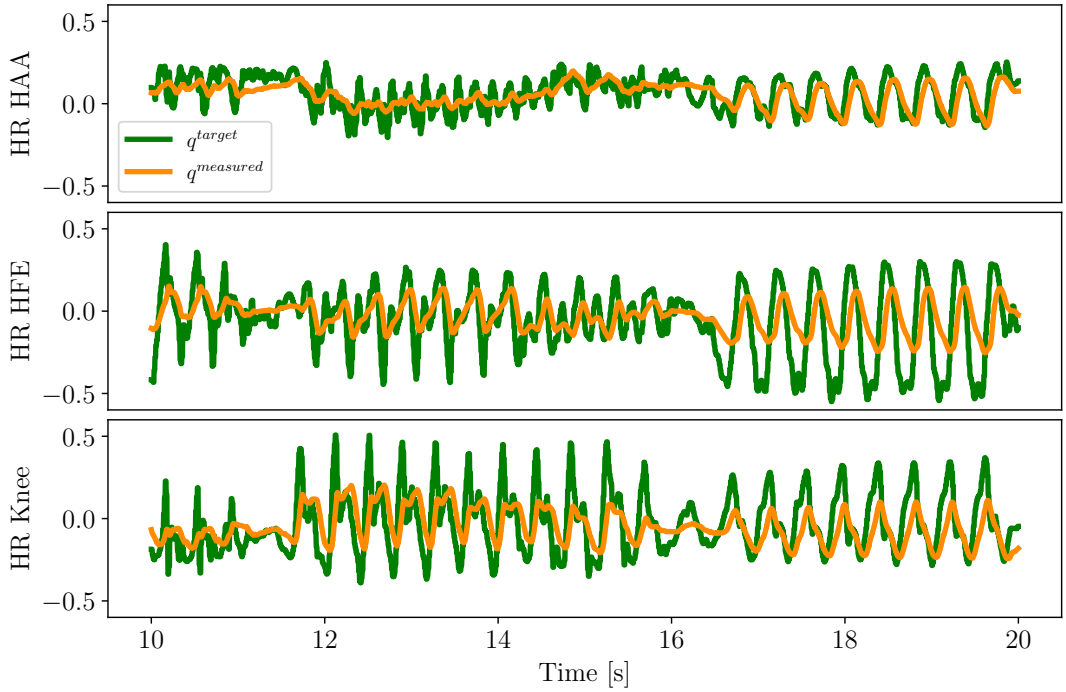


Figure 4.8: Plot of the desired joint angle command vs. the measured joint angles over a random run for the hind right leg. HFE stands for Hip-Flexion-Extension and HAA stands for Hip Aduction-Abduction.

to higher c_E. The velocity error column contains the l_1 norm of the difference between the desired velocity and the achieved velocity. Note that even though the error increases, we see a big decrease in the consumed power, which would make the policies with $c_E \in [3, 4]$ an attractive option since the robot would have slightly less accurate velocity tracking but still save more than 30% on the consumed power.

The base height could be another indicator of energy efficiency, since standing on straighter legs requires less power. In Table 4.4, we list the body height as a function of c_E, and observe a gradual $2\,cm$ increase in the base height when c_E increases from zero to ten. Beyond $c_E = 10$ the RL ceases to produce good policies as mentioned before.

Power vs. torque penalty in simulation

In previous work [Hwa+19; Mik+22; Ji+22a] penalty terms on the torque magnitude, joint velocity magnitude and joint accelerations were used in the reward. We trained several policies using these penalty terms to compare with the proposed power cost. The last row in Table 4.4 shows the power loss vs. velocity error for the policy trained with those penalties. The learned policy is less energy efficient than most of the policies computed with the power term, with high variance between the policies. In practice, we found it easier to tune a single power weight during experimentation rather than tuning three separate weights for torque, velocity and acceleration terms with different units. The power loss formula expresses the relationship between the torque and the velocity by effectively combining the other three penalties into a one single physical and coherent term.

c_E	Power [W]	Velocity error [m/s] (%)	Base height [m]
0.0	17.7	0.079 ± 0.054 (7.9%)	0.23 ± 0.004
0.1	16.2	0.083 ± 0.067 (8.3%)	0.23 ± 0.006
1.0	13.7	0.092 ± 0.065 (9.2%)	0.24 ± 0.009
2.0	12.0	0.121 ± 0.064 (12.1%)	0.24 ± 0.007
3.0	11.0	0.141 ± 0.086 (14.1%)	0.24 ± 0.014
4.0	10.2	0.145 ± 0.091 (14.5%)	0.25 ± 0.004
10.0	7.7	0.198 ± 0.164 (19.8%)	0.25 ± 0.014
20.0	5.5	0.275 ± 0.113 (27.5%)	0.23 ± 0.005
With joint torque, velocity and acceleration penalty			
-	15.5	0.122 ± 0.054 (12.2%)	0.27 ± 0.008

Table 4.4: Average Power vs. velocity error as a function of the power weight c_E.

c_E	0.0	-0.1	-1.0	-2.0	-3.0	-4.0	-10.0
P[W]	82.8	85.4	80.7	71.6	77.5	71.7	81.3

Table 4.5: Power calculation on the real robot as a function of the energy weight c_E.

Electric power on the real robot

We also trained several policies with different power loss weights and then tested them on the real robot by running each policy at a forward velocity command of $1.0m/s$ for five seconds in order to measure the used electrical power (voltage times current). We observed inconsistent results in terms of the measured power vs. the power coefficient weight in Table 4.5. Unlike in the simulation, we did not observe the downwards trend of the power when higher weights are put on the power penalty. We believe this is a clear indication of a sim-to-real gap. A policy that appears to be efficient in the simulation is not necessarily as efficient on the real system. It might be interesting to explore this phenomenon in future work. One possibility is to model the power loss on the real robot and use it in the reward function rather than relying on the calculation from the simulator that does not take into account many contributing aspects in the actuation.

Studying the effect of the curriculum

In order to validate some of the choices made on the reward terms, curriculum and terrain curriculum, we ran a set of ablation experiments. Figure 4.9 shows the training curves that plot the average reward over the training steps for different setups. The blue curve shows our proposed method with the curriculum on the reward and terrain. The orange curve shows the experiments without using a curriculum. The red curve experiments the same reward curriculum but introduces the non-flat terrain from the start of training rather than at the end, as we propose. All the curves are averaged over three different runs of their respective experiments.

Figure 4.9 shows that the proposed method with the curriculum outperforms the rest of the experiments in terms of the final average reward achieved, and that the variance in the performance between the learned policies is low. This indicates that the learning is consistently reaching similar behaviours at the end of training. We also see that the experiments that use a curriculum achieve a higher reward at the start of training, which

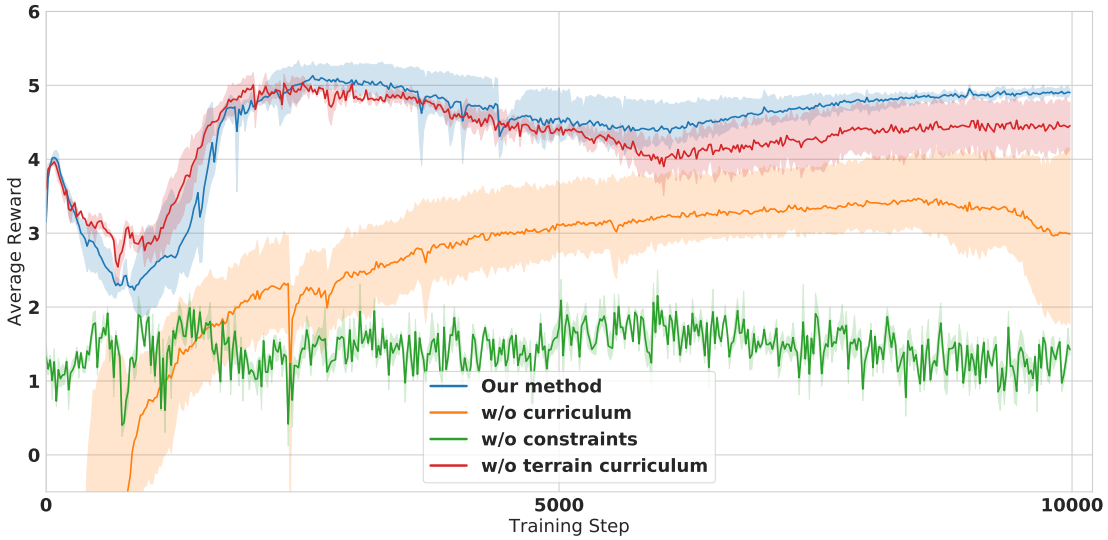


Figure 4.9: Plot of the average rewards over training steps for different setups. Each curve is averaged over three random seeds of the same experiment.

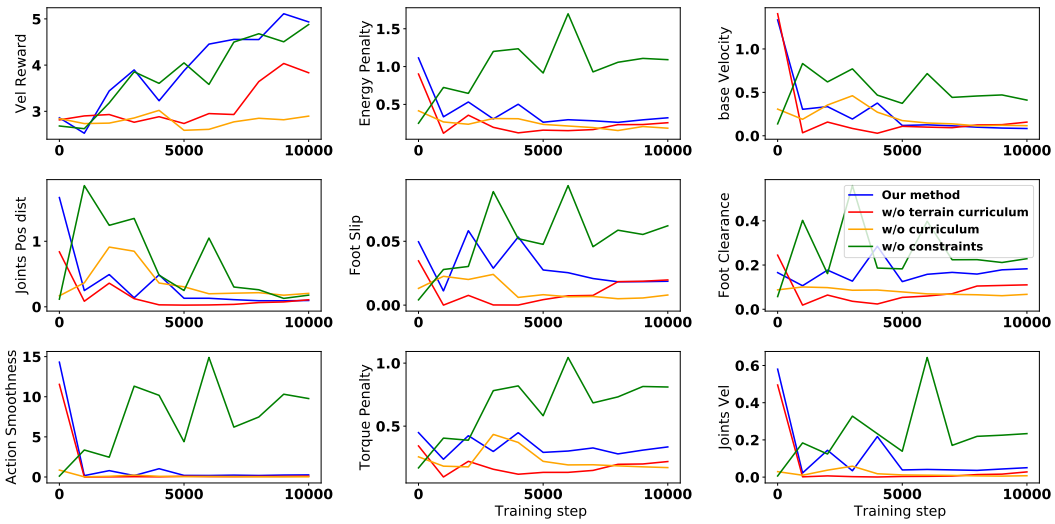


Figure 4.10: Plot of the separate reward penalties of the different curriculum settings over training steps.

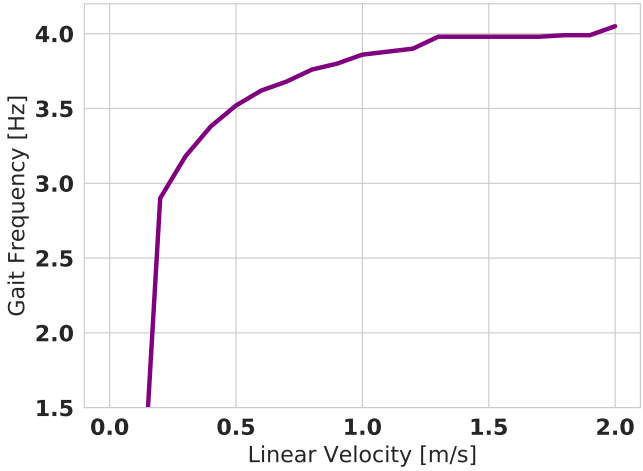


Figure 4.11: Plot of the gait frequency as a function of the linear velocity command

allows it to learn faster and reach a higher performance in the end. On the contrary, not using a curriculum results in slower learning, higher variance between runs and an asymptotically lower performance at the end of training.

The green curve is an experiment where an RL policy is trained only with the velocity tracking reward without the rest of the penalties. The curve plots the value of the complete reward, including the penalty terms, to evaluate the performance of the velocity tracking alone without regards to the other terms. As we see, the average reward performance for that experiment is very low, even though we observe that the velocity tracking term for these experiments is fully optimized.

Figure 4.10 shows the values of the individual reward terms for the same ablation experiments during the training process weighted by their chosen coefficients. The plot displays the average rewards achieved over three random seeds for each experiment. The objective is to maximize the velocity tracking reward while minimizing the rest of the penalties. Our proposed training setup results in the best velocity tracking reward while optimizing the rest of the penalties. The experiments that do not use a reward curriculum (orange) or a terrain curriculum (red) optimize penalties but do not get a good performance over the main velocity tracking reward. This is in line with our motivation for designing the curriculum to learn the best trade-off between following the velocity and respecting the penalties. We notice that the experiment trained on only tracking the reward (green) is able to maximize the velocity tracking term, however it does not respect any penalty terms.

Gait frequency

One of the desired features to have in a controller is the ability to adapt the gait frequency based on the velocity command. We show in our work that using RL, we can learn controllers that adapt their frequencies online. Using Fast Fourier Transform analysis (FFT) on the trajectory of the joint angles of the robot, we are able to deduce the frequency of the gait. Figure 4.11 shows the value of the frequency as a function of the linear velocity command for the same experimental condition as before using the complete reward and curriculum. We see a positive monotone relationship between the velocity and gait frequency. This behaviour emerges naturally during learning and is not hand-designed.

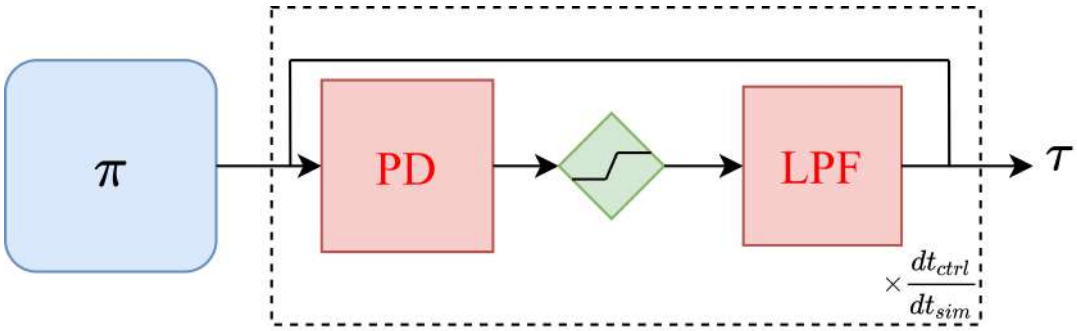


Figure 4.12: The low pass filtering scheme adopted for training and testing the policy.

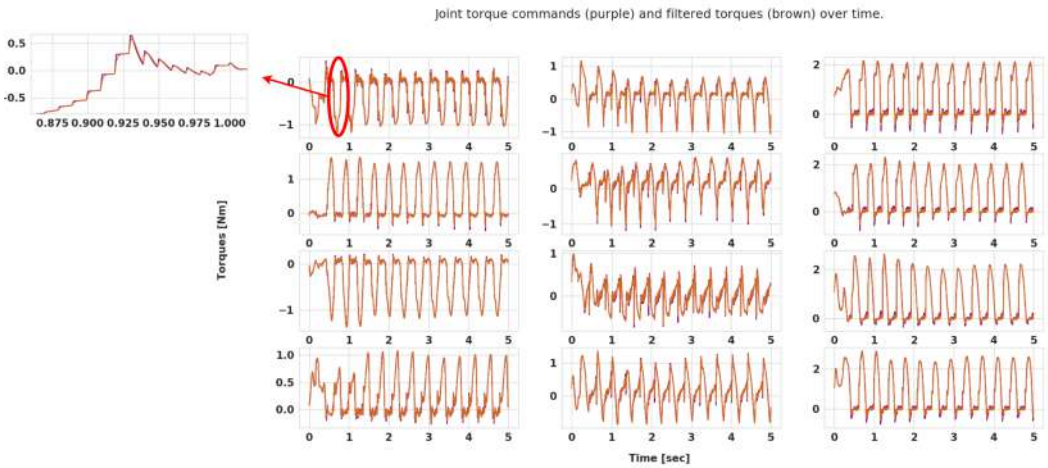


Figure 4.13: The torques output by the PD controller alone (purple) and the torques output by the filter (orange), they are practically the same. The zoomed in plot shows the effect of the filter.

This is an interesting result because adapting the gait frequency to velocity is not that straightforward to obtain through MPC-based controllers.

Adding low pass filter

Putting a low pass filter at the torque control level can simulate the latency that appears in the real system between a commanded torque and the delay before the actuator reaches that torque. We observe that the smoothness terms in the reward function helps the policy to output smooth actions. However, the filter can still be useful at the training level as well as at the deployment. Figure 4.12 shows the low-level control scheme, it consists of a PD step, followed by torque clipping, followed by a low pass filter. The cutoff frequency of the filter is set at 100hz.

Figure 4.13 shows the torque output of the PD controller versus the low pass filter. The torque values of the PD controller before filtering (purple) and after filtering (orange), appear to exactly match. Therefore, the filtering does not seem to have any effect which shows that the policy learns to output smooth actions in the first place when relying solely on the smoothness reward function.

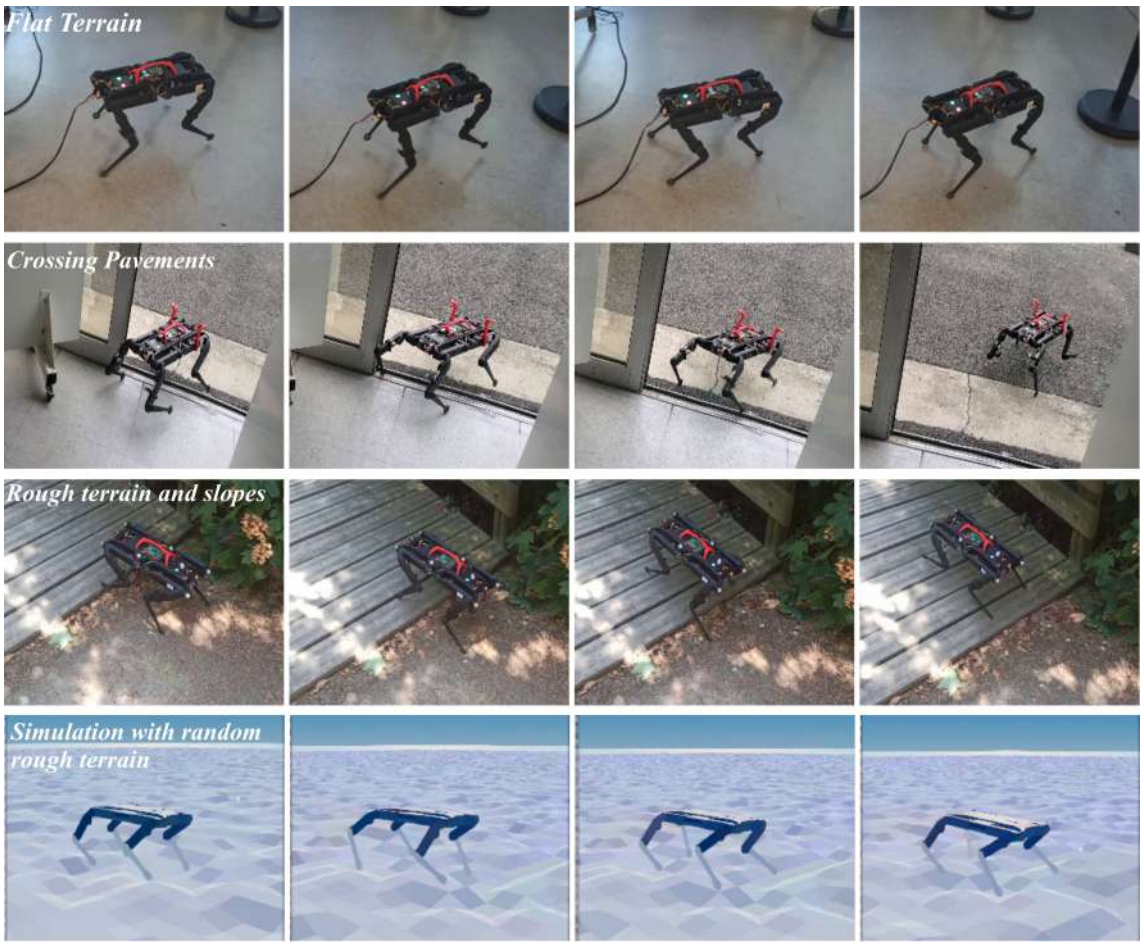


Figure 4.14: Snapshots of the Solo12 quadruped in real settings and in simulation driven by a reactive controller learned through deep reinforcement learning. With learned controllers, the robot can traverse various outdoor environments with slopes and rough ground, full video: https://youtu.be/t-67qBxNyZI.

Comment on the policy transfer to Solo12

As explained earlier, random uniform noise was added to the robot dynamics and state observations during training. This noise was progressively inserted through the curriculum factor $k_{c,noise}$, starting with noiseless simulations and increasing the noise magnitude as the training progressed. The goal was to prepare the policy network for sim-to-real transfer so that, once deployed on a real Solo12, it would still produce robust behavior even if the model did not perfectly fit the system. Such discrepancy is inevitable since different motors have slightly different characteristics that vary as coils get warmer, and the model does not include joint friction, its inertia matrices are not perfectly accurate, etc.

Despite these inevitable model inaccuracies, the policies were successfully transferred almost on the very first try. Even though Solo12 is a lightweight small robot, we were able to run it with our learned policies on various terrains, i.e., indoors, outdoors on grass and pebbles and on an ascending and descending slopes that are relatively steep considering the size of the robot. These results show the robustness of the proposed control approach with respect to model variations. The transfer did not require learning an actuator model, as done in other works [Hwa+19], or modeling the actuation dynamics to include a bandwidth limitation through a low pass filter on the torques. This demonstrates how a simple randomization during training is enough for direct transfer to Solo12, probably by virtue of the fast dynamics of this robot (lightweight quadruped powered by low inertia actuators with high bandwidth) which leads to a limited sim-to-real gap. This all makes the Solo platform an attractive choice for deploying RL schemes. Figure 4.14 shows snapshots of the policy controlling the robot in indoors and outdoors environments and snapshot of the simulated robot running over the rough terrain.

In this section, we provided a detailed account of our approach to design learning-based controllers for Solo-12, focusing on robust velocity tracking tasks. In the next section, we turn our attention to the implementation of RL-based control on the Mini-Cheetah, a heavier quadruped robot with stronger joint torque capabilities.

4.4 Learning Joint Angle Controllers for Mini-Cheetah

This section discussed the modification required to the RL method for learning robust controllers for MIT's Mini-Cheetah. In Section 2.1 we discussed the difference between the Solo-12 robot and Mini-Cheetah. In this section, we witness these differences when transferring policies learned in a similar manner on Solo-12 vs. on Mini-Cheetah.

4.4.1 Evaluating the Baseline RL Approach for Mini-Cheetah

We maintain the same training process explored in Section 4.3 with the state estimation network and policy. However, some control parameters are different on Mini-Cheetah for example the PD gains that have to be increased ($K_p = 20.0, K_d = 0.5$). In addition, more elements of the robot's dynamics model are randomized and the amount of randomization for some common elements with Solo-12 are adapted due to the difference in size and power of Mini-Cheetah.

At a first attempt of deploying a policy learned with the method presented in Section 4.3, we observed that the robot would quickly lose its balance. The robot behaved as

if its center of mass (COM) had been shifted. A rollout of the policy on the real Mini-Cheetah is shown in Figure 4.15. From the snapshots, one can see the tilt of the robot's base, even though the robot should move forward, it diverges quickly to the right due to this imbalance without the ability of recovering.

To combat this issue, we randomized the COM position of the Mini-Cheetah in simulation at the start of each training episode. The random COM allows the policy to observe trajectories where the balance of the base is not constant and varies. Intuitively, this method should produce policies robust to the shifts of the COM on the real-robot. We found that the resulting policy falls much less and is better at keeping balance. However, the robot exhibited jittering behaviour on the joints. Figure 4.16 shows snapshots of the robot after randomizing the COM. The figure shows that the base orientation is more stable and its trajectory no longer diverges when moving forward. However, we observe clear jittering on the legs which causes unpredictable motions in an attempt to recover, which could be dangerous on the real system and could cause the robot to fall. Various factors in the control could cause jittering, such as underdamped servoing, disregarding friction and stiction and insufficient system stiffness. In learning control policies, it is vital to have smooth action sequences. Since we already penalize the lack of smoothness in the reward function and as the output of the network is smooth, then the issue is most-likely related to mismatches between the model in the simulation and real robot dynamics.

Similarly to the Solo-12 baseline, Ji et al. [Ji+22a] proposed to learn a state estimation network for Mini-Cheetah by learning information about the state of the robot that are only available in simulation such as the feet height and the feet contact with the ground. The authors show that the policy transfers well to Mini-Cheetah. Therefore, it seems relevant to try to predict other privileged information with the state estimation network. However, we decided to choose a more general approach for having privileged information injected in the RL training process, without explicitly requiring these privileged information at deployment on the real robot. We decided to use a distillation approach that predicts a latent variable that is a learned representation of the privileged information that are unobservable on the real robot.

4.4.2 Policy Transfer through Distillation-Based Approach

Latent variable models are statistical models that aim to explain observed data by assuming the existence of latent variables. These models assume that the observed data is generated from a set of underlying hidden variables that cannot be directly measured or observed. In latent variable models, the observed variables are considered to be noisy or imperfect indicators of the underlying latent variables [SR07; KW22]. The latent variables are hypothesized to capture the true structure or patterns in the data, which may not be readily apparent from the observed variables alone.

In our case, the state of the robot in simulation can be split into the observable state, that consists of proprioception and IMU readings, and the environment constants, that are unobservable on the real system. These constants shape the dynamics of the simulation in the environment and robot, e.g., mass of the robot, friction of the ground, etc... Latent variables provide a flexible framework for capturing complex relationships and dependencies among state variables and enable us to gain insights into the underlying processes that represent the unobserved data.

Kumar et al. [Kum+21] propose the idea of rapid motor adaptation (RMA). The authors argue that, in order to get robust policies that transfer to the real world, the training

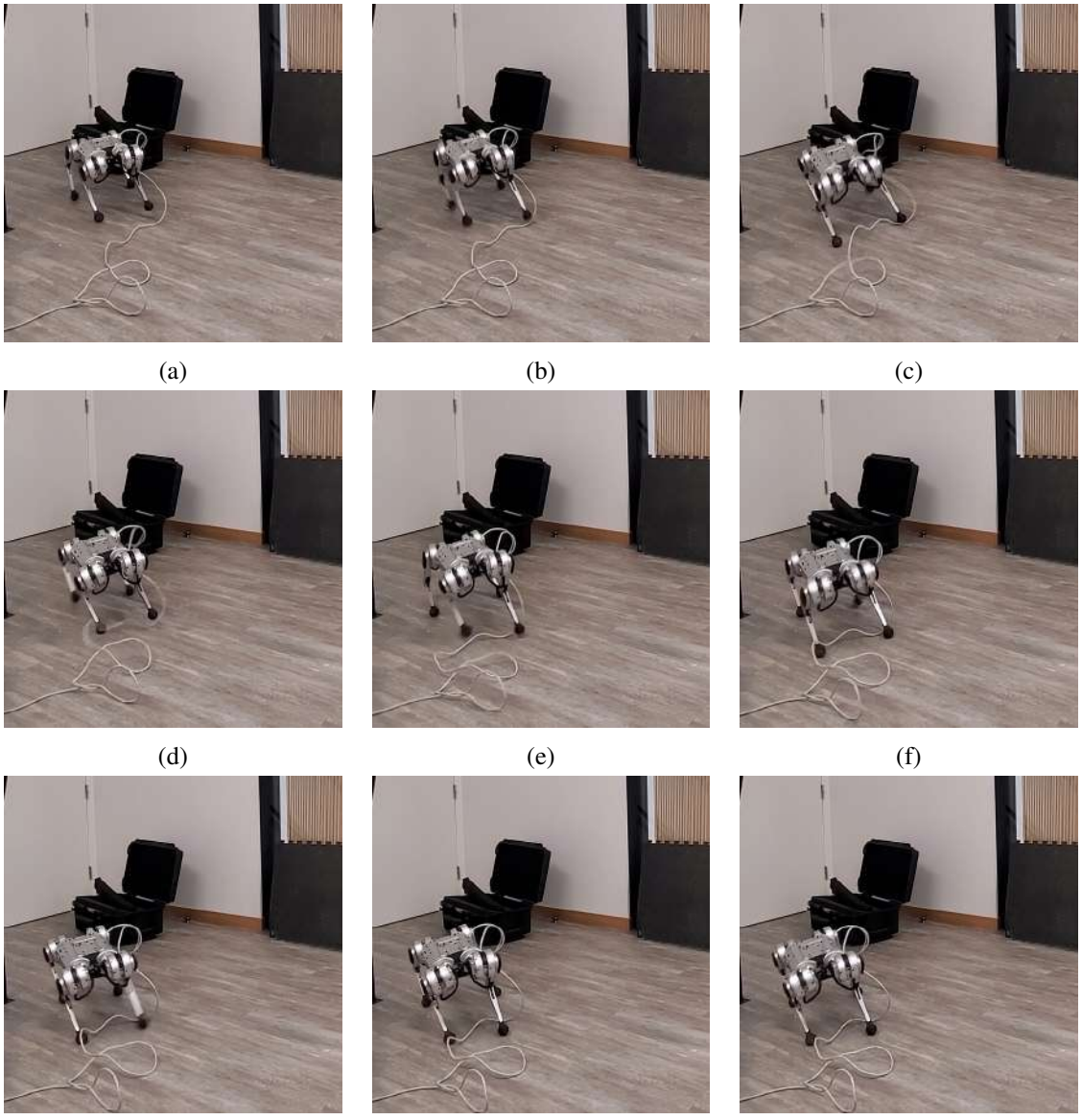


Figure 4.15: Snapshots of a policy rollout on Mini-Cheetah. The policy design is similar to the one that was successful on Solo-12. The robot is commanded to move forward but its base is tilted and the robot is drifting. We can also see clear jittering in the legs in frames (b), (d), (e) and (g), which contribute to the imbalance and eventual failure of the policy. Video of this run can be found here: https://youtu.be/mILgvtutWYM.

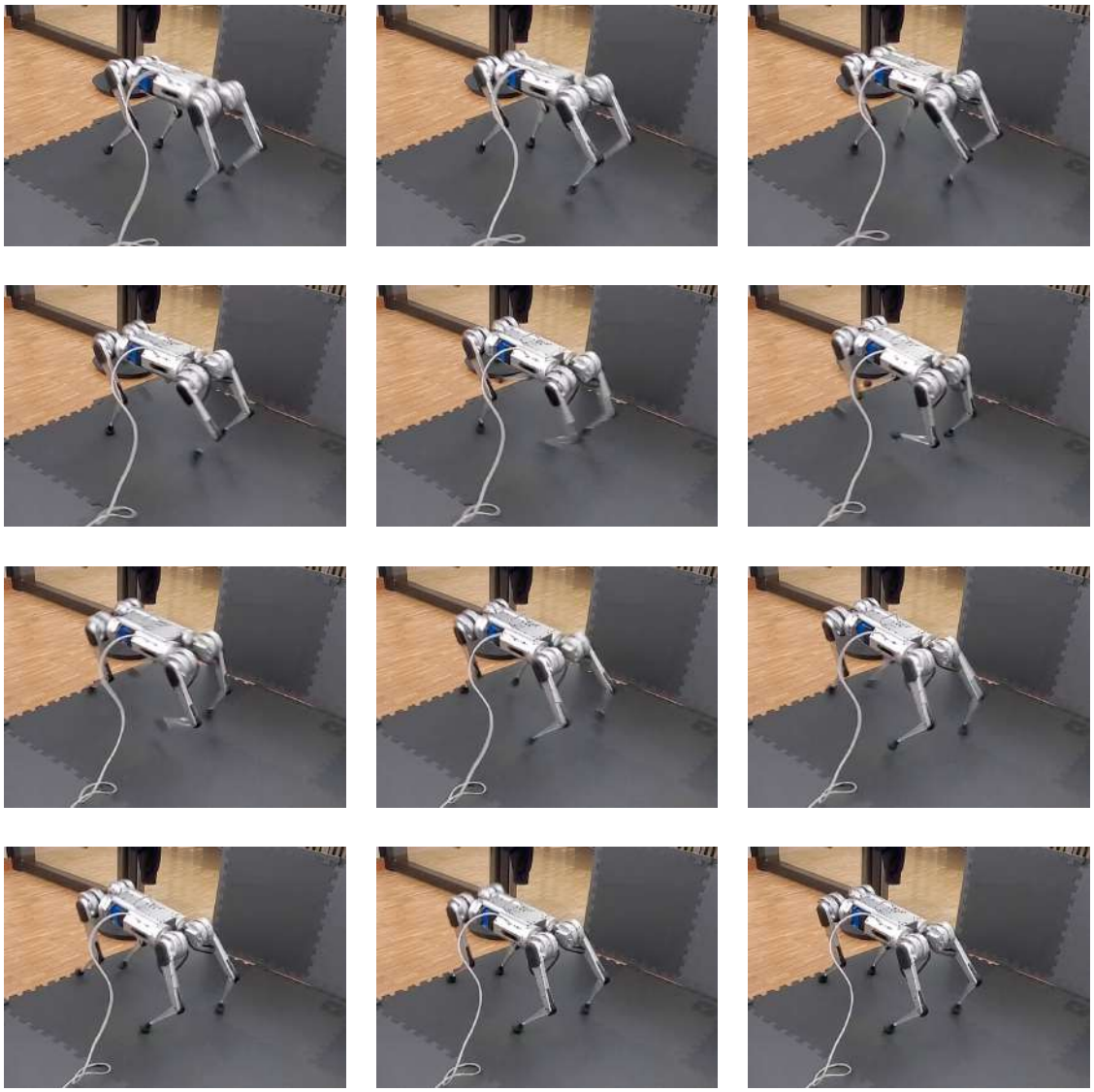


Figure 4.16: Snapshots of a policy rollout on Mini-Cheetah. The added randomness to the COM position results in more stable base motion during the locomotion. However, the resulting jitter makes the policy very dangerous to run on the real robot. The frames show the Mini-Cheetah attempting to stop after moving forward. The jittering motion causes a sequence where most of the robots feet are in the air which almost causes it to fall before it recovers. Video of this trajectory can be found here: https://youtu.be/_BaTmcbxmBQ.

must be done in a diverse environment with various noise and randomization injected to the observations and dynamics of the terrain and the robot that determine the state transitions. The disadvantages of such vast randomization is that it makes the policy learning harder and requires more data to learn and might result in conservative policies that do not produce interesting behaviours. To address these issues, Kumar et al. [Kum+21] proposed adding the randomized coefficient of the terrain and robot dynamics to the input of the policy. This way the policy has direct measurements of the randomized dynamics and can condition its actions on them. The authors proposed learning a latent variable z at time t that encodes a set of privileged information p_t such that,

$$z_t = \text{Encoder}_\mu(p_t), \tag{4.17}$$
$$a_t = \pi_\theta(o_t, z_t). \tag{4.18}$$

where μ is the parameter of the encoder neural network that maps p_t to the latent variable z_t. The paper proposes a two-phase learning approach. The first phase is a standard end-to-end RL where the latent variable acts as a compressed representation of the privileged information that is based on the center of mass of the robot, terrain friction and motor strength. The second phase is a supervised learning step that learns an adaption module ϕ which maps the history of the state to the estimate $\hat{z}_t$ of the latent variable z_t:

$$\hat{z}_t = \text{Adaptation\_Module}_\phi(s_{t-N}, a_{t-N} ..... s_{t-1}, a_{t-1}). \tag{4.19}$$

Therefore, when deployed on the real-system, the final control policy only relies the sequence of the observable states and actions. The adaptation module is estimating a compressed space representing the dynamics from the sequence of state-actions. Figure 4.17 shows the network architecture during the training phase and distillation phase that is later deployed on the real robot.

Many other works use RMA on different quadruped platforms [Mar+22; Fu+21; LKM22] and bipeds [Kum+22]. Other papers propose some variants on the mean squared error loss in the adaptation phase in order to improve convergence and generalization [FCP22; CKP23]. It has also been shown to be an effective technique to distill vision data related to the perception of the terrain by the robot, to a compressed representation space that aids the policy decision mechanism [Aga+22; CKP23; NYM23]. We found this distillation technique to be effective in tasks of navigation in crowded environments with wheeled robots from visual input [MAS22]. The privileged information in this task are the position of the crowd that are visible in the image frame and the observation at deployment are the sequences of images.

Algorithm 1 PPO and adaptation module training

Initialize: policy parameters θ_0, value function parameters ψ_0, encoder parameters μ_0, adaption module parameters ϕ_0, gym environment ENV, on-policy buffer $\mathcal{D} = \{\}$.
for $k = 0, 1, 2, ..., M$ **do**
  Collect a set of trajectories in the data buffer $\mathcal{D}$
  for $t = 0, 1..., T$ **do**
    $o_t, p_t \leftarrow$ Env.Get_States()
    $z_t \leftarrow \text{Encoder}_{\mu_k}(p_t)$
    $a_t \leftarrow \pi_{\theta_k}(s_t, z_t)$
    $r_t \leftarrow$ Env.Simulation_Step(a_t)
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, z_t, a_t, r_t\}$
  end for
  Compute rewards-to-go $\hat{R}_t$ and advantage estimates [Sch+16], $\hat{A}_t$ based on V_{ψ_k}.
  Update policy parameters by maximizing the PPO objective:
    $\theta_{k+1} \leftarrow \arg\max_\theta J_{PPO}(\theta, \hat{A})$,
  fit the critic via regression:
    $\psi_{k+1} \leftarrow \arg\min_\psi \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\psi(o_t) - \hat{R}_t\right)^2$
  Fit adaptation module by regression:
    $\hat{z}_t \leftarrow \text{Adaptation\_Module}_{\phi_k}(s_{t-N}, ...s_t)$
    $\phi_{k+1} \leftarrow \arg\min_\phi \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (z_t - \hat{z}_t)^2$
  $\mathcal{D} = \{\}$
end for

In our work, we used the RMA technique to learn policies that transfer to the Mini-Cheetah. We randomized the following values in the simulation: COM of the base, base mass, ground friction coefficient, ground restitution coefficient and motor strength. A similar scheme has been adopted for Mini-Cheetah [Mar+22]. The exact values of the randomization and noise used for each coefficient and state variable can be found in Table 4.6. We found that we can train the supervised learning step simultaneously with the optimization step of the RL objective. An algorithmic view of the RL training with this distillation approach can be found in Algorithm 1.

4.4.3 Experimental Results on Mini-Cheetah

For this experimentation stage, we switched the simulator from Raisim to IsaacGym [Mak+21]. IsaacGym offers a GPU-accelerated physics engine designed for robot learning tasks. It also provides a PyTorch [Pas+19] wrapper, through a Python-based Tensor API, for the physics buffer which enables direct access to the simulator states via PyTorch tensors, bypassing any bottlenecks that result from transferring data between CPU and GPU. We base our code on the legged gym repository [Rud+21] that uses a massively parallelized gym environments based on IsaacGym for legged robot learning tasks.

Observation space We decided to simplify the previous observation used in Section 4.3, since the policy will get the history of observations and actions via the adaptation module, we decided to put the basic proprioception and IMU measurements in the observation,

$$o_t = (\theta_{\text{body}}, \omega_{\text{body}}, q_t, \dot{q}_r, a_{t-1}). \tag{4.20}$$

We did not include the linear velocity as its estimation on the real robot would require knowledge of the contacts of the feet which are not measured on the robot [Kim+19].

Privileged observations The privileged observation vector is constructed from the elements of the dynamics that are randomized. p_t is an 18-dimensional vector composed of the ground friction coefficient (1D) , the ground restitution coefficient (1D), the robot's base center of mass (3D), the robot's base mass (1D) and the joint motor strength (12D). The motor strength is a scale factor that is applied to the computed torques of the PD controller in order to simulate possible mismatches in the actuators strength simulation. The range and values of the randomized elements can be found in Table 4.6 as well as the noise applied to the sensory observations.

Random Observations:		Random Dynamics:	
θ_{body}	$U^3(-0.05, 0.05)$	Motor Strength	$U^{12}(-0.9, 1.1)$
ω_{body}	$U^3(-0.20, 0.20)$	Ground Friction	$U(0.5, 1.0)$
q	$U^{12}(-0.05, 0.05)$	Ground restitution	$U(0.0, 1.0)$
$\dot{q}$	$U^{12}(-1.00, 1.00)$	Body Mass	$U(-1.0, 1.0)$
		COM displacement	$U^3(-0.2, 0.2)$

Table 4.6: Ranges and dimensions of uniform noise for randomizing the dynamics and observations in the Mini-Cheetah simulation experiments.

Action space The action space is similar to the one in Section 4.3 with a different scale factor $\lambda_q = 0.2$,

$$q_t^{target} = q_{init} + \lambda_q \Delta q_t^{\theta}, \tag{4.21}$$

with $K_p = 20.0, K_d = 0.5$ for the PD gain parameters.

Reward function The reward function is similar in nature to the one designed for Solo-12.

A scheme of the control architecture with the adaptation module is shown in Figure 4.17. The resulting polices transferred well to Mini-Cheetah and produced robust locomotion on a variety of terrains. We show snapshots of the policy running Mini-Cheetah on different terrains for a qualitative analysis of the produced locomotion. Figure 4.18 and Figure 4.19 show snapshots of our robot moving in an indoor environments with mats under its feet. Mats mimic moving ground and feet contact slippage yet the policy is unphased by these conditions. Figure 4.20 shows the robot moving from a rough ground with small pebbles to a harder concrete ground crossing over a pavement. The policy is able to make the transition seamlessly. We also found some icy ground with frozen water over wood. This extremely slippery surface causes the legs of the robot to suddenly extend and slip. The policy reacts robustly to the difficulties on the ground without falling in Figure 4.21. The size and torque strength of Mini-Cheetah allow it to run on more difficult terrains that require faster reactivity, which is in contrast to the capabilities of a lighter robot like Solo-12 .

We also observed robust behaviours that were unexpected. In Figure 4.22, we tried to make the cheetah to go up a very steep ground that was very muddy due to rain. The terrain is very challenging for Mini-Cheetah's size and squash ball feet. We expected the robot to completely fail in this attempt. however, while the robot was about to fall,

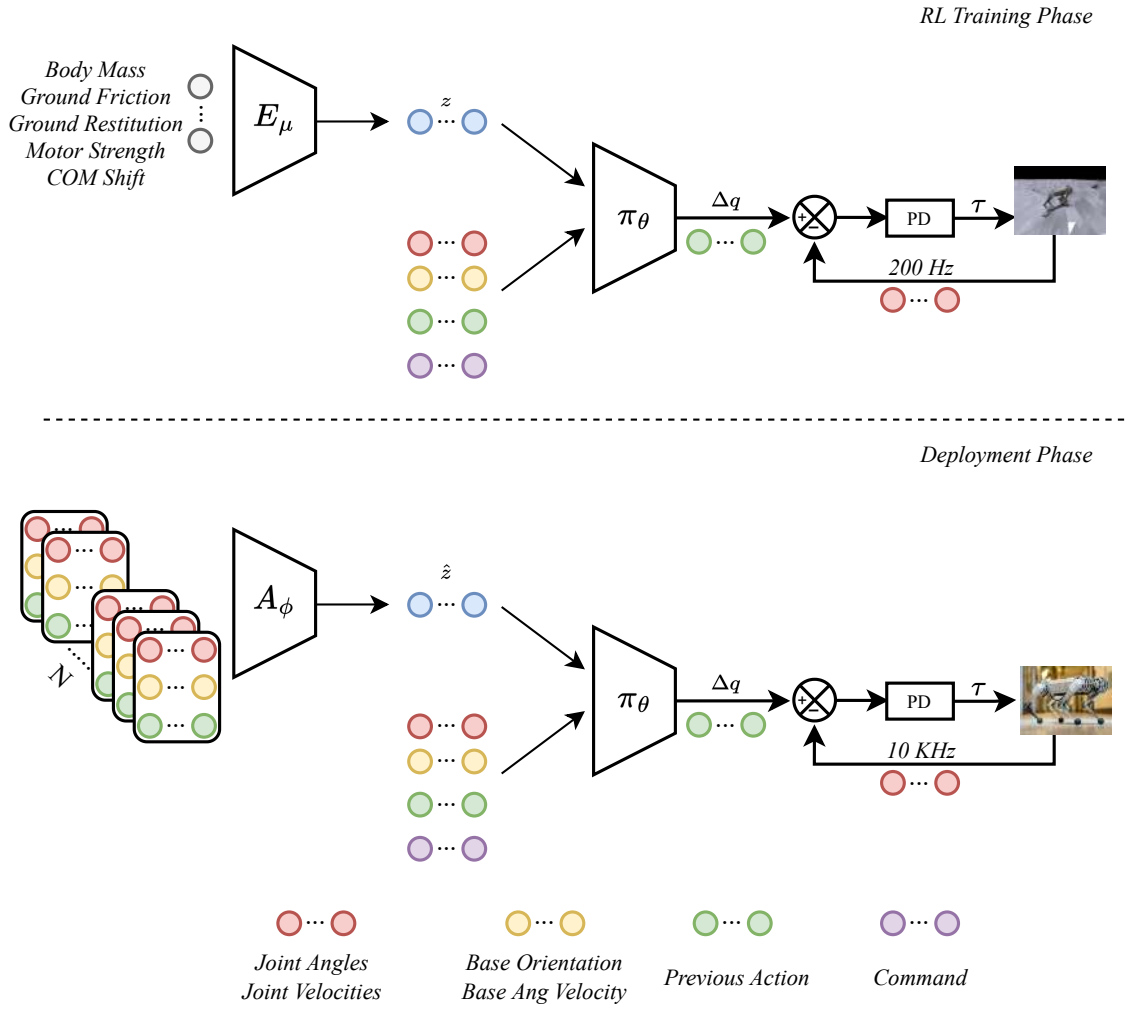
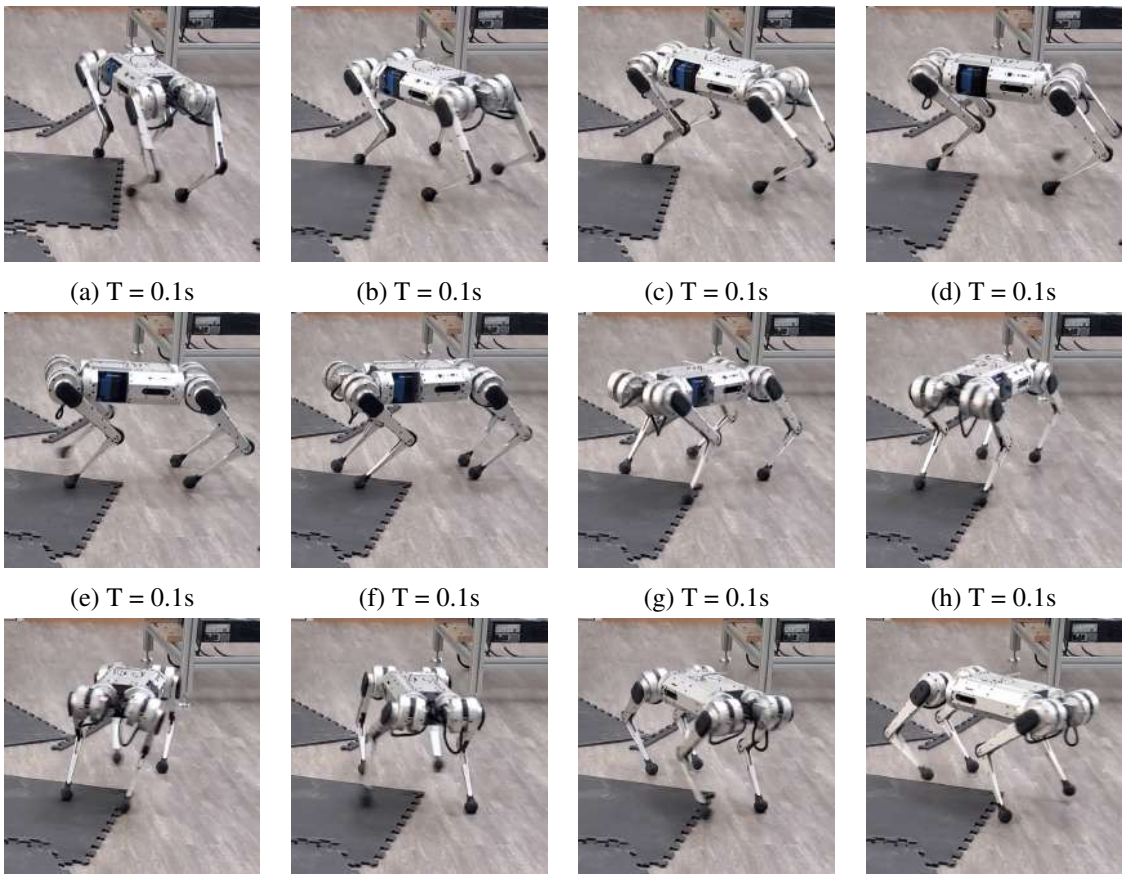


Figure 4.17: Scheme of the control architecture used for in the Mini-Cheetah experiments. During RL training, the encoder receives the privileged information about the dynamics of the environment and feeds and representation of them to the policy. At deployment the encoder is replaces with an adaptation module that is trained to predict the latent representation from the history of the observations and actions. The state, actions, commands and the latent representation are fed to the policy network that outputs joint displacement offsets. A PD controller turns the network's actions into torques.

it suddenly made a very quick shuffling motion of its four legs which allowed it to gain balance. Therefore, though the robot was not able to climb the ground, it was able to make a recovery motion when it should have fallen. Such behaviour is the product of training with different dynamic terrain parameters (as shown Table 4.6) and the mixture of random rough grounds in IsaacGym [Fu+21; Rud+21].

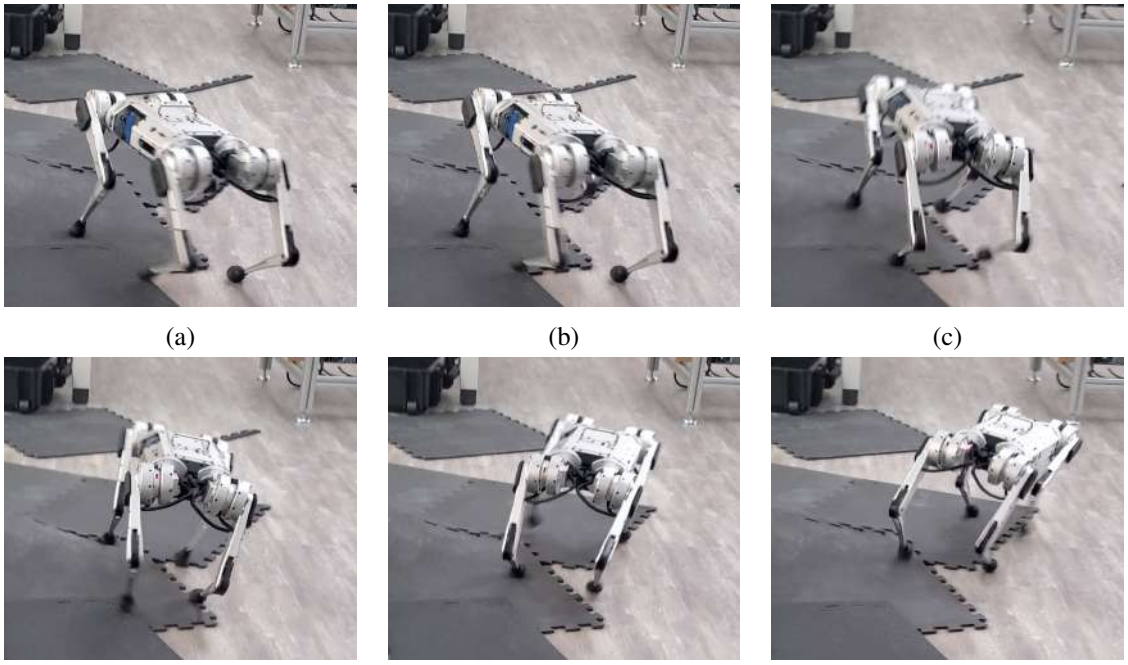


(a) T = 0.1s (b) T = 0.1s (c) T = 0.1s (d) T = 0.1s

(e) T = 0.1s (f) T = 0.1s (g) T = 0.1s (h) T = 0.1s

(i) T = 0.1s (j) T = 0.1s (k) T = 0.1s (l) T = 0.1s

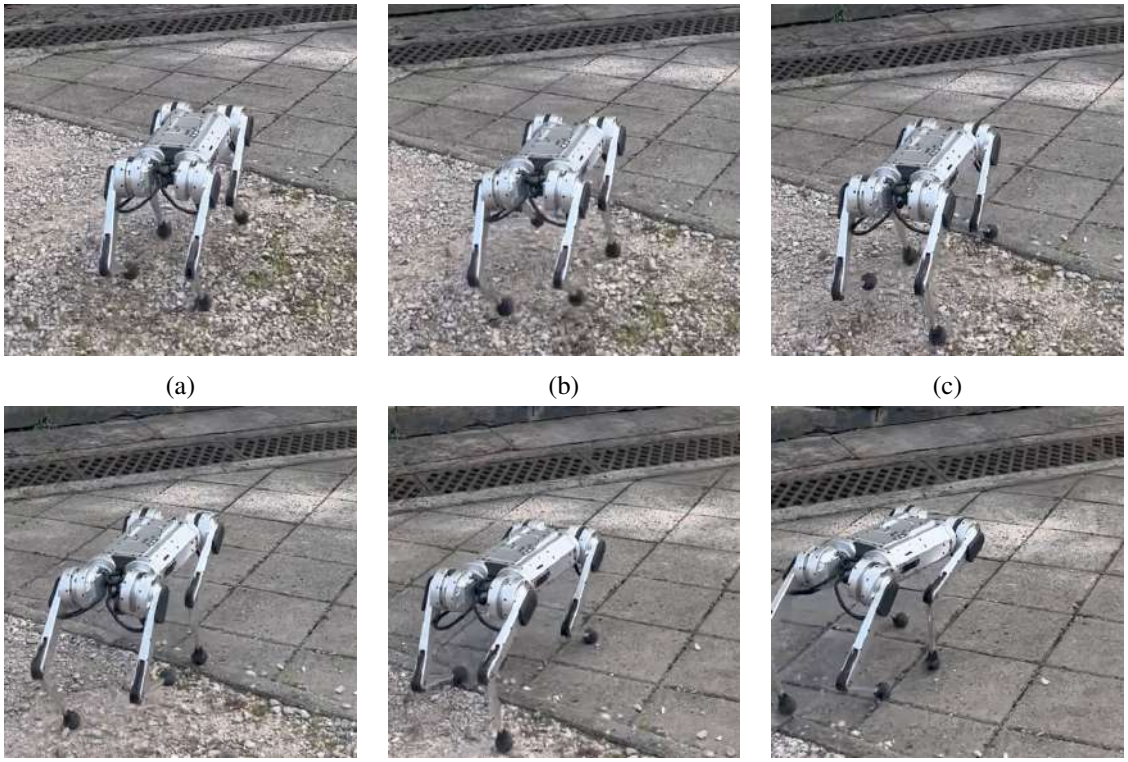
Figure 4.18: Rollout of the learned policy on Mini-Cheetah where the robot is moving over a flat ground in the presence of mats. The mats give the effect of slipping as they easily move when the robot steps on them. However, the learned controller is able to perform the locomotion successfully without any problem. Video of this run can be found: https://youtu.be/CFLp9xTqSwI.



(a) (b) (c)

(d) (e) (f)

Figure 4.19: Another rollout of the Mini-Cheetah policy on the mats where the mats are tilted to give the effect of a slope. Video of this run can be found here: https://youtu.be/CFLp9xTqSwI.



(a) (b) (c)

(d) (e) (f)

Figure 4.20: Rollout of the policy on Mini-Cheetah in outdoor environments. We see the robot navigating from rough terrains to pebbles and rocks to a more regular hard terrain without any problem.

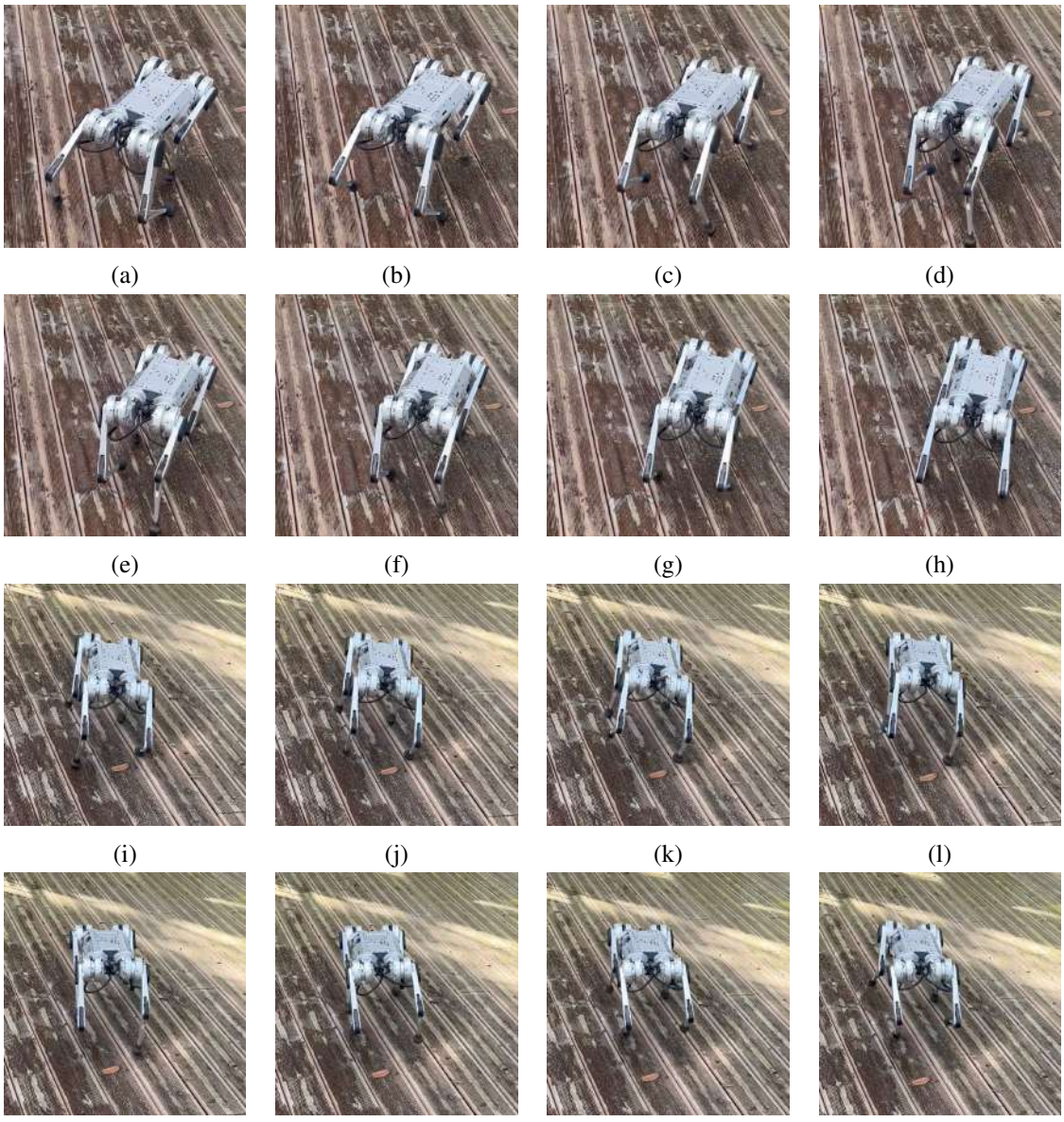


Figure 4.21: Rollout of the learned policy on Mini-Cheetah on patches of frozen ice over a wood terrace. The legs are constantly slipping due to the disturbances of the ice, but they rapidly return close to the nominal pose and the robot can keep its balance while maintaining its velocity. Video of this run available here: https://www.youtube.com/shorts/uwz3pu9TmLk.

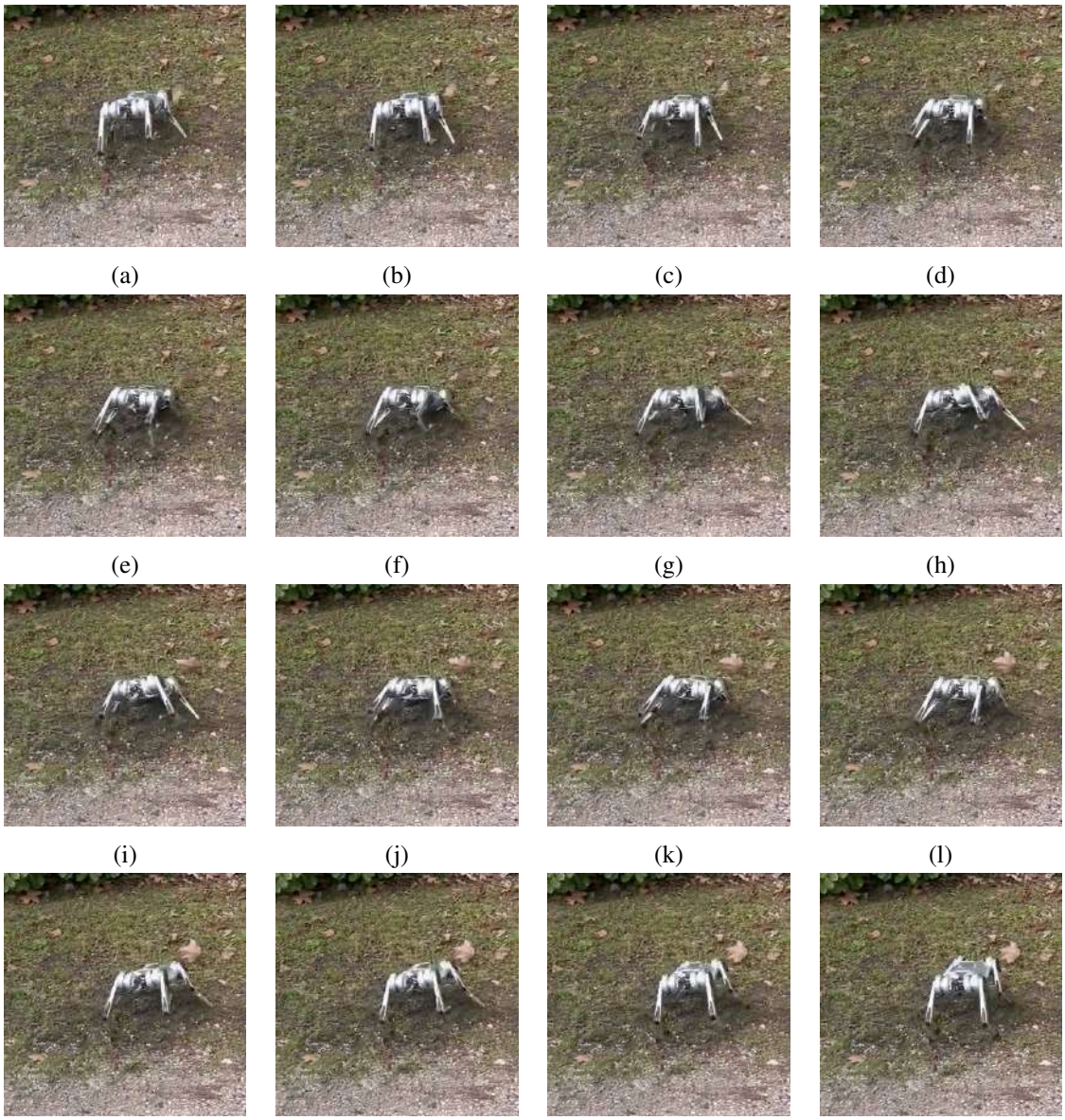


Figure 4.22: Rollout of the learned policy on Mini-Cheetah. The robot is guided through a joystick to move towards a very steep and muddy slope that would be very hard to traverse for a robot of its size. The robot fails to climb up the slope but when it is about to lose balance and fall, it performs a rapid shuffling movement that causes it to regain balance. Video of this run can be found here: https://youtu.be/QIfazKDNGIs.

4.5 Applying the RL Method on a Custom-made Robot

During this thesis, we applied the RL approach that we designed for Mini-Cheetah on a new robot called Sassa that was custom-made and built by the Gepetto team at LAAS-CNRS [Tea] under the expertise of Thomas Flayols. With the Gepetto team, we participated to the IEEE ICRA 2023 quadruped challenge [Jeo] that aimed at pushing the limits of quadruped control by proposing challenging environments with different slopes and on different terrain structures that the quadruped had to traverse. The Sassa was built just before the challenge and we found that we could extend the RL approach from Mini-Cheetah to Sassa very quickly and more easily than extending an optimal controller. The Sassa robot is similar in structure to the Mini-Cheetah robot, only a little larger. We found also that the custom-designed drivers from the open motor driver initiative [Ini] were less noisy than the ones on Mini-Cheetah and we reduced the noise magnitude for the joint angles and velocities and the base orientation in the domain randomization part. Figure 4.23 shows the real Sassa robot and its setup in simulation that was trained to track velocity.

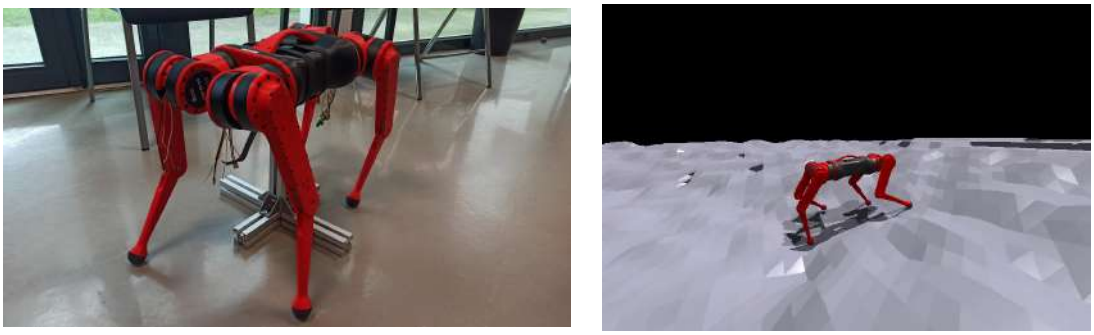


Figure 4.23: **Left:** The real Sassa robot. **Right:** The Sassa robot in the IsaacGym simulation learning to traverse a rough slope terrain.

Further development is needed on the hardware and control policy to get the same impressive results that we got for Mini-Cheetah. However, this was another evidence of the usefulness of RL-based controllers since we were able to quickly develop a training method for Sassa that was able to transfer on the robot and we were able to display the robot in ICRA 2023.

4.6 Possible Extension: Learning Bipedal Locomotion

Bipedal robot control poses a challenging problem, more complicated than quadruped control. Maintaining balance on two legs is a dynamic process that involves constant adjustments in response to internal joint variations and external forces like uneven terrain and disturbances. Bipedal robots must continually sense their surroundings and make rapid adjustments to prevent falling. Achieving this level of stability requires sophisticated control algorithms and advanced sensor systems.

Platforms like Talos from PAL Robotics [Sta+17], HRP-5P from AIST [Kan+19] and Cassie from Agility Robotics [RMA19] and many others allow researchers to explore the problem of bipedal locomotion. However, they are quite expensive, heavy and require considerable effort and manpower for maintenance. With the aim of providing a simple and open platform, the Open dynamic robot initiative provides Bolt (Figure 4.24), a simple lightweight biped robot made of two Solo legs. This provides a robot that is easier to

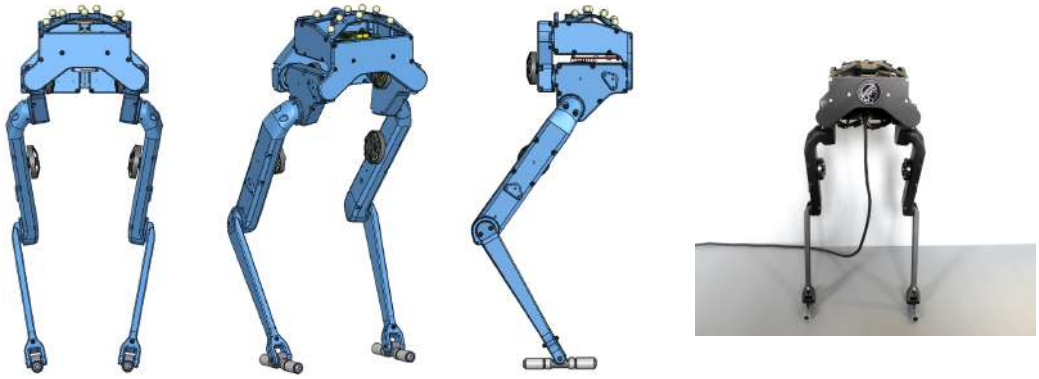


Figure 4.24: The Bolt biped CAD model (left) and the real robot (right).

work and to explore topics around bipedal control [Bor+21].

We have attempted to extend the end-to-end RL approach, of learning velocity tracking policies for Solo-12, for the Bolt biped. Other works have recently explored the possibility of applying RL to bipedal robots. Li et al. [Li+21] proposed an RL based approach for controlling the Cassie robot in velocity tracking, base height tracking and angular yaw tracking. The authors proposed adding parameters that specify a gait pattern as an additional reference to guide the exploration phase. Li et al. [Li+23b] built on that approach and proposed a multi-stage training process that made Cassie perform jumps up to 1.4 meters. In Radosavovic et al. [Rad+23], a novel approach of using autoregressive prediction of future action is proposed for the task of bipedal locomotion. The learned controller shows impressive robustness when deployed on the Digit robot, the new biped prototype from Agility Robotics. Even for robots like the HRP-5P [Kan+19], which suffers from large armature and low backdrivability of its joint, that makes the robot hard to simulate, therefore increasing the Sim2real gap, Singh et al. [Sin+23] managed to learn a policy that transfer to the real robot by relying on current feedback to address the poor torque tracking on the real system.

We adapted the approach described in Section 4.3 for learning velocity tracking tasks for Bolt. Bolt is a 6-dof robot that is torque actuated. We kept the PD control strategy with the same gains since its the same actuators as Solo. We kept the same reward function, but replaced the energy consumption term with only a penalty on the torque magnitude. The initial experiments failed to keep the upright posture expected of bipedal robots. Therefore, We added a base height tracking reward function:

$$r_t^{base_z} = -\|z_t^{base} - z^{desired}\|_2^2, \tag{4.22}$$

where z_t^{base} is the base height at time t and $z^{desired}$ is the desired base height which was set to 60 cm.

Figure 4.25 shows snapshots of the Bolt biped in the Raisim simulator. The robot is commanded to go forward at a speed of 1 [m/s]. The policy manages to drive the robot forward. However, it appears that to obtain a better performance, a lot of improvements would be necessary. For instance, the policy learned a jumping motion while walking rather than alternating the legs. The policy also failed to stand in place at zero velocity, but learned to move in small displacement in order to keep balance. However, the latter issue is more likely related to the feet of the robot as currently they are simulated as sphere and Bolt lacks a good feet design. More discussion around possible extention of this work is discussed at the end of the thesis.

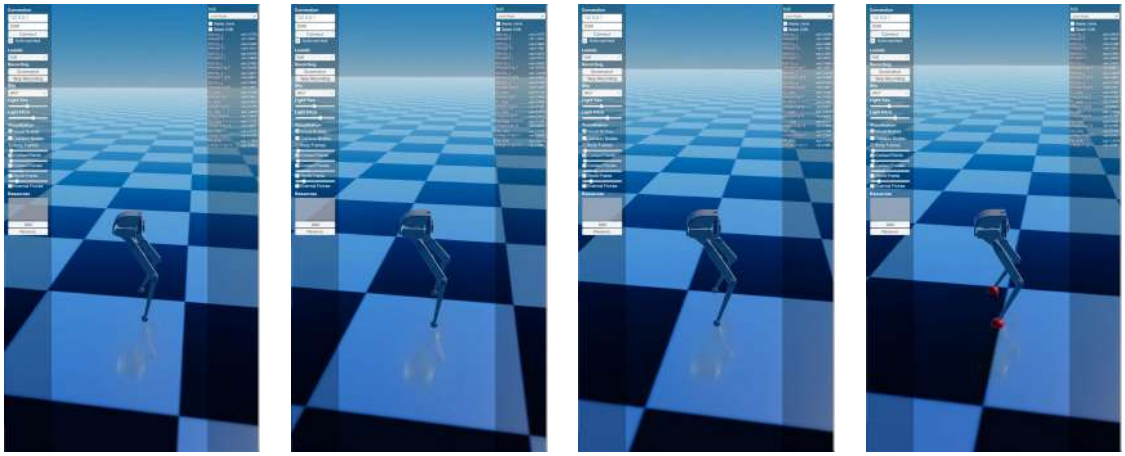


Figure 4.25: Snapshots of the the Bolt biped in the Raisim simulator controller by the the learned policy.

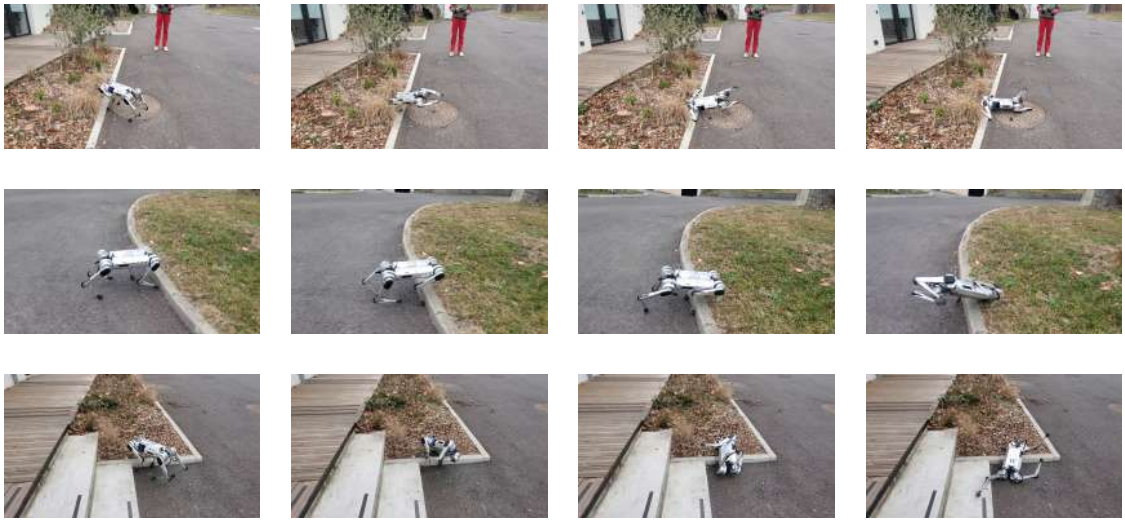
My contribution to the development of RL-based control of Bolt was limited to this first approach. The developed experiments on Bolt were exploited by a group of master interns at ISAE-SUPAERO. This led only to a proof of concept and showing how flexible learned controllers can be as the base learning algorithms can be adapted to other problems rather easily.

4.7 Limitation of the Approach

When deploying these policies on different platforms we observed some failure cases. We can think of two main points of criticism towards these approaches.

Robustness vs. deliberateness. Generally, end-to-end learned controllers have displayed impressive robustness in our work and in the related literature [Ara+23b; Lee+20; Mik+22]. However, this robustness still has its limits. For example, in Figure 4.26 we see snapshots of the robot attempting to walk over pavements, obstacles and steps and immediately failing. One could argue that vision is needed to detect these different levels, but for some steps one would expect the policy to be able to adapt its actions in an agile manner and still cross them. The complexity and proficiency of the learned policy can only be as good as the experience collected during the exploration phase of the RL training. Therefore, if the robot has not encountered such situations, as the ones seen in Figure 4.26, then it will not be able to adapt to them.

Reward function complexity. Another point of criticism is made around the complexity of the proposed reward function to output the required stable and robust locomotion. From an RL perspective, the point of trial-and-error learning is to arrive at a solution, from the agent's experience in the environment, in order to maximize a reward that guides the learning towards *which* task to solve. However, in our proposed reward function where there are around 10-14 reward terms, including costs and penalties, we are not only defining the task, but we are effectively going back to a model-based perspective where we are guiding the learning towards *how* to solve the task. Injecting many constraints in the reward defeats the purpose of learning from scratch as we are adding prior knowledge of the solution that may or may not be well modeled in the environment. Moreover, it poses a practical challenge to find the right way to tune the many terms in the reward. However, when finding the right tuning and with the existing fast simulators, one is able to produce



(a) T = 0.1 s (b) T = 0.2 s (c) T = 0.3 s (d) T = 0.4 s

Figure 4.26: Examples of three failed trajectories on the real robot. When the robot faces certain obstacles at certain heights, it is unable to rapidly lift its feet and recover.

a robust policy in only a few hours, which is faster than adapting an existing model-based controller to the robot.

[Fu+21] argues and shows that learned policies are as complex as the terrain and environment that they have been trained on. For example, if the policy is trained on a flat ground, then it could converge to not lifting its feet very high even in the presence of the foot height reward. In Fu et al. [Fu+21], the authors trained a quadruped on a very difficult terrain with a simple velocity tracking and energy saving reward function. This way, they address the points of criticism that have to do with the elaborate reward function and complexity of the produced policies. After many training iterations (in the scale of billions of samples) they are able to learn a robust policy that transfers well to the robot.

In the next step of this thesis, instead of following a similar approach to Fu et al. [Fu+21], We decided to take advantage of the elaborate reward function to produce complex policies that can adapt their behaviour following several set of commands, e.g., swing feet height, step length, exerted power, etc. This approach will be developed in the next chapter.

4.8 Conclusion

In the first part of this chapter, we presented an end-to-end approach for learning controllers for the Solo12 quadruped robot. We first investigated the possibility of learning direct torque control for solving simple tasks with simple reward function. We found that the policy converges to patterns that looks "unnatural" due to the awkward joint angles and body motion produced when deploying them on the robot. On the real robot such trajectories are not safe and do not resemble motions that are optimal.

We then developed a training method that features more complex state, reward and action designs. We described, in detail, the choice of state space, action space and reward function along with the curriculum strategy and domain/dynamic randomization method, that we made for learning transferable policies for following 3D velocity commands. We

presented results for the velocity tracking and energy loss. For deployment on the real root, we utilized a supervised learning approach to learn a state estimation network that can predict the linear velocity of the base of the robot and give it to the Solo-12 policy as input. Numerous experimental tests on the real robot have shown that robust locomotion policies with different energy profiles can be learned by randomizing the weights of the power loss variables. The results are robust policies that can control Solo-12 in indoor and outdoor environments that are complex compared to the size and weight of the robot.

Later, we adapted the end-to-end approach for learning policies for the MIT's Mini-Cheetah robot. We found that the transfer methodology relying only on domain and dynamic randomization was not enough to produce stable trajectories for long runs on the real robot. We adapted rapid motor adaptation on Mini-Cheetah in IsaacGym to provide the policy with a latent vector that constitutes a representation of the dynamics of the environment and adds other type of information that are useful to the policy to make the right decisions. The results were robust policies that can achieve locomotion over very rough, muddy, grassy and icy terrains.

Finally, we started investigating the development of such RL approaches for the control of a small biped robot Bolt. Even though we could obtain stable walking behaviour, further development would be necessary to obtain reliable locomotion control. we did not develop further in this direction.

Chapter 5

Hierarchical Policies for Locomotion

Contents

5.1 Introduction

In much of the current works on locomotion, whether they come from the model-based literature [Di +18; Kim+19; Léz+20] or from model-free literature [Hwa+19; Lee+20; Kum+21], the objective has been the same: produce robust locomotion that follows a desired velocity command while minimizing the used energy. However, this is a very abstract definition of the locomotion task. As we have seen in this thesis, many low-level features of the motion have to be constrained and penalized, e.g., how much to lift the feet, the deviation of the initial pose, or the best nominal joint pose to center the movement around, etc. These constraints are often determined by fixed parameters that are devised by analyzing the morphology of the robot and rigorous testing.

These different decisions intrinsically make the locomotion problem a highly multi-task one at different levels regarding the motion of the base and the joints [Kim+19]. At the

(a) T = 1s (b) T = 2s (c) T = 3s (d) T = 4s (e) T = 5s

Figure 5.1: Example of running a policy with different desired feet height parameters. The row above shows consecutive snapshots of the robot attempting and failing to climb stairs with a 6 cm desired foot height command as it gets stuck on the first few stairs. The row below shows the same policy succeeding because it was commanded to lifts its feet to 8cm, which is higher than the stairs step height.

high-level, the velocity command determines the trajectory of the center of mass (COM) of the robot, but in more complex environments that require precise intended movements, e.g., going up the stairs or reaching a given location, the velocity on its own is not enough. At the same time, the low-level control of the joint pattern might also need to be modified for such specific tasks. For example, on rough terrains the robot might need to lift its feet higher, and in steep slopes the robot would need to take smaller steps or use more torque [Ton+18].

To demonstrate this point, we conducted an experiment with the Raisim training environment of Solo-12. The experimental setup is the same as the one described in Section 4.3.1, however, instead of training for one value of the desired foot height positions p_z^{max} as in Section 4.3.1, we train the policy to follow two values 6 cm and 8 cm and provide this parameter as input to the policy, so that it can be asked to lift its feet for different height targets. What we found was that the policy was able to learn to lift its feet differently for the two given values of p_z^{max}. Figure 5.1 shows the same policy when communicating different desired foot height parameters. The terrain is composed of stairs with size of 6 cm. The first row shows snapshots of the policy commanded with $p_z^{max} = 6$cm. The robot fails to climb up the stairs. The second row shows the policy with $p_z^{max} = 8$cm and how the robot successfully climbs the stairs. This example shows how useful and important it is to have the ability to adapt some of the fixed parameters that define the behaviour.

We see that there is a clear need to augment the task description and have the ability to convey the desired behavior to dictate the low-level motion. We tackle the multi-task aspect of locomotion using hierarchical reinforcement learning (HRL). With HRL we can decompose a reinforcement learning problem into simpler sub-tasks that are easier to design and learn [Pat+21].

We propose learning a low-level policy with an elaborate reward function that governs many cost terms related to the desired joint-level control. Along with the state, this low-level policy is augmented with *command parameters* which are variables and weights

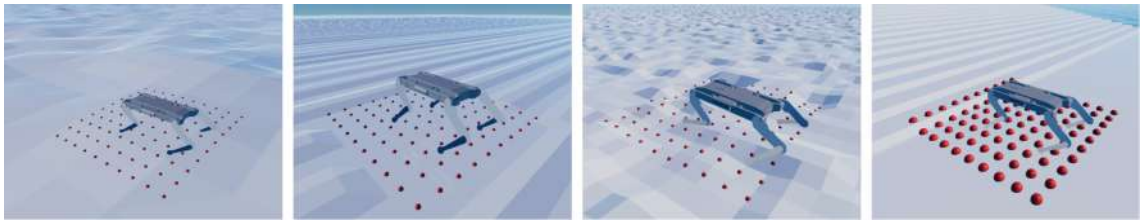


Figure 5.2: Example of the height scan that provides measures of the terrain surrounding the robot. This information is given to the high level policy to infer the right low-level behavior needed to achieve proper locomotion.

that define the terms of the reward function and details of the state dynamics. We use the term *command* here because we view these parameters to be similar to the velocity commands that generally dictate the locomotion task. In addition to the low-level policy, we propose learning a high-level policy that would have access to terrain readings (as shown in Figure 5.2) to determine the suitable command parameters in the environment.

Outline. The main contribution of this chapter is the design and implementation of a hierarchical deep RL scheme that augments the baseline locomotion policy with a command parameter vector ω that manipulates different aspects of the locomotion (swing feet height, step length, initial pose and PD gains). This baseline policy can then be used in a hierarchical setup where a high-level policy determines suitable command parameters to adapt the locomotion to different situations. We show that this two-level design improves the performance of velocity tracking policies by reducing energy consumption and velocity tracking error in various situations while having better sample efficiency. We also show how these policies enable the real robot to cross different structured terrains on which the baseline policy would regularly fail to be robust. The chapter is organized as follows. Section 5.2 discusses the related literature. Section 5.3 introduces some notions and definitions. Section 5.4 presents in detail the approach and how the RL policies are designed and trained. Finally, the experimental results including simulation and experiments on the real robot are outlined in Section 5.5.

5.2 Related Literature

Hierarchical reinforcement learning methods have been proposed for learning locomotion strategies, for example by learning separate policies for recovery, standing and locomotion and then learn a high-level policy that switches between them [LHH19]. Hierarchical methods, in similar spirit, have also been developed in planning-based algorithms. The work in [Ton+18] proposes a contact planner for locomotion in tasks where there are multiple stages of contact and decomposing the planning into sub-problems is necessary. A two-level approach was proposed in [Pen+17b] to control a biped in simulation; where a low-level policy learns to follow desired footstep placements and a high-level policy learns to place the footsteps based on the terrain information. In our work, we propose embedding a policy with several aspects related to the control and the reward function, such that the behaviour of the low-level policy is very flexible and can be adapted in different dimensions via a high-level policy. We also show the effectiveness of the low-level policy on the real system in challenging terrains and not only in simulation.

The proposed approach to learn parameterized policies for quadruped locomotion follows an idea similar to the one developed in [Cho+20]. However, we also use parameters like the gains of the low-level impedance controller that are not part of the reward, but can have a desired effect on the motion. Recently, parameterized policies have also been introduced for versatile quadruped locomotion but without showing benefits of hierarchical RL [MA22].

In the literature of skill learning, some works propose to learn separate skills for different purposes [LHH19; Li+22b; FXP22; Li+23b] often via imitation learning of trajectories generated by an optimal controller. Here, we are more interested in learning a general locomotion policy that we can control to accomplish specific tasks. Though it is also possible to follow a hierarchical approach to learn locomotion policies from visual input [JIC20], the problem we consider in this work is to learn the best blind low-level locomotion policy so that we can control several aspects of it (foot height, stride length). Since our proposed policies are reactive, there is no planning involved.

5.3 Preliminaries

5.3.1 Low-level Parametric Policies

We model the low level learning environment as a set of Markov Decision Processes (MDPs), indexed by parameters ω, with common, continuous state and action spaces [SB18]: $\mathcal{M}_\Omega = \{(\mathcal{S}, \mathcal{A}, \mathcal{R}_\omega, \mathcal{P}, \rho_0) | \omega \in \Omega\}$, where $\mathcal{S}$ is an infinite set of states, and $\mathcal{A}$ is an infinite set of actions. In each $M_\omega \in \mathcal{M}_\Omega$, taking an action a in a state s yields a reward, defined by a function $\mathcal{R}_\omega : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The environment dynamics is described by a conditional transition probability distribution $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, with the interpretation that $\mathcal{P}(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$ is a probability (density) that the next state is s' given that the current state is s, and the action taken is a. ρ_0 is the initial state probability distribution. We assume parameters $\omega \in \Omega \subset \mathbb{R}^d$ to be sampled from a static distribution ρ_Ω, and define the learning task to be finding the common parameters θ for all stochastic policies $\pi_\theta^\omega : \mathcal{S} \to P(\mathcal{A}|\mathcal{S})$ in order to attain a maximum expected discounted sum of rewards:

$$J(\theta) := \mathbb{E}_{\rho_\Omega} \mathbb{E}_{\pi_\theta^\omega, \mathcal{P}_\omega, \rho_0} [\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_\omega(s_t, \pi_\theta^\omega(s_t), s_{t+1})], \tag{5.1}$$

where $\gamma \in [0, 1]$ is a discount factor. In practice, π_θ^ω is implemented as a neural network thus θ corresponds to its weights.

5.3.2 High-level Parameter Controlling Policy

Given a parameterized set of MDPs $\mathcal{M}_\Omega$ (as described above) and a parameterized set of low-level policies π_θ^Ω, indexed by Ω, we define the high-level control setting as an MDP

$$\mathcal{M}^H = (\mathcal{S}^H, \Omega, \mathcal{R}^H, \mathcal{P}^H, \rho_0^H), \tag{5.2}$$

where the superscript H stands for "high-level". The state space $\mathcal{S}^H = \mathcal{S} \times \mathcal{S}^h$ may also contain additional information $\mathcal{S}^h$ (e.g., vision data) and the action space consists of the set of low-level parameters Ω. The transition dynamics is supposed to respect the

low-level dynamics so that it obeys

$$\int_{s'^h} p^H((s', s'^h)|(s, s^h), \omega) ds'^h = \mathcal{P}(s, \pi_\theta^\omega(s), s'), \tag{5.3}$$

for all $s, s' \in \mathcal{S}$, $s^h \in \mathcal{S}^h$, and $\omega \in \Omega$.

The high-level learning objective is to find the parameters ϕ for a high-level stochastic policy $\pi_\phi^h : \mathcal{S}^H \rightarrow Pr(\Omega|\mathcal{S}^H) : \pi_\phi^h(\omega|s) = p_\phi(\omega_t = \omega \mid s_t = s)$ that optimizes the expected discounted cumulative reward:

$$J^H(\phi) := \mathbb{E}_{\pi_\theta^\Omega, \pi_\phi^h, \mathcal{P}^H, \rho_0^H}[\sum_{t=0}^{\infty} \gamma_H^t \mathcal{R}^H(s_t^H, \omega_t, s_{t+1}^H)], \tag{5.4}$$

where $\gamma_H \in [0, 1]$ is a discount factor, $s_t^H = (s_t, s_t^h)$, and ω_t is computed by the high-level policy $\omega_t = \pi_\phi^h(s_t^H)$.

5.4 Learning Parameterized Locomotion Policies

In this section, we will first outline the main procedure for learning general end-to-end locomotion policies on the Mini-Cheetah quadruped. The low-level policy has to be robust while following a set of references determined by the command parameters. The learned policy also has to transfer to the real system. We will then describe the parameters that the low-level locomotion task is conditioned on. Finally, we introduce the high-level policy that adapts the parameterized locomotion. A general scheme of the hierarchical architecture is depicted in Figure 5.3 top.

5.4.1 Low-level Policy

The goal of the low level policy is to produce joint angle control based on the state of the robot and the chosen command parameters.

Observation space. The observation of the robot mainly depends on the IMU readings and the proprioception of the joints. The observation at time t is:

$$o_t = (\theta_{\text{body}}, \omega_{\text{body}}, q_t, \dot{q}_t, a_{t-1}), \tag{5.5}$$

where θ_{body} is the orientation of the base, ω_{body} is the angular velocity of the base, q_t and $\dot{q}_t$ are the joint angles and velocities respectively. We also include the previous actions a_{t-1} to the observation.

State space. Observations in the real system are noisy and the dynamics of the robot and terrain could vary depending on the situation. In order to learn policies that are robust and transferable, one needs to inject noise and randomization in the observation space and environment's dynamics [Hwa+19; Mik+22]. Therefore, relying on a single timestep observation as input to the control policies would often hinder both learning and transfer.

We use the same RMA-based approach in the experiments of this chapter (as detailed in Section 4.4). However, we proposed to augment the encoder input with privileged state information related to the robot's feet positions and contacts, and the base linear velocity,

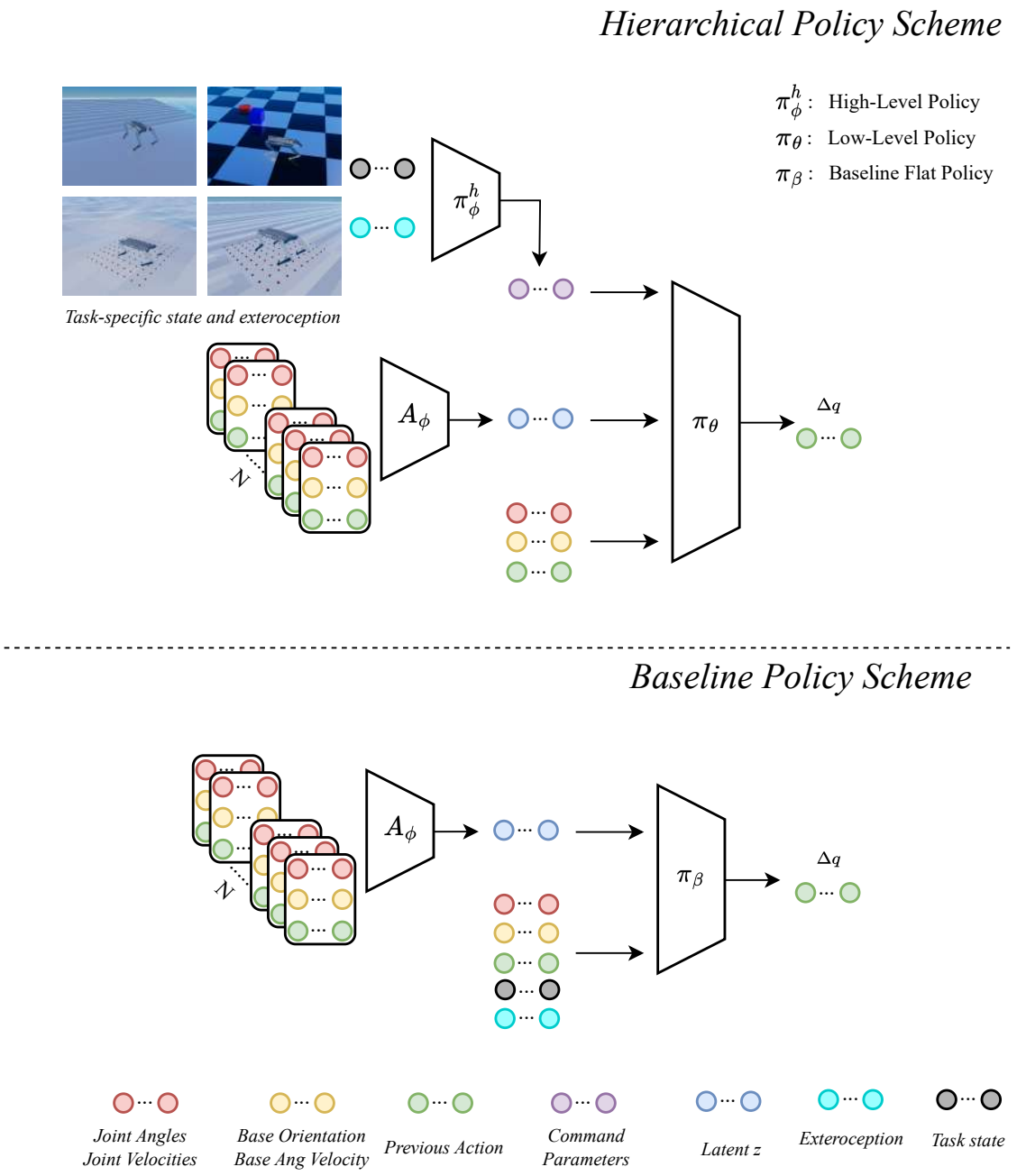


Figure 5.3: **Top:** Full control architecture of the hierarchical policy scheme. The high-level policy get task related information in order to actively control the low-level behaviour by actively adjusting the command parameters and external sensing of the terrain and environment. It then outputs the suitable command parameters to achieve the task. The low-level policy is fed the output of the high-level policy, observations and the latent representation of the dynamics and outputs joint displacement offsets. The low-level policy and the high-level policy are trained separately. **Bottom:** A flat baseline that takes the task and exteroception directly as input to the policy, the baseline has to learn the locomotion behaviour and the task simultaneously.

that is usually not present in the original RMA work [Kum+21]. Ablation studies (see Section 5.5.1) indicate that this helps learning low-level policies that reach the commanded references better and with less training iterations. A full scheme of the RMA approach is depicted in Figure 5.4.

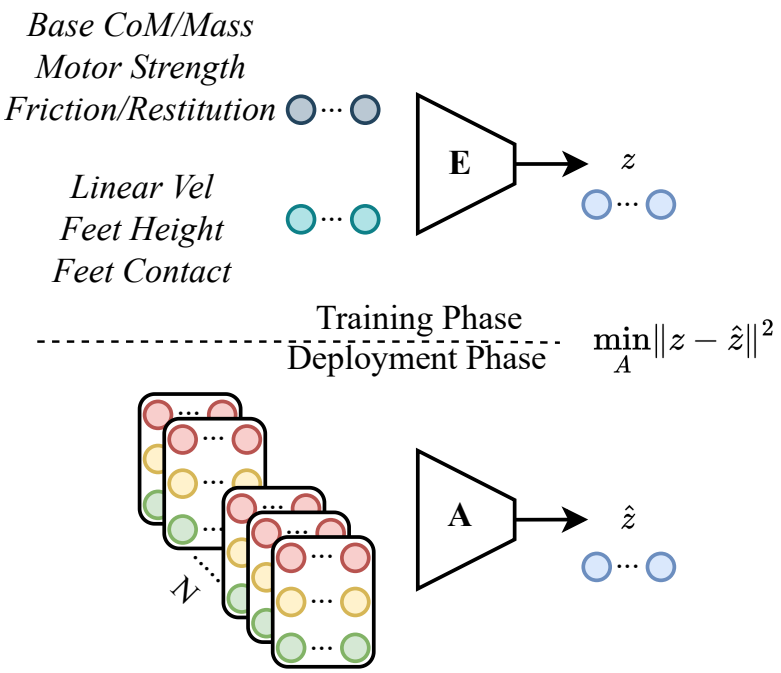


Figure 5.4: The environment encoder E encodes a representation of the privileged information into z. The adaptation module A learns to regress the output of the encoder from the history of the observations and actions to predict $\hat{z}$.

Reward function The objective of the reward function is to achieve a specified base velocity while minimizing costs associated with acquiring resilient and stable locomotion skills. A detailed breakdown of the reward function components and weightings is presented in Table 5.1. $V^{base}_{x,y,yaw}$ refers to the 3D vector consisting of the measured base linear velocity along x and y and angular yaw velocity. $p_{f,z,i}$ is the i^{th} foot height in world coordinates, while $p^{max}_{f,z}$ refers to the desired foot height in world coordinates and $V_{f,xy,i}$ is the i^{th} foot's linear velocity along x and y. The foot clearance reward depends on how close the current foot height is to the desired one, scaled by the foot velocity, e.g., the faster the feet are moving the higher the robot should lift its feet. The total reward function is the weighted sum of these terms. The main positive term in the reward function is related to tracking the desired velocity, while negative costs are used for refining the movement in terms of action magnitude and smoothness, for raising the feet, and maintaining an appropriate posture and base stability.

Locomotion command parameters. The reward design is similar to previous studies on end-to-end RL for locomotion [Lee+20; Hwa+19; Mik+22] and more specifically our work [Ara+23b] that was detailed in section 4.3. However, in these previous works, the velocity command was usually considered the main command parameter in the reward function. When training in simulation it is randomly sampled. On the real robot, it is controlled by the user with a gamepad device. The approach developed in this chapter considers more elaborate command parameters that are hard to be prescribed by the user. The parameters which constitute the vector ω given to the policy is:

$$\omega = (V_{cmd}, p^{max}_{f,z}, c_{jpos}, Kp, Kd, q_{nominal}),$$

where V_{cmd} is 3D the velocity command, linear along x, y and angular about z. $p^{max}_{f,z}$ is the maximum foot height to reach in the swing phase, which is part of the foot clearance reward, c_{jpos} is the coefficient that weights the joint angle deviation penalty, Kp and Kd

Reward Function	Formula
Velocity tracking	$c_{vel} exp(-\|\|V^{cmd} - V^{base}_{x,y,yaw}\|\|^2/\sigma_{vel})$
Base z velocity	$c_{vz}\|\|V_z\|\|^2$
Joint angle deviation	$c_{jpos}\|\|q_t - q_{nominal}\|\|^2$
Joint velocities	$c_{\dot{q}}\|\|\dot{q}_t\|\|^2$
Joint accelerations	$c_{\ddot{q}}\|\|\dot{q}_t - \dot{q}_{t-1}\|\|^2$
Joint torques	$c_\tau\|\|\tau\|\|^2$
Orientation	$c_\theta\|\|\theta_{roll,pitch}\|\|^2$
foot clearance	$c_{fcl}\sum_i^4(p_{f,z,i} - p^{max}_{f,z})^2\|\|V_{f,xy,i}\|\|^{0.5}$
action smoothness	$c_{smooth}\|\|a_t - a_{t-1}\|\|^2$

Table 5.1: Reward function terms and coefficients. The total reward is a weighted sum of all the terms above.

are the proportional and derivative gains for the PD controller, and q_{init} is the initial pose around which the action space is centered.

Action space. The network π_θ outputs the joint angle targets that are fed to a PD controller with zero joint velocity targets. We center the action space around the nominal joint angles at the initial position of the robot, so that the PD target at time t is $q_t^{target} = q_{nominal} + \lambda\pi_\theta^{\omega_t}(s_t)$. This centering is essential for the policy to learn and allows us to control the limits of the joint angles from the initial position. In our experiments we found $\lambda = 0.3$ to be the best choice.

5.4.2 High-level Policy

The actions of the high-level policy are locomotion parameters that modulate the low-level policy. The main intuition is that the elements of ω can be varied to change the behaviour of the robot while still producing stable locomotion that is learned by the low-level policy.

The high-level policy π_ϕ^h can control V_{cmd} when it is not tuned by a user, like for example, in an autonomous point goal navigation task. The maximum desired foot height, $p^{max}_{f,z}$, is an important parameter that has consequences on transfer [Ji+22a; Ara+23b]. On complex terrain the robot might need to lift its feet higher than on flat terrain where the robot can safely save some energy by not lifting its feet much. c_{jpos} determines the joint angle deviation penalty weight. In our experiments in Section 5.5.1, we found that different values of this coefficient result in different stride length. Therefore, we use it as an indirect way of controlling the stride length. Controlling the PD gains adds variable impedance control aspects to learned policies [BKR20]. It also adds robustness to variations in the environment [AS20] and makes it possible to control the expended torques to perform tasks that require more power.

The choice of parameters the high-level policy controls depends on the task. In our study, we used the high-level task for adapting the locomotion for traversing a variety of complex terrains, while the high-level policy only optimizes the velocity tracking and energy consumption. The high-level policy takes a scan of the height measurements surrounding the robot as input (Figure 5.6). The actions of this policy are offsets to the default command parameters vector ω excluding the velocity commands. We chose this task to show how we can easily learn to adapt the low-level policy behaviour that is trained

on a flat terrain to be useful on more complex terrains. The high-level reward function r_h is based on the velocity tracking term (in Table 5.1) and the energy consumption terms:

$$r_t^h = r_t^{vel} - c_{energy}^H |\tau_t^T \dot{q}_t|, \tag{5.6}$$

where τ_t is the torque vector of the motor.

In the next section, an analysis of the resulting low-level policy with the elaborate locomotion parameters input is displayed. We then present some high-level tasks to showcase the benefits of the proposed hierarchical scheme. For each task, the high-level state, action and reward will be explained in detail along with the results.

5.5 Results

This section includes full implementation details about the entire approach. We conduct our experiments to answer the following questions: (1) Which aspects of the locomotion vary when modifying the command parameters of the low-level policy? (2) Does our additions to the encoder network improve the learned performance? (3) How would a hierarchical policy improve the baseline end-to-end approaches on flat terrains and complex structured terrains in the presence of exteroception?

Implementation Details

As mentioned before, we conduct our experiments with the Mini-Cheetah quadruped, which has 12 degrees-of-freedom with 3 actuators per leg [KCK19]. The two levels of our hierarchical policy are trained one after the other. First, the low-level policy is trained to follow different random values of the command parameters. After that, depending on the high-level task, we train a high-level policy that controls the parameters of the already learned parametric low-level policy.

Both policies have the same neural network architecture which is a multi-layer perceptron (MLP) with three hidden layers of sizes 512, 256, 128. The RL algorithm used to train both levels is PPO [Sch+17] with generalized advantage estimation [Sch+15]. We train our policy networks with an actor-critic approach, with the critic having the same architecture as the actor but with a scalar output for the value estimation [KT99]. The policies run at a frequency of 50Hz while the simulation frequency 200 Hz. On the robot the same control frequency is maintained while the low-level PD control runs in a high frequency feedback loop of 40 KHz.

Low-level policy training. The observation space is 45-dimensional while the actions are 12-dimensional, equal to the number of actuated joints. We used the exponential linear unit (ELU) activation function in the neural networks. The critic also receives directly the privileged state of the robot.

We used the IsaacGym simulator from Nvidia [Mak+21]. The environment code is based on the legged_gym repository [Rud+21]. In the current experiments, 4096 agents are run in parallel in an infinite horizon objective where the environment does not reset with each new training episode, but continues with the latest reached state. However, we found that introducing random resets also helps. We were able to learn the low-level policy with command parameters in 5000 iterations.

A linear curriculum was implemented on the penalty terms of the reward. A curriculum factor $k_c \in [0, 1]$ was introduced to scale the reward. This factor is increased as training progresses to give more weight to the penalties. At the start of training the agent is mostly concerned with learning a movement that follows the reference velocity in any possible way. As training progresses the movement is refined to optimize the penalty terms [Hwa+19; Ara+23b]. The reward function term weights in Table 5.1 are chosen to be: $c_{vel} = 1.0$, $c_{v_z} = -1.2$, $c_{\dot{q}} = c_{\tau} = -0.0003$, $c_{\ddot{q}} = -0.00001$, $c_{\theta} = -3.0$, $c_{fcl} = -5.$, $c_{smooth} = -0.01$ and $c^{H}_{energy} = 0.002$.

Command parameters ω are sampled at the start of a new training episode and given as input to the low-level policy. The parameters are resampled when the policy fails (e.g., robot falls) or at random times if the policy does not fail. The sampling range for each element in ω can be found in Table 5.2[1]. Each parameter is sampled from a uniform distribution that samples deviations around the parameter's default value within the specified range. The sampling is also scaled by the curriculum factor k_c, i.e., we use the curriculum on the command parameters as well as on the reward terms.

Element	Range	Default
Foot height target [cm]	[3.0 , 15.0]	7.0
Joint angle deviation	[-1.0, -0.2]	-0.5
Position gain	[17.0, 30.0]	20.0
Velocity reference [m/s]	X: [-2.0, 2.0] Y : [-1.0, 1.0] z_yaw: [-1.0, 1.0]	0.5 0.0 0.0
$q_{nominal}$ [rad]	Shoulder: [0.0, 0.1] Hip : [-2.0, -0.1] Knee: [1.3, 1.9]	0.05 -0.8 1.6

Table 5.2: Default values of the command parameters and ranges in which the commands are sampled.

Domain randomization and RMA. For transferring the learned policy to the robot, it is essential to randomize various aspects of the observation and dynamics in order to mimic the perturbations and imperfections of the real system. The randomized values in the observations, robot dynamics, and terrain dynamics are shown in Table 4.6. However, as explained in Section 4.4, for the Mini-Cheetah, relying solely on domain randomization does not yield good transfer. To enhance transfer, we used rapid motor adaptation (RMA) [Kum+21; Mar+22], in which one first learns a latent embedding z_t that encodes privileged information around the randomized dynamics, and then uses supervised learning to build an adaptation model that estimates this z_t based on the sequence of state variables that are also available on the real robot.

In our setup, the embedding size for z_t is 18 while the input size of the privileged information is 29. The privileged input is constructed from the vector of random dynamics values (ranges presented in Table 4.6) and the privileged data regarding the feet height in world frame, terrain contact indicators and base linear velocities. The encoder E and adaptation model A are both MLPs with two hidden layers of sizes 256 and 128. The

[1]Note the value of the derivative gain is coupled with the position gain, i.e., K_d is chosen to be a scaled value of $K_p^{0.5}$ as in [BKR20].

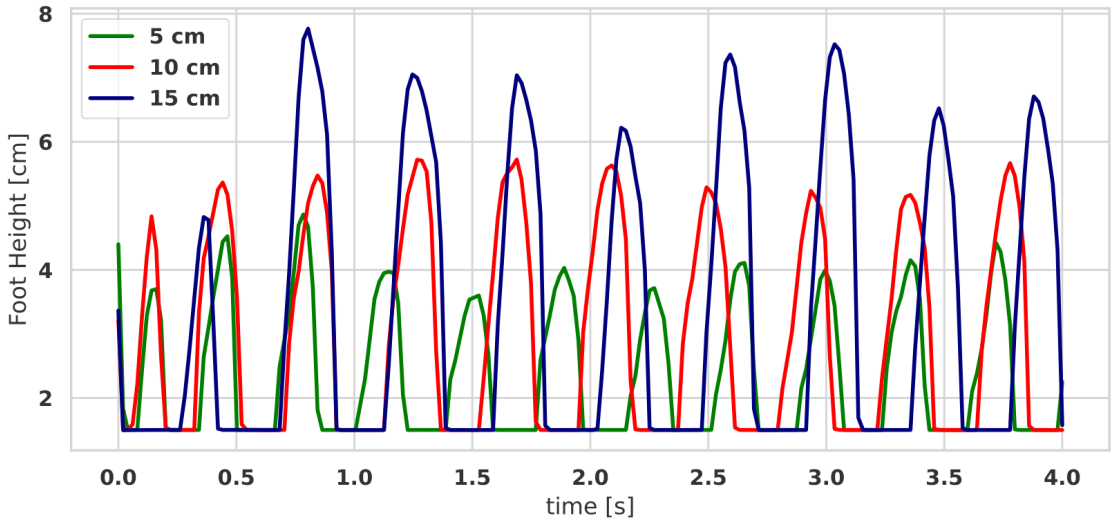


Figure 5.5: Front left foot height for different $p^{max}_{f,z}$ targets (5, 10 and 15cm), when running the robot controlled by the low-level policy at a 1.0m/s forward velocity command.

adaptation model takes as input the last $N = 15$ observations and outputs an estimate $\hat{z}$ of the latent vector z.

5.5.1 Low-level Policy Study

Understanding the Parameterized Low-level Control

It is important to first quantitatively and qualitatively observe the effect that the parameters ω have on the final locomotion in order to be able to define useful tasks that could benefit from a policy that controls ω. In this section we outline the difference in aspects of the resulting locomotion when changing some command parameters individually.

$c_{jpos} = -0.1$			
Vx [m/s]	0.5	1.0	1.5
footstep [m]	0.27 ± 0.03	0.33 ± 0.03	0.42 ± 0.04
$c_{jpos} = -0.5$			
Vx [m/s]	0.5	1.0	1.5
footstep [m]	0.22 ± 0.001	0.32 ± 0.005	0.39 ± 0.02
$c_{jpos} = -1.0$			
Vx [m/s]	0.5	1.0	1.5
footstep [m]	0.17 ± 0.001	0.25 ± 0.001	0.33 ± 0.003

Table 5.3: Average foot step length as a function of the forward velocity command (angular velocity is set to zero) and joint angle deviation penalty weight.

The joint angle deviation term in the reward penalizes the policy based on how far the joint positions are from the nominal pose. Intuitively, modifying this penalty coefficient in the reward should allow us to control the range of joint movement which affects the step length. We verify that the step length in the parameterized low-level policy is a function of the coefficient c_{jpos} by examining the average foot step length of the forward left foot as a function of the forward velocity in Table 5.3. This table shows a clear relationship

between the foot step length and coefficient value for different values of the velocity. For all values of c_{jpos} the average step length increases with the increase in the commanded V_x. However, the average step length for the same velocity command decreases when the absolute magnitude of c_{jpos} is increased. In other words, the more weight given to the joint angle deviation term the smaller the average foot steps are and vice versa. We also notice the high variance of the average step length when $c_{jpos} = -0.1$, which suggests the limit to which c_{jpos} could be varied while the penalty which still retains an effect on the overall behaviour.

Another term in the reward function is the $p_{f,z}^{max}$ parameter which defines the target foot height when the foot is in swing mode. Figure 5.5 shows a plot of the achieved height of the front left foot as a function of time when running the low-level policy with a forward velocity command of 1.0m/s for four seconds. The figure shows the foot height profile alternating between swing phases and stance phases during which the foot is resting on the ground. We can clearly see the increase in foot height for higher values of $p_{f,z}^{max}$. Another interesting behaviour that emerges from changing the foot height target is a change in the stepping period. The 5cm height target results in shorter stepping period than the 10cm target, which in turn results in a shorter period than the 15cm target. Note that the achieved foot height is lower than the target because the foot clearance reward does not only depend on $p_{f,z}^{max}$ but is also scaled by the velocity of the foot, see Table 5.1.

Ablation Studies

In Section 5.4.1, we introduced additional information in the privileged information encoder (see Figure 5.4) regarding the feet height, feet contacts and the base linear velocity as additional input to the encoder network E. We show in Figure 5.7 the average episodic reward achieved during training as a function of the training step, with and without this additional data to the encoder input. Each curve in the plot is the average over five random seeds of the same experiments. We notice the higher overall attained reward with the proposed additions. The additional input allows the learning to optimize rewards like foot clearance whose target value is varied from one episode to another.

5.5.2 High-level Policy Study

The strength of the proposed approach is the ability to reuse the low-level policy for different high-level tasks. In this section, we verify that we are able to learn a high-level policy that adapts the low-level policy, learned on a flat terrain, in order to successfully traverse complex test-bed terrains that includes flat and rough plains, stairs, slopes and obstacles as shown in Figure 5.6.

Elevation map. Access to information about the state of the terrain is necessary in order to be able detect the location of obstacles and steps so the policy can adapt its behaviour correctly. The bottom right picture in Figure 5.6 shows the Mini-Cheetah robot with the elevation map scan that is used in this set of experiments. Each point indicates the terrain height in world coordinates. The elevation map is a constructed as a 1.0mx1.6m rectangle around the robot. The distance between each point is 10cm. Therefore, the final height map is a 178-dimensional vector that conveys the height of a certain patch of the terrain where the robot is at the center.

Non-hierarchic baseline comparison. We compare the results of the proposed hierarchic approach with a baseline end-to-end approach that learns to traverse the terrain

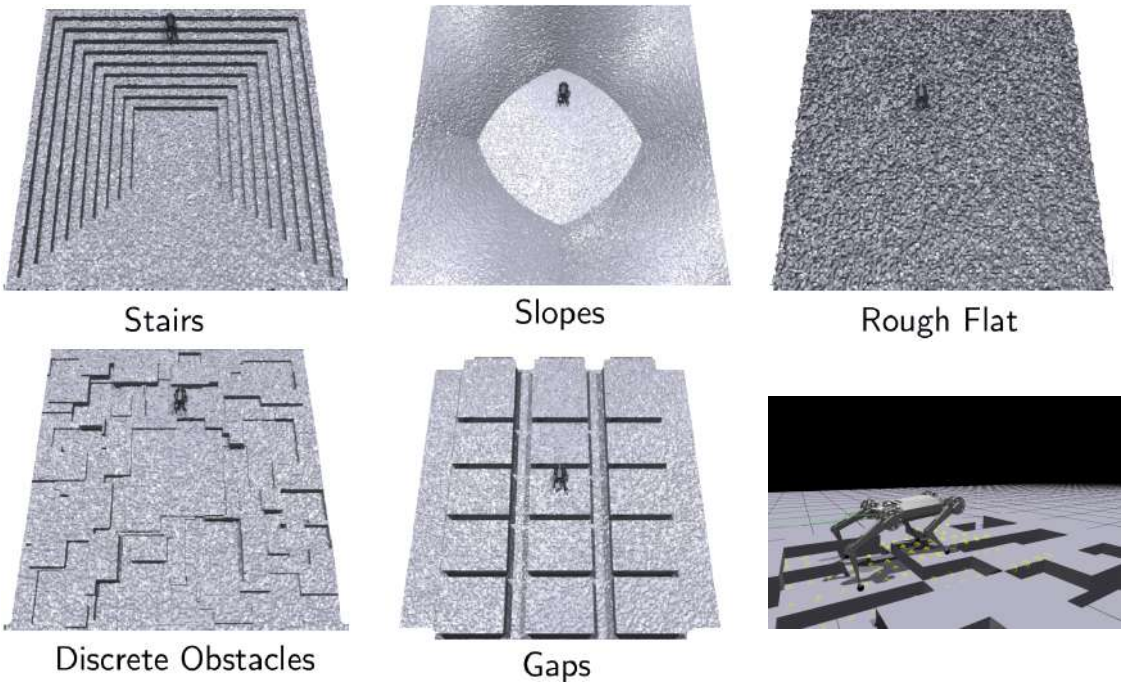


Figure 5.6: Example of the different terrains types various difficulty used to make a big terrain for the training of the high-level policy. Bottom right shows an example of the robot in the obstacle terrain with the height measure that surrounds it.

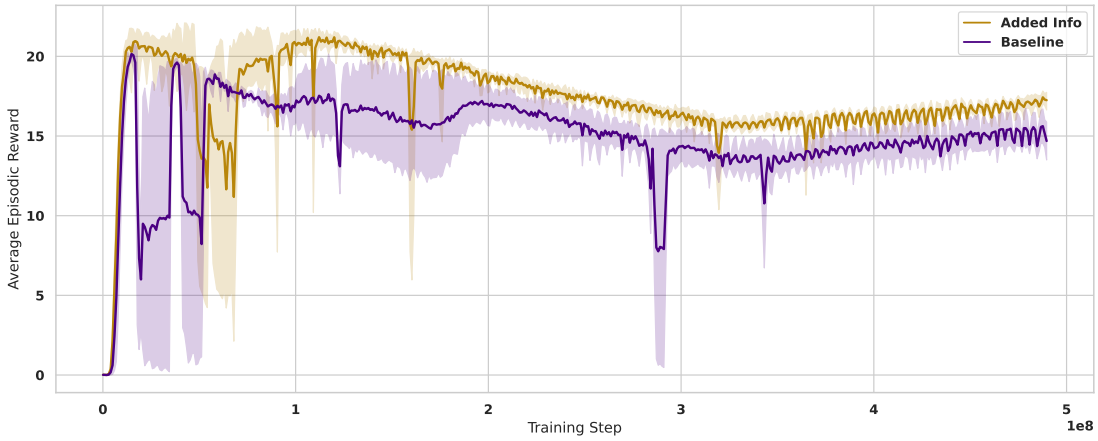


Figure 5.7: The average episodic reward per training iteration. The curves show two experimental setups: when additional information is added to the encoder network (yellow), and when it is not (purple).

from scratch, with the height map state as input, along with the low-level policy observation and reward function. However, we add the energy consumption cost term to the reward function for fairness of the comparison between the baseline and the proposed hierarchical approach. The training approach using the elevation map is inspired by the work done in [Rud+21]. We also use a similar curriculum on the terrain in order to gradually introduce the robot to terrains with increasing difficulty. It is important to note that the baseline training also uses the same RMA technique to enable the transfer of the learned policies to the real robot. Figure 5.3 bottom shows the baseline policy as one neural network receives all the observations with the task information and the elevation

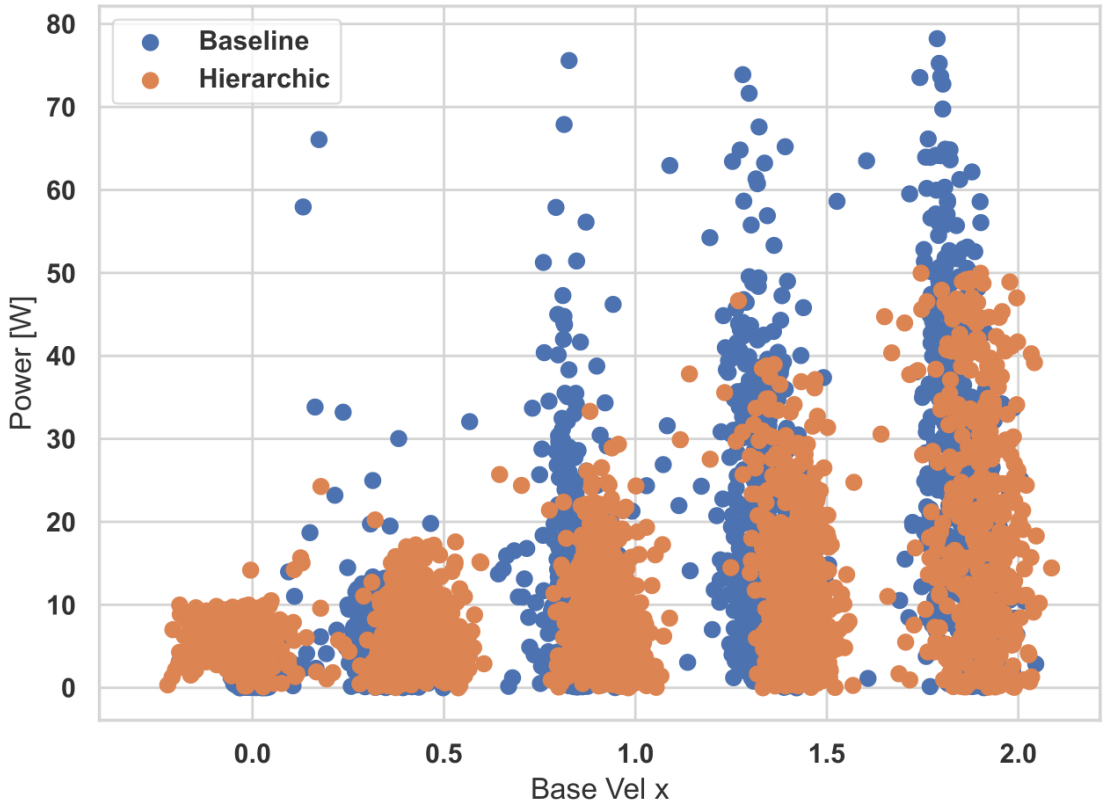


Figure 5.8: Average power vs. measured velocity for the hierarchic and baseline policies.

map. In these particular experiments the task information is the standard velocity tracking command.

Figure 5.8 shows a scatter plot featuring the power consumption as a function of the measured linear velocity. We collected the data points by running the two policies (hierarchic approach and baseline) for different values of the velocity commands $V_x = \{0.0, 0.5, 1.0, 1.5, 2.0\}[m/s]$ for ten seconds at each velocity over a flat terrain. We show the results obtained with those values on a flat terrain in order to judge the performance of the hierarchical policy on the original locomotion task of tracking the velocity while saving energy. The figure shows that the proposed approach leads to lower overall power consumption with less variance than the baseline, especially at higher velocities. The hierarchical policy also tracks the commanded velocity better than the baseline policy both in terms of accuracy and variance.

Figure 5.9 shows the adaptation of the command parameters when the robot has to climb stairs. We see the change from their default value (dashed line). The high-level policy raises the foot height target (Fh) and the position gains which helps the robot use more power to climb the stairs. The angles of the hip and knee joints are lowered from their default values which could explain the overall improved energy efficiency of the robot (seen in Figure 5.8), since standing on straighter legs requires less power.

Sample Efficiency. Our hierarchical learning method also needs fewer samples for training since it splits complex problems into simpler sub-problems. In our experiments, the baseline method that learns to cross terrains from scratch required 20000 training iterations which is around two billion samples. However, with the hierarchical approach we were able to learn a low-level policy with 5000 iterations and a high-level policy with 3000 iterations. This means that our approach requires less than half the number of

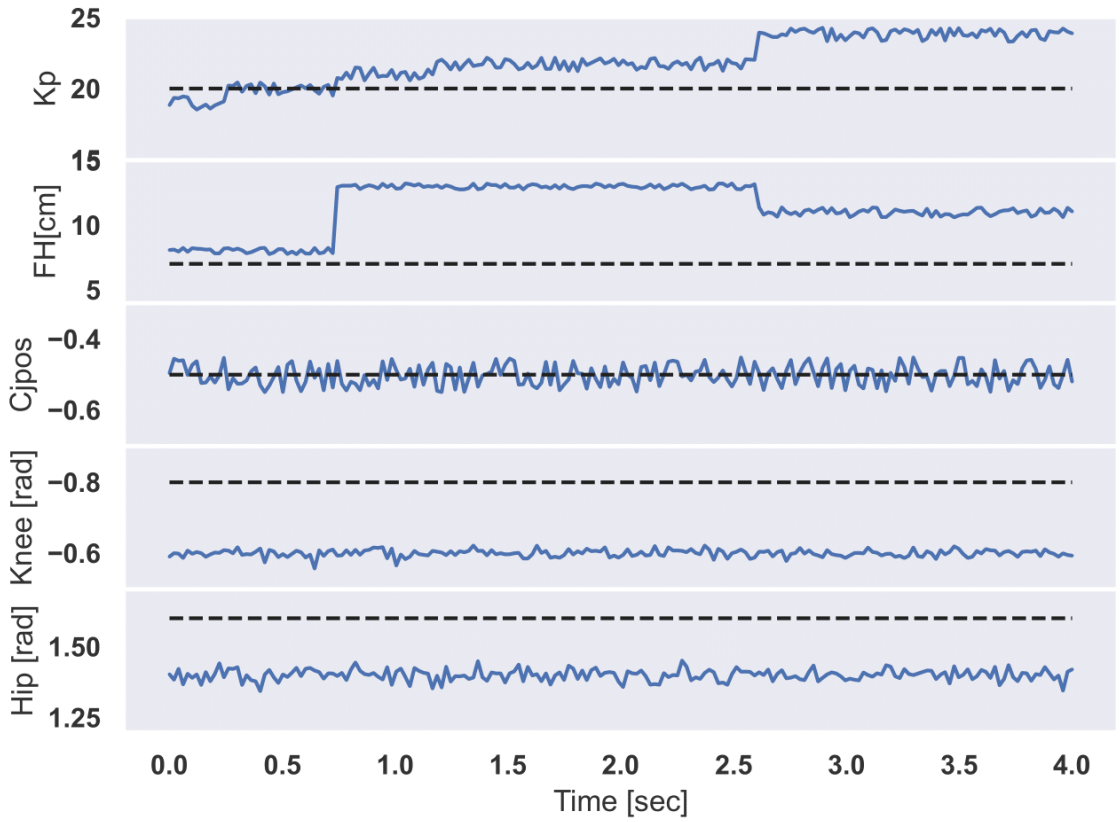


Figure 5.9: Some command parameters values adapted by the high-level policy while climbing stairs. The Knee and Hip represents the angle values in the nominal pose. The dashed lines represent the default values of the command parameters.

Figure 5.10: Snapshots of the Mini-Cheetah going down a set of stairs in a robust manner without losing balance or falling. Video available here: https://youtu.be/B92HB964xq8?si=C5jS1SsEtIbfdWqM.

samples that the baseline needs because the complex problem of learning locomotion is done in the low-level policy on a simple terrain.

5.5.3 Real Robot Transfer

On its own, the low-level policy can be successfully transferred to the Mini-Cheetah robot. We ran the low-level policy on the real robot on multiple difficult surfaces on which the baseline policy was shown to completely fail (as shown in Figure 4.26). Since we do not have access to the elevation map, we manually tuned the command parameters on the real robot to the values that were displayed in similar environments in simulation. For example, we ran the hierarchical policy on stairs in simulation and observed that the policy was increasing the gain and desired foot height in order to traverse the stairs. We took these adapted values and tried them directly with the low-level policy on the real robot on stairs. We observed that the robot was able traverse different terrains in a very robust manner when using the parameters that are similar to the one learned by the high-level policy in those settings.

Crossing obstacles and steps. We observed the robot crossing over pavements and obstacles on the ground that would usually make the baseline policy fail. We see an example in Figure 5.13 of Mini-Cheetah moving from a hard road ground to a wet grass terrain while crossing over a pavement. The wet grass is much softer than the asphalt and more challenging to walk on, yet the robot manages to make the transitions seamlessly.

Stairs. Stairs is one of the most challenging environments for quadruped robots. Our policy was able to climb a set of stairs of length 10 cm shown in Figure 5.11. The robot's feet at some point are planted in the dirt on the side of the stairs and instead of losing balance and failing, the robot is able to robustly continue and reach the stairs. Figure 5.10 also shows the robot going down the same set of stairs without seeming hindered by the delayed contact with the ground due to of the decrease in the elevation of the terrains.

Slippery terrains. We ran the policy on a grass terrain that was very slippery and muddy. We can see the robot right hind foot slip in Figure 5.12 , yet the robot is able to robustly move and recover from disturbances. This also shows that the low-level policy has robust properties in the control with the choice of command parameters.

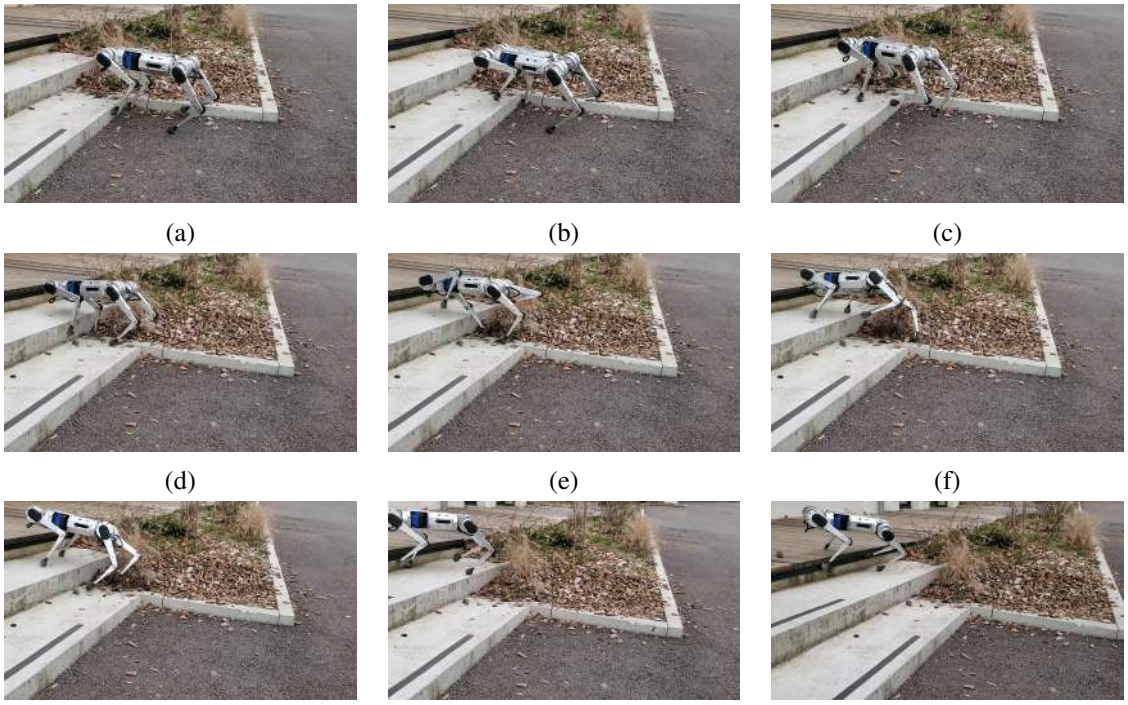


Figure 5.11: Snapshots of Mini-Cheetah climbing stairs successfully and robustly with the low-level policy with hand-tuned parameters taken from simulation experiments. We notice that the robot' feet at frame (d) gets planted on the side in the soil and as the robot is about to fall in frame (e) the policy manages to regain balance and cross the stairs. Video available here: https://youtu.be/B92HB964xq8?si=C5jS1SsEtIbfdWqM.

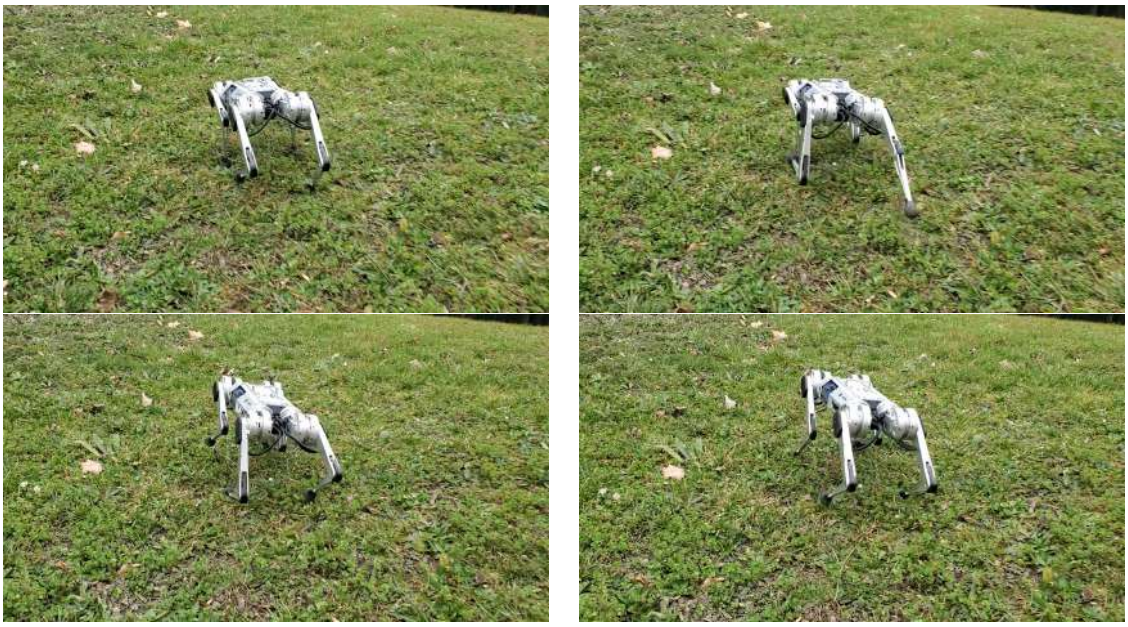


Figure 5.12: We see the robot's right hind leg slipping while Mini-Cheetah is walking on the wet grass. However, the policy is able to recover and keep the robot from falling. Video available here: https://youtu.be/B92HB964xq8?si=C5jS1SsEtIbfdWqM.

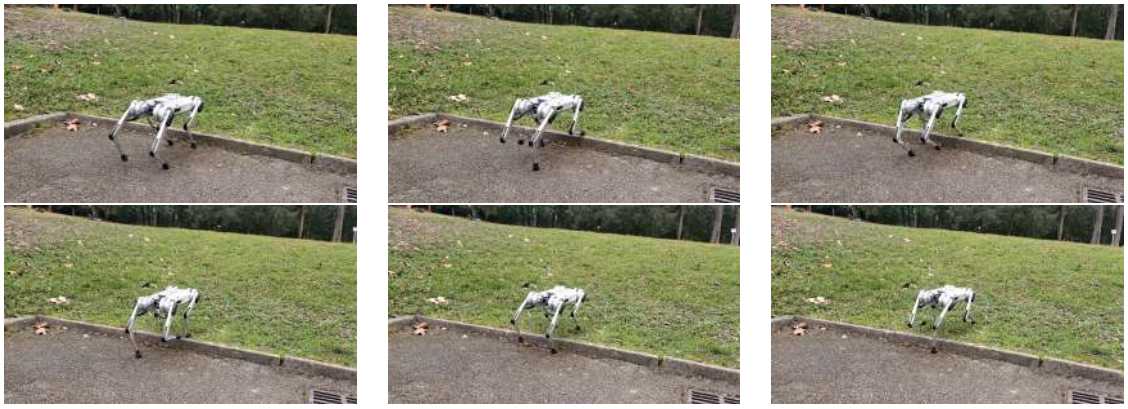


Figure 5.13: Snapshots of the Mini-Cheetah crossing from hard ground to a wet muddy grassy terrain with the low-level policy with hand-tuned parameters taken from simulation experiments. The robot goes over a step that the baseline policy failed to cross. The wet grass is much softer than the asphalt yet the robot manages to make the transitions seamlessly. Video available here: https://youtu.be/B92HB964xq8?si=C5jS1SsEtIbfdWqM.

5.6 Extensions: Learning Gait Policies

The default values that determine the sampling of the command parameters are created from the default values used in the baseline training with some range of sampling. However, the question that remains is: what happens if we use some very different values or ranges for these commands? Would we get the same behaviour?

We ran a few experiments where we shifted some ranges and default values for the purpose of getting different policies with very different behaviours. Maybe certain situations may require a parameterized policy that is specialized, taking very long steps or lifting its feet very high. What we found, is that for some specific ranges and setups of these commands we get policies that execute completely different gaits from walking to trotting to even producing a bounding gait that resembles galloping.

	Walking Gait		Trotting Gait		Bounding Gait	
	Range	Default	Range	Default	Range	Default
$p_{f,z}^{max}$ [cm]	[8.0 , 10.0]	10.0	[3.0 , 15.0]	7.0	[10.0 , 15.0]	10.0
c_{jpos}	[-1.5, -1.0]	-1.5	[-1.0, -0.2]	-0.5	[-0.2, 0.0]	0.0
V_x^{target} [m/s]	[-0.5, 0.5]	0.2	[-2.0, 2.0]	0.5	[1.5, 4.0]	2.0

Table 5.4: Values for different ranges of the command parameters that produce different gaits when applied to the learning procedure.

The walking gait is able to run at low velocities and only lift one foot at a time. The bounding gait exhibits flying phases with a contact pattern where the front feet are lifted and then the hind feet take a big step that makes the robot jump forward with its feet in the air before the front feet land again and the hind feet land and push the body forward. Table 5.4 shows the shifted command parameters that produce different policies.

This is very powerful as learning different gait patterns is still challenging in RL-based controller and usually requires the use of Central Pattern Generators [Mik+22] or having an explicit contact schedule in the reward [MA22].

Figure 5.14 shows the Mini-Cheetah robot running with the bounding gait policy that can achieve much higher velocities than the normal trotting gait. The robot is running at a

Figure 5.14: Example of a parameterized policy where the policy is trained on a specific range of values of the command parameters so that a bounding gait is learned. The policy is able to reach very high velocity up to 4 m/s. Video of this run available here: https://youtu.be/Ml7xGzhaEco.

forward speed of 4 m/s while exhibiting flying phases. The ability to learn different gaits opens all kinds of possibilities in terms of learning multi-gait policies.

Interestingly, we can see a connection between this work, that was developed with learning, and the first work in this thesis in Chapter 3. We tried, in Chapter 3, to learn to control the block tasked with planning the gait pattern in a model-based optimal controller. The goal was to observe diverse gaits in different situations. Unfortunately, we found that controlling the gait pattern alone did not produce different gaits and in the end the policy was learning to keep the trotting pattern only at different frequencies. We argued that one possible reason to this limited behaviour is that there are many parameters that were hand-tuned in different blocks of the optimal controller and they should be adapted with the gait pattern to keep the optimal behaviour. In this section, we found that controlling different parameters that define the low-level behaviour, with a learned policy, was able to produce different gaits that perform better for different situation. These experiments back the arguments made in Chapter 3.

Further work is necessary to develop a full system that is able to use the right gait in the suitable conditions and switch between different gaits. At the end of the thesis we were able to conduct these preliminary experiments to show that this is a promising direction for future work.

Discussion and Limitations

In this chapter, we presented a hierarchical scheme for adapting learned quadruped locomotion. The main strength of the work lies in the versatile low-level policy training that can be reused for different tasks and in different environments and terrains. The low-level policy embeds several behaviour through the command parameters, however the methodology is not without its limits.

Learning specialized policies. Having a single low-level policy could also make

training harder since we have to find one policy that is optimal for all combinations of the command vector. It is possible that training low-level policy on specialized subspaces of the command vectors would produce more specialized diverse skills. For example, we could train one policy that specializes in lifting its feet high and another one that specializes in taking long steps, and then combine these two policies in an ensemble manner [Pen+19]. We have already shown that we can get specialized policies that execute different gaits for different ranges of the command parameters. Hence, one can envision a setup where policies generate various walking patterns that can be learned to switch between or combine their actions.

Expanding the command parameters. We chose a specific set of commands that can directly and indirectly control aspects of the learned behaviour, so that we can simply show how they affect the policy when varied. For future work, an interesting objective could be to expand the command parameters to include all parameters related to the reward coefficients and learned pattern. For example, while we have assumed that PD gains are fixed for all joints, depending on the type of gait pattern we could learn a different gain for each leg or joint.

5.7 Conclusion

In this chapter, we argued for expanding the notion of the command in learned locomotion policies by actively setting targets for locomotion features such as the swing feet height, stepping length, initial pose and gains. We showed that we are able to learn a locomotion policy that reacts to the targets set by the command parameters. This makes it possible to adapt the low-level policy for different tasks. For this effect, we proposed a hierarchical RL setup where a high-level policy learns to adapt the parameters of the low-level policy for versatile locomotion. We also showed that this hierarchical setup speeds up the learning of new tasks and that adapted policies can be better in quality than policies learnt from scratch. With parameterized policies we made our real robot successfully traverse complex terrains on which the baseline policy fails.

Chapter 6

Conclusion

The work done in the framework of this thesis contributed to the literature of locomotion of legged robots by developing learning-based controllers for a lightweight quadruped robot Solo-12 and a heavier one Mini-Cheetah. We mainly developed end-to-end procedures that use deep neural networks for predicting the next suitable joint angle targets from the measured data provided by on-board sensors of the robot. The joint angle targets would then be given to a PD controller that output the necessary torques. The main task of the locomotion is for the robot to follow a user-defined velocity command . We were able to deploy our policies, learned in simulation, on the real platforms (Solo-12 and Mini-Cheetah) and we successfully ran the robots in outdoors environments in very hard conditions. While working on the learning procedure, we encountered many difficulties in terms of collecting the right data in simulation that represent the real system, learning the right behaviour that can be safe to deploy on a robot and adapting what we learned to two different platforms with varying degrees of complexity and difficulty.

6.1 Contributions

The first part of the thesis investigated the ability to modify the gait by modifying the controller that was designed for the Solo-12 robot [Léz+20]. To sum up our contributions:

- We proposed to learn how to adapt the contact patterns for each foot in order to modify the gait that is otherwise fixed and executed by the model-based controller. Changing the gait led to improvements in the velocity tracking and energy efficiency of the low-level control.

- We used a square wave oscillator function that determines, at each timestep, whether the foot should be in stance phase or in a swing phase. The proposed RL approach controls the timings of the oscillation in which a change in the state from stance to swing occurs. The learned actions are offsets to the timings of the nominal gait of the model based controller.

- We designed a reward function that is based on velocity tracking and an energy consumption term that contains elements of the motor torque and friction torque. This term is a better model of the energy consumed on the real Solo-12 robot.

- We proposed the use of self-attention layers commonly used for sequential observations. We showed that the use of a self-attention in the policy architecture results in faster convergence and a better performance in the final reward.

- We showed that the system modifies the gait of the Solo-12 robot by changing the frequency of the trotting pattern. The policy raises the frequency for faster velocity commands and lowers the frequency for low velocity commands. The modified gaits improve the control by achieving higher velocities than the nominal gait and consuming less energy. The work was documented in the report [Ara+21].

The second part of the thesis concentrated on developing learning approaches for joint angle controllers on Solo-12 and later on Mini-Cheetah. To sum up our contribution:

- We displayed our initial effort of learning end-to-end torque control policies for Solo robot. We proposed three tasks for that problem: standing in place, walking forward and navigating towards a goal. We designed simple reward functions for each task based on the task reward and torque magnitude. We then showed that, with PPO and TD3, we are able to learn policies that complete each task. However, the resulting control exploited the simulated environment and learned patterns and maneuvers that did not resemble safe control that could be deployed on a real system.

- With the knowledge acquired from the first attempt, we designed a more complex RL method for learning joint angle controllers. The policy outputs an offset to the nominal pose of the robot that are then translated into torques via a PD controller. We built on the previous approach by adding numerous constraints and penalty terms in the reward function. This elaborate reward allowed us to learn safe actions on the robot and to obtain clear and periodic trotting patterns.

- We introduced a realistic energy loss penalty, incorporating actuator friction and Joules losses identification. We showed extensive experiments in simulations and real robot studying this reward term at different magnitudes and showed how different the learned policies are for different weightings of the penalty terms.

- We detailed the domain and dynamics randomization applied to the environment by adding noise and perturbations in the dynamics parameters and observations. We employed curriculum learning techniques to enable the maximization of all the terms of the reward function and to make the environment more complex (in terms of noise and terrain) as training progressed.

- This resulted in policies that could transfer to the Solo-12 robot and allow it to navigate indoors and outdoors in rough environments such as muddy slopes and gravel with pebbles. This work was published in the journal of Scientific Reports [Ara+23b].

- We then tried to use the same techniques for a heavier robot that exhibits stronger torque output. We found the policies on Mini-Cheetah to lack the same robustness and smoothness in actions and steadiness of the base that was seen on Solo-12. We, therefore, set out to find a more complex training procedures. For learning policies for Mini-Cheetah.

- We decided to use a strategy based on distillation where we learn a latent representation of privileged information, available in the simulator, related to the dynamics of the environment and give that representation as additional information to the policy. Then, we distill the learnt representation through an adaptation module that is tasked with predicting that representation from the sequence of actions and observations available on the real robot. We implement this strategy along with PPO to produce robust policies for Mini-Cheetah.

- We showed the development of the RL procedure, from the first failing experiments to the final system, with experiments on Mini-Cheetah at each stage. We finally made the Mini-Cheetah robustly navigate over complex terrains on wet, muddy grass, hard surfaces, rough pebbles and steep slopes.

- Finally, we showed an extension of the idea of learning joint angle controllers for the Bolt bipedal robot. We showed that we can easily adapt the learning approach for other types of locomotion agents.

Finally, the third part of the thesis built on the foundation laid in the second part and expanded on the approach by proposing a hierarchical method for adapting the learnt locomotion. To sum up our method:

- We showed the limitations of the proposed learning approach on Mini-Cheetah and its lack of adaptability due to many low-level decisions that were hard-coded in the reward function and control architecture. We also generalized the main task of the locomotion to be more than just velocity tracking.

- We proposed a hierarchical learning for quadruped locomotion where the main task of the robot is from just tracking a velocity command, to also contain other commands related to the foot step height, step size, stiffness and more. We proposed to augment the control policies with *command parameters* that consist of values that determine the reward functions and control setup which includes velocity command, desired feet height, joint PD gains, joint deviation penalty coefficient and the nominal pose of the joints.

- We displayed the training of low-level policies in which each command parameter can be varied in order to change some aspect of the locomotion. We performed an extensive study on this low-level policy to show the effect of changing individual elements of the command parameters quantitatively and qualitatively. We showed the Mini-Cheetah lifting its feet to different heights and running with different nominal poses while controlled by the same parameteric policy.

- We then developed a high-level policy that is tasked to control the command parameters to fulfill a high level task. We showed that the high-level policy manipulating the low-level policy, that was trained on a flat ground, was able to make the Mini-Cheetah cross over obstacles, slopes and stairs. We showed experiments of the real robot controlled by the low-level policy going up and down stairs, going over obstacles, pavements, and climbing steep slopes in a wet, grassy terrain. The results were a big improvements over the baseline RL approach developed in the previous chapter. This work was submitted to the IEEE Humanoids conference [Ara+23a].

6.2 Future Directions and Perspectives

The work done in this thesis has laid the ground work for future projects based on exploring robot control and robot behaviour with deep learning and reinforcement learning. We have explored several themes and, therefore, there are many possible directions to take after this work. We present the most promising ones that we would like to work on as a priority in the future.

Vision guided locomotion

We presented our work on hierarchical policies for locomotion where a high-level policy can have access to exteroception data and guide the low-level policy towards the right behaviour. In Chapter 5, the high-level policy had been given a heightmap scan of the terrain surrounding the robot. We did not deploy the high-level policies in the real robot since it was not equipped with a vision system.

We would like to build on this work by having a full vision system using cameras to provide information about the surrounding environments to the policy. One can do this in a number of ways. We can use lidar and depth cameras to perform reconstruction of the environments around the robot. The lidar and depth scans can be used with probabilistic techniques and Kalman filtering to build and update a map of the terrain [FBH18; Mik+22]. However, these techniques aim to reconstruct the entire surrounding of the robot which could be difficult, noisy and unnecessary for many tasks.

A more sensible approach is to use egocentric vision by placing the camera on the front of the robot base to resemble the eyes of the robot. Using egocentric vision gives the robot the ability to see what is in front of it and make right decisions for the locomotion. There has been some works in this direction. Agrawal et al. [Aga+22] proposed an extension of RMA where the depth images are used to predict a representation of the terrain scans, and the vision module is trained through a distillation technique. Loquercio et al. [LKM22] proposed a continual learning approach to collect real RGB images from running the real robot outside, and then train a vision module to predict the terrain. Yang et al. [YYW23] proposed neural volumetric memories to learn a representation of the egocentric depth view aggregated from a sequence of images and poses. We would like to use similar approaches to have the ability to deploy the high-level policy on the robot while keeping the low-level policy blind and responsible for the low-level locomotion.

Task oriented locomotion

In last part of our work, we have attempted to generalize the task of locomotion. Following a commanded velocity is a simple way to communicate to the robot that it has to move but other methods could make more sense. A goal point task, where the robot tries to reach a specific location, would require the robot to move to be able to reach the goal. The velocity that the robot has to have does not need to be predetermined by the user, it could emerge naturally depending on the environment and terrain. The robot should have the ability to slow down or speed up in different circumstances. Lee et al. [Lee+20] hinted at such ideas by specifying the direction a robot needs to be moving without specifying a velocity point.

Another issue of concentrating only on velocity tracking is that it prohibits the addition of other manoeuvers that could be done with the robot's legs. Cheng et al. [CKP23] proposed a system that uses the robot's legs to reach buttons to push. This creates a

task that is a hybrid between point goal navigation and manipulation with the legs of the quadruped. Other works also explore the ability of quadrupeds playing football which would also require the robot to use its legs to push the ball around [Ji+22b; JMA23]. After achieving robust policies that allow the robots to navigate outside, we can use quadruped robots beyond the purpose of basic locomotion.

Bipedal locomotion and mobile manipulation

The work done in learning on quadrupeds inspired researchers to continue towards bipedal robots. We have briefly shown, in this thesis, the ability of adapting the RL procedure for learning locomotion for bipedal robots. Bipedal robots are harder to stabilize and require more complex feet designs to be able to stand without moving. Some of the current works were able to make humanoids perform some locomotion tasks [Rad+23; Sin+23] and even going over stairs and jumping [Li+23b; Li+21]. It seems that the advancement that locomotion research exhibited due to the rise of RL-based controllers is happening for humanoid robots.

The interesting follow-up is the research around simultaneous manipulation and locomotion or generally mobile manipulation. Fu et al. [FCP22] investigated the ability of learning a policy that can control the locomotion of a quadruped and the actions of an arm mounted on top of the quadruped to perform grasping tasks. The resulting policy, trained using PPO with advantage splitting, was able to perform both task together. In humanoid robots, mobile manipulation will be more complex and combine themes from locomotion to bi-manual manipulation which will be a research challenge that is worth exploring.

Multi-modal locomotion

In the past year, the rise of large language models (LLMs) has occupied much of the discussions in all domains of artificial intelligence. The ability of LLMs to generate code [Che+21] has made researchers wonder whether it can be used to reason about tasks, generate goals and provide high-level description about the environment. The use of LLMs aligns nicely with robotics since LLMs receive and output sequences like policies in sequential decision making. In manipulation, Anh et al. [Ahn+22] proposed to ground the output of the language model with the learned primitive skills that the robot is able to achieve resulting in the LLM to output natural language commands that the robot can achieve based on its perception. Liang et al. [Lia+23] took a further step and studies the possibility of using LLMs to translate natural language commands into robot policy code which process perception outputs (position of target objects), parameterize control primitives to recursively generate code to perform the task. While there are a few directions, the current systems require the user to generate commands in a very technical manner, which limits the use of these approaches to experiments and general proof of concepts rather than reliable systems.

In locomotion, recently Tang et al. [Tan+23] introduced a method that uses LLMs to translate human commands via natural language to foot contact patterns that then constitute a gait that is fed to a locomotion controller that generates the corresponding low-level commands. This approach enables the development of an interface between controlling the behaviour of the robot and the LLM with the command statement from the user. While still in its early stage, there is a lot of effort spent developing LLMs. For a future outlook of our work and the work that will be done in this field, automatic robot commands and tasks is likely to be a huge field of future research that will be widely used in all settings.

Learning on the robot

In current robot learning research, control strategies based on optimal control and trajectory optimization are referred to as model-based and controllers based on RL, like the ones proposed in this thesis, are referred to as model-free. Some might object to calling RL-based controller model-free since a model of the robot is necessary for building simulated environments where the policy training happens. However, the terms model-based and model-free refer to the decision making mechanism and whether that mechanism requires a model when deployed on the robot. Modeling the robot is not necessary from the perspective of the model-free RL algorithms as it only requires interaction data to learn and we use simulation as an easy way to collect that data. Simulations are heavily used in the current robot learning paradigm because RL algorithms require a lot of data to converge and training on the robot would be costly and dangerous.

The recent work by Smith et al. [Smi+21; SKL22; Smi+23] investigated learning RL control policies directly on the robot and discarding simulators. They proposed to use soft-actor critic [Haa+18], an RL algorithm that requires much less samples to learn, with specific damping values and restricting the action space to reduce the exploration and learn locomotion policies in around 20 minutes on a real quadruped robot. Discarding simulation and learning directly on the robot allows us to put the robot in complex situations that are hard to model and simulate. It also allows us to develop continual learning methods where we can improve the learned performance in new situations that were not encountered by the policy during training.

Appendix A

Software

Several code bases created over the course of the PhD are available online.

- First source code titled SoloRL which includes the first attempt at solving simple tasks with Solo-12 (Section 4.3.1) and the gait control code that manipulated the gait of the model-based controller (Section 3.4):
  https://github.com/michel-aractingi/soloRL
  The code provides several gym-based [Bro+16] environments which defines pyBullet simulations [CB21] of different Solo tasks standing, walking and point goal. The gym environment contains the exact state, action and reward functions used and mentioned in the thesis. The repository also includes my own implementation of PPO [Sch+17] and TD3 [FHM18] that allows to train the policies. Finally, the repository contains the code to reproduce our work on learning the gait timings of the controller of Solo-12.

- Source for learning joint angle controllers for Solo-12 (Section 4.3.1):
  https://github.com/Gepetto/soloRL
  The codes provides a C++ environment in the style of raisimGymTorch [HLH18] that runs with the Raisim simulator. The environment contains the definition of the action space, state space and all the reward terms. Coupled with this repository is the source code:
  https://gitlab.laas.fr/paleziart/quadruped-rl
  This repository is setup in collaboration with Pierre-Alexandre Léziart and Thomas Flayols that contains the control interface with the real Solo-12 robot along with the checkpoints of the policies displayed in our journal publication [Ara+23b]. Anybody can use this code to control their own Solo-12 with out trained policies that we analyzed in Subsection 4.3.4.

Bibliography

[AS20] Fares J. Abu-Dakka and Matteo Saveriano. "Variable Impedance Control and Learning—A Review". In: *Frontiers in Robotics and AI* 7 (2020). ISSN: 2296-9144. DOI: 10.3389/frobt.2020.590681 (cit. on p. 86).

[Aga+22] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. *Legged Locomotion in Challenging Terrains using Egocentric Vision*. 2022. arXiv: 2211.07638 [cs.RO] (cit. on pp. 40, 65, 102).

[Ahn+22] Michael Ahn et al. *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*. 2022. arXiv: 2204.01691 [cs.RO] (cit. on p. 103).

[AIS] AIST. *National Institute of Advanced Industrial Science and Technology*. https://www.aist.go.jp/index_en.html. Accessed: 2023 - 08 - 20 (cit. on p. 2).

[And+18] Marcin Andrychowicz et al. "Learning Dexterous In-Hand Manipulation". In: *CoRR* abs/1808.00177 (2018). arXiv: 1808.00177. URL: http://arxiv.org/abs/1808.00177 (cit. on pp. 18, 19).

[Ara+21] Michel Aractingi, Pierre-Alexandre Leziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. "Learning to Adapt the Trotting Gait of the Solo Quadruped". working paper or preprint. Oct. 2021. URL: https://hal.laas.fr/hal-03409682 (cit. on p. 100).

[Ara+23a] Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. "A Hierarchical Scheme for Adapting Learned Quadruped Locomotion". working paper or preprint. Aug. 2023. URL: https://hal.science/hal-04174932 (cit. on p. 101).

[Ara+23b] Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. "Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning". To appear in Scientific Reports. July 2023 (cit. on pp. 40, 75, 85, 86, 88, 100, 105).

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.0473 (cit. on p. 1).

[Bel+18] C. Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter. "Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 2261–2268. ISSN: 23773766. DOI: 10.1109/LRA.2018.2794620 (cit. on p. 22).

[BF97] Hamid Benbrahim and Judy A. Franklin. "Biped dynamic walking using reinforcement learning". In: *Robotics and Autonomous Systems* 22.3-4 (Dec. 1997), pp. 283–302 (cit. on p. 38).

[Ben+09] Yoshua Bengio, Jérome Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning". In: *ACM International Conference Proceeding Series*. Vol. 382. New York, New York, USA: ACM Press, 2009, pp. 1–8 (cit. on pp. 19, 51).

[Ber+19] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680 (cit. on p. 1).

[BG08] John Bertram and Anne Gutmann. "Motions of the running horse and cheetah revisited: Fundamental mechanics of the transverse and rotary gallop". In: *Journal of the Royal Society, Interface / the Royal Society* 6 (Oct. 2008), pp. 549–59. DOI: 10.1098/rsif.2008.0328 (cit. on p. 36).

[Ble+18] Gerardo Bledt, Matthew J. Powell, Benjamin Katz, Jared Di Carlo, Patrick M. Wensing, and Sangbae Kim. "MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadru-ped Robot". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain: IEEE Press, 2018, pp. 2245–2252 (cit. on p. 11).

[BKR20] Miroslav Bogdanovic, Majid Khadiv, and Ludovic Righetti. "Learning variable impedance control for contact sensitive tasks". In: *IEEE Robotics and Automation Letters* 5.4 (2020). ISSN: 23773766. DOI: 10.1109/LRA.2020.3011379 (cit. on pp. 86, 88).

[BL05] J.C. Bongard and H. Lipson. "Nonlinear system identification using coevolution of models and tests". In: *IEEE Transactions on Evolutionary Computation* 9.4 (2005), pp. 361–384. DOI: 10.1109/TEVC.2005.850293 (cit. on p. 18).

[Bor+21] Mahrokh Ghoddousi Boroujeni, Elham Daneshmand, Ludovic Righetti, and Majid Khadiv. "A unified framework for walking and running of bipedal robots". In: *CoRR* abs/2110.09172 (2021). arXiv: 2110.09172 (cit. on pp. 5, 74).

[Bro+16] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540 (cit. on p. 105).

[Cam] Tuebingen Campus. *Max Planck Institute*. https://www.tuebingen.mpg.de/en. Accessed: 2023 - 08 - 20 (cit. on p. 5).

[CFH20] Jan Carius, Farbod Farshidian, and Marco Hutter. "MPC-Net: A First Principles Guided Policy Search". In: *IEEE Robotics and Automation Letters* 5.2 (2020). ISSN: 23773766. DOI: 10.1109/LRA.2020.2974653 (cit. on p. 24).

[Car+16] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. "A versatile and efficient pattern generator for generalized legged locomotion". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 3555–3561. DOI: 10.1109/ICRA.2016.7487538 (cit. on pp. 2, 3).

[Car+19] Justin Carpentier et al. "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *IEEE International Symposium on System Integrations (SII)*. 2019 (cit. on p. 31).

[Che+18] Yevgen Chebotar et al. "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience". In: *CoRR* abs/1810.05687 (2018). arXiv: 1810.05687. URL: http://arxiv.org/abs/1810.05687 (cit. on p. 18).

[Che+21] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *CoRR* abs/2107.03374 (2021). arXiv: 2107.03374. URL: https://arxiv.org/abs/2107.03374 (cit. on p. 103).

[CKP23] Xuxin Cheng, Ashish Kumar, and Deepak Pathak. *Legs as Manipulator: Pushing Quadrupedal Agility Beyond Locomotion*. 2023. arXiv: 2303.11330 [cs.RO] (cit. on pp. 40, 65, 102).

[Cho+20] Jinyoung Choi et al. "Fast Adaptation of Deep Reinforcement Learning-Based Navigation Skills to Human Preference". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 3363–3370 (cit. on p. 82).

[Cho+23] Suyoung Choi et al. "Learning quadrupedal locomotion on deformable terrain". In: *Science Robotics* 8.74 (2023), eade2256 (cit. on p. 4).

[Cob+19] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. "Leveraging Procedural Generation to Benchmark Reinforcement Learning". In: *CoRR* abs/1912.01588 (2019). arXiv: 1912.01588. URL: http://arxiv.org/abs/1912.01588 (cit. on p. 20).

[Cob+18] Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. "Quantifying Generalization in Reinforcement Learning". In: *CoRR* abs/1812.02341 (2018). arXiv: 1812.02341. URL: http://arxiv.org/abs/1812.02341 (cit. on p. 20).

[CB21] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2021 (cit. on pp. 31, 41, 105).

[Cub+18] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. "AutoAugment: Learning Augmentation Policies from Data". In: *CoRR* abs/1805.09501 (2018). arXiv: 1805.09501. URL: http://arxiv.org/abs/1805.09501 (cit. on p. 19).

[Cza+18] Wojciech Marian Czarnecki et al. "Mix&Match - Agent Curricula for Reinforcement Learning". In: *CoRR* abs/1806.01780 (2018). arXiv: 1806.01780. URL: http://arxiv.org/abs/1806.01780 (cit. on p. 20).

[Da+20] Xingye Da et al. *Learning a contact-adaptive controller for robust, efficient legged locomotion*. 2020 (cit. on pp. 24, 29, 33).

[Di 20] Jared Di Carlo. “Software and Control Design for the MIT Cheetah Quadruped Robots”. In: 2020. URL: https://dspace.mit.edu/handle/1721.1/129877 (cit. on p. 12).

[Di +18] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. “Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2018. DOI: 10.1109/IROS.2018.8594448 (cit. on pp. 12, 22, 79).

[Dyna] Boston Dynamics. *BigDog Reflexes*. URL: https://www.youtube.com/watch?v=3gi6Ohnp9x8 (cit. on p. 3).

[Dynb] Boston Dynamics. *Spot Mini*. https://www.bostondynamics.com/spot/. Accessed: 2023 - 08 - 10 (cit. on pp. 2, 5).

[Fad+21] Gabriele Fadini, Thomas Flayols, Andrea del Prete, Nicolas Mansard, and Philippe Souères. “Computational design of energy-efficient legged robots: Optimizing for size and actuators”. In: *IEEE International Conference on Robotics and Automation (ICRA 2021)*. 2021 (cit. on p. 50).

[FBH18] Péter Fankhauser, Michael Bloesch, and Marco Hutter. “Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization”. In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3019–3026. DOI: 10.1109/LRA.2018.2849506 (cit. on p. 102).

[Fea08] Roy Featherstone. “Dynamics of Rigid Body Systems”. In: *Rigid Body Dynamics Algorithms*. Boston, MA: Springer US, 2008, pp. 39–64. ISBN: 978-1-4899-7560-7. DOI: 10.1007/978-1-4899-7560-7_3. URL: https://doi.org/10.1007/978-1-4899-7560-7_3 (cit. on p. 18).

[Fre99] Robert M. French. “Catastrophic forgetting in connectionist networks”. In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135 (cit. on p. 52).

[FCP22] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. *Deep Whole-Body Control: Learning a Unified Policy for Manipulation and Locomotion*. 2022. arXiv: 2210.10044 [cs.RO] (cit. on pp. 40, 65, 103).

[Fu+21] Zipeng Fu, Ashish Kumar, Jitendra Malik, and Deepak Pathak. “Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots”. In: (Oct. 2021) (cit. on pp. 65, 68, 76).

[FXP22] Yuni Fuchioka, Zhaoming Xie, and Michiel van de Panne. *OPT-Mimic: Imitation of Optimized Trajectories for Dynamic Quadruped Behaviors*. 2022. DOI: 10.48550/ARXIV.2210.01247 (cit. on pp. 11, 40, 82).

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *CoRR* abs/1802.09477 (2018). arXiv: 1802.09477. URL: http://arxiv.org/abs/1802.09477 (cit. on pp. 41, 105).

[Geh+13] Christian Gehring, Stelian Coros, Marco Hutter, Michael Bloesch, Markus A. Hoepflinger, and Roland Siegwart. “Control of dynamic gaits for a quadrupedal robot”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 3287–3292 (cit. on p. 3).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 1).

[Goo+14] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014 (cit. on p. 18).

[Gra22] Michael C. Granatosky. "Quadrupedal". In: *Encyclopedia of Animal Cognition and Behavior*. Ed. by Jennifer Vonk and Todd K. Shackelford. Cham: Springer International Publishing, 2022, pp. 5828–5834. ISBN: 978-3-319-55065-7. DOI: 10.1007/978-3-319-55065-7_1442. URL: https://doi.org/10.1007/978-3-319-55065-7_1442 (cit. on pp. 22, 23).

[Gra+17] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. "Automated Curriculum Learning for Neural Networks". In: *CoRR* abs/1704.03003 (2017). arXiv: 1704.03003. URL: http://arxiv.org/abs/1704.03003 (cit. on p. 20).

[Gri+20] F. Grimminger et al. "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3650–3657. DOI: 10.1109/LRA.2020.2976639 (cit. on pp. 1, 5, 10, 24, 40, 41).

[Gul95] Vijaykumar Gullapalli. "Skillful control under uncertainty via direct reinforcement learning". In: *Robotics and Autonomous Systems* 15.4 (Oct. 1995), pp. 237–246 (cit. on p. 38).

[Haa+18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290 (cit. on pp. 16, 104).

[HGS15] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461 (cit. on p. 15).

[HT81] Donald F. Hoyt and C. Richard Taylor. "Gait and the energetics of locomotion in horses". In: *Nature* 292.5820 (July 1981), pp. 239–240. ISSN: 00280836. DOI: 10.1038/292239a0. URL: https://www.nature.com/articles/292239a0 (cit. on pp. 21–23, 35, 42).

[Hut+16] Marco Hutter et al. "ANYmal - a highly mobile and dynamic quadrupedal robot". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 38–44. DOI: 10.1109/IROS.2016.7758092 (cit. on pp. 1, 3, 5, 39).

[HLH18] Jemin Hwangbo, Joonho Lee, and Marco Hutter. "Per-contact iteration method for solving contact dynamics". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 895–902. URL: www.raisim.com (cit. on pp. 18, 52, 105).

[Hwa+19] Jemin Hwangbo et al. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* 4.26 (2019). ISSN: 24709476. DOI: 10.1126/scirobotics.aau5872 (cit. on pp. 4, 19, 37, 39, 48, 51, 55, 61, 79, 83, 85, 88).

[Hyu+14] Dong Jin Hyun, Sangok Seok, Jongwoo Lee, and Sangbae Kim. “High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the MIT Cheetah”. In: *The International Journal of Robotics Research* 33.11 (2014), pp. 1417–1445 (cit. on p. 4).

[Ini] Open Robots Dynamics Initiative. *Open Motors Drivers Initiative*. https://github.com/open-dynamic-robot-initiative/open-motor-driver-initiative. Accessed: 2023 - 08 - 30 (cit. on p. 73).

[JIC20] Deepali Jain, Atil Iscen, and Ken Caluwaerts. “From Pixels to Legs: Hierarchical Learning of Quadruped Locomotion”. In: (Nov. 2020) (cit. on p. 82).

[Jeo] Jeongmin Jeon. *ICRA 2023 Quadruped Robot Challenges*. https://quadruped-robot-challenges.notion.site/. Accessed: 2023 - 08 - 30 (cit. on p. 73).

[Ji+22a] Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. “Concurrent Training of a Control Policy and a State Estimator for Dynamic and Robust Legged Locomotion”. In: *IEEE Robotics and Automation Letters* 7.2 (Feb. 2022), pp. 4630–4637 (cit. on pp. 12, 39, 47, 49, 51, 53, 55, 62, 86).

[JMA23] Yandong Ji, Gabriel B. Margolis, and Pulkit Agrawal. *DribbleBot: Dynamic Legged Manipulation in the Wild*. 2023. arXiv: 2304.01159 [cs.RO] (cit. on pp. 40, 103).

[Ji+22b] Yandong Ji et al. “Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills using a Quadrupedal Robot”. In: *International Conference on Intelligent Robots and Systems* (2022) (cit. on pp. 40, 103).

[KAI] KAIST. *Korea Advanced Institute of Science and Technology*. https://www.kaist.ac.kr/en. Accessed: 2023 - 08 - 20 (cit. on p. 2).

[KL02] Sham Kakade and John Langford. “Approximately Optimal Approximate Reinforcement Learning”. In: *Proceedings of the Nineteenth International Conference on Machine Learning*. ICML ’02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 267–274. ISBN: 1558608737 (cit. on p. 16).

[Kal+12] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. “Learning force control policies for compliant robotic manipulation”. In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 1 (Sept. 2012), pp. 4639–4644 (cit. on p. 38).

[Kan+04a] F. Kanehiro et al. “Locomotion planning of humanoid robots to pass through narrow spaces”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 1. 2004, 604–609 Vol.1. DOI: 10.1109/ROBOT.2004.1307215 (cit. on p. 2).

[Kan+04b] K. Kaneko et al. “Humanoid robot HRP-2”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 2. 2004, 1083–1090 Vol.2. DOI: 10.1109/ROBOT.2004.1307969 (cit. on p. 1).

[Kan+19] Kenji Kaneko et al. “Humanoid Robot HRP-5P: An Electrically Actuated Humanoid Robot With High-Power and Wide-Range Joints”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1431–1438 (cit. on pp. 1, 73, 74).

[Kat18] Benjamin Katz. “A Low Cost Modular Actuator For Dynamic Robots”. In: 2018. URL: https://dspace.mit.edu/handle/1721.1/118671 (cit. on p. 11).

[KCK19] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. “Mini cheetah: A platform for pushing the limits of dynamic quadruped control”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 6295–6301 (cit. on pp. 1, 4, 11, 22, 87).

[Kim+19] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. *Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control*. 2019 (cit. on pp. 12, 22, 24, 37, 67, 79).

[KW22] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML] (cit. on p. 62).

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015 (cit. on p. 53).

[Kob+10] Jens Kober, Katharina Mülling, Oliver Krömer, Christoph H. Lampert, Bernhard Schölkopf, and Jan Peters. “Movement templates for learning of hitting and batting”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2010), pp. 853–858 (cit. on p. 38).

[KS04] Nate Kohl and Peter Stone. “Policy gradient reinforcement learning for fast quadrupedal locomotion”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2004.3 (2004), pp. 2619–2624 (cit. on p. 38).

[KT99] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999 (cit. on pp. 16, 87).

[Kou+13] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. *Evolving large-scale neural networks for vision-based reinforcement learning*. New York, New York, USA, 2013 (cit. on p. 39).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *ImageNet classification with deep convolutional neural networks*. 2012 (cit. on pp. 1, 39).

[Kum+21] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. “RMA: Rapid Motor Adaptation for Legged Robots”. In: *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*. Ed. by Dylan A. Shell, Marc Toussaint, and M. Ani Hsieh. 2021 (cit. on pp. 39, 40, 48, 62, 65, 79, 84, 88).

[Kum+22] Ashish Kumar, Zhongyu Li, Jun Zeng, Deepak Pathak, Koushil Sreenath, and Jitendra Malik. *Adapting Rapid Motor Adaptation for Bipedal Robots*. 2022. arXiv: 2205.15299 [cs.RO] (cit. on p. 65).

[LAA] LAAS/CNRS. *Laboratoire d’Analyse et d’Architecture Des Systèmes*. www.laas.fr/public/. Accessed: 2023 - 08 - 20 (cit. on p. 5).

[Lab] Leg Lab. *MIT Leg Laboratory*. www.ai.mit.edu/projects/leglab. Accessed: 2023 - 08 - 20 (cit. on p. 2).

[LR13] Thomas Lampe and Martin Riedmiller. "Acquiring Visual Servoing Reaching and Grasping Skills using Neural Reinforcement Learning". In: *IEEE International Joint Conference on Neural Networks (IJCNN 2013)*. Dallas, TX, 2013 (cit. on p. 15).

[LBH15] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444 (cit. on p. 39).

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2323 (cit. on pp. 1, 39).

[LeC+89] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989 (cit. on p. 1).

[LHH19] Joonho Lee, Jemin Hwangbo, and Marco Hutter. "Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning". In: (Jan. 2019) (cit. on pp. 81, 82).

[Lee+20] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning quadrupedal locomotion over challenging terrain". In: *Science Robotics* 5.47 (2020). ISSN: 24709476 (cit. on pp. 4, 19, 24, 37, 39, 51, 75, 79, 85, 102).

[LK13a] Sergey Levine and Vladlen Koltun. "Guided Policy Search". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1–9. URL: https://proceedings.mlr.press/v28/levine13.html (cit. on p. 24).

[LK13b] Sergey Levine and Vladlen Koltun. "Guided policy search". In: *30th International Conference on Machine Learning, ICML 2013*. ICML'13 PART 2. JMLR.org, 2013, pp. 1038–1046 (cit. on p. 39).

[Léz22] Pierre-Alexandre Léziart. "Locomotion control of a lightweight quadruped robot". Theses. Université Paul Sabatier - Toulouse III, Oct. 2022. URL: https://hal.laas.fr/tel-03936109 (cit. on pp. 11, 25).

[Léz+22] Pierre-Alexandre Léziart, Thomas Corbères, Thomas Flayols, Steve Tonneau, Nicolas Mansard, and Philippe Souères. "Improved Control Scheme for the Solo Quadruped and Experimental Comparison of Model Predictive Controllers". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 9945–9952. DOI: 10.1109/LRA.2022.3192581 (cit. on p. 11).

[Léz+20] Pierre-Alexandre Léziart, Thomas Flayols, Felix Grimminger, Nicolas Mansard, and Philippe Souères. "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12". working paper or preprint. Dec. 2020. URL: https://hal.laas.fr/hal-03052451 (cit. on pp. 6, 11, 22–25, 35, 38, 52, 79, 99).

[Li+23a] C. Li, S. Blaes, P. Kolev, M. Vlastelica, J. Frey, and G. Martius. "Versatile Skill Control via Self-supervised Adversarial Imitation of Unlabeled Mixed Motions". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May 2023 (cit. on p. 40).

[Li+22a] Chenhao Li, Marin Vlastelica, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius. "Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations". In: *Conference on Robot Learning* (2022) (cit. on pp. 11, 40).

[Li+22b] Chenhao Li, Marin Vlastelica, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius. "Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations". In: *6th Annual Conference on Robot Learning*. 2022 (cit. on pp. 11, 82).

[Li+23b] Zhongyu Li, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. *Robust and Versatile Bipedal Jumping Control through Reinforcement Learning*. 2023. arXiv: 2302.09450 [cs.RO] (cit. on pp. 74, 82, 103).

[Li+21] Zhongyu Li et al. "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2021-May (2021), pp. 2811–2817. ISSN: 10504729 (cit. on pp. 39, 74, 103).

[Lia+23] Jacky Liang et al. *Code as Policies: Language Model Programs for Embodied Control*. 2023. arXiv: 2209.07753 [cs.RO] (cit. on p. 103).

[Lil+16] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1509.02971 (cit. on p. 15).

[LKM22] Antonio Loquercio, Ashish Kumar, and Jitendra Malik. *Learning Visual Locomotion with Cross-Modal Supervision*. 2022 (cit. on pp. 40, 65, 102).

[MHN13] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cit. on p. 52).

[Mak+21] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021 (cit. on pp. 39, 66, 87).

[MA22] Gabriel B Margolis and Pulkit Agrawal. "Walk These Ways: Tuning Robot Control for Generalization with Multiplicity of Behavior". In: *Conference on Robot Learning* (2022) (cit. on pp. 82, 96).

[Mar+22] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. *Rapid Locomotion via Reinforcement Learning*. 2022. DOI: 10.48550/ARXIV.2205.02824 (cit. on pp. 12, 39, 65, 66, 88).

[Mar+21] Gabriel B. Margolis et al. *Learning to Jump from Pixels*. 2021. arXiv: 2110.15344 [cs.RO] (cit. on pp. 12, 40).

[Mas+19] Carlos Mastalli et al. “Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control”. In: *CoRR* abs/1909.04947 (2019). arXiv: 1909.04947. URL: http://arxiv.org/abs/1909.04947 (cit. on p. 4).

[Mat+17] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. “Teacher-Student Curriculum Learning”. In: *CoRR* abs/1707.00183 (2017). arXiv: 1707.00183. URL: http://arxiv.org/abs/1707.00183 (cit. on p. 20).

[MR15] Francisco Melo and Isabel Ribeiro. “Convergence of Q-learning with linear function approximation”. In: *2007 European Control Conference, ECC 2007* (Mar. 2015) (cit. on p. 14).

[Mik+22] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. “Learning robust perceptive locomotion for quadrupedal robots in the wild”. In: *Science Robotics* 7.62 (2022), eabk2822 (cit. on pp. 4, 5, 37, 39, 51, 55, 75, 83, 85, 96, 102).

[MP69] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969 (cit. on p. 52).

[Mni+13] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Dec. 2013 (cit. on p. 39).

[Mni+15] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature 2015 518:7540* 518.7540 (Feb. 2015), pp. 529–533 (cit. on pp. 1, 14, 17, 39, 42).

[Mni+16] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783 (cit. on p. 16).

[MAS22] Gianluca Monaci, Michel Aractingi, and Tomi Silander. “DiPCAN: Distilling Privileged Information for Crowd-Aware Navigation”. In: *Robotics: Science and Systems (RSS) XVIII*. 2022 (cit. on p. 65).

[Mun+16] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. “Safe and Efficient Off-Policy Reinforcement Learning”. In: (2016) (cit. on p. 15).

[NYM23] Aswin Nahrendra, Byeongho Yu, and Hyun Myung. *DreamWaQ: Learning Robust Quadrupedal Locomotion With Implicit Terrain Imagination via Deep Reinforcement Learning*. 2023 (cit. on p. 65).

[NAA03] K. Nakada, T. Asai, and Y. Amemiya. “An analog CMOS central pattern generator for interlimb coordination in quadruped locomotion”. In: *IEEE Transactions on Neural Networks* 14.5 (2003), pp. 1356–1365. DOI: 10.1109/TNN.2003.816381 (cit. on p. 5).

[Nel+12] Gabe Nelson et al. “PETMAN: A Humanoid Robot for Testing Chemical Protective Clothing”. In: *Journal of the Robotics Society of Japan* 30.4 (2012), pp. 372–377. DOI: 10.7210/jrsj.30.372 (cit. on p. 3).

[NHR99] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287. ISBN: 1558606122 (cit. on p. 43).

[NLE] NLE. *Naver Labs Europe*. https://europe.naverlabs.com/. Accessed: 2023 - 08 - 20 (cit. on p. 6).

[ODR] ODRI. *Open Dynamic Robot Initiative*. https://open-dynamic-robot-initiative.github.io/. Accessed: 2023 - 08 - 10 (cit. on pp. 9, 11).

[PWK15] Hae-Won Park, Patrick Wensing, and Sangbae Kim. "Online Planning for Autonomous Running Jumps Over Obstacles in High-Speed Quadrupeds". In: *Proceedings of Robotics: Science and Systems*. Rome, Italy, July 2015. DOI: 10.15607/RSS.2015.XI.047 (cit. on p. 4).

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035 (cit. on p. 66).

[Pat+21] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. "Hierarchical Reinforcement Learning: A Comprehensive Survey". In: *ACM Comput. Surv.* 54.5 (June 2021). ISSN: 0360-0300 (cit. on p. 80).

[Pen+17a] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *CoRR* abs/1710.06537 (2017). arXiv: 1710.06537. URL: http://arxiv.org/abs/1710.06537 (cit. on pp. 18, 19).

[Pen+17b] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning". In: *ACM Transactions on Graphics*. Vol. 36. 4. 2017. DOI: 10.1145/3072959.3073602 (cit. on pp. 23, 81).

[Pen+19] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. "MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies". In: (2019) (cit. on p. 98).

[Pen+20] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang Wei Edward Lee, Jie Tan, and Sergey Levine. *Learning agile robotic locomotion skills by imitating animals*. 2020. DOI: 10.15607/rss.2020.xvi.064 (cit. on p. 23).

[PP16] Xue Bin Peng and Michiel van de Panne. "Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?" In: *Proceedings - SCA 2017: ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Nov. 2016) (cit. on pp. 23, 43, 48).

[PS08] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4 (May 2008), pp. 682–697 (cit. on p. 38).

[PR92] R.R. Playter and M.H. Raibert. “Control Of A Biped Somersault In 3D”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 1. 1992, pp. 582–589. DOI: 10.1109/IROS.1992.587396 (cit. on p. 2).

[PW95] G.A. Pratt and M.M. Williamson. “Series elastic actuators”. In: *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*. Vol. 1. 1995, 399–406 vol.1. DOI: 10.1109/IROS.1995.525827 (cit. on p. 3).

[Rad+23] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. “Learning Humanoid Locomotion with Transformers”. In: *arXiv:2303.03381* (2023) (cit. on pp. 74, 103).

[Raf+21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 268 (2021), pp. 1–8 (cit. on p. 53).

[RCB86] M. Raibert, M. Chepponis, and H. Brown. “Running on four legs as though they were one”. In: *IEEE Journal on Robotics and Automation* 2.2 (1986), pp. 70–82. DOI: 10.1109/JRA.1986.1087044 (cit. on p. 2).

[Rai90] Marc H. Raibert. “Trotting, pacing and bounding by a quadruped robot”. In: *Journal of Biomechanics* 23.1 (Jan. 1990), pp. 79–98. ISSN: 00219290. DOI: 10.1016/0021-9290(90)90043-3 (cit. on p. 25).

[RHC84] Marc H. Raibert, Jr H. Benjamin Brown, and Michael Chepponis. “Experiments in Balance with a 3D One-Legged Hopping Machine”. In: *The International Journal of Robotics Research* 3.2 (1984), pp. 75–92 (cit. on p. 2).

[RMA19] Jacob Reher, Wen-Loong Ma, and Aaron D. Ames. “Dynamic Walking with Compliance on a Cassie Bipedal Robot”. In: *CoRR* abs/1904.11104 (2019). arXiv: 1904.11104. URL: http://arxiv.org/abs/1904.11104 (cit. on p. 73).

[Rud+21] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*. 2021 (cit. on pp. 39, 66, 68, 87, 91).

[RSC19] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. “Learning To Simulate”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HJgkx2Aqt7 (cit. on p. 19).

[RN94] G. A. Rummery and M. Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. Cambridge, England: Cambridge University Engineering Department, 1994 (cit. on p. 14).

[Rut+08] Simon Rutishauser, Alexander Badri-Spröwitz, Ludovic Righetti, and A.J. Ijspeert. “Passive compliant quadruped robot using Central Pattern Generators for locomotion control”. In: Nov. 2008, pp. 710–715. DOI: 10.1109/BIOROB.2008.4762878 (cit. on p. 5).

[SL16] Fereshteh Sadeghi and Sergey Levine. “(CAD)

$2RL : RealSingle-ImageFlightwithoutaSingleRealImage$". In: *CoRR* abs/1611.04201 (2016). arXiv: 1611.04201. URL: http://arxiv.org/abs/1611.04201 (cit. on p. 19).

[Sak+02] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. "The intelligent ASIMO: system overview and integration". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 3. 2002, 2478–2483 vol.3. DOI: 10.1109/IRDS.2002.1041641 (cit. on pp. 1, 2).

[STD19] Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. "Data Parameters: A New Family of Parameters for Learning a Differentiable Curriculum". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019 (cit. on p. 20).

[Sch+19] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: (2019). arXiv: 1911.08265 (cit. on p. 1).

[Sch+15] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. "Trust region policy optimization". In: *32nd International Conference on Machine Learning, ICML 2015* 3 (Feb. 2015), pp. 1889–1897 (cit. on pp. 16, 39, 87).

[Sch+16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016 (cit. on pp. 17, 66).

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: (2017) (cit. on pp. 16, 31, 39, 41, 47, 52, 53, 87, 105).

[Sem+11] Claudio Semini, Nikos G Tsagarakis, Emanuele Guglielmino, Michele Focchi, Ferdinando Cannella, and D G Caldwell. "Design of HyQ – a hydraulically and electrically actuated quadruped robot". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 225.6 (2011), pp. 831–849 (cit. on p. 3).

[Sem+17] Claudio Semini et al. "Design of the Hydraulically Actuated, Torque Controlled Quadruped Robot HyQ2Max". In: *IEEE/ASME Transactions on Mechatronics* 22.2 (2017), pp. 635–646 (cit. on p. 3).

[Seo+13] Sangok Seok, Albert Wang, Meng Yee Chuah, David Otten, Jeffrey Lang, and Sangbae Kim. "Design principles for highly efficient quadrupeds and implementation on the MIT Cheetah robot". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 3307–3312. DOI: 10.1109/ICRA.2013.6631038 (cit. on pp. 4, 11).

[Sin+23] Rohan Pratap Singh, Zhaoming Xie, Pierre Gergondet, and Fumio Kanehiro. *Learning Bipedal Walking for Humanoids with Current Feedback*. 2023. arXiv: 2303.03724 [cs.RO] (cit. on pp. 74, 103).

[SR07] Anders Skronal and Sophia Rabe-Hesketh. “Latent Variable Modelling: A Survey”. In: *Scandinavian Journal of Statistics* 34.4 (2007), pp. 712–745. ISSN: 03036898, 14679469. URL: http://www.jstor.org/stable/41548578 (visited on 07/11/2023) (cit. on p. 62).

[Smi+21] Laura Smith, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. *Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World*. 2021. arXiv: 2110.05457 [cs.RO] (cit. on p. 104).

[SKL22] Laura Smith, Ilya Kostrikov, and Sergey Levine. *A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning*. 2022. arXiv: 2208.07860 [cs.RO] (cit. on p. 104).

[Smi+23] Laura Smith et al. *Learning and Adapting Agile Locomotion Skills by Transferring Experience*. 2023. arXiv: 2304.09834 [cs.RO] (cit. on p. 104).

[Son+20] Xingyou Song et al. *Rapidly adaptable legged robots via evolutionary meta-learning*. 2020 (cit. on p. 23).

[SS20] Yunlong Song and Davide Scaramuzza. *Learning High-Level Policies for Model Predictive Control*. 2020 (cit. on p. 24).

[Sta+17] O. Stasse et al. “TALOS: A new humanoid research platform targeted for industrial applications”. In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. 2017, pp. 689–695. DOI: 10.1109/HUMANOIDS.2017.8246947 (cit. on pp. 1, 73).

[Suk+17] Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. “Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play”. In: *CoRR* abs/1703.05407 (2017). arXiv: 1703.05407. URL: http://arxiv.org/abs/1703.05407 (cit. on p. 20).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215 (cit. on p. 1).

[Sut+99] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999 (cit. on p. 15).

[Sut88] Richard S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. In: *Mach. Learn.* 3.1 (Aug. 1988), pp. 9–44. ISSN: 0885-6125. DOI: 10.1023/A:1022633531479. URL: https://doi.org/10.1023/A:1022633531479 (cit. on p. 14).

[SB81] Richard S. Sutton and Andrew G. Barto. “Toward a Modern Theory of Adaptive Networks: Expectation and Prediction”. In: *Psychological Review* 88.2 (1981), pp. 135–170. DOI: 10.1037/0033-295x.88.2.135 (cit. on p. 14).

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: www.incompleteideas.net/book/the-book-2nd.html (cit. on pp. 6, 12, 14, 38, 82).

[Sze10] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Vol. 4. Jan. 2010 (cit. on pp. 12, 13).

[Tan+23] Yujin Tang, Wenhao Yu, Jie Tan, Heiga Zen, Aleksandra Faust, and Tatsuya Harada. *SayTap: Language to Quadrupedal Locomotion*. 2023. arXiv: 2306.07580 [cs.RO] (cit. on p. 103).

[Tea] Gepetto Team. *Movement of Anthropomorphic Systems, LAAS-CNRS*. www.gepettoweb.laas.fr/index.php/. Accessed: 2023 - 08 - 30 (cit. on p. 73).

[Tob+17] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. arXiv: 1703.06907 [cs.RO] (cit. on pp. 18, 19).

[Ton+18] Steve Tonneau, Andrea Del Prete, Julien Pettre, Chonhyon Park, Dinesh Manocha, and Nicolas Mansard. "An efficient acyclic contact planner for multiped robots". In: *IEEE Transactions on Robotics* 34.3 (June 2018) (cit. on pp. 80, 81).

[Tso+20] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. "DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 5.2 (2020). ISSN: 23773766. DOI: 10.1109/LRA.2020.2979660 (cit. on p. 24).

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. Vol. 2017-December. Neural information processing systems foundation, June 2017, pp. 5999–6009. arXiv: 1706.03762. URL: https://arxiv.org/abs/1706.03762v5 (cit. on pp. 23, 30, 31).

[Wan18] Xingxing Wang. *Laikago Pro, Unitree Robotics*. 2018. URL: http://www.unitree.cc/e/action/ShowInfo.php?classid=6&id=355 (cit. on p. 1).

[Wan] Xingxing Wang. *Unitree Robotics*. https://www.unitree.com/. Accessed: 2023 - 08 - 10 (cit. on p. 4).

[Wat89] Christopher Watkins. "Learning From Delayed Rewards". In: (Jan. 1989) (cit. on p. 14).

[WC18] Daphna Weinshall and Gad Cohen. "Curriculum Learning by Transfer Learning: Theory and Experiments with Deep Networks". In: (2018). arXiv: 1802.03796 (cit. on p. 20).

[Wen19] Lilian Weng. "Domain Randomization for Sim2Real Transfer". In: *lilianweng.github.io* (2019). URL: https://lilianweng.github.io/posts/2019-05-05-domain-randomization/ (cit. on p. 18).

[Wil92] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696 (cit. on pp. 15, 42).

[Win+18] Alexander W. Winkler, C. Dario Bellicoso, Marco Hutter, and Jonas Buchli. "Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1560–1567. DOI: 10.1109/LRA.2018.2798285 (cit. on p. 26).

[Wüt+20] Manuel Wüthrich et al. "TriFinger: An Open-Source Robot for Learning Dexterity". In: *CoRR* abs/2008.03596 (2020). arXiv: 2008.03596. URL: https://arxiv.org/abs/2008.03596 (cit. on p. 5).

[YYW23] Ruihan Yang, Ge Yang, and Xiaolong Wang. *Neural Volumetric Memory for Visual Locomotion Control*. 2023. arXiv: 2304.01201 [cs.RO] (cit. on p. 102).

[Yan+21] Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. "Fast and Efficient Locomotion via Learned Gait Transitions". In: *CoRR* abs/2104.04644 (2021). arXiv: 2104.04644. URL: https://arxiv.org/abs/2104.04644 (cit. on pp. 24, 29).

[YLT17] Wenhao Yu, C. Karen Liu, and Greg Turk. "Preparing for the Unknown: Learning a Universal Policy with Online System Identification". In: *CoRR* abs/1702.02453 (2017). arXiv: 1702.02453. URL: http://arxiv.org/abs/1702.02453 (cit. on p. 18).

[ZS14] Wojciech Zaremba and Ilya Sutskever. "Learning to Execute". In: *CoRR* abs/1410.4615 (2014). arXiv: 1410.4615. URL: http://arxiv.org/abs/1410.4615 (cit. on p. 20).

Abstract

For many years, researchers have been trying to develop and study legged machines that imitate animals and humans. The legs allow the agent to navigate on different terrains and through obstacles and steps where wheeled robots fail. In recent years, several quadruped robots have been invented that can produce high torque density and withstand impact. As a result, research on locomotion accelerated because many mechanical challenges could be solved on these new platforms. However, due to the complexity of controlling these robots and their underactuated nature, generating robust locomotion remains a problem. Traditional methods based on modeling and optimization produce efficient locomotion, but are difficult to adapt to different situations and lack robustness. Over the past decade, progress in the field of « deep reinforcement learning » have led many researchers to use these techniques for learning robots. The goal of these methods is to learn control policies from interaction data by maximizing a reward function that represents the desired task.

In this thesis, we explore and develop deep reinforcement learning methods for quadruped locomotion. We had access to two quadruped robots, the Solo-12 and MIT's Mini-Cheetah. First, we developed a learning based method that is complementary to the model-based controller. The proposed approach modifies the nominal gait of Solo-12 that is executed by a controller based on model predictive control (MPC) which controls the trade-off velocity tracking and energy consumption. We then describe a method based on end-to-end policy learning of joint angle controllers for Solo-12. The goal of this policy is to control the robot to follow on a user-defined velocity command. We detail the definition of states, actions, reward functions and propose a term based on the energy losses in order to represent the real energy consumption on the real robot. We then show an attempt to transfer the method developed for Solo-12 to Mini-Cheetah. Several difficulties were encountered in transferring the policy to the Mini-Cheetah. To overcome them, we have developed a more complex approach based on distillation in order to learn a representation of unobservable privileged parameters, linked to the dynamics of the environment and the robot.

Finally, we propose a hierarchical approach to locomotion where the low-level policy is tasked to optimize different parameterization of the reward and control. We argue that many features of the underlying locomotion, are not represented in the high-level task of velocity tracking, such as, swing feet height, step length and expended energy. We propose an approach to learn control policies augmented with parameters that modify different aspects of the reward function and control setup which, in turn, results in variations of the locomotion. We can then define a hierarchical architecture where a high level policy infers the suitable parameters to complete a given task.

This thesis contributes to the locomotion of legged robots as we implemented and deployed joint angle controllers learned with deep reinforcement learning on the Solo-12 robot and Mini-Cheetah. We conducted many experiments on the real robots and documented the complications and difficulties that arise from working with both systems.

Keywords

Deep Reinforcement Learning, Locomotion, Quadruped Robots

Résumé

Depuis de nombreuses années, les chercheurs tentent de développer et d'étudier des machines à pattes imitant les animaux et les humains. Les pattes permettent à l'agent de naviguer sur différents terrains et de franchir les obstacles et les marches là où les robots à roues échouent. Ces dernières années, plusieurs robots quadrupèdes ont été développés, capables de produire une densité de couple élevée et de résister aux chocs. En conséquence, la recherche sur la locomotion s'est accélérée car de nombreux défis mécaniques ont pu être résolus sur ces nouvelles plates- formes. Cependant, en raison de la complexité du contrôle de ces robots et de leur nature sous- actionnée il est difficile de leur conférer une locomotion robuste. Les méthodes traditionnelles basées sur la modélisation et l'optimisation produisent une locomotion efficace, mais sont difficiles à adapter aux différentes situations et manquent de robustesse. Au cours de la dernière décennie, les progrès dans le domaine de « l'apprentissage par renforcement profond » ont incité les chercheurs à utiliser cette nouvelle approche en robotique. Ces méthodes permettent d'apprendre des politiques de contrôle à partir des données d'interaction en maximisant une fonction de récompense qui permet d'exécuter la tâche souhaitée.

Dans cette thèse, nous explorons et développons des méthodes d'apprentissage par renforcement profond pour la locomotion des quadrupèdes. Nous avons eu accès à deux robots quadrupèdes, le Solo-12 du LAAS et le Mini-Cheetah du MIT. Tout d'abord, nous avons développé une méthode d'apprentissage venant en complément d'un contrôleur basé-modèle. L'approche proposée modifie la locomotion nominale de Solo-12 produite par un contrôleur basé sur l'optimisation prédictive basée-modèle (MPC), assurant un compromis entre le suivi de la vitesse et la consommation d'énergie. Nous décrivons l'approche que nous avons développée pour apprendre de bout en bout d'une politique de commande des angles des liaisons de Solo-12. Cette politique permet de contrôler le robot pour suivre une vitesse de commande définie par l'utilisateur. Nous définissons les états, les actions et la fonction de récompense, ainsi que d'un terme représentant les pertes énergétiques sur le robot réel, que nous introduisons dans la récompense afin de simuler la consommation énergétique réelle. Nous montrons ensuite que la méthode qui nous a permis de réaliser le transfert des politiques sur Solo-12 ne peut pas être directement utilisée pour Mini-Cheetah. Pour y parvenir, nous avons développé une approche plus complexe basée sur la distillation afin d'apprendre une représentation de paramètres privilégiés inobservables, liés à la dynamique de l'environnement et du robot.

Enfin, nous proposons une approche hiérarchique de la locomotion dans laquelle la politique de bas niveau est chargée d'optimiser différents paramétrages de la récompense et du contrôle. Nous mettons en évidence que de nombreuses caractéristiques sous-jacentes de la locomotion ne sont pas représentées dans la tâche de suivi de la vitesse, telles que la hauteur des pieds en vol, la longueur des pas et l'énergie dépensée. Nous proposons une approche pour apprendre les politiques de contrôle augmentées de paramètres permettant de modifier différents aspects de la fonction de récompense et de la configuration du contrôle, ce qui, en retour, entraîne des variations de la locomotion pouvant être contrôlées par une politique de haut niveau.

Cette thèse contribue à la locomotion des robots à pattes en développant et en déployant des contrôleurs d'angle articulaires appris par apprentissage par renforcement profond sur les robot Solo-12 et Mini-Cheetah. De nombreuses expériences ont été menées sur chacun de ces robots pour valider ces développements et les difficultés rencontrées ont été détaillées.

Mots clefs

Apprentissage par Renforcement, Locomotion, Robots Quadrupède