Imperial College London

Imperial College London

Department of Computing

Open-Ended Curriculum Learning for Dynamic Robot Locomotion

Supervisor:
Dr. Antoine Cully

Author:
Davide Paglieri

Second Marker:
Dr. Edward Johns

Submitted in partial fulfillment of the requirements for the MSc degree in Computing (Artificial Intelligence & Machine Learning) of Imperial College London

September 2, 2021

Abstract

Legged robots have the potential to revolutionise the world as we know it, taking up dangerous jobs that put the lives of humans at risk. However, from cognitive intelligence to simple locomotion, a great deal of work remains to be done to make that a reality. Reinforcement Learning holds the promise to radically improve upon existing control technologies in robotics while drastically reducing the engineering time. Besides the athletic prowess and reliability of the physical robot, two critical aspects needed for Reinforcement Learning to succeed in the above are outstanding control architectures and training techniques. This research focuses on the latter. After building a solid control architecture for a quadruped robot, we experiment with different Curriculum Learning techniques in simulation to achieve dynamic locomotion in challenging terrains. We compare the training of generalist against specialist agents, open-ended against close-ended learning, and show that training a generalist agent with open-ended learning achieves the best results allowed by the control architecture. The performance of the resulting agent can be seen here.

Acknowledgments

I want to thank my supervisor, Dr Antoine Cully, for the invaluable support throughout the project and for allowing me the freedom to explore the research domains I wanted while guiding me through what was feasible and what was not. Thanks to Bryan Lim for the long and enlightening discussions and the many pieces of advice, I appreciated your help a lot.

Thanks to Atil Iscen at Google Brain for answering all my questions regarding his research and helping me implement his control architecture.

Thanks to all the friends I have made at Imperial and to the members of the AIR-Lab, you made this year a special one.

Lastly, thank you to my Mum, without whom all of this wouldn't have been possible, for believing in me every step of the way.

Contents

Chapter 1

Introduction

1.1 Motivation

Legged animals can traverse a remarkable variety of challenging environments that wheeled robots could never access. From climbing rocks to nimbly running through highly unstructured terrains, animals provide incredible examples of what can be achieved by legged locomotion. It comes as a consequence that robot locomotion is an attractive field of research with extraordinary potential. Legged robots can carry weights, handle tools and shape the environment in ways that airborne drones are incapable of doing. These robots could one day be deployed instead of humans in various dangerous situations, such as search and rescue missions, working in factories, aiding humans as construction workers, and even acting as terrestrial probes for exploring planets and moons alongside rovers (Figure 1.1). The foundations of all the above must be built on dynamic and robust locomotion, which this project will focus on. We will experiment with quadruped robots; however, the same methods can be used by all kinds of legged robots.

Today, the two main approaches for legged locomotion are robot learning and the more classical Model Predictive Control (MPC) with hand-crafted models. Both techniques have their advantages that we will discuss more in detail later. However, in this project, we will use model-free Deep Reinforcement Learning, one of the most popular and promising robot learning techniques.

Robot learning solutions that attempt to learn locomotion from scratch are trained in simulation and can take the equivalent of years of real-time training to learn how to walk. On the other hand, several animals like horses and giraffes can walk within an hour of being born, suggesting that locomotion is not learnt tabula rasa but is built on innate primitives [3]. Starting from this idea, many robot learning techniques inject prior knowledge to make the learning process significantly faster. We will adopt the same strategy and use a periodic prior to locomotion [4] that will speed up training. We will include historical measurements and tune the architecture to provide a good baseline for further experiments.

After building a solid control architecture, we will explore different curriculum learning methods to achieve the best results possible in challenging terrains. In particular, we

Figure 1.1: On the top, two ANYmal robots by ANYbotics [1] deployed in a search and rescue mission (left) and a dangerous factory (right). On the bottom, two Spot robots by Boston Dynamics [2] deployed in a construction site (left) and a Martian-like cave (right)

will set out to answer two critical questions. Firstly: is it more effective to train a single generalist agent capable of solving a vast amount of tasks with good generalisation, or is it better to train a population of specialised individuals, highly performing in a single task, and switching among them when necessary? Secondly: is open-ended learning in a robotics setting a better solution than a close-ended and carefully crafted curriculum to achieve good athletic performance? The answers to these questions will help to guide future robot learning research in a promising direction.

1.2 Contributions

The contributions of the thesis are the following:

- We explore two different curriculum learning paradigms, train a generalist agent and a population of specialist agents in a robotics setting, and compare their performances, showing that generalist achieves the best results.
- We explore and compare how open-ended learning and close-ended learning perform in a robotics setting.
- We show that open-ended generation of terrains produces a curriculum of high-quality and diverse terrains, better than those hand-crafted with domain knowledge.
- We demonstrate that open-ended curriculum learning of a single generalist agent is the best way to achieve high performance in dynamic legged robot locomotion.

1.3 Report outline

The report is organised as follows:

- Chapter 2 introduces concepts relevant to Reinforcement Learning, gradient-based methods, gradient-free methods and quality-diversity.
- Chapter 3 reviews some related literature on quadruped robot locomotion and learning techniques
- Chapter 4 describes the control architecture used
- Chapter 5 details the different curricula through which the robot will be trained and underlines the key ideas that the experiments test
- Chapter 6 discusses the experiments and analyses their results
- Chapter 7 concludes the report by highlighting the findings made in the arguments of specialist vs generalist agents and open-ended vs close-ended learning, and suggests where future research should focus.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement Learning is a Machine Learning framework for learning how to solve control problems by optimal sequential decision-making. Intuitively, an autonomous system - the agent - learns a policy by interacting with the environment. A reward function guides the agent and provides feedback useful to obtain the desired behaviour. More formally, Reinforcement Learning problems are described with Markov Decision Process (MDP) tuples $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}_{ss'}$ is a state transition probability matrix, $\mathcal{R}_s = [r_{t+1}|S_t = s]$ is an expected immediate reward, and $\gamma \in [0, 1]$ is the discount factor of the reward. The policy is a function that maps the state s to a probability distribution of actions $\pi(A|S = s)$. At each discrete time step, the agent observes its state s_t and draws an action a_t from its policy conditioned on the state, it will then transition to a state s_{t+1} and receive a reward r_{t+1}.

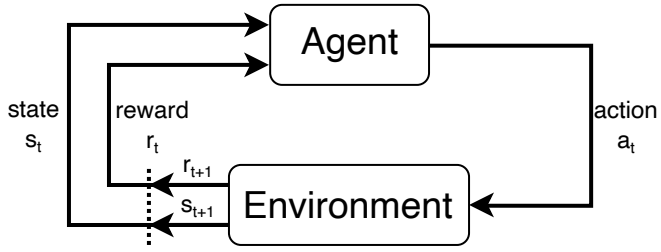

Figure 2.1: Agent-Environment interaction

The agent's transition from a state to another depends on the policy $\pi(a_t|s_t)$, and the transition probability $P(s_{t+1}|s_t, a_t)$. The goal of the agent is to maximize the expected return $\mathbb{E}[R_t]$, that is the sum of the discounted rewards over time

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.1}$$

For practical reasons, we do not allow the agent to interact with the environment for an infinite amount of time. Hence we reduce it to a maximum time length T. After time T has expired, we reset the agent's state in the environment and make them interact again. We

call each sequence of agent-environment interaction an episode. We will continue to use ∞ instead of T as the upper time limit for mathematical convenience.
We can define the value of a certain state for the agent given its policy π as

$$v_\pi(s) = \mathop{\mathbb{E}}_{\pi}[R_t|S_t = s] = \mathop{\mathbb{E}}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s], \tag{2.2}$$

which we call state-value function for policy π. We can also define the action-value function for policy π, which gives information on how good it is for the agent to be in a given state s and choosing a particular action a:

$$q_\pi(s, a) = \mathop{\mathbb{E}}_{\pi}[R_t|S_t = s, A_t = a] = \mathop{\mathbb{E}}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s, A_t = a] \tag{2.3}$$

which is also called Q value function.
We can estimate the values of v_π and q_π with Monte Carlo methods by averaging over many random sample returns. Going back to the objective of Reinforcement Learning, we want our agent to learn an optimal policy that maximizes the return. There might exist more than one optimal policy, however they all share the same optimal state-value functionv_* which is defined as

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.4}$$

or equivalently with the optimal action-value function q_* as

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

q_* and v_* are related as follows:

$$q_*(s, a) = [r_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{2.6}$$

When the state and action spaces are sufficiently small, we can store the approximate value functions in tables. In these cases, the optimal policy can be found with so-called Tabular Methods such as Dynamic Programming, Monte Carlo methods, and Temporal Difference learning[5][6].

2.2 Deep Reinforcement Learning

Limitations of Tabular Methods

Often the state-action space is too large to use tabular methods, and we have to settle for approximate solutions. One of the main limitations of tabular methods is memory, as it might be unfeasible to store the value functions in arrays when the state-action space is vast. This limitation is particularly true in the case of robot locomotion, as the number of Degrees of Freedom (DoF) is high and even worse in the case where visual input is needed, and the number of possible states vastly outnumbers the number of atoms in the universe [5]. Another problem is exploration: each state-action pair must be visited at least once to find out its reward and compute the Q value in tabular methods. Knowledge is not shared in a tabular representation, and this exploration would be too inefficient for big state-action spaces.

Approximate Solutions

These problems can be addressed using approximate solutions, that is, using a continuous function to approximate the Q values that in tabular methods would be stored in a table. The memory problem is solved because we use Neural Networks (NNs) to approximate the underlying Q function. Thus, the number of parameters is constant regardless of how large the state space is. The exploration problem is solved because NNs are continuous functions, and exploring a state will also update its nearby states' Q values. Thus the agent does not have to explore every possible state-action pair, as some Q values will be automatically inferred.

Deep Neural Networks

Artificial Neural Networks are used as universal function approximations [7], and have been one of the leading architects of the incredible success of Deep Reinforcement Learning in the past years [5]. They are inspired by biology, and just like real neural networks, the building blocks are the neurons. Each artificial neuron is linked with a set of input features. Each link is associated with a weight θ that leverages the influence of the corresponding input to that neuron. Next, the weighted inputs are summed, then a bias term b is added. Finally, a non-linear activation function g is applied. We can connect many neurons in parallel so that they will learn to model different features. If we stack more than one layer of neurons in sequence and fully connect the outputs of a layer to the inputs of the next one, we get a Multilayer Perceptron (MLP).

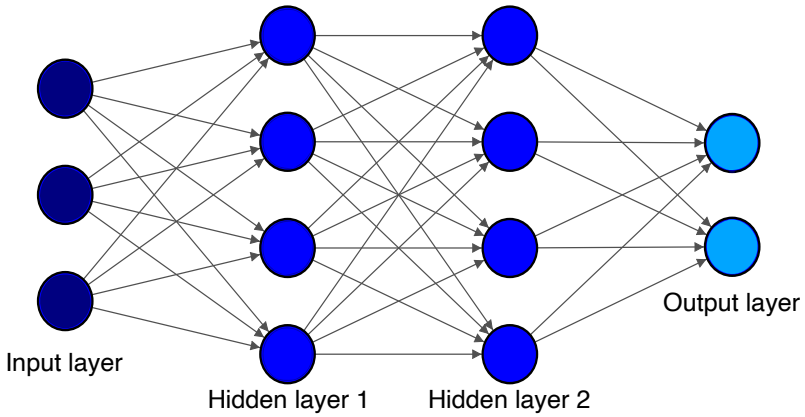

Figure 2.2: Example of a Multilayer Perceptron (MLP) with two hidden layers.

Layers between the first one that takes the original input and the last one that produces an output are called hidden layers. MLPs can model any arbitrary function with a single hidden layer made up of a sufficient number of neurons with suitable weights. However, this is very computationally expensive. What is often done instead is to use deeper and narrower neural networks as they can approximate complex functions while being more compact [7]. These are called Deep Neural Networks, which, as mentioned previously, can be used in Reinforcement Learning to approximate the Q value function, hence the name Deep Reinforcement Learning.

When the inputs to be processed are images, Convolutional Neural Networks (CNNs) are used instead of normal Multilayer Perceptrons. CNNs use weight sharing between neurons to exploit redundancies in pictures and decrease the number of parameters needed.

The weights of Neural Networks can be learnt by training the NNs to provide an output that minimizes a loss function given an input. This is usually achieved through an algorithm called backpropagation, which alternates forward and backwards passes through the network to compute gradients that are used to update the network's weights so that the new output is closer to the desired one [5].

2.3 Gradient-Based methods

Arguably the most popular techniques to train agents in Deep Reinforcement Learning settings are gradient-descent-based methods, meaning that the parameters of the Neural Networks are learnt via the backpropagation algorithm with a Reinforcement Learning objective. Furthermore, these methods are MDP-based, meaning that they work by following the tuple-like structure described in section 2.1 to improve their performance. We will briefly present the most well-known methods in this category and compare them with gradient-free methods like Evolution Strategies (ES).

Q-Learning

The Reinforcement Learning algorithms we have described so far are called Q-Learning, as the aim is to approximate the underlying Q value function. This approximation is usually made with a Neural Network, where the input is a continuous state, and the outputs are the Q-values of the discrete actions. The training is done by using the temporal difference [6], a method that leverages experience to predict future behaviours. The exploration necessary to find the Q-values that most closely approximate the underline Q-value function is done by using a method called ϵ-greedy, which balances exploration and exploitation. In particular, experience is generated by the behaviour policy, which is ϵ-greedy, meaning that it will choose a random action (exploration) with probability ϵ and the best action (exploitation) with probability $1 - \epsilon$. This exploration-exploitation balance is fundamental to find the best possible approximation of the underlying Q-value function. Q-learning is an off-policy algorithm, meaning that it learns by experience that does not follow the policy. On the other hand, the target policy is greedy and always chooses the action with the highest Q-value when deployed on the task. The most famous variant of Q-learning algorithms is the DQN algorithm by DeepMind [8][9].

Policy Gradients

Another class of RL algorithms are the Policy Gradients (PG). Policy Gradients algorithms differ from Q-Learning, as the agent learns a parameterized policy that directly maps the state to actions without the need to use a value function [10]. PG improve the policy by performing gradient ascent with respect to the policy parameters θ, while trying to

maximize an objective function $J(\theta)$ 2.3

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta). \tag{2.7}$$

A requirement is that the policy needs to be differentiable, so Neural Networks are used in practice. The policy can be deterministic, meaning that it outputs a single action for each input state, or stochastic, meaning that it outputs the parameters of a distribution which is then sampled to get the action. In the former, exploration is done by adding noise to the action; in the latter, the policy does exploration directly. Policy gradients methods have a few advantages over Q-Learning. Namely, they are more effective in continuous spaces and can learn stochastic policies. An example of a PG algorithm is the Deep Deterministic Policy Gradients (DDPG) [11].

Some Policy Gradient algorithms learn an approximate value function to assess the actions. They are known as Actor-Critic. The actor refers to the learnt policy and the critic to the learnt value function [10]. Actor-Critic algorithms like TD3 [12], and SAC [13] are currently some of the best performing algorithms by the Reinforcement Learning community.

2.4 Evolution Strategies

An increasingly popular alternative to using gradient-descent-based methods like Q-Learning and Policy Gradients to train Deep Neural Networks for Reinforcement Learning is to use black-box optimization techniques like Evolution Strategies. Although Evolution Strategies have been around for a long time, they have recently been getting more attention due to their competitiveness with gradient descent-based approaches [14][15]. Evolution Strategies are a class of optimization techniques inspired by how species evolve in the real world: they work by evolving a population of individuals through natural selection. At each iteration, the performance of every individual is evaluated by a fitness function. After a selection process, only the fittest survive and pass their genes —parameters— to the offspring with some random mutations —noise—, while the worst individuals die. Although some ES methods estimate gradients by finite difference, they do not explicitly calculate or perform gradient descent.

Neuroevolution, in particular, is a direct policy search technique that evolves the parameters of the policy while, in our case, optimizing a Reinforcement Learning reward function. ES have some notable advantages over gradient-based methods like Q-Learning, Policy Gradients and Actor-Critic, the most obvious one being that no gradient descent and backpropagation is needed. This is especially important when the gradient's landscape is non-smooth, and gradient-based methods struggle to escape local minima, which can often be the case in complex Reinforcement Learning tasks, where sub-optimal solutions are found. Furthermore, exploration in ES is significantly better than in gradient-based methods, as it is done in the parameter space of the policy and not by adding noise or using stochastic policies like PG. Other benefits of ES are that they are embarrassingly parallel, can learn in tasks with very long time horizons and are robust to hyper-parameter choices [14]. ES are better at learning in tasks with long horizons because they do not

strictly follow the tuple-like MDP structure as Gradient-Based algorithms do. ES does not apply any discount to the reward; instead, it evaluates an individual's performance by the total reward achieved during an entire episode with the individual's policy. This makes ES incredibly far-sighted, as the performance in the later stages of the task is just as important as the performance in the first few time steps.

2.5 Quality Diversity

Quality Diversity (QD) is a field of nature-inspired algorithms that explore a diverse set of high performing skills to solve a task [16, 17]. One of the key ideas of QD is that the path to solving a task is not linear; instead, the optimization process will benefit from open-endedness, and the best solutions might come up when not actively searching for them. The two most prominent branches of QD algorithms are Novelty Search (NS) [18, 19] and MAP-Elites [20]. QD algorithms like MAP-Elites excel at exploring the search space to find high-performing solutions. MAP-Elites, in particular, is an easy-to-implement algorithm that can be easily adapted for a variety of learning tasks. In this project, we will use the structure of the MAP-Elites grid to explore and compare the training of a population of specialist agents to the training of a single generalist agent. We explain how we perform this in detail in section 5.3. We will then use modified versions of POET [21], an open-ended curriculum learning algorithm, to answer our questions regarding open-ended and close-ended learning, which we explain in more detail in section 5.4.

Chapter 3

Related Work

3.1 Robot Locomotion

The research in Robot Locomotion aims to develop agile and robust locomotion behaviours that enable legged robots to traverse a wide variety of terrains. Previous to the advent of contemporary Deep Reinforcement Learning, the primary technique used for legged robot locomotion was Model Predictive Control (MPC) on top of a handcrafted model of the robot's dynamics. We here provide a quick introduction to Model Predictive Control and compare it to model-free Reinforcement Learning techniques.

3.1.1 Model Predictive Control

Model Predictive Control is a feedback optimization technique to control a process subject to constraints. MPC is a model-based approach, meaning that at each step, the model is queried over a specific time horizon to solve an optimization problem that finds the optimal control sequence to reach the desired behaviour. Only the first action of the sequence is executed, and at the next step, the model is queried again to plan the following best control sequence based on the new state. The model used by MPC to predict the effect of the action on the future states can be learnt or hand-designed.

MPC and its variants have shown a lot of potential in dynamic robot locomotion [22][23]. It has been disclosed [24] that Boston Dynamics' Atlas humanoid robot, possibly the most advanced legged robot in the world, is using Trajectory Optimization together with nonlinear MPC, while Spot only uses the latter. Their products, however, are patented, and no papers with the exact implementation details are published.

3.1.2 Reinforcement Learning

Model Predictive Control has been around for many decades. Thus its stability, robustness, feasibility and constraint handling theories are well developed, all things which Reinforcement Learning lacks [25]. As a result, systems based on MPC can be very reactive and robust; however, they do have some drawbacks. More specifically, the online complexity of

running MPC is usually high, and it takes a lot of engineering effort and handcrafting to design a proper model and objective function to solve a task with it.

Reinforcement Learning, on the other hand, is a more recent and arguably more immature framework that has traditionally focused more on learning speed and scaling properties [26]. RL takes a more model agnostic approach, which promises to significantly cut down on engineering costs by providing a flexible and robust system, learnt completely end-to-end. It is worth noting that MPC and model-based Reinforcement Learning can be used together to achieve excellent sample efficiency [27]. Another interesting idea is to use RL for cognitive intelligence to build on top of the athletic performance of other control algorithms. Boston Dynamics' founder Marc Raibert has hinted that they are headed in this direction [24].

There are still many challenges to be solved for Reinforcement Learning to become widely adopted for robot locomotion. Although RL sample efficiency has improved, RL agents require a significant amount of data, often limiting the training phase to simulation only. Another problem is that designing a well-shaped reward function that encourages the emergence of the desired behaviour is not always trivial. Nonetheless, the promises of Reinforcement Learning are many, and research in robotics has been gradually shifting towards using end-to-end RL for robotics. For example, Hwangbo et al. [28] showed that it is possible to train quadruped robots with RL to achieve agile and dynamic locomotion. Their work was the first to achieve such a level of athletic performance that only MPC could previously reach, and their follow-up work is arguably state-of-the-art in robot locomotion and inspired a core part of this project [29].

3.1.3 Hierarchical Learning for Locomotion

A popular research trend in robot locomotion is to use Hierarchical Reinforcement Learning (HRL). The idea is to combine an exteroceptive higher level with a proprioceptive lower level. The former guides the latter, whose only job is to handle motor coordination [30, 31, 32, 33, 34]. However, to achieve robust locomotion, the lower level should be able to walk in challenging terrains while being completely blind. Fully-blind locomotion is undoubtedly limited, but it is the foundation on which the higher level must build upon for truly autonomous and robust locomotion, and it is what we will focus on in this project. Future work might include a higher level of vision to enable the robot to fulfil high-level commands without being guided by a human operator.

3.2 Inductive Biases

Inductive biases are a means to increase the sample efficiency and performance of a learning algorithm by using prior knowledge on the task to facilitate the agent's training. Of course, this is only valid if the bias is true to the problem that needs to be solved. Crafting priors that ease the training can be a tedious job, and in general, the more we bias the solution, the less flexible it becomes. This is a well-known trade-off between performance and generalisability of the solution [35].

Reinforcement Learning can notoriously find unexpected solutions to achieve high rewards, sometimes by exploiting the task or physics engine in which the agent runs. The use of prior knowledge can put constraints on the solution, making the RL agent significantly better behaved while also speeding up the training. Due to its periodicity and ease of understanding, locomotion naturally lends itself well to incorporate biases. The PMTG architecture [4] does precisely that: by providing a prior to locomotion, it speeds up the training of the agent while also making it more performing. As we said previously, inductive biases can lead to poor flexibility and generalisation. However, that is not the case for PMTG, as the policy can still learn to output the action residuals necessary to make the robot acquire agile skills such as jumping over obstacles [36].

An alternative to adding inductive bias to a solution reliant on locomotion would be to use imitation learning on some motion-captured data of quadruped animals. Research papers like "DeepLoco", "DeepMimic", and "Learning Agile Robotic Locomotion Skills by Imitating Animals" by Peng et al. [31, 37, 38] showed truly impressive results in this direction. Perhaps in the future, we might want to experiment in removing the PMTG prior and pre-train our agent by motion imitation while later improving its robustness with more Reinforcement Learning training.

3.3 Curriculum Learning

Curriculum Learning is a Reinforcement Learning method that makes the agent's task progressively more difficult as the agent gets better. The task proposed at every episode should be challenging enough to push the skills learnt by the agent to the limit, but not too demanding so that it completely fails and does not allow the agent to improve. This curriculum enables the agent to gradually learn to solve tasks that would be too challenging to learn from scratch.

The effectiveness of curriculum learning for robot locomotion was shown by the paper "Emergence of Locomotion Behaviours in Rich Environments" by Heess et al. [39] and "Automatic Goal Generation for Reinforcement Learning Agents" by Florensea et al. [40]. In particular, Heess et al. show that training an agent in progressively more challenging but still solvable environments is key to obtaining complex skills without carrying extensive reward engineering.

An interesting curriculum learning approach is the "Paired Open-Ended Trailblazer (POET)" [21, 41]; a QD open-ended curriculum learning algorithm. In contrast to other curriculum learning approaches that seek to optimize a single generalist agent, POET aims to evolve a population of specialised agents, each paired with a different environment.

Chapter 4

Control Architecture

The barrier to entry into the realm of legged robot locomotion research is quite steep. Robot locomotion is a challenging task on which big teams of experienced researchers have worked for years. Up until relatively recently, MPC was still the only way to go for practical robotics, and very little success had ever been achieved with Reinforcement Learning, with the first method comparable to previous techniques being the one by Hwangbo et al. in 2019 [28]. In order to set up a good baseline on which to build further research, we had to look at the paper "Learning quadrupedal locomotion over challenging terrain" by Lee et al. [29], which is arguably the current state-of-the-art in legged locomotion. In their paper, Lee et al. achieved unprecedented robustness for an RL based controller, even surpassing the published state-of-the-art MPC architectures. A significant effort in this project was spent to get up to speed and produce a high-quality implementation of the core parts of their architecture. However, due to the time limit and the sheer scale difference of this project, it was not feasible or realistic to reproduce their entire setup. We decided to focus on the key components, which are the PMTG [4] prior and the historical measurements. A quick overview of our architecture can be seen in the following figure 4.1. We will now dive deep

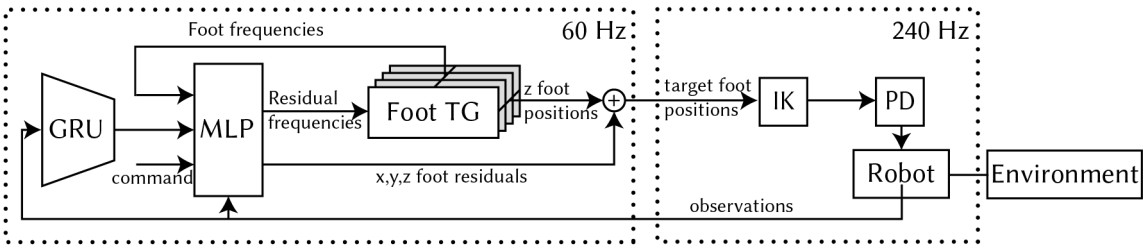

Figure 4.1: Control Architecture

into each component, outlining the progress and the differences compared to the original implementation.

4.1 PMTG

The first part of the project consisted of replicating the Policies Modulating Trajectory Generator (PMTG) low-level controller [4], upon which the rest of the control architec-

tures builds upon. We managed to contact Atil Iscen, the paper's main author, who kindly provided more details on the implementation of his work and some of his follow-up architectural changes that are not yet published.

As stated previously, Reinforcement Learning can produce solutions that exploit the flaws of the physics engine, giving way to unrealistic behaviours, the most common one being vibrating locomotion, where the robot does not lift the legs high enough and instead just vibrates very quickly to achieve the desired movement. Different solutions exist to this problem, the main ones being: using reference motion, as done by Peng et al. [38]; handcrafting a highly engineered reward function [42, 43]; or by biasing the solution with prior knowledge, as done by the PMTG architecture [4].

We chose to use the PMTG solution, as it provides a robust and flexible baseline as shown in the original paper and in other architectures that adopt it, such as the one by Lee et al. [29]. Some papers perform the learning process directly with the physical robot [44]; however, we argue that in order to achieve robust locomotion and avoid damaging expensive equipment, simulation is more compelling, and the sim2real transfer can be performed later.

The original PMTG architecture consists of two main components, a periodic prior to locomotion called Trajectory Generator (TG), and a Policy that modulates the TG parameters such as swing (how much it should swing the leg forward and backwards) and lift (how much it should lift the leg upwards) and outputs residuals to the x, y, z positions of the foot links. The Trajectory Generator outputs the foot links' x, y, z absolute coordinates based on the internal phase ϕ and parameters. The legs alternate between swing when $\phi \in [0, \pi]$, and stance when $\phi \in [\pi, 2\pi]$. The policy is a linear Neural Network that takes as input the phase of the TG $(\sin\phi, \cos\phi)$, the observations of the robot and the command, outputs the x, y, z residuals, the frequency multiplier that is added to the internal phase ϕ, and the TG parameters.

The original PMTG was developed by the Google Robotics team and was made to work on a more straightforward 8 DoF Minitaur robot. The policy used to issue commands in the form of swing/extend parameters [4]. Under the advice of Atil Iscen, who kindly informed us of some changes they made to the original architecture, we switched to commands entirely in Cartesian space, then using analytic inverse kinematics to get the target joint angles from the foot links. This change was also previously adopted by Lee et al. [29] in their version of PMTG. A PD controller is then used to output the torque necessary to track the desired joint angles. The use of the target joint angles together with local feedback by the PD controller was shown to be the best action space for legged locomotion by Peng & van de Panne [45].

Different versions of the TG exist, the main ones being the new architecture Atil Iscen informed us about, and the variant of the original architecture used by Lee et al. [29]. The main difference between the two is that Lee et al. focus purely on locomotion robustness in challenging terrains. Thus, their TG only outputs vertical stepping movement of the foot links, while the rest of the motion is guided by the policy's x, y, z residuals. On the other

hand, the version by Iscen et al. is more flexible, has more TG parameters that influenced the absolute x, y, z positions of the foot links to which the residuals are added, resulting in a stronger prior, and is more geared towards agility rather than robustness. In this project, we experimented extensively with both versions and decided to use the one by Lee et al. as it more closely aligns with the research direction we are following. A high-level scheme of PMTG can be seen in figure 4.2

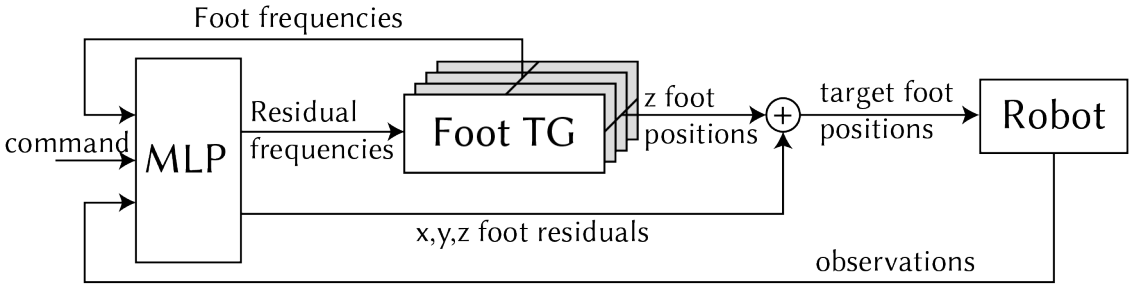

Figure 4.2: PMTG high-level overview

The observation of the robot consists of the 12 joint angles, 12 joint velocities, 5 IMU measurements (pitch, roll, pitch rate, roll rate, yaw rate) and estimated x,y,z velocities. Our architecture consists of 4 trajectory generators, one for each leg. The internal frequencies are offset so that the base gait is set to a trot gait with a base frequency of 1.25 Hz. It is possible to have all the legs coupled with a single TG, however having one TG per leg means that the internal frequency of each leg can be more flexibly adjusted, allowing for better locomotion robustness when the robot is walking in uneven terrains, or when for example a foot slips and loses contact with the ground. However, this added flexibility comes at the cost of a weaker prior to locomotion, which means that we need a slightly more engineered reward to get the same elegant walking behaviour and a bigger policy. Our policy consists of an MLP with two hidden layers of 64 neurons each and tanh activation functions. We discuss the reward function in the following section.

4.2 Reward Function

In order not to spend too much time and resources in crafting a reward function from scratch, we decided to adapt and and tune a reward inspired by the one used by Lee et al. [29]. We define our reward r_t as follows:

$$r_t = 0.05r_{lv} + 0.05r_{av} + 0.025r_s + 0.005r_{br} + 0.01r_{bp} + 0.00002r_t \tag{4.1}$$

The individual terms of the reward are:

1. Linear Velocity Reward: $r_{lv} := \begin{cases} 0.5\exp(-15(v_x - 0.25 * desired\_v_x) + \\ 0.5\exp(-15(v_y - 0.25 * desired\_v_y) & \text{for } v_{pr} < 0.25 \\ 1.0 & \text{for } v_{pr} \geq 0.25 \end{cases}$

2. Angular Velocity Reward: $r_{av} := \begin{cases} \exp(-15(\omega_{yaw} - 0.3 * desired\_\omega_{yaw}) & \text{for } \omega_{pr} < 0.3 \\ 1.0 & \text{for } \omega_{pr} \geq 0.3 \end{cases}$
3. Target Smoothness Reward: $r_s := -||(r_{f,d})_t - 2(r_{f,d})_{t-1} + (r_{f,d})_{t-2}||$
4. Base Roll Reward: $r_{br} := \exp(-2|\omega_{roll}|)$
5. Base Pitch Reward: $r_{bp} := \exp(-2|\omega_{pitch}|)$
6. Torque Reward: $r_t := -\sum_{i \in joints} |\tau_i|$

where:

•v_x, v_y are the velocities in the x and y axes
•$desired\_v_x$, $desired\_v_y$ are the desired velocities scaling factors
•ω_{yaw}, ω_{pitch}, ω_{roll} are the yaw, pitch and roll rates.
•$desired\_\omega_{yaw}$ is the desired yaw rate scaling factor
•v_{pr} is the dot product between the vector of desired and actual velocity;
•ω_{pr} is the product between desired yaw rate direction and actual yaw rate;
•r_f is the foot linear position in Cartesian coordinates;
•τ is the torque of a joint.

The above effectively makes the maximum desired velocity $0.25m/s$ and the maximum desired yaw rate $0.3rad/s$, the agent does not get rewarded more for moving or turning faster than that.

4.3 Historical Measurements

Crucial to locomotion robustness and dynamic locomotion are historical measurements, which are used to make the neural network aware of the robot state and recent history and act accordingly. Ideally, we would have liked to use a Temporal Convolutional Network (TCN) with teacher-student training, using the past 100 states of the robot to allow for better robustness, similarly to how it has been done by Lee et al. [29]. Unfortunately, due to the project's time constraint, we decided to adopt a smaller and easier-to-implement architecture, consisting of a Gated Recurrent Unit (GRU) instead of the TCN. This is also motivated by the fact that the GRU has significantly fewer parameters to train and can be trained gradient-free without the problem of having to truncate the gradient backpropagation. That being said, the TCN was proven to be slightly better performing than GRU used by Lee et al. [29]. Our GRU takes as inputs the 12 joint angles, 12 joint velocities and 5 IMU measurements. We decided not to feed the GRU with the TG frequencies as in our experiments, they caused some peculiar gait changes when obstacles perturbed the robot walking pattern. The GRU encodes the recent history in a hidden internal state of 8 dimensions, which is then concatenated to the most recent complete observation and fed into the MLP policy. Including the MLP described in section 4.1, our architecture has a total of 9272 parameters.

This approach has some limits, namely the fact that we cannot be sure of the amount of history taken into account by the GRU at any given time and the fact that without the teacher-student training, we are not adequately teaching what the GRU should learn, but only hope that it will learn by itself using RL. In our initial validating experiment, we saw only minor improvements in pushed robustness when using historical measurements. This is backed by the findings by Lee et al. [29] who have shown that both the teacher student-training is crucial to help the robot acknowledge foot-trappings, thus telling the policy to lift the leg higher up and enabling the robot to walk up the stairs. The work by Iscen et al. [36] also shows that using a mentor, a framework similar to teacher-student training, can be critical to achieving some complex skills even when a curriculum is used. In future work, where the project's time frame is not as limited, we might want to implement a complete architecture that considers these findings and perform some ablation studies to weigh their importance. However, as our contributions lie more in curriculum learning and we are not trying to surpass state-of-the-art for legged locomotion at the moment, the current architecture will suffice.

4.4 Training Algorithm

The algorithm used for the training is Augmented Random Search (ARS) [15]. We decided to start with this algorithm to replicate the initial PMTG paper more closely and ensure that the architecture was working correctly. ARS is an ES-inspired gradient-free method that requires a minimum amount of hyperparameter tuning to work; it is highly parallelisable and easy to implement. However, it does have some drawbacks compared to gradient-based methods, the most notable one being that we have no guarantees if it can properly train big architectures, as the original paper only tested it on small linear policies [15]. For this reason, in the future, we might want to experiment with other learning algorithms that might give us more freedom to train larger policies.

4.5 Validation

In order to validate the implemented PMTG architecture, we performed a series of experiments to compare it with the original implementation to be sure that it was working properly. In subsection 4.5.1 we analyse the standard PMTG architecture [4] on a few simple experiments that track forward speed. We then extensively experimented with making the robot fully command conditioned to track any command given the desired x and y velocity and the yaw rate. We discuss our approach and results in subsection 4.5.2.

4.5.1 Preliminary PMTG validation

After having implemented our variant of the PMTG architecture, we started performing some experiments to analyse better how the policy modulates the TG parameters and how the x, y, z residuals help with stability. The first experiment consisted of training the robot to follow an x velocity profile in a flat hallway. It can be seen how closely the robot manages

to follow the desired speed and how the TG parameters change to satisfy the goal (Figure 4.3). Even though in training the robot was only tasked to follow an x-velocity whose maximum magnitude was $1m/s$, we were able to test the same learnt policy on a variety of different speeds that it never learnt to track during training (Figure 4.4). We also managed to train the robot to trot successfully at $3.3m/s$ $(11.88km/h)$, the maximum velocity claimed by UniTree for the A1 robot. We believe the simulated robot could go faster than that; however, it is not in the project's scope. These validation experiments were run using only the original PMTG architecture with a linear policy instead of a two-layer MLP and did not include historical measurements like it is done for the rest of the project. The algorithm used for training is ARS [15], exactly as used in the original PMTG paper [4].

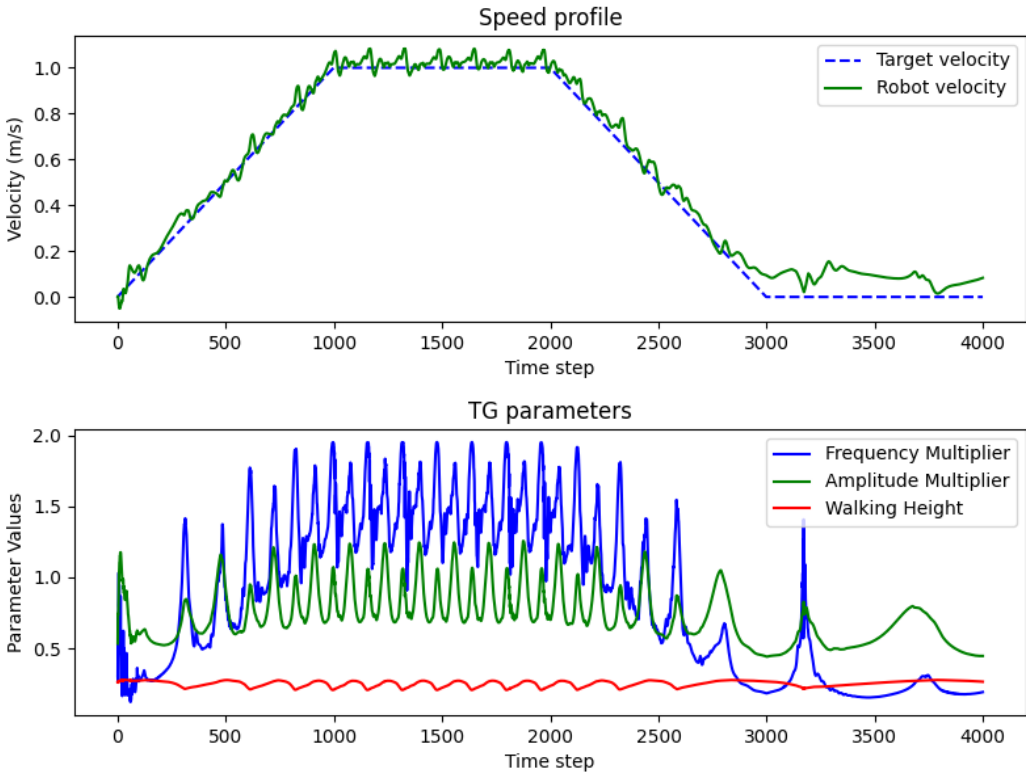

Figure 4.3: Desired speed profile used in training, and relative TG parameters

The x,y,z residuals in the original PMTG provide just minor adjustments to the robot's movement. A well-tuned TG using the original implementation, which used parameters that modified the trajectory of the legs, could walk surprisingly well on flat terrains and could even withstand and adjust to small perturbation. On the other hand, the modified version without TG parameters that we use requires the policy to do most of the work, outputting x,y,z residuals to make the robot walk, as the TG does not aid in any way forward movements of the legs. The residuals of the latter version of PMTG can be seen in the figure 4.5 and are obtained for forward motion at $0.25m/s$.

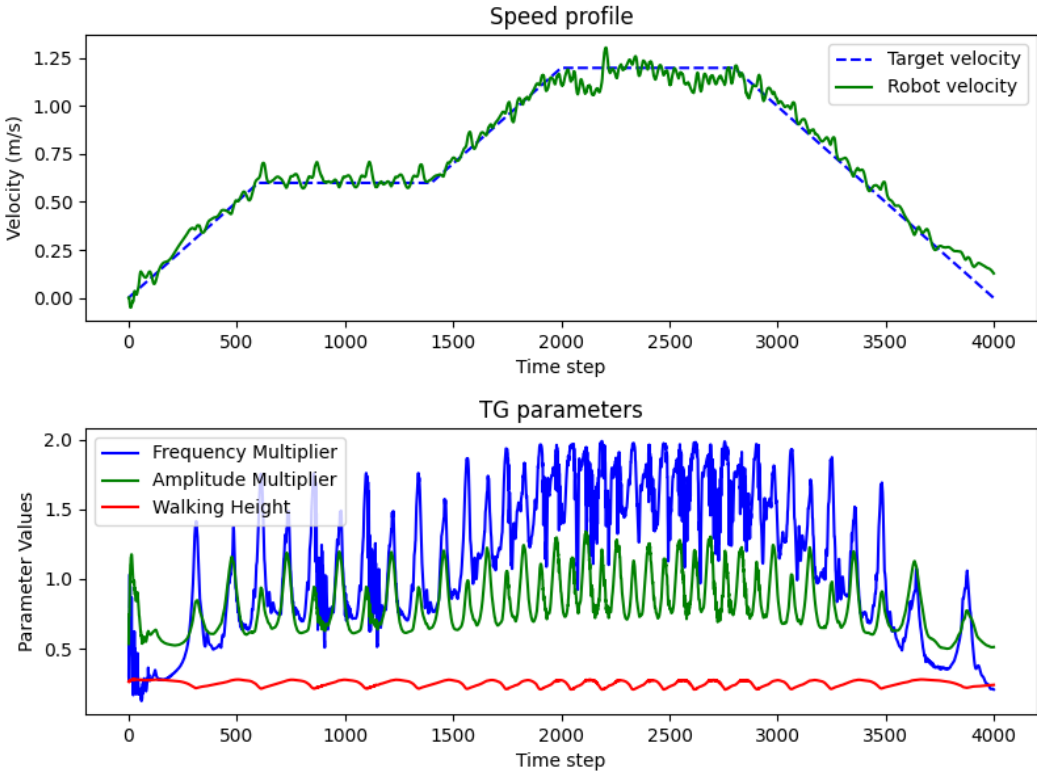

Figure 4.4: Desired speed profile used in testing, and relative TG parameters

4.5.2 Command conditioned

Throughout the project, we experimented significantly with making the policy fully command conditioned so that the robot could move in any direction. The original PMTG architecture was not fit to perform fully command conditioned, as the TG parameters such as the swing made it difficult for the robot to learn to stand still or walk laterally or backwards in our experiments. This was another contributing factor of why we switched to the PMTG version used by Lee et al. [29], where the only action of the Trajectory Generator is the lifting motion of the legs, and the policy does all the work. However, even when changing the architecture, training the robot to track x velocity, y velocity, and yaw rate was not trivial. Arguably the most significant challenge posed was a much longer training time, more sophisticated reward engineering and capability of the learning algorithm used.

We define two different 'levels' of command conditioned controller: the first one can modulate forward velocity and yaw rate; the second one can modulate forward, backwards, lateral velocity and yaw rate, all independently from one another. In order to train the 'first level', we randomly sample a different desired forward velocity and yaw rate from a uniform distribution for every training step. For the 'second level', we randomly sample a goal in the (x,y) coordinates and a random desired yaw from a uniform distribution. For every time step of the episode, we recalculate the required x and y velocities necessary to reach the goal while keeping the random desired yaw, thus giving shape to a rotating behaviour in training.

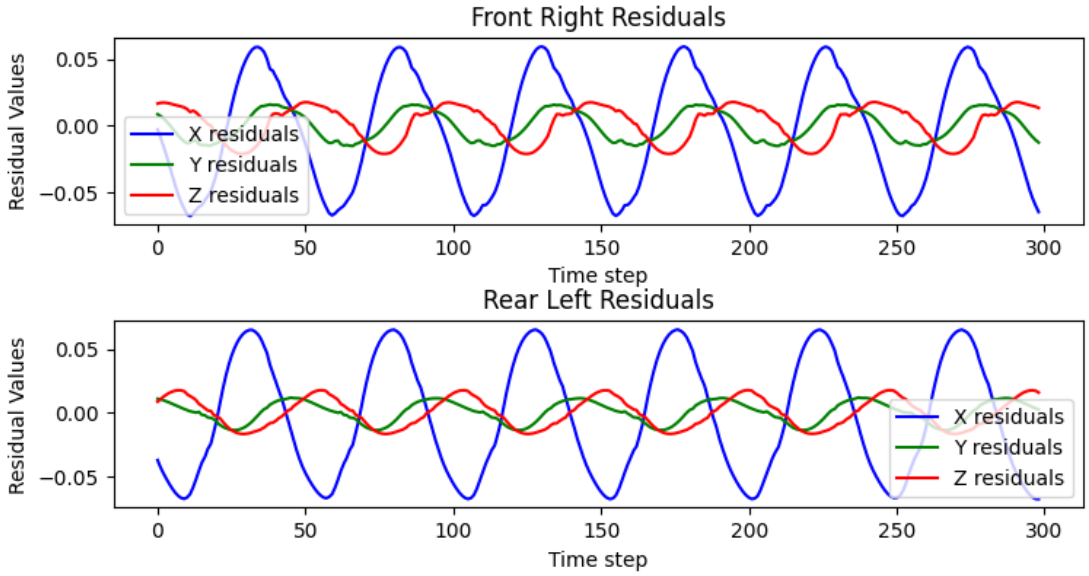

Figure 4.5: x,y,z residuals output by the MLP policy.

We achieved 'level one', meaning that we trained the robot to follow desired forward speed and yaw rate, allowing it to turn (video 1 linked in the appendix A). Although we had some degree of success with full command conditioned, as the robot tried to follow the desired commands in all directions, the walking behaviour was not as smooth as we hoped. The two most relevant research projects that achieved fully command conditioned locomotion on a quadruped robot [28, 29] both use TRPO, a policy gradient training algorithm [46]. In the future, we might want to experiment with other training algorithms such as TRPO and longer training time to achieve full command conditioned. However, for the time being, to not deal with the increased training time of our experiments, we will keep using forward movement only.

Chapter 5

Curriculum Learning

In order to achieve robust and dynamic locomotion in a variety of previously unseen terrains, it is necessary to make the agent high performing and robust over many environments with curriculum learning. In this project, we decided to compare different philosophies of curriculum learning. In particular, we will compare the training of a single generalist agent with a population of specialist agents, and we will compare close-ended and open-ended learning. We will perform the curriculum training by first pre-training the agent on flat terrain and using domain randomisation; then, we will fine-tune this policy using different techniques to compare and contrast.

5.1 Pre-Training with Domain Randomisation

We decided not to start training the entire curriculum in a single session for several reasons. The first one is the practicality of training length; in order not to significantly increase the training time of every experiment, we use a pre-trained baseline on flat terrain. Furthermore, to achieve dynamic locomotion in challenging terrain, the robot must learn how to walk on a flat plane to learn how to walk correctly and not to acquire any bizarre and inefficient gaits. The same baseline will be fine-tuned in a few different ways to perform some experiments and get the necessary data to answer the questions that we set out to answer at the beginning.

The pre-training phase consists of a simple forward running task with domain randomisation. In particular, the agent will experience random pushes $\sim \mathcal{N}$(0,30 N) for $0.016s$ every $0.016s$, and random friction coefficients $\sim \mathcal{U}$(0.4, 0.8). The random pushes make the policy more robust to sudden changes, while the random friction coefficients are used so that when performing sim2real transfer, the actual friction coefficient will have a chance of falling into the distribution of coefficients experienced in training. In the future, we might want to randomise more parameters, such as the proportional and derivative gains, motor torques, mass and damping. This kind of extensive domain randomisation has already proven to be very successful in Deep Reinforcement Learning for robotics [47, 43] and we expect that it will enable a smooth sim2real transfer of our work too. The effectiveness of this domain randomisation is discussed later and further shown in the video 1 in the appendix A.

5.2 Procedural Terrain Generation

The key to curriculum training is to have different terrains on which to train the robot; this has two goals, learning to walk in more challenging terrains by following a curriculum and acting as further domain randomisation, thus making the sim2real transfer of the policy more straightforward. We decided to propose four different types of terrain to our agent, each with its peculiarities that should encourage the agent to learn a particular skill necessary to locomote robustly in the real world. The terrains used are steps, stairs, hills and mountains (figure 5.1). The first three types of terrains are inspired by the ones used by Lee et al. [29]; however, the exact implementation details differ.

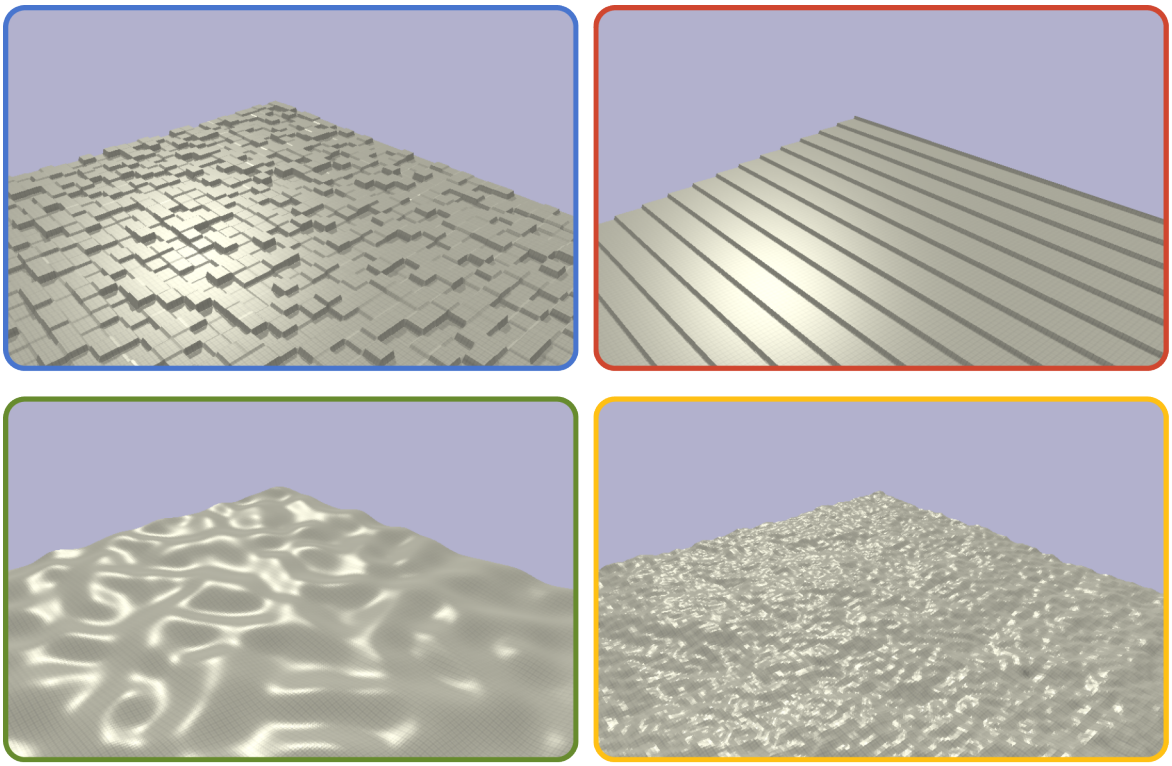

Figure 5.1: Procedurally generated terrain types. Top-left steps, top-right stairs, bottom-left hills, bottom-right mountains

Each terrain type has 2 parameters that we can change to modify its difficulty, for a total of 8 combined parameters. They are generated as follows:

- Steps: consist of squares of random *height*, sampled from a uniform distribution. Each environment can have a different, but fixed, *side length* of the squares and different maximum height.
- Stairs: consist of fixed *height* and *width* changes. Each environment can have a different, but fixed, height and width changes.
- Hills: generated by modulating *amplitude* and *frequency* of *OpenSimplex* noise.

- Mountains: generated by modulating *amplitude* and *persistence* of *fractal* noise. The *Fractal* noise used is made up of six different octaves of *Perlin* noise.

Each terrain type provides a different set of challenges:

- In the steps-environment, the robot will often have to deal with foot-trapping when the foot gets stuck in between squares of different heights, and the robot will have to acknowledge that it has to lift it more.
- In the stairs-environment, the robot will have to learn how to overcome discrete height changes. Similarly to the steps environment, it will have to lift the legs more once it detects that they are being blocked
- In the hills-environment, it will have to learn how to overcome slippages that are likely to occur when pushing down the feet on an inclined terrain to generate forward motion
- In the mountains-environment, it will have to learn how to walk in highly unstructured terrains, which resembles the kind of rocks found when walking on a dirt or gravel road.

5.3 Specialist vs Generalist

One of the main questions we asked was whether it is better to train a single generalist agent or a population of specialised agents in a robotics setting. The question is not as straightforward as it seems, and neither is the answer. Regardless of which one is easier to train and leads to better results, what we usually want to have in the end is a generally capable agent who can perform various tasks. This can be achieved by simply deploying the trained generalist agent or picking a specialist agent trained for the task we are facing. Algorithms like the "Intelligent Trial and Error" [48] and "Reset-Free Trial and Error" [49] do allow to switch and adopt different policies when needed by the robot. However, although fast, the switch is never seamless, and abrupt switches in challenging terrains may cause loss of balance that could easily damage a quadruped robot. Supposed that we could perform these switches seamlessly and we had a comprehensive enough population of specialist agents, if these specialist agents were more performing in their respective tasks than the generalist one, then it would make sense to use a population of trained agents over a single one trained to be a generalist.

If the specialist agents were actually significantly better than the generalist agent in their task, but we still wanted to have a single agent to deploy in order to avoid going through the trouble of switching policies, distillation of the specialist policies into a generalist one would be a viable approach. Several methods exist for this, and it might be possible to have a distilled policy that performs better than the agent trained as a generalist. However, for the moment, we refrain from answering these questions as they are not in the scope of the project.

In order to come up with a more objective answer to the specialist vs generalist agent dilemma, we will use cumulative reward in a curriculum training setting to evaluate the differences between specialist and generalist agents. In order to test this fairly, we design a close-ended curriculum learning algorithm inspired by Multi-Task MAP-Elite [50]. Each cell of the grid corresponds to a different environment, parameterised as detailed in section 5.2. We organised the four types of environments in a grid at increasing difficulties; a visualisation is shown in figure 5.2.

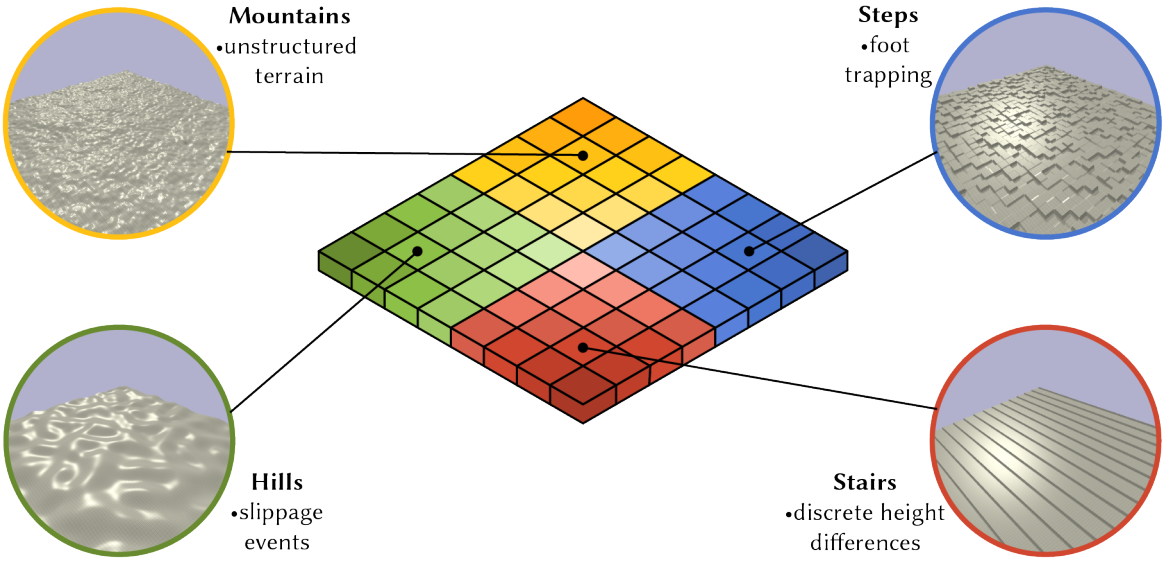

Figure 5.2: Curriculum on a grid visualised. The darker the shade the more challenging the environment.

The algorithm works as follows: we start with the pre-trained agent on a flat terrain described in section 5.1, we then evaluate the pre-trained policy on all the environments of the curriculum, collect the rewards, sum them and set the sum as a starting baseline. We then proceed to train in two different ways; the first one will train a single generalist agent in all the environments, the second one will evolve a population of specialised agents, where there will be an agent in each cell. For the specialist training at each iteration, we will choose a starting policy either randomly or by using a particle filter based on the rewards achieved, while for the generalist, it will always be the same one. We then select a destination environment randomly or using the particle filter and train the selected policy in that terrain for several training steps. If we get a reward better than the previously saved one, we update the reward, and in the case of the specialist agents, we also save the new policy in the corresponding cell of the grid. Algorithm 1 describes the training for the specialist agents in detail, while Algorithm 2 describes the training of the generalist agent.

At this point, another important consideration needs to be made. If we were to use Algorithm 1 and 2 to train a population of N individuals for M training iterations, the computational effort put into each specialist agent would be M/N, compared to a computational effort of M for the generalist agent. If N is large, it becomes clear that the generalist agent will be

Algorithm 1 Curriculum Learning of Specialist Agents

▸ Initialise
envs - a curriculum of diverse terrains at increasing difficulties
A - a pre-trained agent
EAR_list - list of lists that will contain environment, agent and reward

▸ Evaluation
for E in envs **do**
    reward ← *evaluate(A, E)*
    Add (copy(A), E, reward) to EAR_list

▸ Training loop
for *step = 1,...,N* **do**
    A ← *sample starting agent*
    E, old_reward ← *sample destination env and correspoding reward*
    for *exploration = 1,...,M* **do**
        train *A on E for 1 iteration (any algorithm could be used)*
        new_reward ← *evaluate(A, E)*
        if new_*reward* > *old_reward* **then**
            update (E, A, reward) paired with E in EAR_list

Algorithm 2 Curriculum Learning of a Generalist Agent

▸ Initialise
envs - a curriculum of diverse terrains at increasing difficulties
A - a pre-trained agent
ER_list - list of lists that will contain environment and reward

▸ Evaluation
for E in envs **do**
    reward ← *evaluate(A, E)*
    Add (E, reward) to ER_list

▸ Training loop
for *step = 1,...,N* **do**
    E, old_reward ← *sample destination env and corresponding reward*
    for *exploration = 1,...,M* **do**
        train *A on E for 1 iteration (any algorithm could be used)*
        new_reward ← *evaluate(A, E)*
        if new_*reward* > *old_reward* **then**
            update (E, reward) paired with E in ER_list

trained significantly more than each specialist if few successful transfers are made.

This motivates whether it makes sense to train specialist agents on tasks that require similar athletic skills to be solved. Training a population of agents to the same extent as a single agent requires significantly more training time, and how much learning different athletic skills in legged robotics settings can be considered a different task is still to be answered. For example, even though the movements and skills required for walking up a flight of stairs are different from walking on flat ground, the domain is still legged locomotion, and there is a severe overlap between the two types of movements needed. Therefore, we expect the specialists to outperform the generalist only if the various skills needed to solve each terrain differ enough to make the generalist forget previously learnt skills. On the other hand, we expect the generalist to be better if that is not the case, and all the skills needed to achieve robust dynamic locomotion in various challenging terrains are similar enough to be quickly learnt by the same policy. We will experiment and answer these questions in section 6.2.

5.4 On Open-Ended Learning

Open-Ended learning is a field of Machine Learning that deals with learning intelligent systems of unbounded complexity with minimal to no human supervision. The three pillars of true open-ended learning can be considered meta-learning architectures, meta-learning algorithms, and the generation of effective learning environments [51]. This project deals only with the latter: we create new procedurally generated terrains for the agent to solve, while the task will always be forward locomotion and the learning algorithm and architecture are hand-designed.

DeepMind's Open Ended Learning Team showed that Open-Ended learning might hold the keys to produce generally capable agents [52]. However, there are many questions still to be answered on the use of open-ended learning. Is open-ended learning better than a carefully crafted curriculum learning approach in a robotics setting? In a legged robotics setting, what matters is the athletic performance, and there is not a need for the general intelligence of an agent. The trend in Machine Learning of going from hand-designed pipelines to end-to-end learning the entire architecture is transferring to Reinforcement Learning too; however, that begs the question of whether this is truly necessary for every aspect of Reinforcement Learning. Open-ended learning algorithms like the minimal criterion [53], and POET [21] are excellent examples of what open-endedness in generating terrains together with the coevolution of the agents can achieve. When the task itself is open-ended, and there is no particular skill that we want the agent to learn, open-ended training is the way to go. Rather than training with a target in mind, POET surprises us by evolving some skills that we did not specifically ask for or expect. POET was initially tested on an unrealistic 2D mini-game that might encourage or need this diversity of behaviours and lend itself well to open-ended learning [21]. However, how does this translate to a more realistic athletic performance tasks? Is open-ended learning the best way to teach our robot to walk up the stairs with a reasonable computational effort? Is using open-ended learning for everything potentially better?

It might be the case that we humans are not smart enough to create true general intelligence, and we should instead let AI figure out the way to general intelligence on its own [54]. This argument might also be true regarding crafting a curriculum for an agent to learn new skills. Are we good enough to select a diverse range of environments that will enhance the learning progress of the agent, or will we fall short on the task and be outperformed by AI? Moreover, if that was the case, how would the open-ended curriculum compare to ours?

To answer these questions, we will use a revised version of the Paired Open-Ended Trailblazer (POET) [21] and a modified close-ended version of the same algorithm in a robotics setting. Both will be used to train a population of agents to develop dynamic locomotion over challenging terrains. The open-endedness of POET sparks from the coevolution of agents and terrains via the Minimal Criterion [53], meaning that as the population of agents gets better, new and more challenging terrains are proposed. We modify this algorithm to make it close-ended by simply cutting the coevolution part and introducing a fixed, hand-designed, curriculum of terrains, similar to the one described in section 5.2 and figures

5.1-5.2. High-level overviews of how they work can be found in Algorithm 3 and Algorithm 4; for more details, consult the original paper [21]. We call Algorithm 3 open-ended POET and Algorithm 4 close-ended POET to make it clear which one is what; however, it is crucial to understand that the standard POET algorithm in itself is already open-ended [21]. In order to make the two algorithms comparable for later experiments, we initialise both populations as a set of 32 agents paired with 32 environments. The agents are all a copy of the pre-trained one explained in section 5.1. The 32 environments are all flat terrains for the open-ended version, while for the close-ended version, they are all predefined hand-crafted encodings.

Algorithm 3 Open-Ended POET

▸ **Input:**
A - a pre-trained agent
E - initial flat environment

▸ **Initialize:**
Set *EA_list* empty
Add *M* copies of *(E, A,)* to *EA_list*

▸ **Training loop:**
for *step = 1,...,N* **do**

▸ **Add a new environment**
child_envs ⟵ mutate_envs(EA_list)
child_envs ⟵ satisfy_Minimal_Criterion(child_envs, all_agents)
E ⟵ select_most_novel(child_envs)
A ⟵ select_best_agent(all_agents, E)
add (E, A) to EA_list
if *len(EA_list) > capacity* **then**
remove_oldest(EA_list)

▸ **Optimise EA pairs**
for *E, A in EA_list* **do**
train A on E

▸ **Attempt transfers**
for *E in EA_list* **do**
A ⟵ select_best_agent(all_agents, E)
update (E, A) in EA_list

Algorithm 4 Close-Ended POET

▸ **Input:**
A - a pre-trained agent
envs - a curriculum of diverse terrains at increasing difficulties

▸ **Initialize:**
Set *EA_list* empty
for *E in envs* **do**
    add (E, A) to EA_list

▸ **Training loop:**
for *step = 1,...,N* **do**

    ▸ **Optimise EA pairs**
    for *E, A in EA_list* **do**
        train A on E

    ▸ **Attempt transfers**
    for *E in EA_list* **do**
        A ← select_best_agent(all_agents, E)
        update (E, A) in EA_list

Chapter 6

Experiments

We here present the experiments run to evaluate and compare the effectiveness of the curriculum learning techniques discussed in chapter 5.

6.1 Simulation and Quadruped Robot

The physics simulator of choice used for the experiments is PyBullet [55], as it is well documented, accessible and relatively easy to pick up by following the examples available on the official GitHub repository. The quadruped robot that we use is the A1 robot by UniTree [56]. The A1 is a 12 DoF medium-sized quadruped robot that has gained much popularity in the RL community, as more and more teams are picking it up due to its good stability and competitive price. The hope is to be able to perform sim2real transfer later on with the real A1 robot.

Figure 6.1: On the left, the real A1 robot. On the right, the A1 robot simulated in PyBullet

6.2 Specialist vs Generalist

To answer the questions we posed ourselves in section 5.3, we train Algorithm 1 and 2 for 1000 iterations. The training algorithm used in the exploration phase is ARS [15] for 20 iterations at every optimisation step, exploring 32 directions every time, of which we keep

and update the policy towards the 16 most performing ones. The number of environments used is 32, divided into four groups, one for each terrain type, where each group is made up of 8 terrains at varying difficulties. The easiest and hardest difficulty setting of every terrain of the curriculum can be seen in figure 6.2

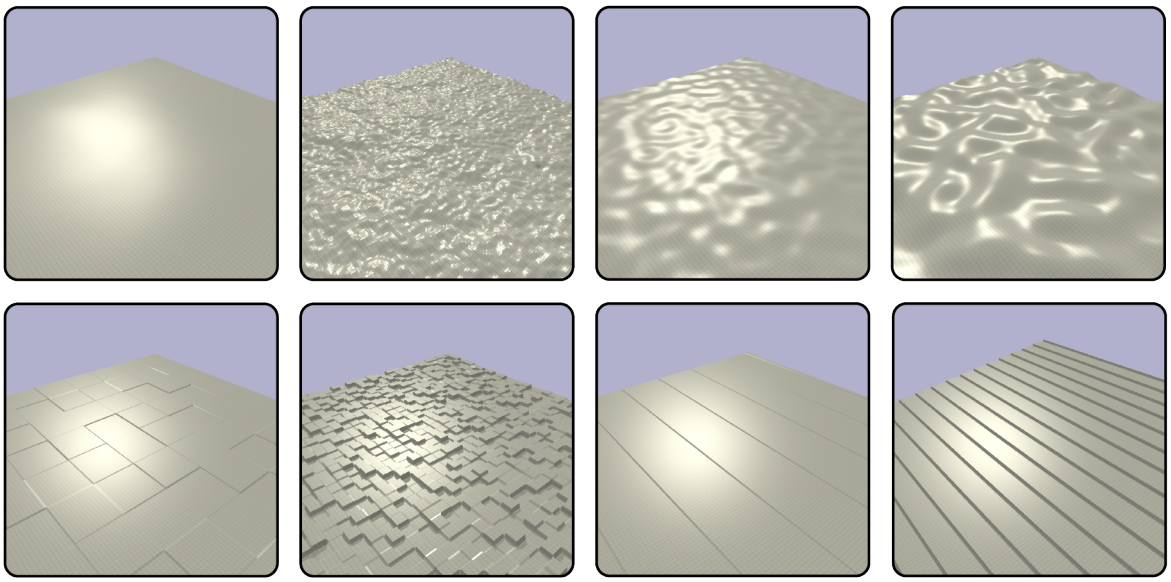

Figure 6.2: The figure shows the easiest and hardest parameterised environment of each type of terrains.

In order to evaluate the performance of specialists and generalists, every 100 iterations of the outer loop, we evaluate all the specialist policies in their respective terrains and the generalist agent on all terrains, sum up the rewards and compare them. Figure 6.3 shows the result.

It can be seen that the generalist agent performs better than the population with the same total computational budget offered. This indicates that the different locomotion skills needed for different athletic tasks are highly overlapping and it does not make sense to retrain basic robust locomotion for every agent. It might be possible that over a long training period the performance of the population will surpass the generalist agent, however according to the results this doesn't seem likely to happen unless given a significant amount of more training time. It is now apparent that training a generalist agent is definitely the better solution for dynamic robot locomotion. This is because the skills needed for robust locomotion in challenging terrains are highly overlapping, and if the computational budget is limited it does not make sense to train a policy for each one.

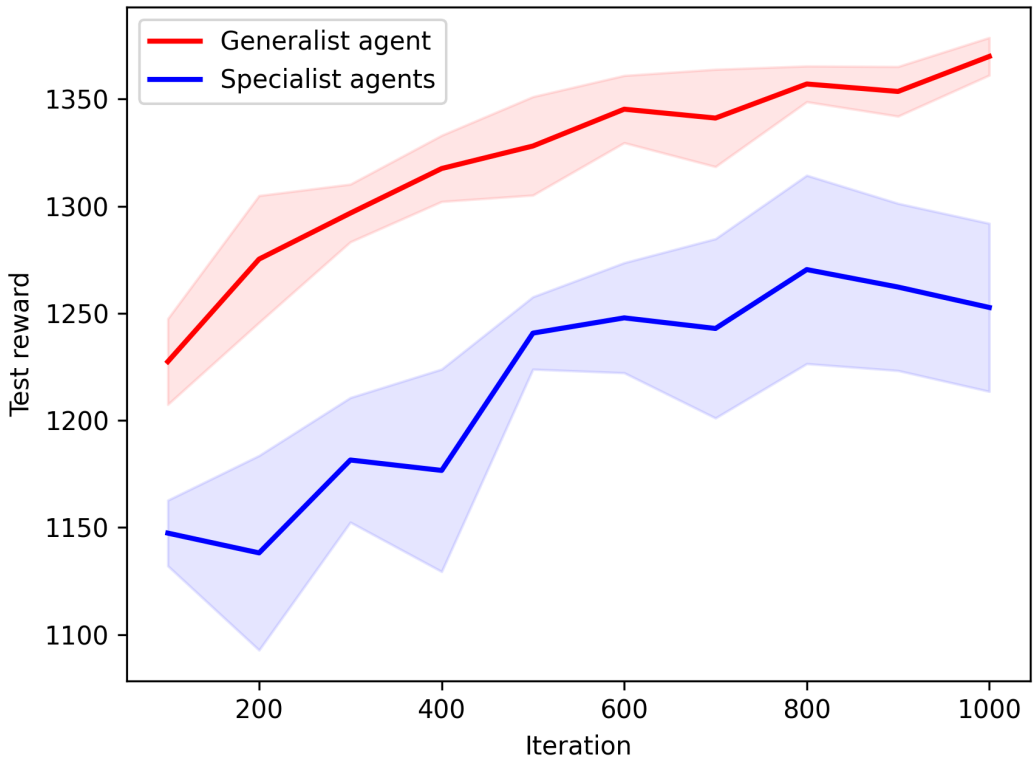

Figure 6.3: Results of the generalist and specialists learning algorithms. Statistics are collected over five runs using random seeds. The lines indicate the mean and the shaded areas represent the standard deviation

6.3 Open-Ended vs Close-Ended

In our quest to answer the questions regarding open-ended learning that we posed ourselves in section 5.4, we use Algorithm 3 and Algorithm 4 to train a population of agents in an open-ended and a close-ended learning fashion. As a quick summary, we want to find out how open-ended and close-ended learning compare in a robotics setting, where there is no need for a generally capable intelligent agent, but only athletic performance is desired. Another important consideration is that we have significant task knowledge of the athletic performance we want the agent to acquire. Therefore, hand-crafting an appropriate curriculum of environments should not be too challenging for us and should be able to rival the open-ended learning curriculum easily. Or so we expected.

We train both open-ended and close-ended algorithms starting from the same pre-trained agent described in 5.1 for 100 iterations. The two algorithms differ only in the coevolution of the paired environments, which is the open-ended part of POET [21]. At every iteration, to optimise the paired environment-agent, we use ten steps of ARS [15] on 32 random directions, of which we optimise the policy towards the most performing 16. In order to test how the two algorithms compare, every ten iterations of the outer loop, we evaluate the current population on a held-out test set comprising eight environments. For fairness of the evaluation, four of the environments are similar to the ones hand-crafted by us, while the other four are similar to those automatically produced by the algorithm. During the testing, every one of the 32 agents is evaluated on all 8 test environments. The test environments have been hand-designed to pose a good evaluation benchmark to test the algorithms on. They can be seen in the figure 6.4.

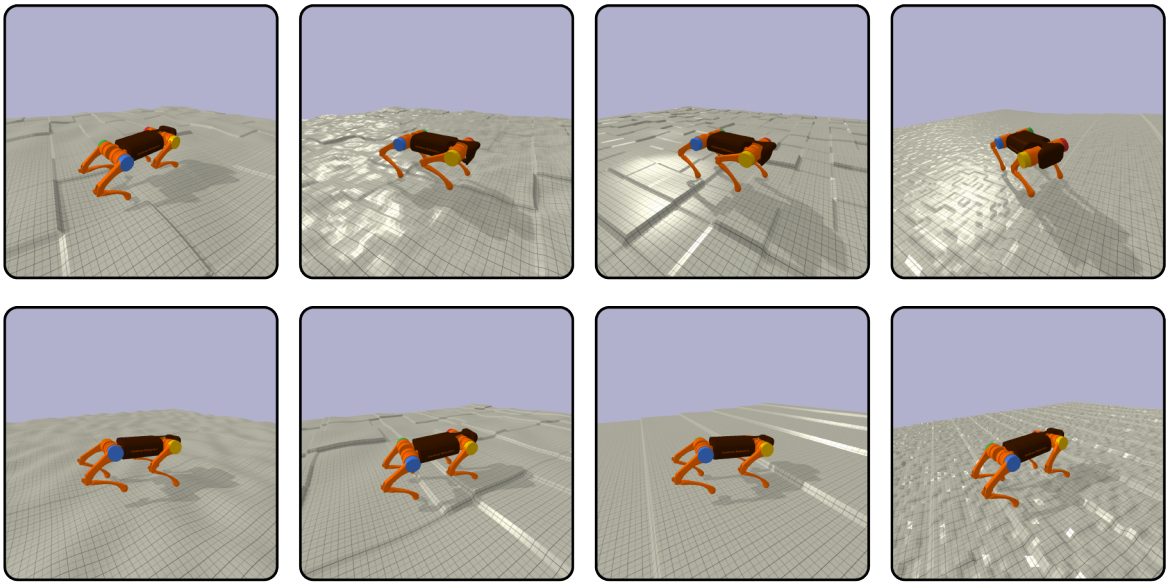

Figure 6.4: Test set of predefined terrains of different types.

A flaw of this kind of evaluation is that some of these terrains might be so challenging that

the robot falls within a few seconds of deployment, while others might be not challenging enough. This, however, is equally taxing on both algorithms, so we decided not to change the test set further so as not to go into excessive hand-tuning. This downside is a recurring theme for the close-ended curriculum, which is built similarly to the test environments. As a result, the agent will often surprise us and perform significantly better than expected on what appears to us as challenging terrains, but it will often disappoint our expectations and perform poorly on what we deem easy terrains. This is particularly true for the mountains and hills parameterised terrains, as their highly unstructured and inclined surfaces proved to be significantly more difficult than expected for the robot. The results on the test set performed during training can be seen in figure 6.5.

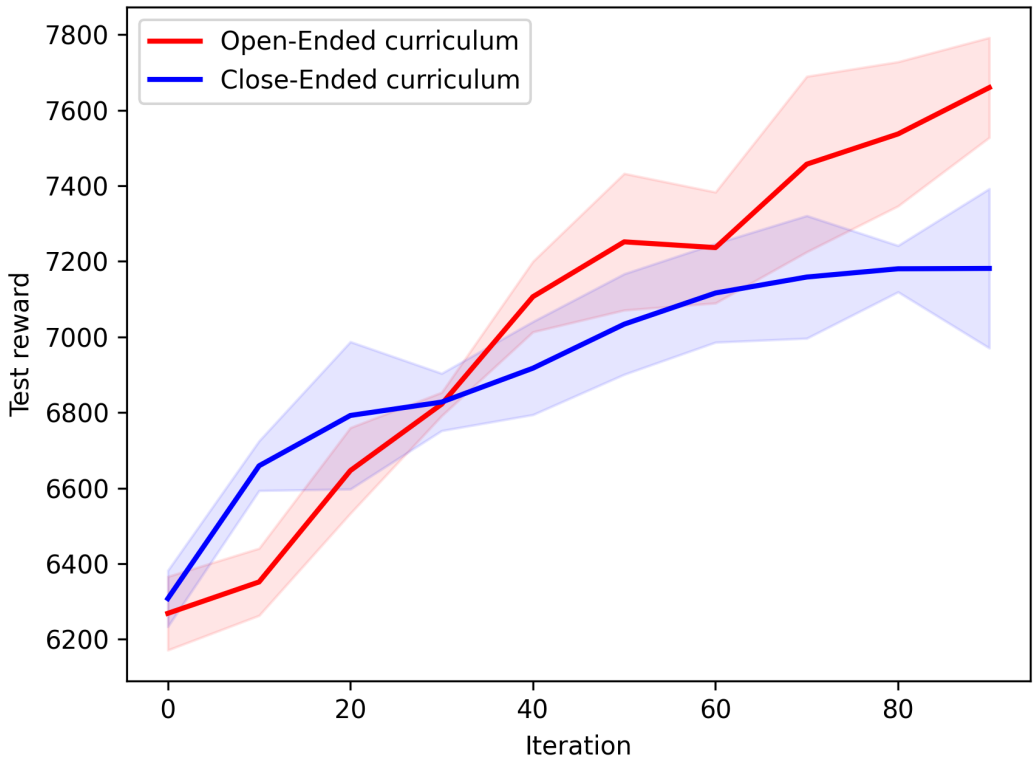

Figure 6.5: Results of the open-ended and close-ended learning algorithms. Statistics are collected over five runs using random seeds. The lines indicate the mean and the shaded areas represent the standard deviation.

The performance of the open-ended curriculum quickly surpasses the close-ended one. Within the length of the experiment, we did not observe the open-ended curriculum plateau yet, which we expect to happen once it reaches the limit imposed by the control architecture and the physical robot. A similar seemingly-unbounded learning curve was also found in the Enhanced POET paper [41]. The initial difference in performance is likely due to how we set up the algorithms to make them more comparable. This is because we

initialised the open-ended curriculum with a population of 32 agents, all paired with an initial flat environment. One by one, the flat environments are substituted by new novel ones that are sufficiently challenging to stimulate the improvement of the agent. Effectively, the two algorithms are comparable starting from iteration 32, when all the environments of the open-ended curriculum are self-proposed and not predetermined. This happens approximately when we also see the open-ended curriculum test performance surpassing the close-ended one. The potential for improvement on the initial 32 flat environments is limited, hence why we see the close-ended curriculum having the edge initially.

Some interesting considerations can be made when comparing the curriculum of environments that we hand-designed for the close-ended algorithm to those developed by the open-ended learning one. The close-ended curriculum, which is the same ones used in section 6.2, and showed in figure 6.2, is made up of 32 environments that span a range of difficulties among the four terrain types. To not complicate the crafting of our close-handed curriculum, we decided not to mix different types of terrains, as we knew the challenges offered by each terrain type, which required the emergence of some specific athletic skills. The self-proposed terrain by the open-ended algorithm, on the other hand, are generated by mutating an encoding vector comprising of all the eight parameters, which we discussed in section 5.2, resulting in environments that mix all of the parameterised terrains. The self-proposed terrains were chosen based on the Minimal Criterion [53] and novelty of the encoding vector, prioritising the most novel environments first. This search for novelty leads to an incredibly diverse set of environments, which we can see in figure 6.6, that dramatically improves the learning performance of the agents. On the other hand, it is possible to have different difficulties with a close-ended curriculum, but it is not easy to have a good diversity of terrains at different difficulties, making progress slower. An other interesting observation we made was that all the policies of the population had very similar behaviour, almost like if the population was slowly converging to a single high-performing.

With our close-ended hand-crafted curriculum, we accidentally selected environments that we expected the robot to learn how to solve, yet it never did. This is one of the limits of inducing biases in our solutions and proves that maybe we humans are not good enough to design and train complex, intelligent robots properly, and what might work instead is a kind of AI-Generating Algorithms [51]. The open-ended curriculum with a well-tuned minimal-criterion will always propose solvable yet challenging terrains. This, however, is also a drawback we have found with open-ended learning. For the algorithm's performance to shine, it is necessary to appropriately tune the Minimal-Criterion lower and upper bound rewards. For the sake of time and simplicity, in our experiments, we spent a minimum amount of time tuning the minimal criterion, and other POET-related hyper-parameters [21], which means that if well-tuned, it could quickly achieve even better performance. The lower we set the MC lower bound reward, the more challenging the proposed terrains; however, when they are too challenging, the agent cannot learn anything meaningful. On the other hand, if the MC lower bound is set too high, only close-to-flat terrains will be proposed, and the agents will not have a rich enough curriculum on which to train, significantly limiting its performance. This can happen both if the MC or the terrain generating functions are not correctly tuned. Some examples of self-proposed terrains generated when the algorithm was not adequately tuned can be seen in figure 6.7.

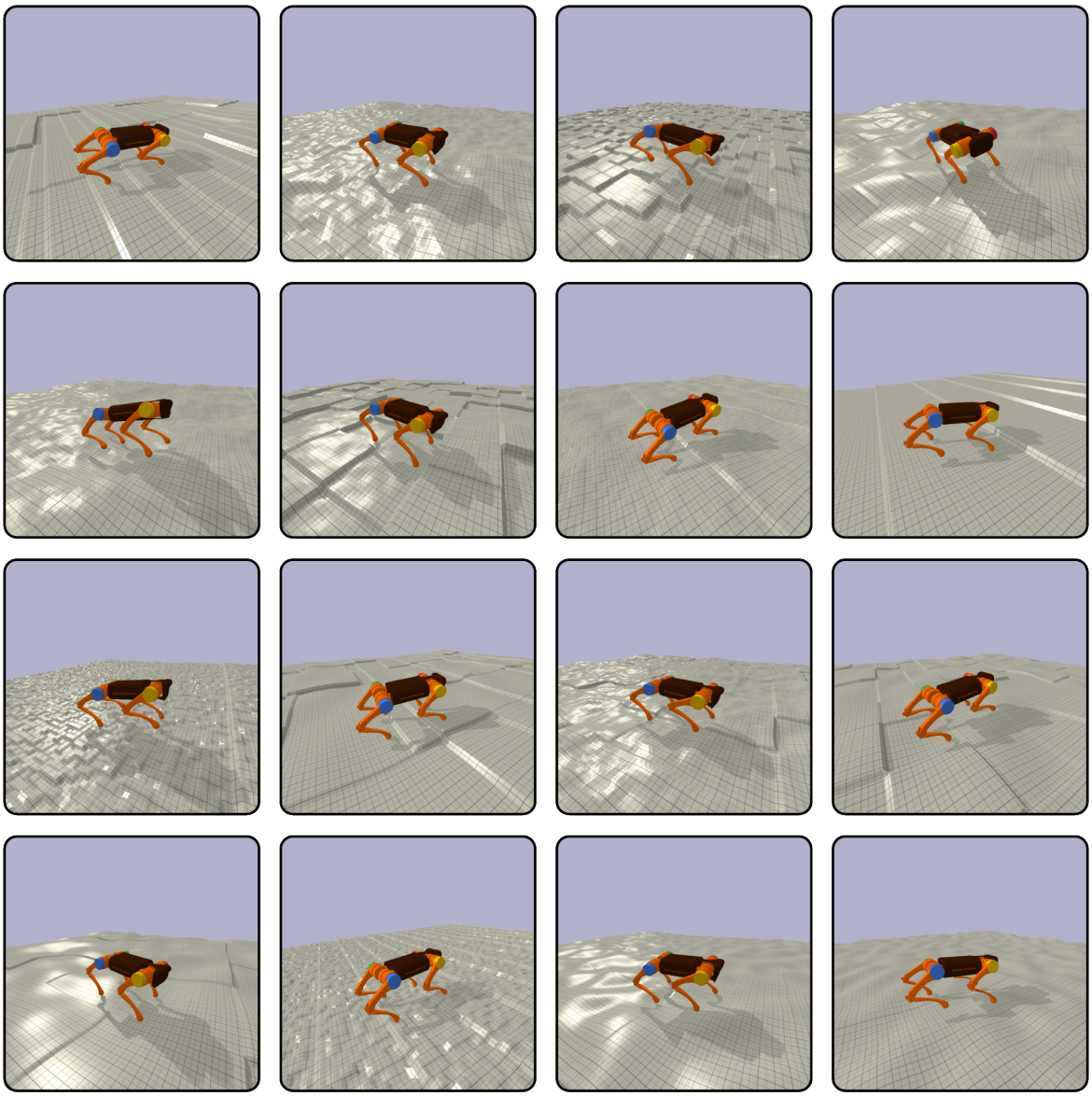

Figure 6.6: Examples of self-proposed terrains by the open-ended algorithm.

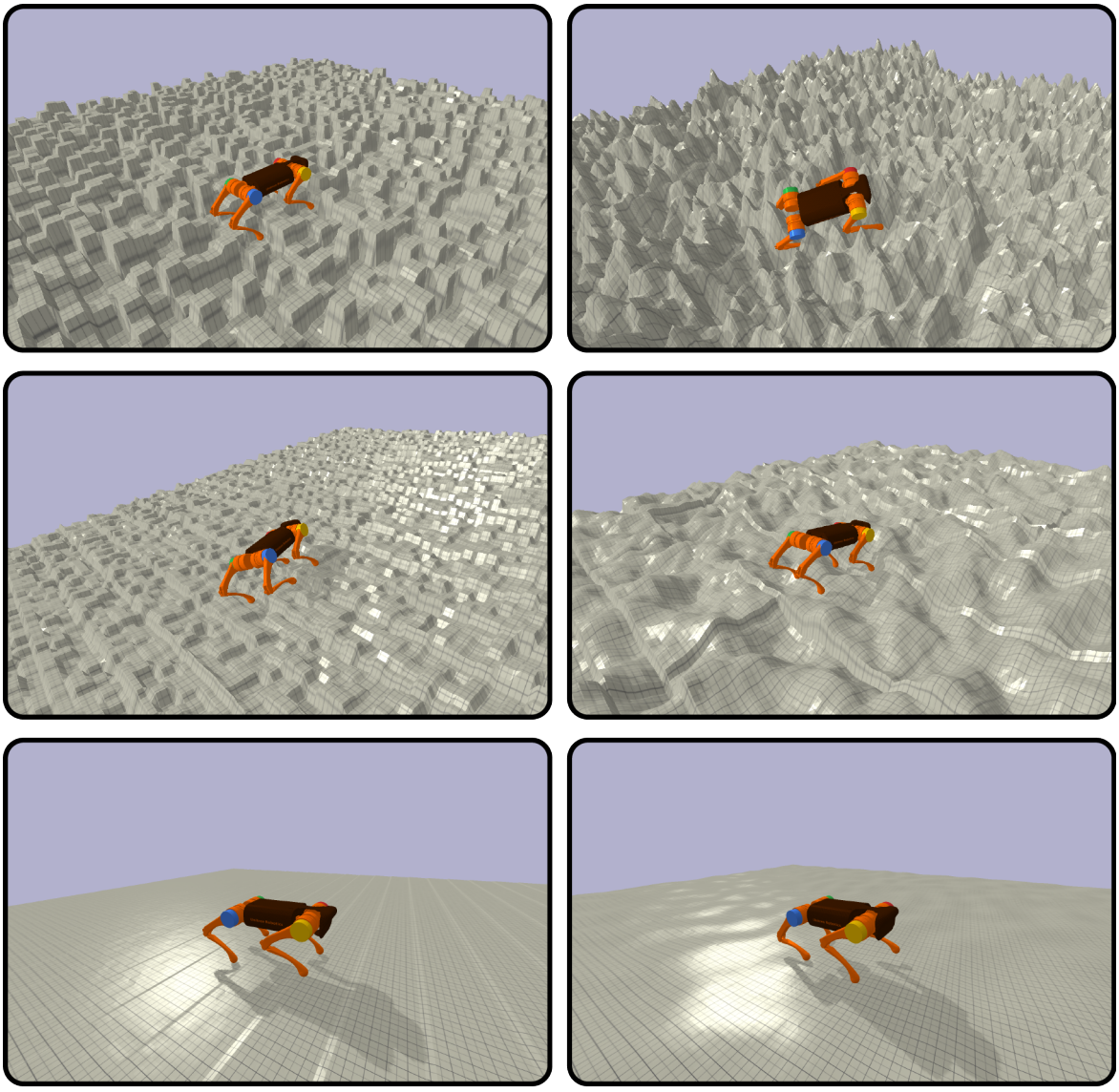

Figure 6.7: Examples of self-proposed terrains when the minimal criterion lower bound is not properly tuned. They vary from being too challenging when the lower-bound reward is too low (top four) to being too easy when the lower-bound reward is too high (bottom two).

Open-Ended Generalist

In light of the results of the experiments in section 6.2, we asked ourselves whether the population of POET, one of the main features of the algorithm, was really necessary for a robotics task. As we mentioned previously, POET was tested on a 2D game, which means that its findings might not translate to robots' real-world athletic performance. Our results so far point out that it is better to train a generalist agent instead of dividing the computational effort among many specialist agents in the hope that robust legged locomotion requires different skills that a generalist alone could not learn alone. Lee et al. [29] also advocated for a generalist agent instead of a population of specialists, without, however, showing any experimental results in support of their statement. We experimented by eliminating the population and training with open-ended curriculum learning always the same generalist agent. The results can be seen in figure 6.8. The learning curve of the open-ended generalist dwarfs the performance of the other two algorithms on the same test set. It can be seen that it also reaches a plateau, likely to correspond either to the upper constraint of the control architecture, or the maximum reward achievable in test environments. However, it does not necessarily imply that the agent could not improve its performance in more challenging terrains. The environments automatically generated by the open-ended curriculum of the

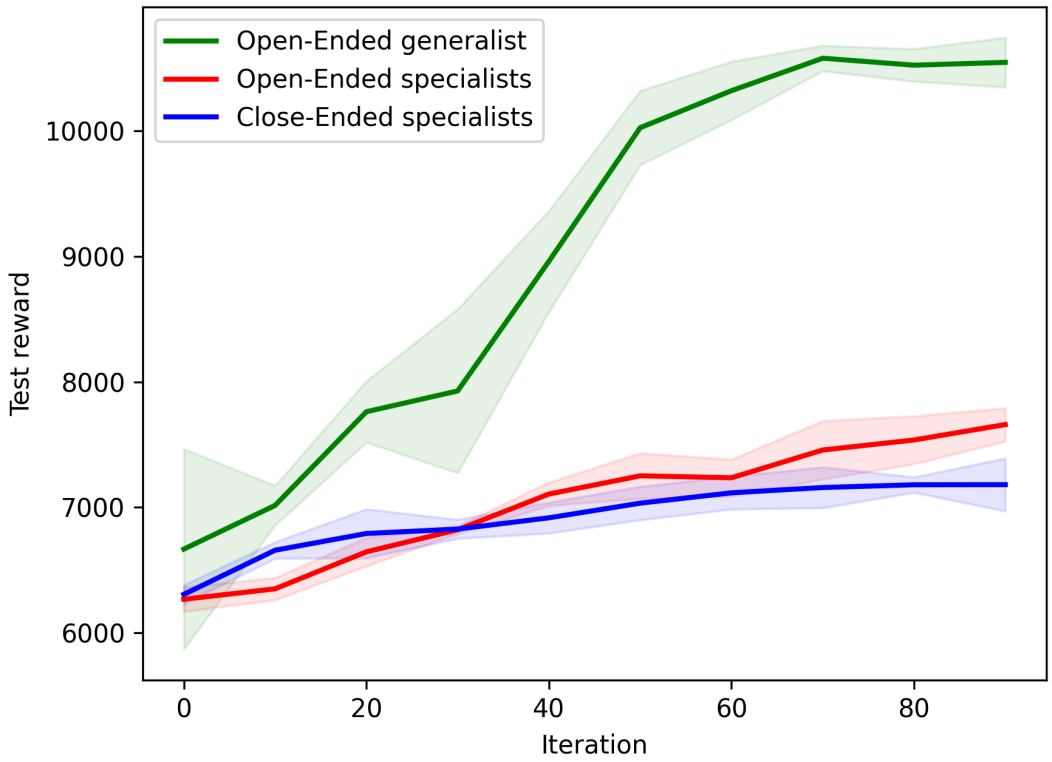

Figure 6.8: Open-Ended generalist agent results compared to the same algorithm trained with a population with open-ended and close-ended curriculum learning. Statistics are collected over five runs using random seeds. The lines indicate the mean and the shaded areas represent the standard deviation.

generalist agent are also much more challenging than the ones generated by the specialists, as the performance of the generalist one was superior, and it could solve difficult proposed terrains. The self-proposed environments of the generalist agent can be seen in figure 6.9. Video 2 in the appendix A shows the behaviour of the agent across several of these challenging terrains.

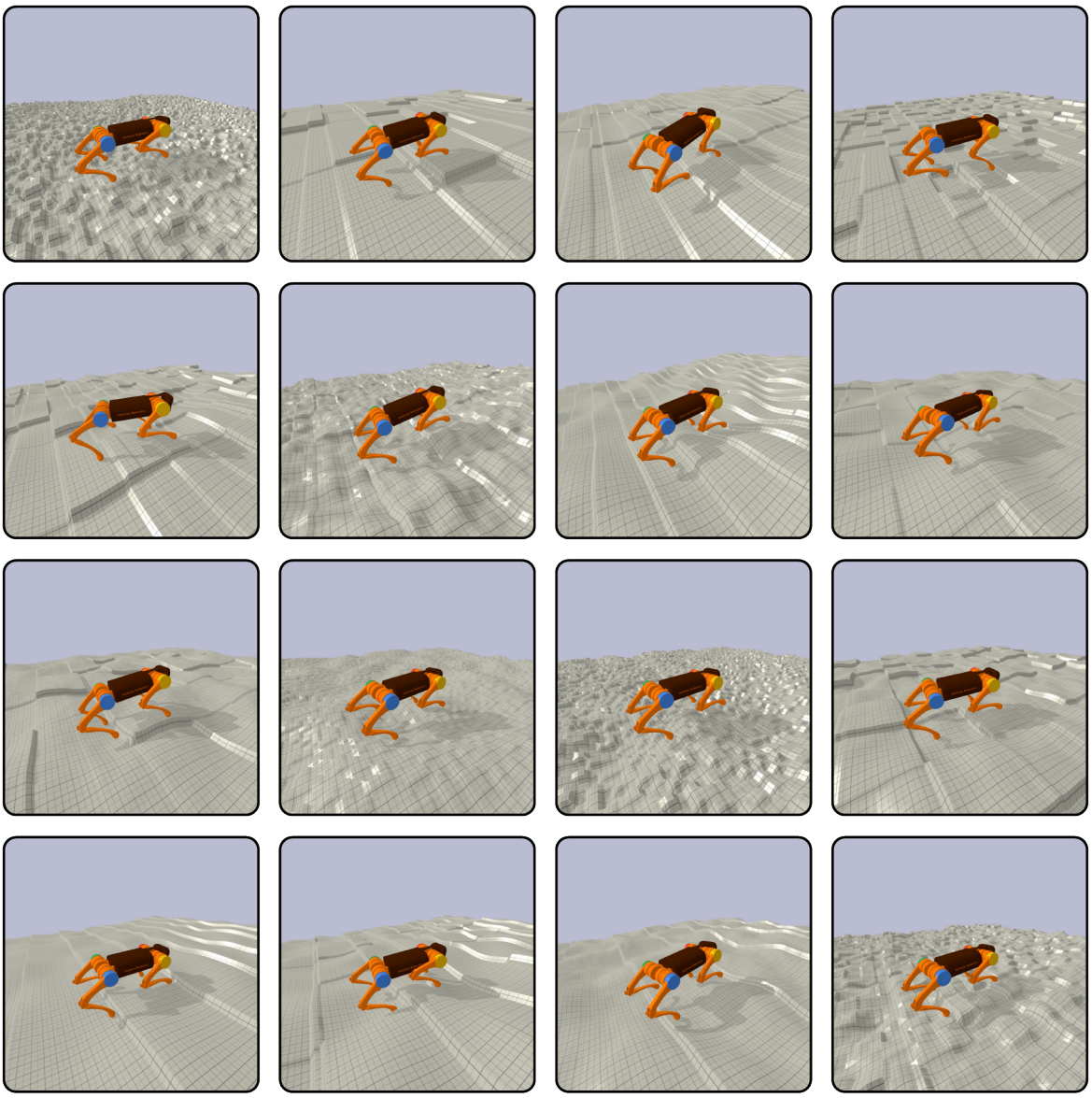

Figure 6.9: Examples of self-proposed terrains of the generalist agent trained with open-ended curriculum learning.

We detail how our algorithm works in Algorithm 5. Our method differs from POET [21] as it does not evolve a population of agents, each one paired with a single environment, but rather trains a single generalist agent. However, similarly to POET, we use the Minimal Criterion [53] and the concept of novelty in the encoding vector to select the most novel terrain at each iteration, which helps to improve the agent's robustness to a variety of

terrains. Our work also differs from the one by Lee et al. [29] as they use a complex particle filter to select their terrains, but they do not mix the terrain types, and thus, they also do not encourage the emergence of novel terrains.

Algorithm 5 - Open-Ended Curriculum Learning of a generalist agent

▸ **Input:**
A - a pre-trained agent
E - initial flat environment

▸ **Initialize:**
Set *E_list* empty
Add *M* copies of *(E)* to *E_list*

▸ **Training loop:**
for *step = 1,...,N* **do**

▸ **Add a new environment**
child_envs ⟵ mutate_envs(E_list)
child_envs ⟵ satisfy_Minimal_Criterion(child_envs, A)
E ⟵ select_most_novel(child_envs)
add (E) to E_list
if *len(E_list) > capacity* **then**
remove_oldest(E_list)

▸ **Optimise generalist A**
for *E in E_list* **do**
train A on E

6.4 Dynamic Locomotion Evaluation

Even though we did not start out with the aim to achieve state-of-the-art in robot locomotion, our open-ended generalist policy still shows some impressive results in a variety of challenging terrains, learning all the locomotion skills we wanted it to learn. We compare it with the pre-trained policy in figures 6.10-6.11-6.12-6.13 and show that the fine-tuned generalist policy trained with open-ended curriculum learning was able to walk in all the environments, displaying excellent dynamic locomotion, which can be seen in video 1 and 2 in the appendix A.

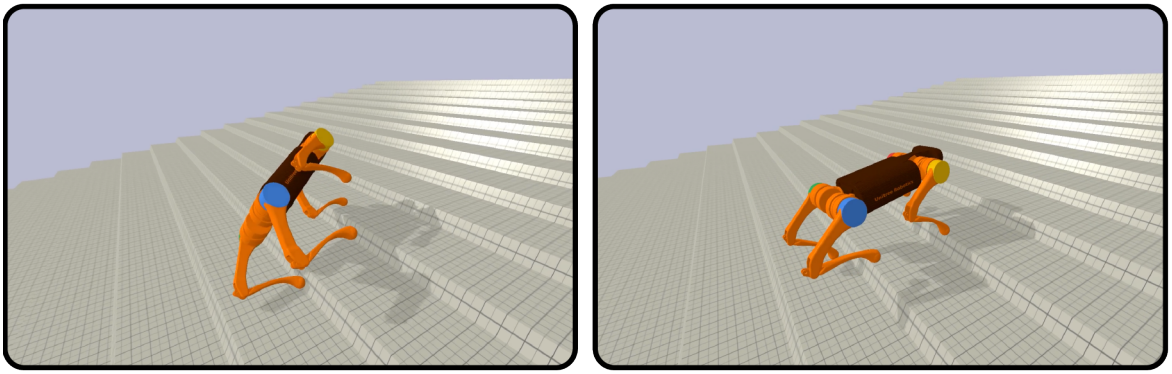

Figure 6.10: STAIRS. The pre-trained agent cannot properly handle discrete height changes, it quickly loses balance and falls while walking towards the steps (on the left). The fine-tuned agent consistently manages to walk up the stairs without loss of balance (on the right).

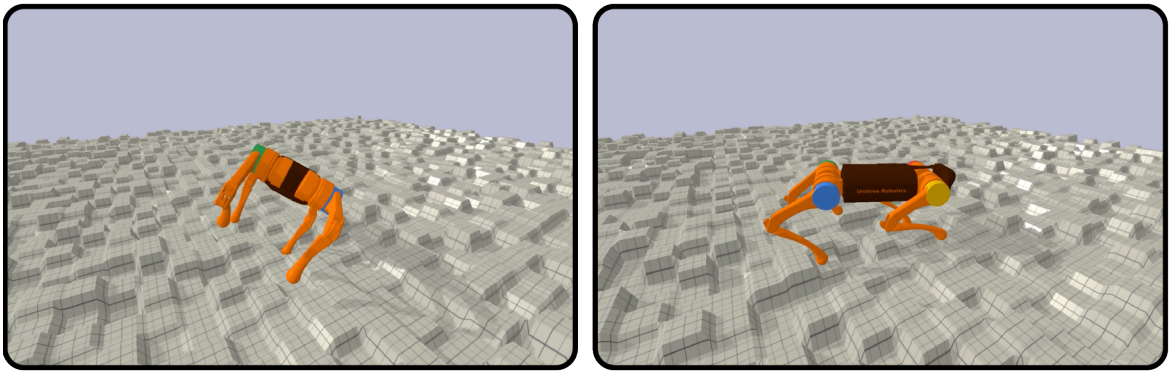

Figure 6.11: STEPS. The pre-trained agent gets its feet trapped between some small steps, overamplifies the reaction and falls (on the left). The fine-tuned agent quickly recovers from trapped feet and consistently lifts the leg high enough to clear the obstacle (on the right).

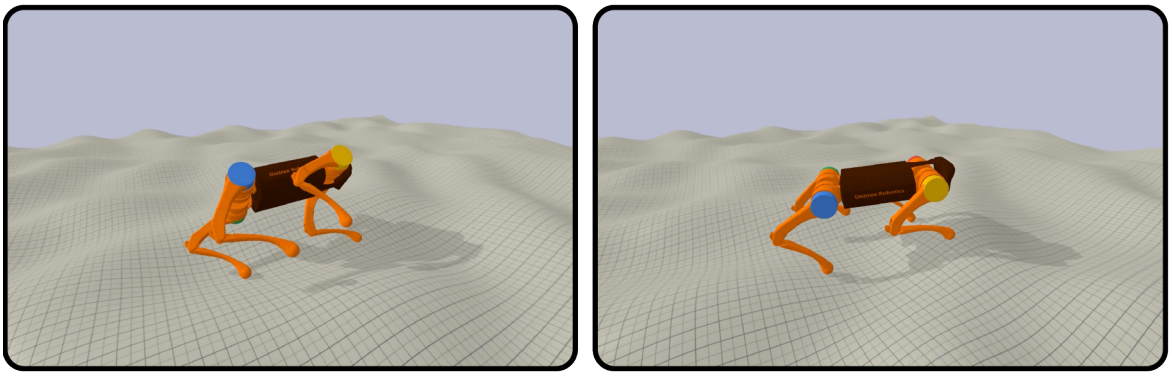

Figure 6.12: HILLS. The pre-trained agent cannot handle inclined terrains and immediately tips over after being deployed (on the left). The fine-tuned agent handles inclined terrains exceptionally well, it manages to recover from slippages and walks straight without problems (on the right).

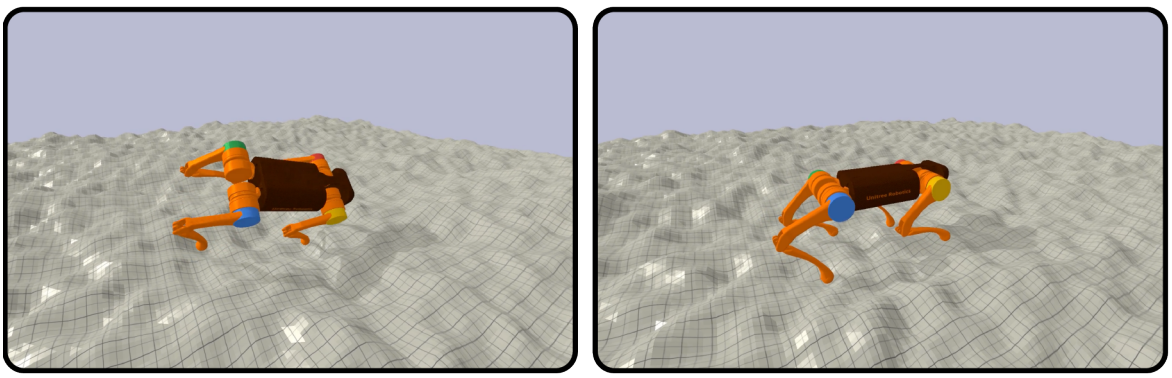

Figure 6.13: MOUNTAINS. The pre-trained agent cannot adjust the frequency of its legs dynamically enough to keep balancing in this highly unstructured terrain, it falls after a few seconds of being deployed (on the left). The fine-tuned agent dynamically adjusts the frequency of its legs in order to stabilise in the unstructured terrain, it never loses balance and continues to walk (on the right).

6.5 Push robustness

One of the main reasons why the pre-trained agent described in section 5.1 received tiny random pushes during training was to make the overall policy more robust to sudden changes that may occur in the real world. These changes may occur in various situations, such as when the robot walks on moving surfaces, when it loses contact with the ground, or when it gets pushed. The agent that sustained random pushes during training proved to be significantly more robust to pushes order of magnitudes larger (4000 N applied for $0.016s$) than the ones it was used to ($\mathcal{N}$(0,30 N) applied for $0.016s$ every $0.016s$). On the contrary, agents that never experienced random pushes during training immediately fell when hit with a strong force, as seen in figure 6.14 and the video linked in the appendix A. We decided

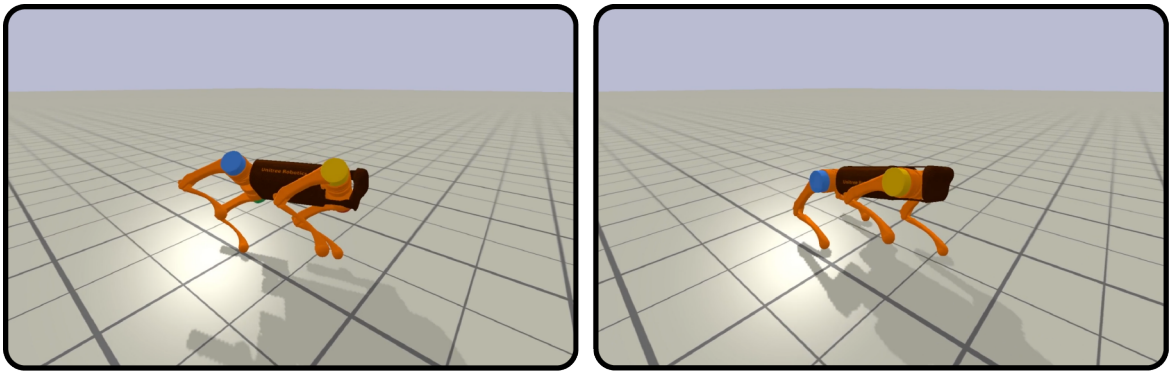

Figure 6.14: On the left, the agent that never experienced pushed during training fails to take a powerful side hit on its chassis. The agent trained while receiving small pushes manages to recover swiftly from the same hit on the right.

not to continue applying this kind of domain randomisation during the curriculum learning fine-tuning to not further increase the training time required. Because of this, none of the fine-tuned policies trained with the various approaches described exhibit the same kind of push robustness as the initial pre-trained policy, and they all fall when pushed with the same strength. Although they are still more robust than the agent that never experienced pushes in training, this certainly means that there has been some forgetting, and they do not have the same level of reactivity and robustness as previously needed. In light of this results, in the future, we might want to keep randomly pushing the robot and use domain randomisation techniques for the entire duration of the training.

Chapter 7

Conclusions

In this project, we set out to answer some critical questions related to Open-Ended learning and Curriculum Learning in a legged robotics setting; we present our conclusions, discuss the main limitations, and propose some compelling grounds where future research might focus on.

Firstly, We found out that training a single generalist agent instead of a population of specialised agents for dynamic robot locomotion is better if the computational budget is limited. This is because there is considerable overlap between the domain skills needed to perform various athletic tasks. For example, it makes more sense to keep building on a single generalist controller rather than re-discovering the basics of robust locomotion every time by individually training different individuals. This, however, is not to say that population training is a bad thing altogether. When what is needed is truly quality diversity, training a population of agents to cover a broader range of diverse skills might be the best option.

We then explored the differences between open-ended and close-ended curriculum learning and clearly showed that the former is a better performer and creates a high-quality and diverse curriculum of terrains that encourage the improvement of the agents. Even though humans have a vast amount of domain knowledge in legged locomotion, and we might think that we can design an excellent curriculum to train and test athletic performance, open-ended learning decisively surpassed our performance. We found out that the overall population of agents trained open-endedly was more performing than those trained with a close-ended curriculum. Each open-ended trained agent was surprisingly generally capable, and the population was seemingly converging to a single individual.

After answering our two initial questions, we decided to put what we learnt to the test and train a single generalist agent with open-ended curriculum learning. The results achieved were excellent for a blind robot using a relatively simple control architecture compared to the state-of-the-art [29]. The performance of the generalist agent dwarfed the two populations of agents given the same total computational budget. These results suggest that open-ended learning of a generalist agent should be the way to approach robotics tasks where overlapping skills are required in the future. As we explain in the next section, we are also interested in seeing how far we can push open-ended learning and use it not only

for the curriculum but also for the architecture and the overall learning process.

7.1 Limitations and Future work

The main limitation of our work arises from the fact that we did not train the final agent with a full command conditioned controller. The problems we faced when experimenting with making the robot fully command conditioned were mainly increased training time and capability of the training algorithms, as we discussed in section 4.5.2. We are not alone in not having deployed a fully command conditioned policy for legged locomotion. A recent and prominent research paper using the A1 robot to explore dynamic locomotion on challenging terrains also deployed their robot with forward locomotion capabilities only [43]. Once we have adequately trained an agent capable of omnidirectional fully command conditioned locomotion, we expect it to be even more robust in challenging terrains, as it will be more used to moving its legged in all directions, making it overall more stable.

However, our work also leaves us with many exciting ideas, potential research paths and future work that we hope to explore soon. We can roughly divide our future work into two different directions. The first one aims to reach and eventually surpass the state-of-the-art in legged locomotion in challenging terrains, while the second one aims to explore some interesting ideas in the realm of open-ended learning, which could also help us reach the first goal.

Throughout the project, we mentioned that the control architecture used was just a means to answer some questions regarding curriculum learning in a robotics setting, and it was never really meant to challenge the state-of-the-art in legged locomotion. As we continue to develop more expertise in this field, we want to improve upon the work done so far and implement a more capable control architecture that could rival the current state-of-the-art one, which is more complex, has more than ten times the amount of NN parameters that we used, and goes through a carefully thought out teacher-student training and curriculum learning [29]. However, after seeing the surprisingly good performance displayed by open-ended learning, we wonder whether crafting such a complicated and carefully tuned control architecture is necessary or whether it would be possible to evolve the control architecture and training algorithm itself open-endedly. Thus, this research frontier in open-ended learning is of high interest for us and could represent some promising future work that we might want to venture into.

Appendix A

Videos

Video 1 shows forward + yaw command conditioned, push robustness, and compares the pre-trained agent with the fine-tuned agent trained with open-ended curriculum learning on four types of terrains.

https://www.youtube.com/watch?v=IE6DSlAliQ8

Video 2 shows the performance of the generalist agent trained with open-ended curriculum learning in a variety of challenging self-proposed environments.

https://www.youtube.com/watch?v=PUApftlqQ_Y

Appendix B

Ethics considerations and checklist

The experiments conducted during this project were performed in simulation using PyBullet physics; no harm could have been made to objects, persons or animals. Although some companies like Ghost Robotics [57] are conducting research in quadruped robot locomotion for military purposes, we are confident that our work, as is, does not provide ground for misuse, as it follows a different philosophy to robotics than the one used by the company mentioned above. Furthermore, the scope of this project is purely for locomotion. A complete ethics checklist following the template provided by the Imperial College Department of Computing can be found in the following few pages B.1 B.2.
Parts of the code related to the A1 robot PyBullet interface and URDF have been adapted from code open-sourced by Google under the Apache License and used accordingly. As such, no copyright claims can be made.

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from "Human Embryos/Foetuses" i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓

Figure B.1: Ethics checklist part 1 - as required by the Department of Computing at Imperial College

	Yes	No
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
SECTION 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
SECTION 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Figure B.2: Ethics checklist part 2 - as required by the Department of Computing at Imperial College

Bibliography

[1] ANYbotics. *"ANYbotics Homepage"*. https://www.anybotics.com. (2021).

[2] Boston Dynamics. *"Spot"*. https://www.bostondynamics.com/spot. (2021).

[3] Nadia Dominici et al. "Locomotor primitives in newborn babies and their development". In: *Science* 334.6058 (2011), pp. 997–999.

[4] Atil Iscen et al. "Policies modulating trajectory generators". In: *Conference on Robot Learning*. PMLR. 2018, pp. 916–926.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[7] Aston Zhang et al. "Dive into deep learning". In: *arXiv preprint arXiv:2106.11342* (2021).

[8] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[9] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[10] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[11] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[12] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[13] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[14] Tim Salimans et al. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[15] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple random search provides a competitive approach to reinforcement learning". In: *arXiv preprint arXiv:1803.07055* (2018).

[16] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. "Quality diversity: A new frontier for evolutionary computation". In: *Frontiers in Robotics and AI* 3 (2016), p. 40.

[17] Antoine Cully and Yiannis Demiris. "Quality and diversity optimization: A unifying modular framework". In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 245–259.

[18] Joel Lehman and Kenneth O. Stanley. "Exploiting open-endedness to solve problems through the search for novelty". In: *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI*. MIT Press, (2008).

[19] Joel Lehman and Kenneth O Stanley. "Abandoning objectives: Evolution through the search for novelty alone". In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[20] Jean-Baptiste Mouret and Jeff Clune. "Illuminating search spaces by mapping elites". In: *arXiv preprint arXiv:1504.04909* (2015).

[21] Rui Wang et al. "Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions". In: *arXiv preprint arXiv:1901.01753* (2019).

[22] Jared Di Carlo et al. "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control". In: *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2018, pp. 1–9.

[23] Gerardo Bledt. "Regularized predictive control framework for robust dynamic legged locomotion". PhD thesis. Massachusetts Institute of Technology, 2020.

[24] Marc Raibert and Sethu Vijayakumar. *"Turing Lecture: Building dynamic robots - Marc Raibert, Boston Dynamics"*. https://www.youtube.com/watch?v=yLtdzJ6mVMk&t=0s. (2020).

[25] Daniel Görges. "Relations between model predictive control and reinforcement learning". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 4920–4928.

[26] Damien Ernst et al. "Reinforcement learning versus model predictive control: a comparison on a power system problem". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2008), pp. 517–529.

[27] Yuxiang Yang et al. "Data efficient reinforcement learning for legged robots". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1–10.

[28] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019).

[29] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 5.47 (2020).

[30] Nicolas Heess et al. "Learning and transfer of modulated locomotor controllers". In: *arXiv preprint arXiv:1610.05182* (2016).

[31] Xue Bin Peng et al. "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13.

[32] Deepali Jain, Atil Iscen, and Ken Caluwaerts. "Hierarchical reinforcement learning for quadruped locomotion". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 7551–7557.

[33] Deepali Jain, Atil Iscen, and Ken Caluwaerts. "From pixels to legs: Hierarchical learning of quadruped locomotion". In: *arXiv preprint arXiv:2011.11722* (2020).

[34] Josh Merel et al. "Hierarchical visuomotor control of humanoids". In: *arXiv preprint arXiv:1811.09656* (2018).

[35] Matteo Hessel et al. "On inductive biases in deep reinforcement learning". In: *arXiv preprint arXiv:1907.02908* (2019).

[36] Atil Iscen et al. "Learning Agile Locomotion Skills with a Mentor". In: *arXiv preprint arXiv:2011.05541* (2020).

[37] Xue Bin Peng et al. "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14.

[38] Xue Bin Peng et al. "Learning agile robotic locomotion skills by imitating animals". In: *arXiv preprint arXiv:2004.00784* (2020).

[39] Nicolas Heess et al. "Emergence of locomotion behaviours in rich environments". In: *arXiv preprint arXiv:1707.02286* (2017).

[40] Carlos Florensa et al. "Automatic goal generation for reinforcement learning agents". In: *International conference on machine learning*. PMLR. 2018, pp. 1515–1528.

[41] Rui Wang et al. "Enhanced POET: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9940–9951.

[42] Jonah Siekmann et al. "Sim-to-Real Learning of All Common Bipedal Gaits via Periodic Reward Composition". In: *arXiv preprint arXiv:2011.01387* (2020).

[43] Ashish Kumar et al. "Rma: Rapid motor adaptation for legged robots". In: *arXiv preprint arXiv:2107.04034* (2021).

[44] Tuomas Haarnoja et al. "Learning to walk via deep reinforcement learning". In: *arXiv preprint arXiv:1812.11103* (2018).

[45] Xue Bin Peng and Michiel van de Panne. "Learning locomotion skills using deeprl: Does the choice of action space matter?" In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–13.

[46] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[47] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[48] Antoine Cully et al. "Robots that can adapt like animals". In: *Nature* 521.7553 (2015), pp. 503–507.

[49] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. “Reset-free trial-and-error learning for robot damage recovery”. In: *Robotics and Autonomous Systems* 100 (2018), pp. 236–250.

[50] Jean-Baptiste Mouret and Glenn Maguire. “Quality diversity for multi-task optimization”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 121–129.

[51] Jeff Clune. “AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence”. In: *arXiv preprint arXiv:1905.10985* (2019).

[52] Ended Learning Team et al. “Open-Ended Learning Leads to Generally Capable Agents”. In: *arXiv preprint arXiv:2107.12808* (2021).

[53] Jonathan C Brant and Kenneth O Stanley. “Minimal criterion coevolution: a new approach to open-ended search”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 67–74.

[54] Jeff Clune. *"GECCO 2021 - AI-Generating Algorithms"*. https://www.youtube.com/watch?v=FJJpmxO2Ci8&t=0s. (2021).

[55] Erwin Coumans and Yunfei Bai. “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In: (2016).

[56] UniTree Robotics. *"A1"*. https://www.unitree.com/products/a1/. (2020).

[57] Ghost Robotics. *"Ghost Robotics Homepage"*. https://www.ghostrobotics.io. (2021).