Universidade Estadual de Campinas
Instituto de Computação

INSTITUTO DE COMPUTAÇÃO

Yuri Corrêa Pinto Soares

Deep Reinforcement Learning for Bipedal Locomotion

Aprendizagem por Reforço Profundo para Locomoção Bípede

CAMPINAS
2020

Yuri Corrêa Pinto Soares

Deep Reinforcement Learning for Bipedal Locomotion

Aprendizagem por Reforço Profundo para Locomoção Bípede

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientadora: Profa. Dra. Esther Luna Colombini

Este exemplar corresponde à versão final da Dissertação defendida por Yuri Corrêa Pinto Soares e orientada pela Profa. Dra. Esther Luna Colombini.

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Soares, Yuri Corrêa Pinto, 1994-
So11d Deep reinforcement learning for bipedal locomotion / Yuri Corrêa Pinto Soares. – Campinas, SP : [s.n.], 2020.

Orientador: Esther Luna Colombini.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Aprendizado por reforço profundo. 2. Caminhada bípede. 3. Robótica. I. Colombini, Esther Luna, 1980-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Aprendizado por reforço profundo para caminhada bípede
Palavras-chave em inglês:
Deep reinforcement learning
Biped locomotion
Robotics
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Esther Luna Colombini [Orientador]
Marcos Ricardo Omena de Albuquerque Maximo
Eric Rohmer
Data de defesa: 23-10-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-9327-0189
- Currículo Lattes do autor: http://lattes.cnpq.br/9562053106843512

Universidade Estadual de Campinas
Instituto de Computação

INSTITUTO DE
COMPUTAÇÃO

Yuri Corrêa Pinto Soares

Deep Reinforcement Learning for Bipedal Locomotion

Aprendizagem por Reforço Profundo para Locomoção Bípede

Campinas, 23 de outubro de 2020

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

This study was financed in part by the São Paulo Research Foundation (FAPESP), grant #2017/21426-7.

I would like to thank the Unicamp community and staff for all the cherished years of learning and personal growth. I also want to thank FAPESP and CAPES for not only supporting this work, but also advancing the scientific and technological research in our country as public foundations.

I would like to thank my advisor, Esther. She supported me enthusiastically ever since the beginning of the program, both in academia and also in my personal and professional development outside the university. I appreciate that Esther would always allow me the freedom to explore my own path while always being kind and friendly when mentoring me. Esther encouraged and inspired me to give my best at all times.

I also want to thank the friends I made in our research laboratory: Rafael, Gabriel, Samuel, Guilherme and Renan for the relatively brief, but equally valuable companionship.

I would like to thank the friends I made at Unicamp: Douglas, Jardel, Luan, Fernando, Luciano, Jucélio, André, Rafael, Murilo, Sachs and many others whose companies during classes, projects, games and meals comprises the best memories I made during my time in Unicamp.

I also want to thank my high school friends that, even from afar, still manage to be very much present in my life after all these years: Bruno, Isabela, Rian, Carol and Borsato.

Lastly, I would like to thank my family: José, Vânia, Erick and Malu, for the unconditional love and support. Ever since moving out for studying, I started appreciating every single opportunity I get at being with them and I feel truly fortunate for being a part of this family.

Resumo

Robótica e suas aplicações de serviço com robôs bípedes tem se expandido recentemente devido a possibilidade de se usar robôs desta categoria em ambientes originalmente planejados para operação humana. No entanto, locomoção bípede tem se mostrado um desafio teórico e prático devido a alta dimensionalidade do problema, visto que a ação de andar tipicamente envolve o controle preciso em tempo-real de múltiplos atuadores e sensores em conjunto com sistemas dinâmicos complexos. Concomitantemente, aprendizado por reforço (RL) e sua versão com redes neurais profundas (DRL) estão se tornando uma abordagem prominente para solucionar tais problemas, devido a sua capacidade de lidar com processos contínuos e livres de modelo. Neste trabalho, modelamos a tarefa de locomoção como um problema de aprendizagem por reforço, propondo representações práticas baseada em MDPs e estratégias generalizáveis para funções de reforço. Em seguida, prosseguimos desenvolvendo um framework para integrar nosso simulador de escolha (CoppeliaSim [11]) com a interface corrente padrão para aprendizagem por reforço (OpenAI Gym [5]). Finalmente, nós aplicamos algoritmos do estado-da-arte em aprendizado por reforço profundo com nosso framework em experimentos configuráveis para validar nossa modelagem e aprender uma política de caminhada estável em simulação para o robô Marta, um sofisticado robô humanoide com 25 graus de liberdade.

Abstract

Robotics and its service applications with biped robots have faced an upsurge lately as this category of robots is suitable for deployment in environments originally designed for operation by humans. However, bipedal locomotion has proven to be a challenge in theory and practice due to the problem's high dimensionality: as walking gaits typically involve precise real-time control of multiple actuators and sensors, coupled with complex dynamical systems. Concomitantly, reinforcement learning (RL) and its deep neural network version (DRL) are becoming a prominent approach in solving such challenging control problems due to their capacity to work on continuous and model-free processes. In this work, we modeled a locomotion task as an RL problem by proposing practical MDP representations and generalizable reward engineering strategies. We then proceeded to develop a framework for integrating our simulator of choice (CoppeliaSim [11]) with the de facto standard interface for Reinforcement Learning (OpenAI Gym [5]). Finally, we applied state-of-the-art DRL algorithms within our framework in configurable and reproducible experiments to validate our modeling and learn a stable walking gait in simulation for the Marta robot, a sophisticated humanoid robot with 25 DOFs.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Artificial intelligence and automation are continually driving the frontiers of society in recent times as whole industries have perished while new ones have flourished in the last few decades. As a growing market, robotics is playing an important role in research nowadays because, as an interdisciplinary field, its applications demand growing collaboration in research and education.

Robotic locomotion plays a key role in these developments, enabling an ever-increasing pace of technological progress. Due to the simplicity of control and the high efficiency in flat terrains, wheeled robots are still the most commonly used. However, they face problems such as soil unevenness and adaptability. These problems, on the other hand, tend not to be an issue for legged robots. Among the various legged arrangements, there is a particular interest in humanoids since these would be the most adapted to interact with tools and environments already created for humans.

Bipedal locomotion, in particular, has proven to be a long-standing challenge in theory and practice. One of the most challenging aspects is the high dimensionality of the problem: walking gaits typically involve precise real-time control of multiple actuators and sensors coupled with complex dynamical systems and energy requirements to produce usable thrust. Hand engineering these attributes is an error-prone process and, unfortunately, has to be reworked and adapted for each new model, environment or task.

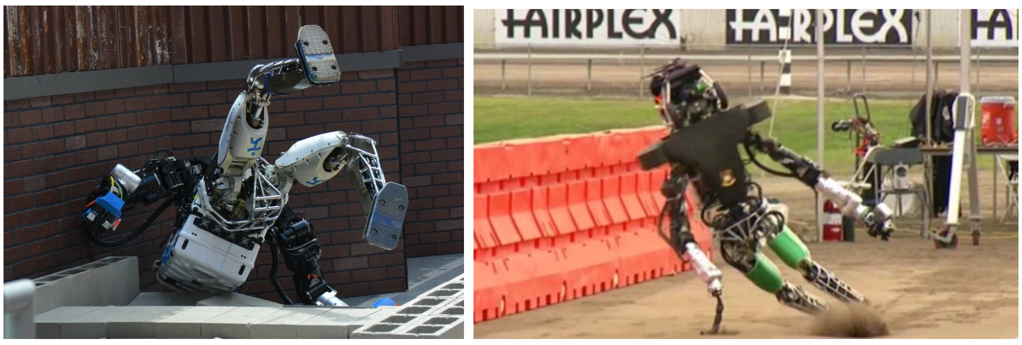

Figure 1.1: Even under controlled environments and expensive budgets, bipedal locomotion has been a notoriously difficult task with a myriad of real missteps. [24]

To tackle these contemporary challenges, classical and vanguard methods in artificial intelligence are frequently employed on a large-scale. Machine learning (ML), in particular, has enjoyed a recent upsurge in development and interest across both academia and industry, gradually becoming a household name that has received uncommon mass media coverage for a scientific field. One of the most recent of such methods in ML is Reinforcement Learning (RL). As we will discuss in the following sections, RL is increasingly becoming one of the top choices to tackle these and other related problems. Deep Reinforcement Learning (DRL), in particular, employ deep neural networks to cope with high dimensional inputs, as well as modeling non-linear behavior and fitting good predictions.

Over the next sections, we shall delve more formally in reinforcement learning, drafting some key aspects that sets it apart from other machine learning approaches while adopting a predominantly academic perspective.

1.1 Motivation

As DRL methods upsurge, many tasks related to bipedal control are now successfully addressed [48, 19, 50] in the literature. However, most of the test scenarios where they are applied run over simulated platforms that contain a non-negligible number of limitations and constraints regarding the dynamics of the simulation or the robot model. Despite their accomplishments, such simplifications in the controlled model could lead to the unsuccessful application of these algorithms in more realistic scenarios.

1.2 Objectives

Our objective is to study and apply DRL algorithms to develop and improve a stable walking gait for the humanoid robot Marta and possibly similar models. More details of the Marta model are discussed in the following chapters. Our objective comprises both the use of state-of-the-art DRL algorithms as a black box to assess how feasible the use of reinforcement learning is in the automation of gait discovery and comparing and tuning different algorithms parameters and reward functions for our particular case.

The following hypotheses were formulated to measure the attainment of our objectives:

- H_1: DRL algorithms can be used to conceive a walking gait for Marta from scratch.
- H_2: Reward engineering can be used to improve discovered gaits.
- H_3: It is possible to transfer the policy learned in simulation to the real robot.

1.3 Main contributions

In this project, we widely investigated and applied Deep Reinforcement Learning to many environments, architectures, and algorithms. Due to the field's very nature, the work was mostly experimental and heavily focused in rapid iterations and empirical results.

We list the following main contributions:

- Developed an extensible and transparent framework for integrating the V-REP [11] simulator (now CoppeliaSim) with the de facto standard interface for Reinforcement Learning (OpenAI Gym [5]).
- Applied DRL state-of-the-art algorithms to discover novel gaits for Marta without explicit supervision or human design.
- Proposed and compared different reward functions and how they affect the discovered gaits.

1.4 Text structure

We organized this Master's thesis as follows:

- In **Chapter 1**, we introduced our motivation, objective, hypotheses, and contributions.
- In **Chapter 2**, we discuss the theoretical background, and introduce useful mathematical notation for later chapters.
- **Chapter 3** outlines related work as well as recent developments in the field.
- Our main approach to modeling the problem is thoroughly discussed in **Chapter 5**.
- The experimental results of our models are presented and discussed in **Chapter 6**.
- **Chapter 7** concludes the dissertation considering our main contributions and contemplating possible opportunities for future work.

Chapter 2

Theoretical Background

In this chapter, we present the theoretical background behind state-of-the-art Reinforcement Learning methods and the common issues faced by these algorithms.

2.1 Computers and cognition

Machine learning problems are often divided into three categories [47]: supervised, unsupervised, and reinforcement learning.

Supervised learning. This task category involves learning inference through labeled training data. While supervised learning systems have displayed powerful classification and inferring capabilities for a myriad of tasks, they usually require vast amounts of correctly labeled input data [36]. While this requirement may be acceptable for problems that possess a considerable amount of examples, it poses a prohibitive obstacle to original or dynamic problems, such as robotic locomotion.

Unsupervised learning. On the other hand, unsupervised learning techniques do not have the same requirements as supervised ones. This mode of learning is already extensively used in statistical summarizing and clustering [18], but it has gained substantial attention in recent years, enjoying and contributing to advancements in other areas of artificial intelligence. Most notably, Generative Adversarial Nets (GANs) [14] were originally proposed as a generative model for unsupervised learning, but have found use in Supervised and Reinforcement learning as well.

Reinforcement Learning. In this project, we shall focus on Reinforcement Learning (RL). In this kind of problem, training data are not available or are hard to be defined. However, behaviors may be objectively assessed automatically by one or more measurements. These objective values are called **rewards**, and are intrinsically related to the task at hand. For example, in a robotic locomotion task, the reward may be defined as the final distance attained in one single episode. Another possible reward is the instantaneous velocity at each moment of the episode. The first case is usually called a type of **sparse reward** and the latter, a **dense reward**.

2.2 Reinforcement learning

In reinforcement learning, the system does not require pre-labeled data to train and, instead, relies on a reward signal to improve and construct policies that dynamically interact with the problem environment. Differently from the previous tasks, RL tasks can be modeled as an agent interacting with a simulated or real stochastic environment. These interactions may span variable time frames and be episodic, giving rise to the notion of long-term rewards or penalties. As such, RL is not limited to traditional classification or regression, but incorporates decision-making, planning, knowledge representation and responsiveness to new and unfamiliar elements. These concepts will be more formally defined in Section 2.3. RL bears a strong theoretical foundation for modeling robotic tasks, and developing this technique has bolstered plenty of practical applications.

Common challenges

Most RL methods face some common challenges that are addressed in different ways depending on the selected technique. We detail next the most relevant of these challenges.

Exploration vs. exploitation. One of the major challenges in RL is the exploration vs. exploitation dilemma [25]. This trade-off is of major practical importance and consists of finding a balance between exploiting the current knowledge for amassing rewards, or exploring unknown settings to improve such knowledge at the potentially cost of some reward.

Credit assignment problem. When interacting with an environment, the agent may observe a reward that is a consequence of a remote past action [52]. These delayed rewards represent a problem because it encumbers the agent to identify and exploit favorable actions correctly.

Partial observability. In RL problems, the agent only has access to observations, which, in most cases, do not contain all the information about the environment state [6]. In Robotics, this problem is ubiquitous: it is impractical to have sensors to account for every dynamic element of a complex system.

Reward shaping and specification gaming. For many useful tasks, it is hard to evaluate performance in an objective and numerical way. Assessing the quality of a walking gait, for example, tends to be an intuitive and subjective human ability, hard to be translated analytically. For these kinds of tasks, a technique called Reward Shaping [37] is used, in which additional reward signals are used to guide learning and approximate the intended behavior. The drawback of reward shaping is specification gaming: in which the agent learns how to satisfy the defined reward, but the learned policy is not satisfactory for the original task, often in an ingenious way. By far, specification gaming was the most frequent challenge that manifested in our project, and we will discuss it separately in the following chapters.

We will also analyze how the different RL techniques approach these challenges throughout the text.

2.3 Markov decision process

As we previously discussed, RL is mainly concerned with the dynamic interactions of an agent in a changing environment. Such a framework is often described as a Markov decision process (MDP) [3]. Using the same notation as [49], an MDP consists of the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$ with $\mathcal{S}$ and $\mathcal{A}$ being the sets of states and actions, respectively, and the transition probability distribution is described as:

$$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1] \tag{2.1}$$

The reward function can be defined as:

$$r : \mathcal{S} \mapsto \mathbb{R} \tag{2.2}$$

The distribution of the initial state s_0:

$$\rho_0 : \mathcal{S} \mapsto \mathbb{R} \tag{2.3}$$

and the discount factor $\gamma \in [0, 1]$. In regard to the agent formulation, we denote a stochastic policy:

$$\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1] \tag{2.4}$$

A policy can be understood as the main determinant of an agent, it is the decision-making process we wish to optimize. In this stochastic formulation, the policy assigns a probability value to all state-action pairs, expressing the probability of the action being carried out. In the particular case that all probabilities are zero or one, the policy is said to be deterministic. As we will discuss more in Section 2.4, there are a number of different ways to represent a given policy, such as look-up tables, graphs, automata and neural networks, to name a few. The agent main objective is to maximize the expected cumulative reward, called return:

$$J(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right] \tag{2.5}$$

We may also define secondary functions, such as the state-action value function:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right] \tag{2.6}$$

The state-action value function can be understood as the cumulative reward of taking the action a_t in state s_t and following the policy π thereafter. The value function is similarly defined, but the action is implicitly sampled from π:

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, \dots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right] \tag{2.7}$$

Some algorithms also define an advantage function such as:

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

with states and actions sampled accordingly. In the context of RL environments, the MDP may be partially observable. In that case, the subset of the state space $\mathcal{S}$ that is observable to the agent can be denoted as the observation space $\mathcal{O}$. Such definitions may facilitate further discussion on policy optimisation.

It is important to note that formal RL definitions and terminology are not completely standardized and slight variations can be seen in different works.

2.4 RL approaches

As defined in Section 2.3, the main RL problem is to find an optimal policy π^* that maximizes the expected rewards for any given s_0. In other words, RL can be understood as a policy optimisation problem. Regarding this optimisation, RL can be tackled by two main approaches: value-based or policy-based.

2.4.1 Value-based RL

This approach attempts to learn the value function (or, similarly, the state-action value function). With it, an implicit policy is derived: for each state s_t, the policy consists of choosing the action $a_t = max_a Q(s_t, a)$. The prime example of value-based RL is Q-learning [60]. In this method, the values of $Q_{\pi}(s_t, a_t)$ are progressively approximated by using the Bellman equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.9}$$

Where $\alpha \in (0, 1]$ is the learning rate parameter. It has been demonstrated that, with appropriate values of α and infinite iterations, such approximation indeed converges to the optimal policy [22]. In practice, an ε-greedy approach is used when deriving a policy π from a Q function. Under this guideline, the action with the highest approximated Q value is picked with probability $(1 - \varepsilon)$, and a random action is selected otherwise. In this way, ε-greedy procedures are an efficient and easily parameterized response to the exploration vs. exploitation dilemma previously mentioned [57].

2.4.2 Policy-based RL

Policy-based RL avoid dealing with value functions and, instead, attempts to directly learn the policy. This approach can be further classified into three categories [49].

Policy iteration methods These methods iterate in policy space in an attempt to improve the policy. They generally alternate between estimating the value function and improving the policy [4]. These methods include more theoretical ones, such as brute force, where all possible policies are iterated. While this may be suitable for small MDPs, it is unfeasible for stochastic and large (sometimes infinite) MDPs in which RL is generally concerned. Please note that including value-based RL inside policy iteration methods is simply a matter of definition.

Policy gradient methods As the name suggests, these methods employ the gradient (or an estimate) of the total reward concerning the policy parameters [45]. The purpose of these methods is to update the policy with stable monotonic improvement.

Derivative-free optimization methods This last category does not require the use of gradients, viewing all returns as a black box function [53]. Derivative-free stochastic optimisation methods have been widely applied with good results, despite their apparent simplicity. Standard techniques include the cross-entropy method (CEM) and covariance matrix adaptation evolution strategy (CMA-ES).

As we shall see in Chapter 3, novel methods such as Actor-critic attempt to bridge the gap between value-based and policy-based RL, learning the value function and policy concomitantly.

2.5 Soft Actor-Critic

As our algorithm of choice in this work is the Soft Actor-Critic (SAC) [17], we will discuss its theoretical framework into more detail here and its historical context at Chapter 3 along with other recently proposed algorithms.

SAC is an off-policy algorithm that optimizes a stochastic policy. It is a successor of Soft Q-Learning SQL [16], incorporating ideas from earlier algorithms such as DQN [35] and DDPG [32] while bearing similarities with TD3 [13]. It is based upon both actor-critic methods and entropy-regularized RL, with the latter being its most distinctive feature. Instead of maximizing the standard return, SAC's objective is to maximize a trade-off between expected return and entropy by using entropy regularization. Increasing entropy makes the policy act more randomly which aids in exploration. Reminding that, for a random variable x and probability mass or density function P, the entropy H of x can be computed from its distribution P as:

$$H(P) = \underset{x \sim P}{\mathrm{E}} [-\log P(x)] \tag{2.10}$$

The entropy bonus is included directly in the definition of the optimal policy π^* and weighted by a new hyper-parameter α:

$$\pi^* = \arg\max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}} \left[\sum_{t=0}^{\infty} \gamma^t \bigg(R(s_t, a_t, s_{t+1}) + \alpha H \left(\pi(\cdot|s_t) \right) \bigg) \right] \tag{2.11}$$

Similarly, the equations of V^π and Q^π are also updated to include the entropy bonuses:

$$V^\pi(s) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t\bigg(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\bigg)\bigg|\, s_0 = s\right] \tag{2.12}$$

$$Q^\pi(s,a) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi(\cdot|s_t)\right)\bigg|\, s_0 = s, a_0 = a\right] \tag{2.13}$$

Similarly to TD3 [13], SAC concurrently learns two Q-functions, a technique known as Clipped Double-Q Learning [13]. A SAC algorithm formulation [2] can be seen in Algorithm 1.

Algorithm 1: Soft Actor-Critic

```
 1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay
    buffer $\mathcal{D}$
 2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
 3: repeat
 4:   Observe state $s$ and select action $a \sim \pi_\theta(\cdot|s)$
 5:   Execute $a$ in the environment
 6:   Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is
      terminal
 7:   Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
 8:   If $s'$ is terminal, reset environment state.
 9:   if it is time to update then
10:     for $j$ in range(however many updates) do
11:       Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:       Compute targets for the Q functions:
            $$\tilde{a}' \sim \pi_\theta(\cdot|s')$$
            $$y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right)$$
13:       Update Q-functions by one step of gradient descent using
            $$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \qquad \text{for } i = 1,2$$
14:       Update policy by one step of gradient ascent using
            $$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$
          where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt $\theta$ via the
          reparametrization trick.
15:       Update target networks with
            $$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1,2$$
16:     end for
17:   end if
18: until convergence
```

Chapter 3

Related work and Recent Developments

As discussed so far, the area of reinforcement learning is teeming with diverse and promising ideas. RL could take advantage of solid mathematical frameworks such as MDPs while also drawing inspiration from higher-level fields such as psychology. However, in order to be practically useful, the methods previously described must be carefully implemented and validated, demonstrating the ability to converge in a, preferably, finite amount of time. In this section, we analyze some of the most recent and successful ideas found in the literature while also broadly replicating some results and applications.

3.1 Artificial neural networks

Artificial neural networks (**ANN**) and, in particular, deep neural networks (**DNN**) also have enjoyed widespread use in machine learning and other domains [29]. In the context of RL, DNNs have been used primarily as approximators for the defined functions or policies, whereas ANNs and DNNs tackle sensorial and high dimensional data, RL tackles decision making and planning.

3.1.1 DQN

Two of the major problems in using sparse representation with the Q-learning method, presented in Section 2.4.1, are the rate of convergence and the number of states. Nonetheless, DNN has demonstrated proficient capability in the task of approximating functions and can be applied to approximate the Q function and mitigate both convergence and combinatorial explosion problems [35]. One of the most significant advantages of this variety is the ability to have good approximations of Q-values for states that have never been visited. Instead of the traditional update, the network can be optimized with a squared error loss:

$$L = \frac{1}{2}\big[r + \gamma max_{a'} Q(s', a') - Q(s, a)\big]^2$$

Unfortunately, the introduction of DNNs produces some other practical obstacles. One such problem is caused by the similarity of consecutive samples, which might converge the network into a local minimum. To avoid this, all the transitions are stored in a **replay memory**, which is sampled randomly during training. This practical consideration is called **experience replay**. A high-level DQN algorithm that considers the points as mentioned earlier is described in Algorithm 2, using a notation scheme similar to OpenAI Gym [5].

The seminal work in DQN of Mnih et al. [35] represented a breakthrough not only in RL but in ML in general. Their work represented one of the first general-purpose agent able to excel in a task of high-dimensional input, encouraging novel research on the topic. An example game that their system was able to master is shown in Figure 3.1.

Algorithm 2: Deep Q-learning Algorithm

```
Input: Discrete environment $\mathcal{E}$
Output: Q network with implicit policy
1  initialize replay memory $\mathcal{D}$ ;
2  initialize action-value function $Q$ with random weights;
3  observe initial state s;
4  while $learning\_condition = true$ do
5      with probability $\epsilon$: select $a = random.action()$ ;
6      otherwise select $a = argmax_{a'} Q(s, a')$ ;
7      $(s', r) = \mathcal{E}.transition(s, a)$ ;
8      store experience $< s, a, r, s' >$ in replay memory $\mathcal{D}$ ;
9      sample random transitions $< s_D, a_D, r_D, s'_D >$ from replay memory $\mathcal{D}$ ;
10     calculate target for each mini-batch transition ;
11     if $s'_D$ is terminal state then
12         $t_D = r_D$ ;
13     else
14         $t_D = r_D + \gamma max_{a'} Q(s'_D, a'_D)$ ;
15     end
16     train the $Q$ network using $(t_D - Q(s_D, a_D))^2$ as loss ;
17     $s = s'$ ;
18 end
```

Following that, the original tabular Double Q-learning was combined with DQN, resulting in Double DQN, with even better performance [59].

3.1.2 Actor-critic

In this method, the action-value function and policy are generally parameterized to support gradient methods.

The action-value and policy parameters are usually labeled as w and θ, respectively. As mentioned in Section 2.2, actor-critic methods aims to benefit from the advantages of both value-based and policy-based approaches, and have succeeded on a wide variety of problems [34]. To achieve that, these algorithms maintain two entities with different roles: the *critic* estimates the action-value function and updates w, while the *actor* updates θ as directed by the *critic*. A diagram of a generic actor-critic framework can be seen in Figure 3.2.

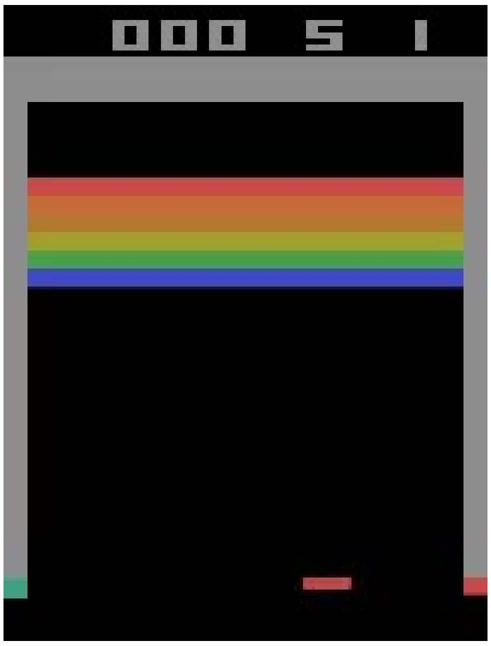

Figure 3.1: A deep reinforcement learning agent proposed in [35] was able to achieve super-human mastery on the simulated Atari game Breakout by using high-dimensional input: the pixels on the screen.

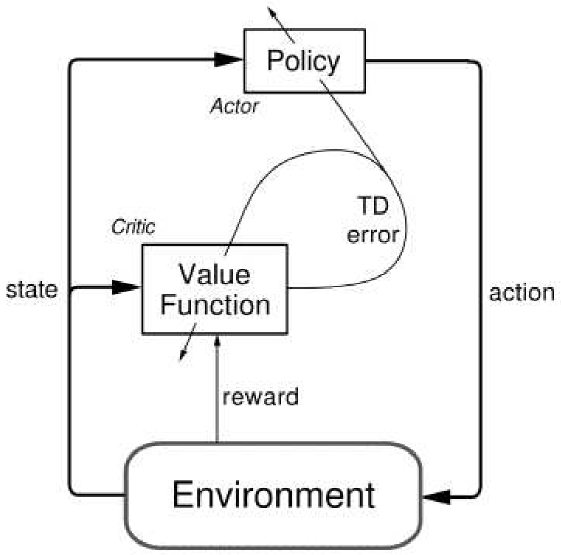

Figure 3.2: Diagram of actor-critic generic strategy. Extracted from [51]

Actor-critic strategies have recently become part of the most popular algorithms in the RL framework thanks to their manageable architecture and successful practical results [44] being later extended as asynchronous actor-critic methods [43]. Actor-critic reinforcement learning has also been applied with multiple policy gradients [15], which we review in Section 3.2.

By using the actor-critic approach, Lillicrap et al. [31] proposed the Deep Deterministic Policy Gradient (DDPG), which extends DQN to continuous action spaces, allowing its application to more diverse tasks.

3.2 Policy gradients

Advancements in RL methods have not been limited to neural networks. Methods that draw inspiration from classic optimization are also constantly improving. In this subsection, we discuss some improvements regarding direct policy optimization.

3.2.1 Natural policy gradient

In [26], Kakade provided a natural gradient method along with experimental results that show great improvements over the standard gradient method in both simple and complex MPDs, which translates as a drastic contribution to policy iteration methods.

3.2.2 Trust Region Policy Optimization (TRPO)

Schulman et al. [49] built upon natural policy gradient methods, describing an algorithm with strong guarantees of monotonic improvement. In this method, the size of the policy update is constrained by a parameter, which derive the name “Trust Region”. The proposed Trust Region Policy Optimization (TRPO) can be used effectively for optimizing large nonlinear policies such as neural networks. In the same article, experimental results indicate robust performance for diverse tasks: from 2D robotic tasks to Atari playing with screen inputs. One of the major contributions of their paper is a generic policy search method able to learn such diverse controllers from scratch. Some models used in the tasks are replicated in Figure 3.3.

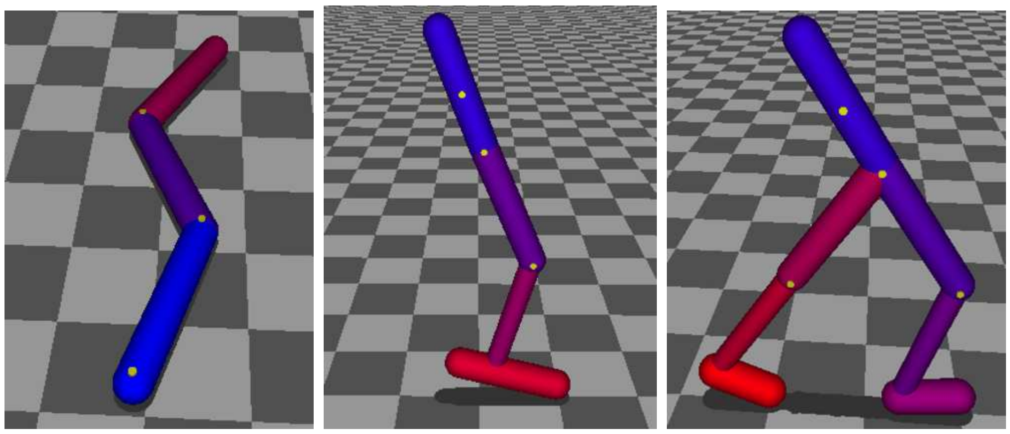

Figure 3.3: 2D robot models used in [49] for locomotion experiments. Notice that the models are visually rendered in three dimensions, but they are still constrained to a bi-dimensional plane.

Subsequently, the same team combined TRPO for both the policy and the value function with an estimator of the advantage function [50]. Using this approach, they were able to create policies for controlling simulated 3D robots, with up to 33 state dimensions and 10 actuators.

Proximal Policy Optimization (PPO)

TRPO has been further developed and simplified, giving rise to Proximal Policy Optimization (PPO) [48], outperforming previous online policy gradient methods. In PPO, the policy update's size is no longer constrained but, instead, is only penalized. The substitution of the hard constraint for a penalty has multiple benefits, in addition to simplifying the implementation. At the time of writing, PPO is considered the best and default algorithm for a wide range of DRL problems. We reproduce the method in Algorithm 3, noting that it also adopts an actor-critic strategy.

Algorithm 3: PPO, Actor-Critic Style

```
1 for iteration = 1, 2, ... do
2     for actor = 1, 2, ..., $N$ do
3         Run policy $\pi_{\theta_{old}}$ in environment for $T$ timesteps ;
4         Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ ;
5     end
6     Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$ ;
7     $\theta_{old} \leftarrow \theta$ ;
8 end
```

In [19], DeepMind researchers described the Distributed PPO (DPPO), an improved algorithm able to learn locomotion behaviors in rich environments such as those presented in Figure 3.4.

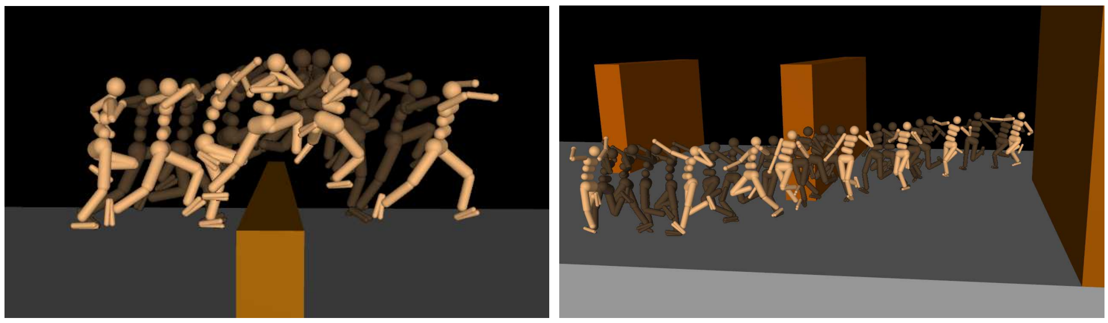

Figure 3.4: The simulated DPPO agent was able to learn to leap over obstacles and to avoid obstacles [19].

3.2.3 Soft Actor-Critic (SAC)

Recently, a new off-policy actor-critic deep RL algorithm called SAC (Soft Actor-Critic) [17] has been proposed. This algorithm tackles the two major issues of applying DRL to real-world robotics directly: high sample complexity, which demands a longer training time and hyper-parameter tuning. Instead of maximizing only the expected reward, SAC also attempts to maximize the policy's entropy, which promotes policies that act as random as possible while also achieving a high reward. This has a series of desirable consequences: it improves its sample efficiency and makes the policy more robust to hyper-parameters and environment changes. All of these characteristics make SAC a viable algorithm for learning robotic skills, and the authors have already demonstrated this by successfully applying SAC to real-world tasks [17], as seen in Figure 3.5.

Given those advantages and after initial tests, SAC was our algorithm of choice for discovering policies for the walking gait in the Marta robot.

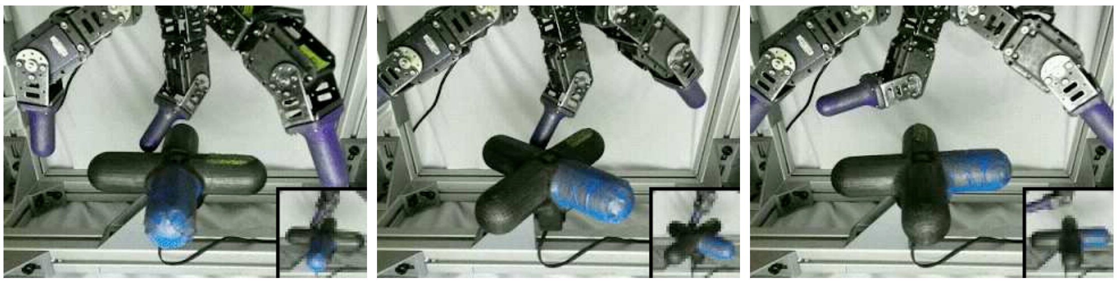
Figure 3.5: SAC was able to train a policy to learn how to operate the Dynamixel Claw to rotate a valve [17].

3.3 Deep Reinforcement Learning and Locomotion

Despite all the recent successes in DRL, most control tasks are still being learned in simulation only. Such simulations usually exhibit simplifications in the physical or robot model [19][41][33].

Among the control tasks, the locomotion task, in particular, still presents a challenge for realistic robot models and simulation parameters. However, the field is progressing fast towards learning in the real world, with recent works applying DRL in varying degrees to learn gaits for real quadrupedal robots [17][54][42][21][9] and real bipedal (non-humanoid) robots [62]. However, learned policies applied in the real world are often brittle and may fail to generalize well to unaccounted changes in the environment. Learning walking gaits for real humanoid robots remains unsolved and currently represents the next frontier for DRL.

3.3.1 Locomotion in simulation

Recent DRL methods are demonstrating to be a very suitable approach for locomotion tasks in simulation. In the works from [41] and [33], DRL was used to train a simulated humanoid in navigation tasks using high-dimensional input such as an RGB camera. This tasks could be learned using limited amount of prior knowledge, usually in the form of motion capture trajectories. An example result can be seen in Figure 3.6.

In [55], the authors demonstrate that motion capture can be avoided, and instead, the model can rely on a simpler walking target trajectory. By using this trajectory as an initial guide force, they propose a 3-stage curriculum to train DRL policies. During one of the stages of the curriculum, the guide forces are gradually reduced. By using this proposed curriculum and Proximal Policy Optimization [48], the agent could learn complex walking policies for terrain with hurdles, stairs, and gaps. Some of these tasks can be seen in Figure 3.7.

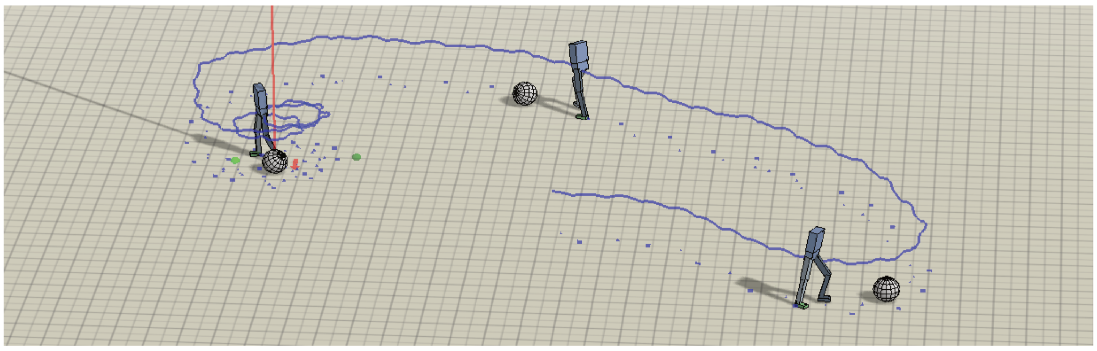

Figure 3.6: “Soccer dribbling” task learned with a mixed actor-critic approach [41].

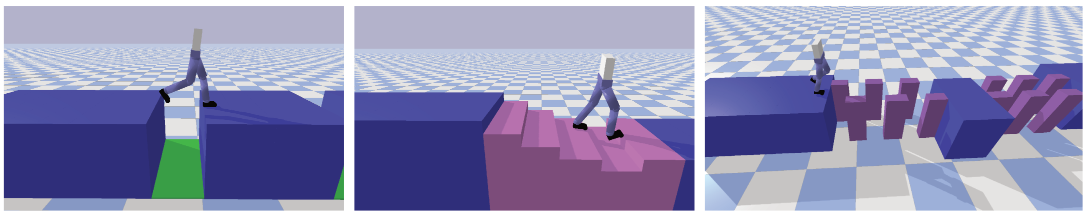

Figure 3.7: Walking tasks learned using Curriculum Learning and PPO [55].

In [63], Xi et al. show that reinforcement learning alone, without curriculum learning, is insufficient for solving the stepping stones task. Hence, they employ four different curriculum learning strategies with PPO to train three different task agents. The agents are simulated using PyBullet, the Humanoid has 21 joints and is torque-controlled.

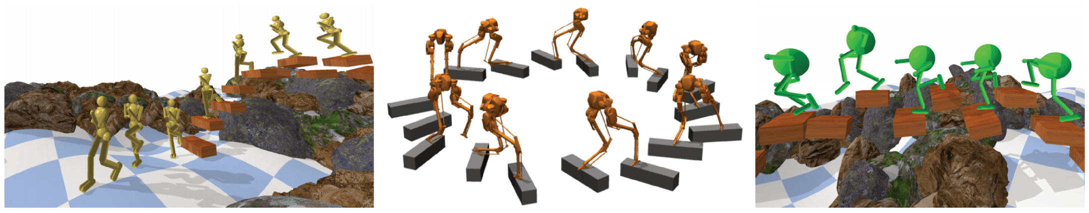

Figure 3.8: Curriculum Adaptive DRL for different agents architectures proposed by [63].

3.3.2 Quadrupedal locomotion in the real world

Haarnoja et al.[17] described one of the first applications of DRL algorithms to learn walking gaits in real robots. They applied the SAC algorithm in the Minitaur robot, a small-scale quadruped with eight actuators (2 in each leg) [28]. Our formulation discussed in Chapter 5 was heavily inspired by this work. In the observation space, some similarities include the use of motor angles, angular velocities of the base and the exclusion of *yaw* from the observation. In the reward function, the authors included penalties for preventing odd angles and the extension of legs, two problems that we also encountered in our experiments (Section 6.4) and had to counteract (Section 5.4). Their robot training pipeline achieved a remarkable result, being able to learn stable walking gaits in the real Minitaur robot in approximately 2 hours of real-world training time. Furthermore, the policy was able to walk on varied terrains and obstacles even though it was trained only on flat terrain. This robustness was mostly attributed to the entropy maximization in SAC. Photos of the learned gait can be seen in Figure 3.9.

In [9] a combination of model-based control and RL is able to learn robust policies for the Unitree Laikago real quadruped robot. Their approach splits the task into a high-level and low-level controllers and use a DQN-like algorithm to train the high-level policy. Their deployment to the real robot did not require domain randomization.

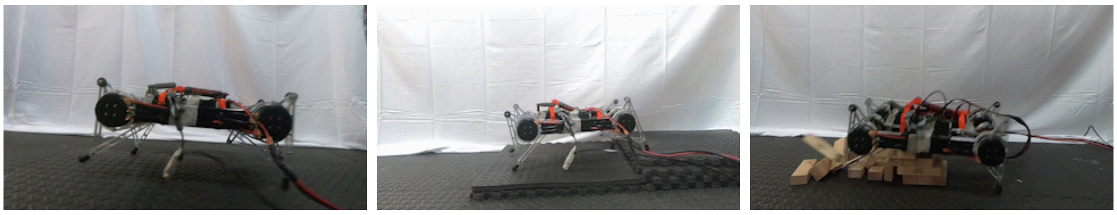
Figure 3.9: DRL policy gait learned for the Minitaur quadrupedal robot [17].

3.3.3 Bipedal locomotion in the real world

Zhaoming et al.[61] proposed a mixed method that allows an iterative design process for locomotion skills. Their work tackles directly the difficulties related to shaping and predictability of reward function changes. In this proposed method, reference motions and aggregated state-action pairs are used in combining Reinforcement Learning and Imitation Learning. This data collection method is referred to in the paper as Deterministic Action Stochastic State (DASS). One of the advantages of this approach is that it allows the iterative design of reward functions while limiting the previous iteration deviation. The method was evaluated in the Cassie bipedal robot, a robot designed and built by Agility Robotics that stands approximately 1 meter tall. Despite using a reference motion and imperfections in the walking gait, this technique demonstrates a practical method for leveraging DRL ideas to develop a useful gait. Once more, photos of the learned gait can be seen in Figure 3.10.

3.4 Summary

In this chapter, we show how DRL-based methods have evolved and are now employed in various tasks, from action video games to robotics navigation. We discussed the main approaches followed, showing that real robot models are not yet tackled for the model-free learning algorithms despite the quality of the methods. Indeed, agents like those presented in [19] and the environment where they were simulated are far from considering the complexity of joints' dynamics as other aspects of the agent's model. We also showed how DRL worked with real Cassie and Minitaur robots. However, no model-free DRL algorithm was applied to a humanoid robot in more realistic simulations as far as we are concerned. In this scenario, we aim at learning a DRL based policy to control the gait of our humanoid robot Marta.

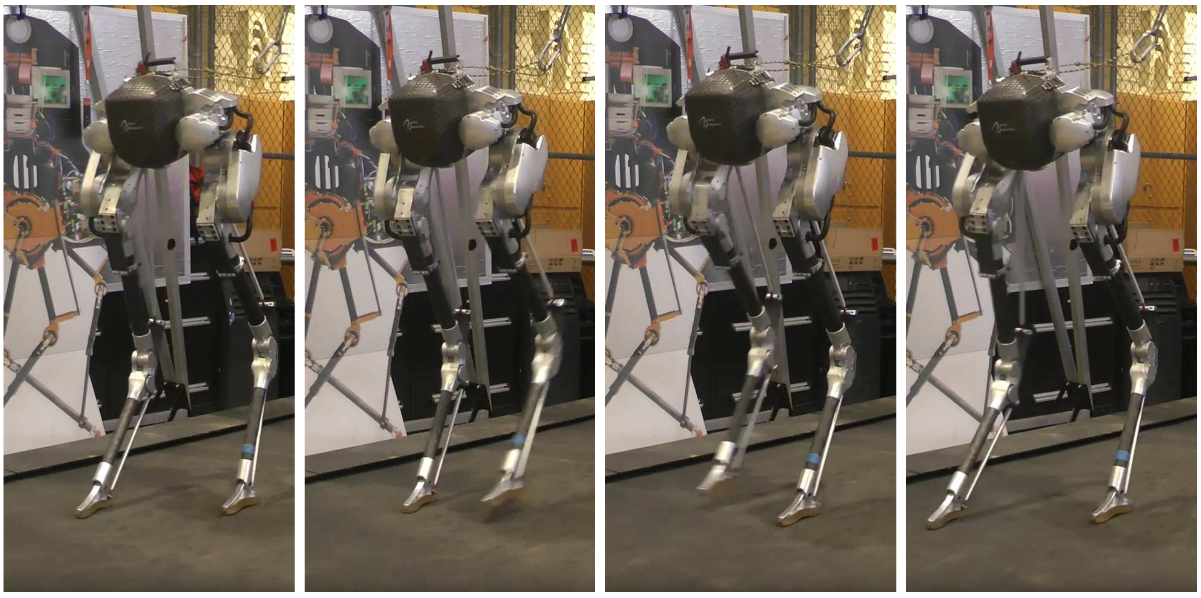

Figure 3.10: By mixing RL-based and Imitation Learning, an iterative-design approach could successfully develop locomotion skills for the Cassie robot [61] .

Chapter 4

Materials and Methods

As discussed in Chapter 3, the number of reinforcement learning applications has increased rapidly in the last few years. This project aims to contribute to this growth by both performing experiments using the most successful recent techniques while also investigating novel approaches on a new humanoid robot model developed by our team.

This project's source code can be found on our laboratory's Github repository. [1]

4.1 Materials

Fortunately, academia and industry have been working closely on such problems, collaborating considerably in the rapid development on RL. This collaboration fostered the invention of multiple robust and accessible virtual environments designed to test, validate, and compare RL methods. One of these environments, OpenAI Gym [5], became the de-facto standard interface for Reinforcement Learning, and its tasks are commonly used as benchmarks in recent papers [48]. All the environments and tasks proposed use OpenAI Gym as its interface, facilitating reproducibility, and comparing algorithms.

4.1.1 Simulated Environment

The chosen simulator in this project was V-REP [11]. During this project, V-REP was discontinued and succeeded by CoppeliaSim. CoppeliaSim and V-REP are largely compatible with each other, and, currently, the major differences are performance and improved interfaces. For this work, CoppeliaSim can be seen as a major version enhancement over V-REP. This simulator is exceptionally accessible and user-friendly with an intuitive scene editor, flexible remote API, and multiple alternatives for the physical back-end. Particularly, CoppeliaSim's dynamics module currently supports Vortex Dynamics, a physics engine that produces high fidelity physics simulations. It is worth noting that, currently, most DRL papers use a class of fast simulators that lack the intuitiveness of CoppeliaSim when creating realistic control tasks and robotic models. Three of this simulators are MuJoCo (Multi-Joint dynamics with Contact) [56], PyBullet [8] and Roboschool [48], now discontinued. MuJoCo has seen a recent rapid growth in adoption [39] due to its

[1]Github repository: `https://github.com/larocs/msc_yuri_soares`

fast and accurate simulation engine [12]. PyBullet and Roboschool share the same underlying physical engine (Bullet), behave similarly in most environments and, differently from MuJoCo, do not require a license to run. We eventually came in contact with all these three additional simulators during our work as they are used by different research groups and are interchangeable to some degree. These simulators were also used when the sporadic requirement of a faster training phase for testing purposes made CoppeliaSim less suitable.

4.1.2 Physical Model

In this section, we will describe the physical model that was used to generate our simulated model.

The model technical specifications are described along the following list:

- 25 DOFs (Dynamixel MX-64T/AX-12)
- Height: 1,10m
- Sensors: 1 Logitech C920 HD Pro webcam and a 9 axis IMU UM7-LT
- CPU: Intel Nuc Mini-PC 2.4GHz quad core 4GB, 120GB SSD (wifi, Bluetooth and 6 USB ports)
- Battery: 2 Lipo 4S 14.8V - 4400mAh

The model also incorporates several innovative ideas that could influence the quality of the dynamic learned process, such as: a compact foot size for reduced surface contact; a spheroidal joint in the waist with an actuation point, and an extra DOF in the foot. The real robot and the simulated model are shown in Figure 4.1.

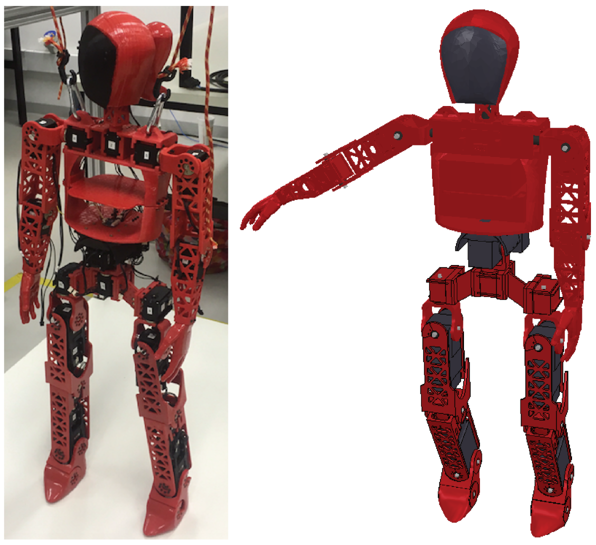

Figure 4.1: The real Marta robot and the simulated CoppeliaSim/V-REP model.

4.1.3 Programming languages and software libraries

Most experiments were conducted using the *Python3* programming language. A combination of multiple available libraries, tools and simulator bindings contributed to this choice:

- **NumPy** [58]: Performant n-dimensional array and numerical computation library.
- **OpenAI Gym** [5]: Common interface for RL environments.
- **TensorFlow** [1]: Software library for dataflow and differentiable programming, widely used for machine learning research and applications.
- **CoppeliaSim** [11]: The chosen robot simulator.
- **PyRep** [23]: Toolkit for efficient interfacing between CoppeliaSim and *Python3*.
- **Stable Baselines** [20]: Set of implementations of RL algorithms based on OpenAI Baselines [10].

Using pre-existing libraries also aids in reproducibility and easier comparison with other methods. We also developed a framework called `vrep_env` for wrapping the low-level CoppeliaSim API by using Gym environment semantics. A diagram showing how CoppeliaSim, `vrep_env` and other components interoperate is shown at Figure 4.2.

4.1.4 Algorithms

To test our hypothesis, we applied two algorithms that are close to the state of the art on reliability and sample efficiency among policy-learning algorithms [2]: PPO [48] and SAC [17]. However, after initial tests, the sample-efficient SAC algorithm, when empirically compared to PPO, reduced training time considerably and allowed for faster iteration of experiments. In fact, in our experiments, PPO has never been able to learn a control policy for Marta control gait. Therefore, we choose to focus our efforts in experimenting with multiple MDP formulations and reward functions using SAC only.

4.1.5 SAC hyper-parameters

The hyper-parameters used in the SAC algorithm can be seen in Table 4.1.

Hyper-parameter	Value
discount factor (γ)	0.99
learning rate (α)	0.0003
replay buffer size	50000
minibatch size	64
soft update coefficient (τ)	0.005

Table 4.1: SAC hyper-parameters.

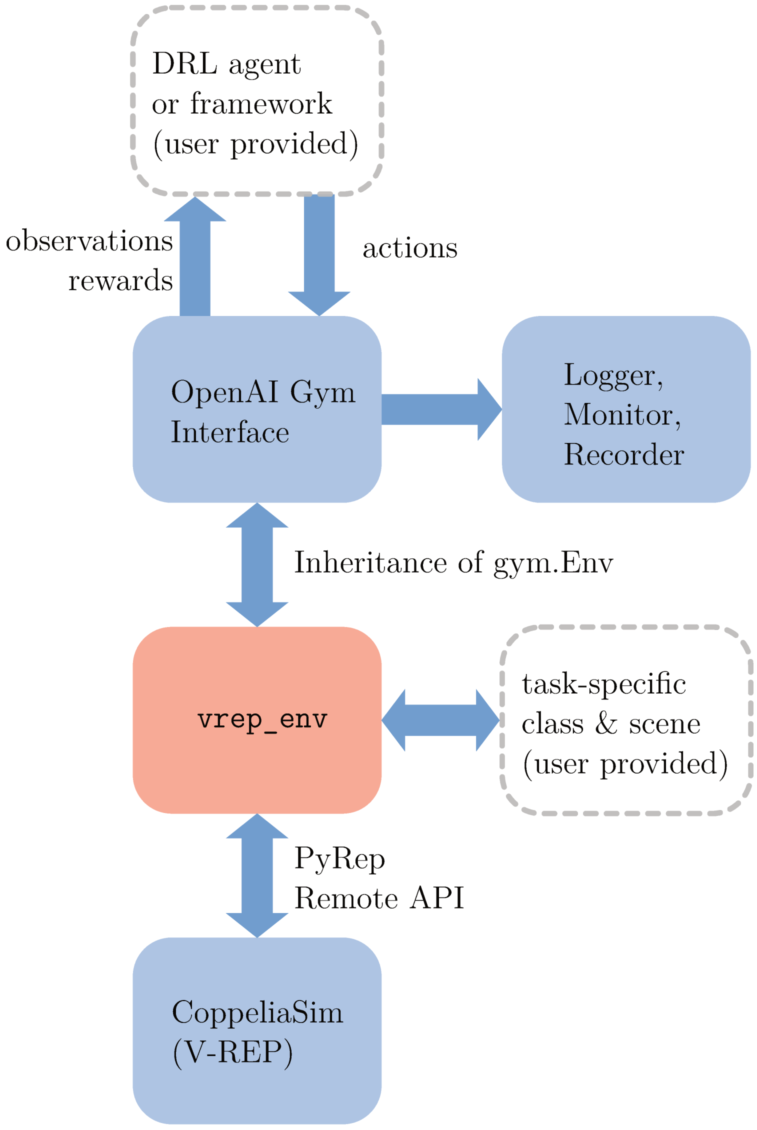

Figure 4.2: `vrep_env` and its directly related components.

This choice of parameters was taken from the original authors [17] and the chosen algorithm implementation [20]. Given the results obtained, the parameters did no require further tuning.

4.1.6 Artificial Neural Network Architecture

All neural networks used in SAC for the present work were multilayer perceptron networks. These networks had 3 layers of 128 neurons each. These 3 layers do not include input and output layers, which depends on the observation and action spaces.

4.2 Methods

4.2.1 Quantitative evaluation criteria

In order to compare the learned gaits, we use a set of metrics that are aggregated throughout each episode. These following metrics are collected once at the end of each episode and evaluate the overall episodic performance:

- Distance walked in x (bigger is better).
- Distance walked in y (smaller absolute value is better).
- Whether the agent reached 50m and was terminated (true is better).

We also collected measurements made in each time-step of an episode to compare different gaits in their intermediary steps:

- Simulated energy use (smaller is better).
- Feet z position (bigger is better, capped by an empirical parameter).
- Variations in *pitch* (smaller absolute value is better).
- Variations in *roll* (smaller absolute value is better).

All of these results are averaged across 40 different runs for each policy compared in Section 6.6.

4.2.2 Qualitative evaluation criteria

As we discussed in Section 2.2, it is hard to evaluate the quality of a gait learned by optimization techniques. Even by devising metrics that take into account multiple objective factors such as energy efficiency, stability, balance, and measurements, the ultimate sieve will almost always reside on whether the gait looks "natural" or not. For this qualitative evaluation, we also rendered videos of the walking gaits in motion along with relevant plotting, which allow the detection of possible issues in the learned gait. While we admit that this criterion is a highly subjective one, it tends to be consensual among different individuals since walking is a natural skill possessed by most humans. This situation is also alleviated when considering the similarities between the Marta robot and the human anatomy, making it easier to envision a natural gait for the model. It would be possible to take an approach similar to [7] with a human evaluator, but our goal is to attempt a more automated approach.

As we shall see in Chapter 6, the agent was deceptively clever on specification gaming, learning policies that could walk on a single leg, sideways, backward, among other arrangements to avoid penalty factors and exploit rewards in an unintended way.

4.2.3 Transfer Learning

In order to accelerate our reward design iteration process, transfer learning [46] was greatly used. As we shall see in Chapter 5, multiple different RL formulations are proposed, with each building up in complexity. Instead of training the policy from scratch each time, the learned parameters from simpler tasks could be reused to bootstrap learning in more complex ones. As discussed in Section 6.7 this process did not substantially change the obtained policy, only accelerated the training phase.

Chapter 5

Locomotion as a Reinforced Learning problem

As discussed in Section 2.4, MDPs provide a very general mathematical framework for modeling control problems [3] and RL ones. In the particular case of locomotion, RL is already demonstrating success in both simulated and real tasks [17].

In order to apply RL algorithms in our problem, we first need to model the Marta locomotion task as a RL problem. Using the notation from Section 2.3, we must define our observation space $\mathcal{O}$, action space $\mathcal{A}$ and reward function r. The state-space $\mathcal{S}$ and transition probability P are implicit from the physics simulation engine and only require minor additional configuration, e.g., setting the gravity constant and time step dt.

As we will show throughout the text, practical RL also tends to define additional conventions and signals. One of the most important ones is the “done” signal, which would technically belong to $\mathcal{O}$, but tends to influence the choice of r considerably.

5.1 Observation space

To choose an observation space, we should take into account which information is necessary for the agent to learn a walking gait and how this information should be represented. Redundant or extra information should be avoided for the sake of efficiency in the learning process. But its presence does not cause any major problems, as neural network policies can learn to ignore any input if advantageous. Therefore, when in doubt regarding an input's relevance, it is better to err on the side of caution and include it anyway, at the smaller cost of efficiency.

5.1.1 Proprioception observation inputs

As we can see in Figure 5.1, the robot has 25 revolute joints: 5 central and 10+10 symmetrical joints. Of all these joints, only *HeadPitch* and *NeckYaw* are not used in the observation vector. All the remaining 23 joints contribute to the observation space with 2 scalars per joint: one for the normalized joint angular position and the other for the joint angular velocity.

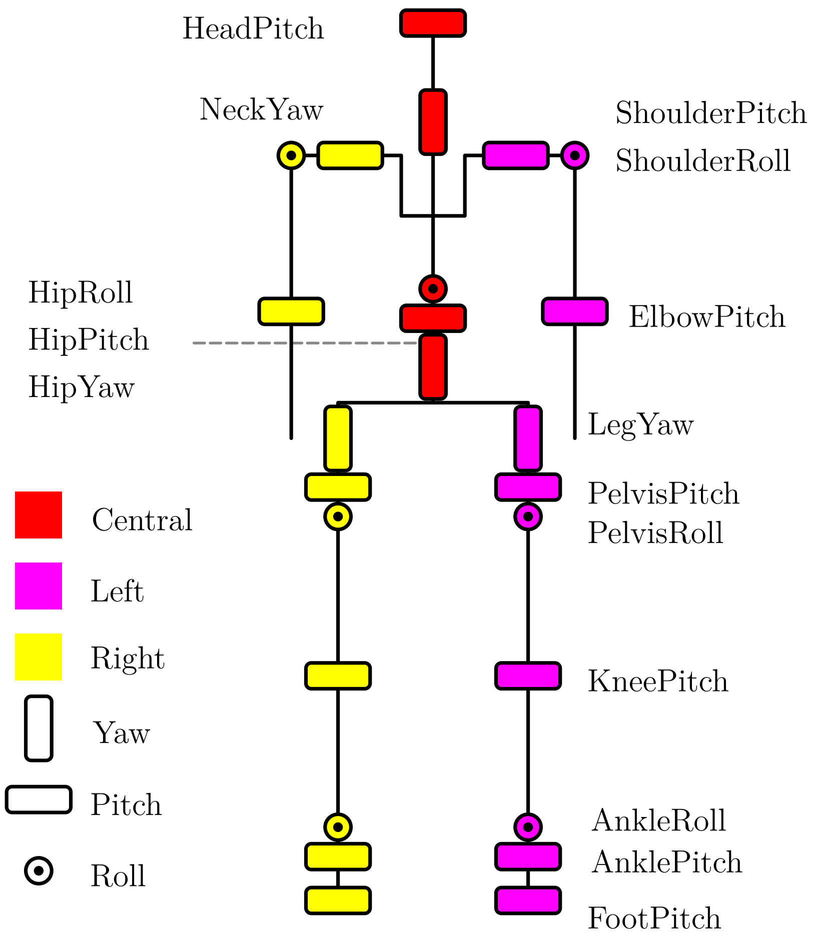

Figure 5.1: Positions and orientations of Marta's joints, highlighting symmetrical versus central joints.

Bounded coordinates It is preferred not to include values in the observation vector that could increase or decrease without bounds during the walking cycle. Due to the nature of how neural networks compute functions, this kind of values may help in the initial frames but become meaningless as the robot moves forward and away from the initial position. The absolute x and y coordinates of the robot, for example, should be irrelevant to compute the actions during the walking cycle, because the robot walks in the xy plane and these coordinates can assume any value. The z coordinate, on the other hand, is bounded and could provide relevant information during the cycle. A similar argument could be made to discard the *yaw*, but include the *roll* and *pitch* rotation axes [17]. With that in mind, we include these 3 additional scalars in our observation vector: z, *roll*, and *pitch*, all of them from the central robot link.

Relative linear velocities Using the same ideas from the previous paragraph, the absolute linear velocity along the x and y axes would have different meanings for the robot because the agent is, in principle, free to rotate in that plane. To correct this, we consider the velocity from the robot's planar reference. To compute this, we simply rotate by the yaw:

$$v_{pov} = \begin{bmatrix} cos(-yaw) & -sin(-yaw) & 0 \\ sin(-yaw) & cos(-yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{5.1}$$

Adding v_{pov} to the observation vector yields 3 additional scalars to the observation space.

Adding all together These $(23 \times 2) + 3 + 3 = 52$ scalars set represents our "core" observation inputs and are included in all experiments.

5.1.2 Additional observation inputs

In addition to the "core" observations mentioned, we also defined additional observational inputs for different experiments and depending on each strategy for learning walking gaits.

Foot z information We experimented including the information of each foot into the observation, as it could potentially benefit the agent's decision-making. Using a similar argument from Section 5.1.1, only the z axis component of each foot's position and velocity were included, adding $2 \times 2 = 4$ new inputs.

Floor contact With similar reasoning, we choose to experiment including floor contact information. Differently to the previously mentioned observation inputs, floor contact is represented as a boolean value. Since all inputs are mapped to the interval $[-1.0, +1.0]$, these boolean values are simply mapped to $\{-1, +1\}$. As the contact is measured at the front and back of each foot, these inputs also add $2 \times 2 = 4$ values to our observation input.

5.1.3 Sensorial observation inputs

So far, all the inputs discussed were related to the robot's proprioception. However, healthy humans rely on visual cues for walking [64]. In order to be able to walk in a fixed direction, our agent needs at least some information from the environment. Without this sensory information, any deviation from the intended walking direction could never be corrected and would likely accumulate over time. For that reason, we choose to also experiment with additional sensory information in the observation vector.

Robot target position One of the simplest possible sensory information we could include the intended direction of movement. To model this, we pick a point p_{target} in the xy plane located at a distance d_{max} away and straight ahead of the robot initial position p_0. The exact value of d_{max} is chosen so the robot can barely reach this point in the allotted time, even at full theoretical speed. Then we compute θ_{target}, which is the angle difference between the robot's yaw and the angle to the target. An example can be seen in Figure 5.2. To avoid angle discontinuities in the neural network input, we do not include θ directly in the observation, but instead $[sin(\theta_{target}), cos(\theta_{target})]$.

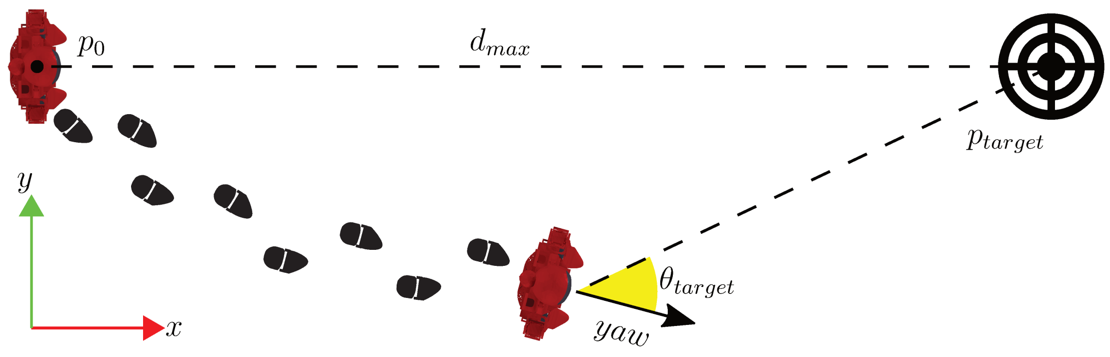

Figure 5.2: Example of θ_{target} after a possible walking trajectory. p_0 is usually chosen at $(0, 0, 0)$, but this is not a requirement. This diagram is not in scale, and d_{max} is usually much larger.

Foot target position As we will see in Chapter 6, providing only the distant target position sometimes led to unnatural gaits, with the agent learning to take very small steps, essentially trying to walk with both legs simultaneously. In order to alleviate this problem, we also experimented with foot target positions, inspired by ideas from [63]. Foot target positions work somewhat like stepping stones: at any given moment, there are two target positions, one for each foot that is constantly advancing when a foot reaches its respective stone. The observation angles θ_{left} and θ_{right} are calculated similarly as θ_{target} in the previous paragraph, but considering the (x, y) position of each foot:

$$\begin{aligned} \Delta_{LR} &= stone_{LR} - foot_{LR} \\ \theta_{LR} &= arctan(\Delta_{LR}.y, \Delta_{LR}.x) - yaw \end{aligned} \tag{5.2}$$

Where LR subscripts indicate that a variable is duplicated for both $left$ and $right$, $stone_{LR}$ are the 3-dimensional position of both next two stones and $foot_{LR}$ the positions of the feet. Besides $sin(\theta_{LR})$ and $cos(\theta_{LR})$, we also include $\|\Delta_{LR}\|$ and two booleans b_{LR} that indicate which leg is expected to move next. In total, $2 \times (2 + 1 + 1) = 8$ new inputs are added when foot target position is used. Two renderings of this setup can be seen in Figure 5.3.

5.2 Action space

Similarly to the observation space, we must define our action space with all controls that would be required for a walking gait. In an analogous manner, including joints that are not relevant to walking has a marginal cost, so including all 25 joints would be a sensible choice. However, both *HeadPitch* and *NeckYaw* are once again discarded as their effect on locomotion is negligible.

As we will see in the Chapter 6, sometimes the agent heavily relied on the six lower joints (*AnkleRoll*, *AnklePitch* and *FootPitch* of both legs) so we also conducted learning experiments with these 3 pairs of joints underactuated.

Another possible choice on the action interpretation would be to choose between position, velocity, and torque controls. Turns out that both velocity and torque control

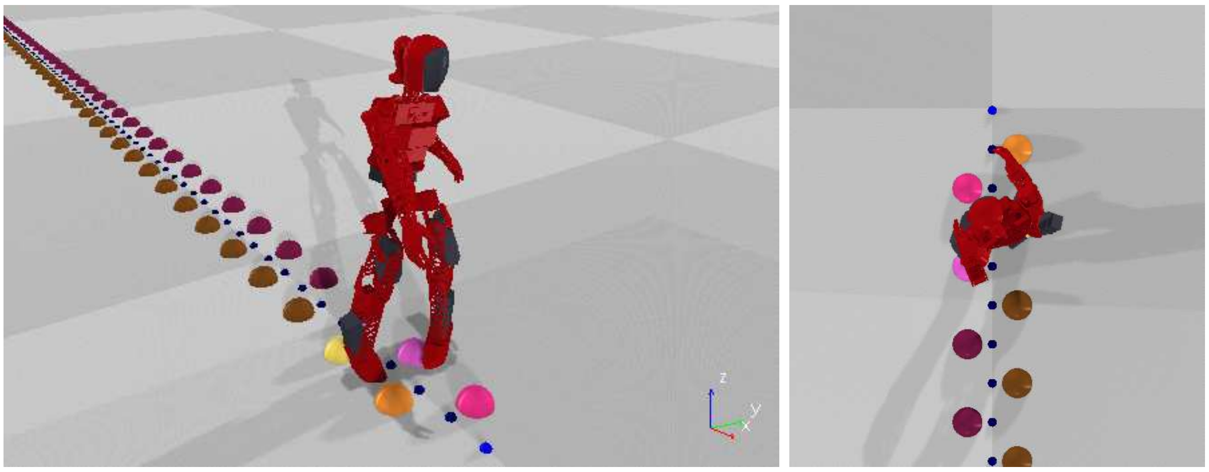

Figure 5.3: Rendering of the simulated environment using foot target position as "stepping stones". Left image is a perspective view and right is a top-view orthogonal one. Past, current, and next stones can be seen for each foot.

yield similar results. Position control turned out to be a bit more tricky to work in CoppeliaSim as it would require setting additional PID parameters for each joint. Position control would, in turn, cause the agent to learn how to indirectly control the PID controller. Therefore, by setting appropriate PID parameters, position control, should be, in theory, similar to velocity control in simulation.

As previously mentioned, all 25 joints in Marta are revolute joints (its spherical joint is treated as 3 revolute joints) and all have upper and lower bounds of angular freedom, hence unable to perform full rotations. In addition to position limits, all joints are also simulated with velocity and torque limits, which the agent must learn to manage.

5.3 Episode termination

Defining the criteria for episode termination has many different consequences. Although it may seem trivial, this decision is almost as important as the choice of r itself. In fact, both the termination criteria and the reward function should be constructed in unison [40]. Doing otherwise could lead to an unnecessarily inefficient learning or even change the optimal policy.

In theory, the most straightforward way would be to always terminate an episode after reaching a predefined number of frames. This way is, in fact, the easiest way to theoretically examine the incentives: the policy would learn to maximize $J(\pi)$ in the allotted frames and we would be able to define the reward function r with greater freedom.

However, terminating the episode early in the case of hard-to-recover states (such as falling) is desirable since ending these episodes earlier can reduce the total time required for training. Although it may seem like a harmless modification this kind of early termination could introduce a dilemma for our agent. Suppose that, in our environment, we are training a policy to find a low cost regarding some metrics. In our case, this could be the energy expenditure during the walking cycle, for example. We may be tempted to introduce in r a negative term that computes this cost. If this term is not appropriately scaled, our agent could, instead of learning how to minimize this cost, “prefer” to learn how to terminate the episode as soon as possible (which tends to be much easier). The incentive has shifted towards having shorter episodes to amass less negative rewards.

This kind of counter-intuitive optimization logic is also observed in evolutionary computation [30] and is often called “specification gaming”. We will see more ingenious examples in Chapter 6.

Fortunately, in this case, this kind of gaming is relatively easy to counteract: we must simply guarantee that r evaluates to strictly positive values. In practice, this is done by always adding a positive constant term to the reward, usually called *keep_alive*. To be able to cancel any possible negative values, this term should have the absolute value of the largest negative number that the original r could assume.

Nonetheless, as we shall see in the next subsection, introducing a large positive term in our reward can introduce its problems that require additional adjustments.

5.3.1 Potential-based rewards and practical considerations

As demonstrated in [37], rewards that are not potential-based could change the optimal policy. Because of this result, all positive reward factors were carefully chosen to comply with potential functions. Even when accounting for potential-based rewards, some practical considerations are worth addressing: Negative reward factors (penalties) do not necessarily share this restriction because they usually cannot be exploited by the agent in infinite positive loops and negative loops should not be a concern in this task as they are naturally disincentivized. However, to comply with potential functions, our terms must span negative values, demanding an even higher value for the *keep_alive* constant. To make matters worse, the *keep_alive* term itself, by its own nature, is not potential-based.

A workaround is to reintroduce the upper limit on the number of frames mentioned in the beginning of this section, even when also using early termination. It is also worth noting that all terms in the reward function (including *keep_alive*) should be scaled appropriately. A very large value for *keep_alive* could lead to a lethargic policy: one that avoids “dying”, but does not accomplish anything else.

With all this being said, even though using early termination and *keep_alive* might seem not worth the additional complexity, analogous constant factors are used in similar locomotion tasks from other environments [8][5].

5.4 Reward function

In this section, we discuss the main reward terms experimented with Marta, as well as the motivation behind each of them. To ease the discussion, we will present the reward terms as if negative components were not a problem, assuming that *keep_alive* was scaled properly as described in Section 5.3.

As we saw in Section 2.3, r must evaluate to a single number. It is notoriously hard to evaluate a complex task such as walking and reshape it as a single scalar. Nonetheless, we attempted to construct a reward function by weighting multiple derived data from the model that should be proxies for a good gait.

The general form of our reward function is a weighted linear combination of multiple reward signals:

$$r(s_t) = keep\_alive + \sum_i w_i \rho_i(s_t) \tag{5.3}$$

where each $\rho_i : \mathcal{S} \mapsto \mathbb{R}$ is a function that computes a scalar from the current state s_t and w_i is how much this scalar contributes to the final reward. In the next subsections, we will elaborate more on each ρ_i and its intended effects.

5.4.1 Velocity to target field

As we already discussed, [37], provides the rationale for choosing reward terms that are potential-based.

As a small intuitive example for that proof, let us suppose that we define a simple r that incurs a small positive reward when the agent moves forward and no other terms. While our hopes may be that the robot learns to walk forward as far as possible, it is much more likely that it will learn instead how to make some kind of small cyclic movements and, essentially, stay in the same place. The reason this happens is that we do not punish the backward movement, only reward the forward one.

In the case of the walking gait for Marta, that is exactly what we did: we simulated a physical field that emanates from the p_{target} and use those field lines to project the robot velocities into. First, let's define p_{dif}:

$$p_{dif} = p_{target} - p_{xyz} \tag{5.4}$$

Then, we can define this projected velocity towards the target as $\rho_{progress}$:

$$\rho_{progress}(s_t) = \rho_{progress}(p_{xyz}, v_{xyz}) = v_{xyz} \cdot \hat{p}_{dif} \tag{5.5}$$

The only info from s_t required by $\rho_{progress}$ are the robot's spatial position p_{xyz} and velocity v_{xyz} (p_{target} is not considered part of the state, but a constant defined for the environment). The final result is simply a scalar projection as illustrated in Figure 5.4.

5.4.2 Penalties

In our first experiments, $\rho_{progress}$ and $keep\_alive$ were the only reward terms in r. As we will see in Chapter 6, the algorithm with this simple r was able to learn a reasonable policy for simulation, but impractical for the real robot. Therefore, the process of adding penalties was an iterative one, with each new penalty added with the intent of adjusting perceived flaws in the gait and shifting the policy to a more desirable solution.

Knee extension penalty One of the first observed flaws on the learned gaits was that the agent initially tried to walk with the knees fully extended (similar to infants). To prevent that, a penalty term ρ_{knee} was added:

$$\rho_{knee}(s_t) = (\text{\# of knees extended more than } \beta_{knee})/2 \tag{5.6}$$

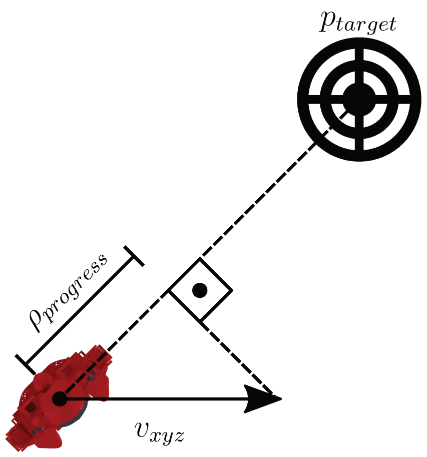

Figure 5.4: Example of $\rho_{progress}$ computation. Differently from θ_{target}, the *yaw* value is not used.

All joints angular position are normalised (1 being fully extended and 0 being fully contracted). So here, β_{knee} is a constant threshold to penalize and is usually set to around 80%. As the Marta model only has two legs, we can see that $\rho_{knee} : \mathcal{S} \mapsto \{0, 0.5, 1\}$. As this measures a undesirable effect, we simply choose $w_{knee} < 0$. In other words, no penalty is applied if no knees are not fully extended and the full penalty is applied if both knees are extended.

Body orientation penalties This set of penalties were introduced to prevent the robot tilting to odd angles. Examples include: walking sideways, backwards, leaning to one side or not facing the target.

In the case of the *roll* and *pitch*, the penalty is straightforward, as both the desirable angles are 0°:

$$\begin{aligned} \rho_{roll}(s_t) = \rho_{roll}(roll) &= |roll| \\ \rho_{pitch}(s_t) = \rho_{pitch}(pitch) &= |pitch| \end{aligned} \tag{5.7}$$

For the *yaw*, as usual, additional steps are needed:

$$\rho_{yaw}(s_t) = \rho_{yaw}(yaw) = |arctan(p_{dif}.y, p_{dif}.x) - (yaw + \pi)| \tag{5.8}$$

This makes sure that ρ_{yaw} outputs higher values when the robot faces away from the target and smaller otherwise.

Chapter 6

Results and discussion

In this chapter, we will discuss the experimental results and compare the walking gaits learned by the algorithm using different rewarding shapes strategies.

Graphs and plots In order to compare the different learned gaits, we generated graphs and plots from the multiple roll-outs of each policy. Due to the temporal and spatial nature of walking gaits, videos were also rendered for better visualization and evaluation of the gaits. In this chapter, locomotion plots provide different "trail" lines for better temporal perception. For these lines, we use the following color convention: red for the center of mass, magenta for the head, blue for the left foot and green for the right foot.

6.1 Policy derived from target field

In the first experiments, we choose to design a minimalist reward function, using only $\rho_{progress}$ as defined in Section 5.4.1. The policy derived from this reward function would be highly informative for the robot model and environment parameters, as it is a highly unbiased policy. Due to the simplicity of r and absence of penalties, the algorithm has a relatively high freedom to learn a policy with the sole objective of walking forward.

For these reasons, this policy was analyzed as a baseline and all of our attempts to refine the gait are compared against the gait learned by this policy, which will be referred as $\pi_{progress}$.

Policy results

The gait time-lapse with marked points can be seen in Figure 6.1 and the trajectory of the CoM can be seen in Figure 6.2. As we can see from the time-lapse, the learned policy gait is unnatural: the agent learned to take very low steps, bend backward and use the arms for balancing unusually. The robot also moves using what we called "micro-steps", that is, steps that are so low and short that it looks like the robot is moving with both legs simultaneously. However, this policy did manage to move the robot forward, frequently reaching the predefined limit of 50 m. Remember that r does not explicitly penalize any deviation on the y axis, since it mostly uses a single potential field. The reward weights can be seen in Table 6.1. Even without explicit shaping, the agent managed to keep the

y deviation low. The following reward engineering and policies attempt to improve on $\pi_{progress}$ by modifying r to avoid its shortcomings while preserving its ability to succeed in the task.

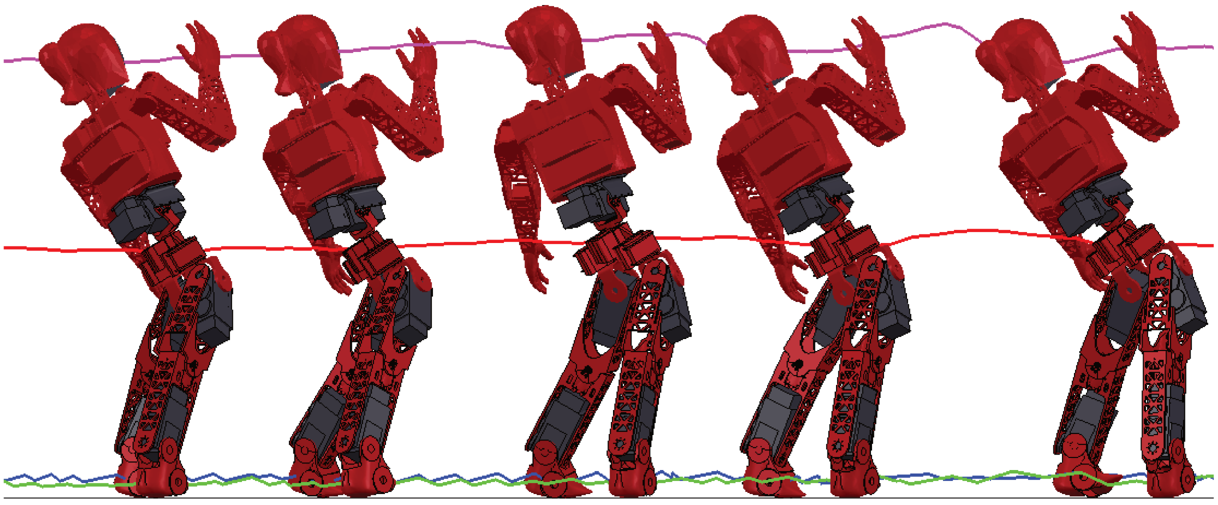

Figure 6.1: Time-lapse of the gait produced by $\pi_{progress}$.

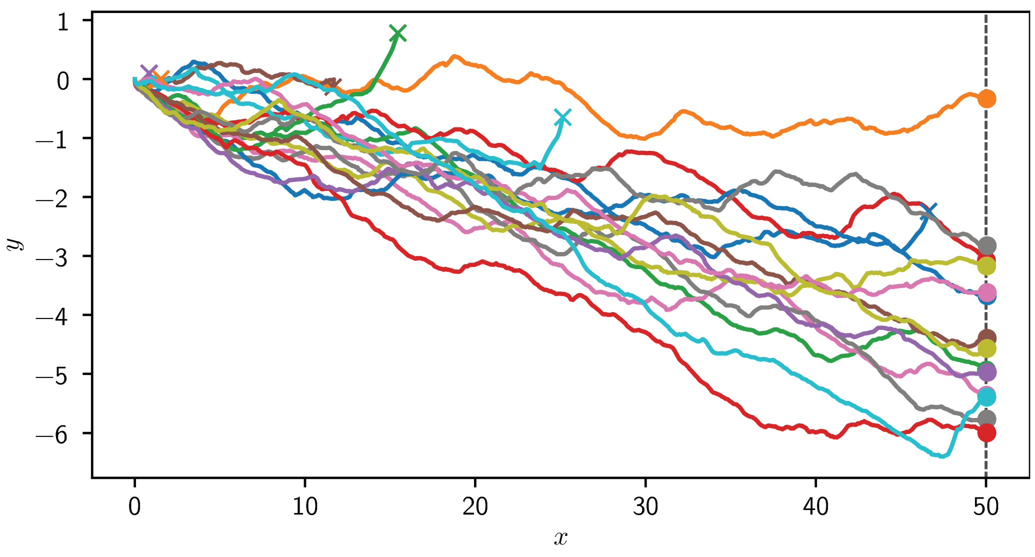

Figure 6.2: CoM X-Y position for 20 runs of $\pi_{progress}$ capped at 50 m.

Parameter	Value
keep_alive	+1.0
$w_{progress}$	+1.0

Table 6.1: Reward weights used to train $\pi_{progress}$.

6.2 Foot target position

As we discussed in Section 5.1, there are multiple possible approaches when defining our observational inputs. As an attempt to improve on $\pi_{progress}$, we experimented with the foot target position strategy discussed in Section 5.1.3. Using the notation introduced there, we will refer to the derived policy as π_{stone}.

As before, a gait time-lapse and trajectories can be seen in Figure 6.3 and Figure 6.4, respectively. The weights can be seen in Table 6.2. Using the stepping stones did improve the gait in some aspects. Most notably, the body no longer shifts backward and the divergence in the y axis is practically eliminated. Unfortunately, the "micro-steps" issue persists since stones provided only a potential reward but did not forbid missteps. However, the biggest drawback of assigning target positions for each foot is that we are now imbuing the gait development with parameters such as step dimensions instead of having them automatically discovered by the agent. This is error-prone, and it lacks adaptability. For these reasons, we choose to explore other more general reward strategies instead of relying on more opinionated reward shaping.

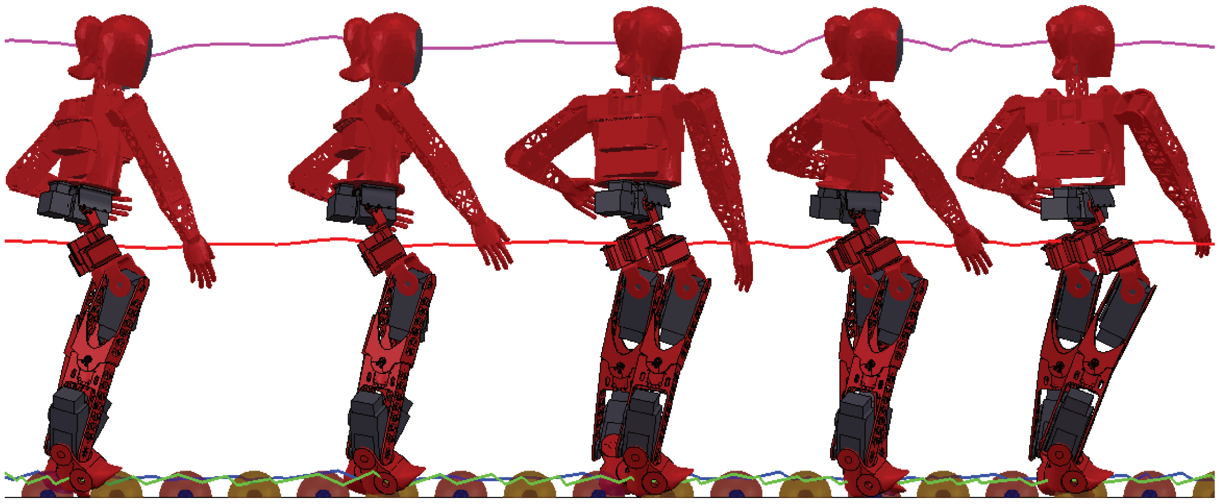

Figure 6.3: Time-lapse of the gait produced by π_{stone}. Stepping stones can also be seen.

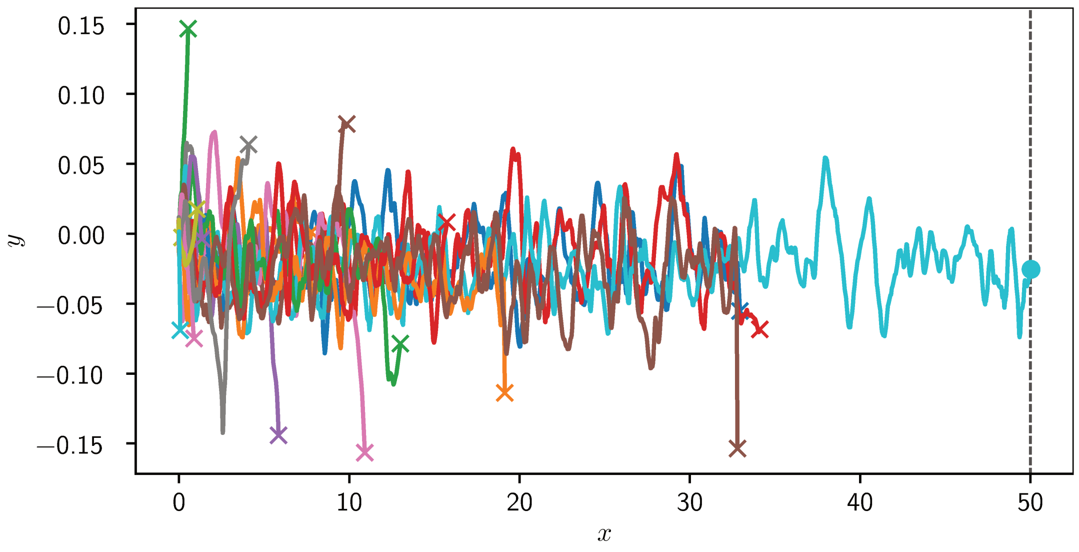

Figure 6.4: CoM X-Y position for 20 runs of π_{stone} capped at 50m.

Parameter	Value
keep_alive	+2.0
w_{stone}	+1.0

Table 6.2: Reward weights used to train π_{stone}.

6.3 Specification gaming

At this point, it is worth mentioning that many different reward strategies experimented with suffered from "specification gaming", when agents exploit the reward in an unintended way. E.g., in an attempt to mitigate the "micro-steps" issue discussed before, we experimented with a reward factor that encouraged the agent to increase the z difference of each foot. This made the agent learn how to briefly walk on one leg and keep the other raised (π_{1leg}) as can be seen in Figure 6.5. The weights can be seen in Table 6.3.

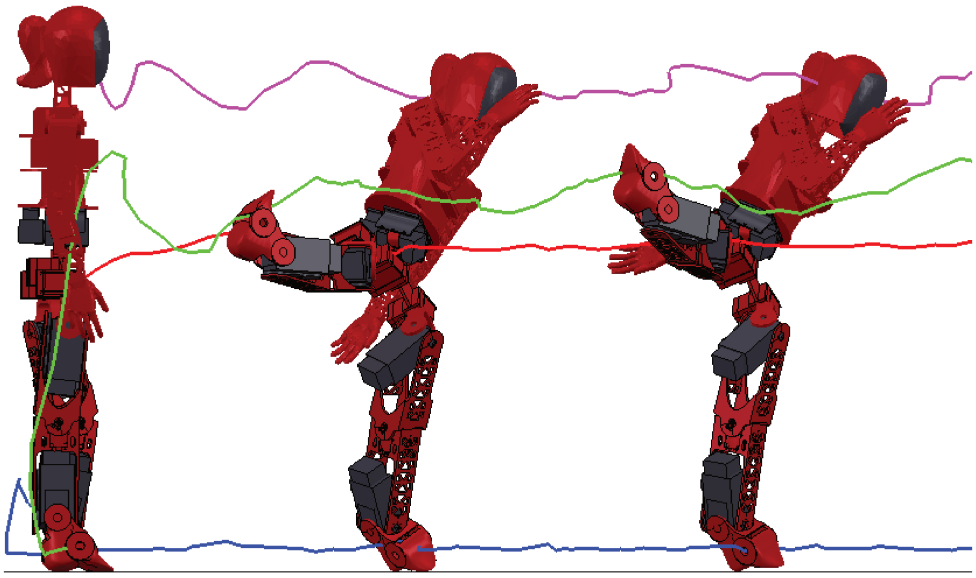

Figure 6.5: Specification gaming: when encouraged to increase feet height difference in order to improve the bipedal gait, the agent learns to walk on only one foot. This policy is referred as π_{1leg}.

Parameter	Value
keep_alive	+0.2
$w_{progress}$	+2.0
w_{footz}	+6.0

Table 6.3: Reward weights used to train π_{1leg}.

Another undesirable behavior that kept appearing on experiments was that the agent learned to walk with knees fully extended, so we introduced the knee extension penalty described in Section 5.4.2. To also prevent "micro-stepping" caused by the overuse of *FootPitch* this joint was underactuated as discussed in Section 5.2. The combination of the knee extension penalty and underactuation of *FootPitch* resulted in another specification gaming, in which the agent evade these adversities by learning to walk backward. These policy results (π_{back}) can be seen in Figure 6.6 and Figure 6.7. The weights can be seen in Table 6.4.

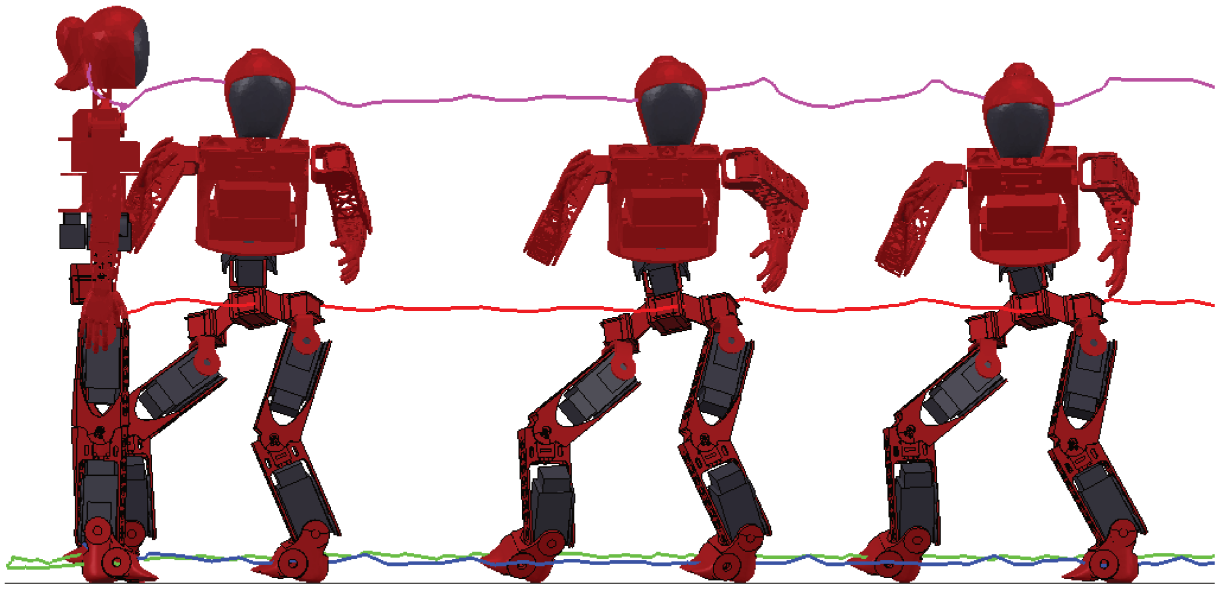

Figure 6.6: Specification gaming: introducing specific penalties unintentionally made the agent learn to walk backwards. This policy is referred as π_{back}.

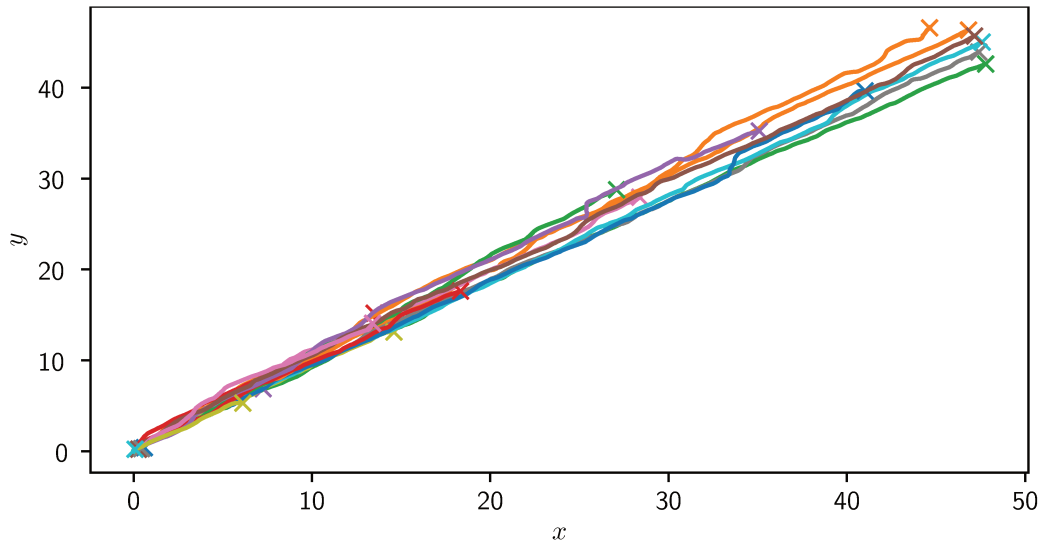

Figure 6.7: CoM X-Y position for 20 runs for π_{back}.

Parameter	Value
$keep\_alive$	$+5.0$
$w_{progress}$	$+2.0$
w_{knee}	-5.0
β_{knee}	$+0.75$

Table 6.4: Reward weights used to train π_{back}.

6.4 Body orientation penalties

In order to prevent the backward gait learned by the agent, the body orientation penalties described in Section 5.4.2 were added. We refer to this resulting policy as π_{ypr}, due to the inclusion of all 3 rotation axes in r. Once more, results can be seen in Figure 6.8 and Figure 6.9. The weights can be seen in Table 6.5. This policy improved on some aspects of previous ones, such as the knee extension, backward walking. Some new ones also appeared, such as the unnatural hip position. However, instead of overusing *FootPitch* the agent now misuses *AnkleRoll* and *AnklePitch* to keep walking using micro-steps.

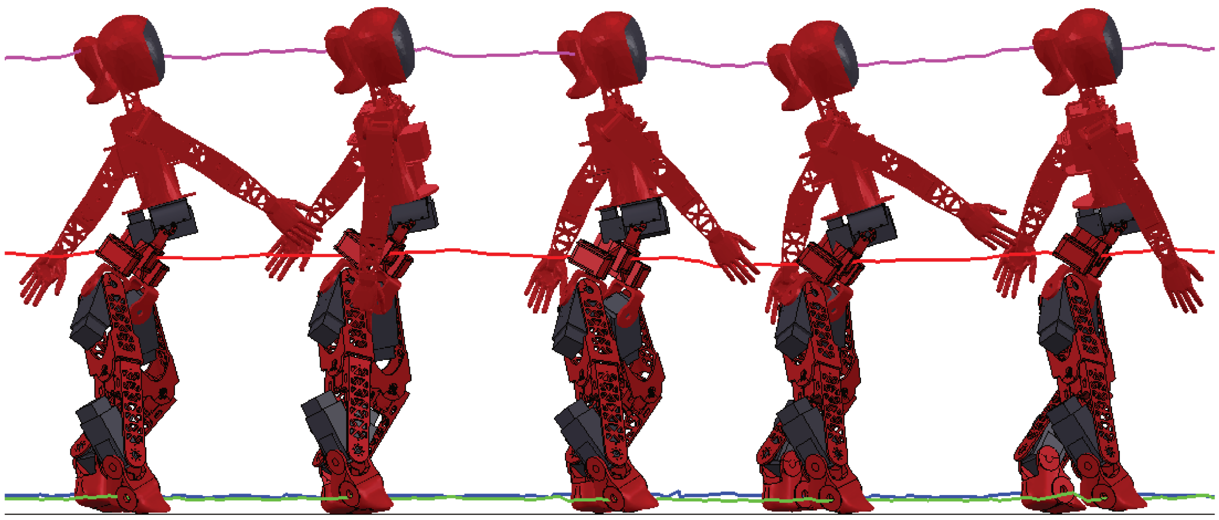

Figure 6.8: Time-lapse of the gait produced by π_{ypr}.

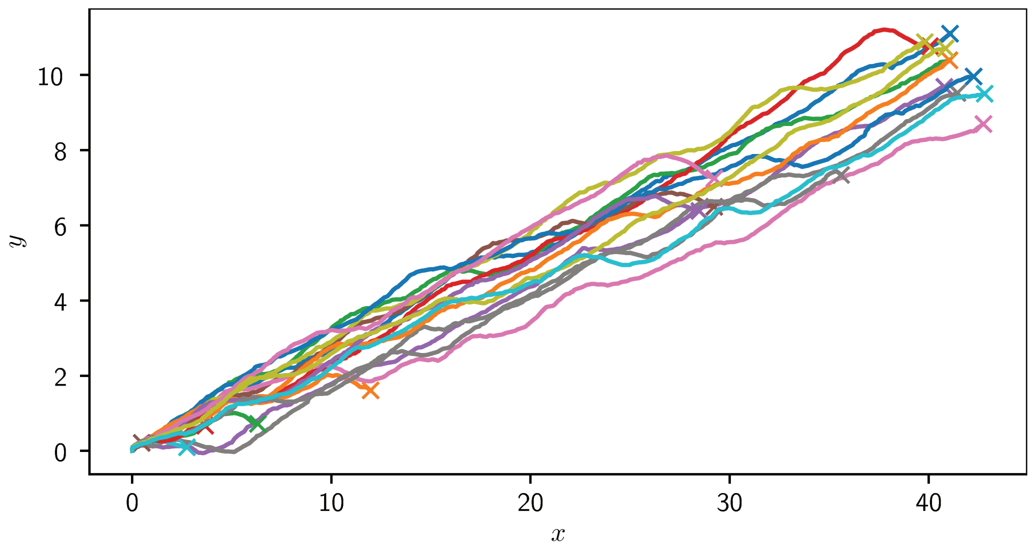

Figure 6.9: CoM X-Y position for 20 runs. π_{ypr} capped at 50m.

Parameter	Value
$keep\_alive$	$+10.0$
$w_{progress}$	$+10.0$
w_{knee}	-5.0
β_{knee}	$+0.76$
w_{yaw}	-6.0
w_{pitch}	-6.0
w_{roll}	-6.0

Table 6.5: Reward weights used to train π_{ypr}.

6.5 Under-actuation of joints below knee

From the results of π_{ypr}, we decided to under-actuate *AnkleRoll* and *AnklePitch* in addition to *FootPitch*. In other words, all joints strictly below the *KneePitch* are under-actuated. For this reason, we refer to the resulting policy as π_{knee}. Once more, results can be seen in Figure 6.10 and Figure 6.11. The weights can be seen in Table 6.6. This policy was the first to mitigate the micro-steps issue. Without enough controllable lower joints, the agent was forced to use more its knees and pelvis to walk. There was also a natural emergence of some kind of alternating movement between the two legs. As opposed to π_{stone}, where the movement phase was more enforced into the r itself, π_{knee} discovered the movement phase entirely by itself. Unfortunately, at the cost of this higher quality gait, the required training time was longer than usual. This longer training time requirement made it even harder to train the policy to reach the end of the simulated environment. π_{knee} shares with π_{ypr} a similar problem of unnatural hip position.

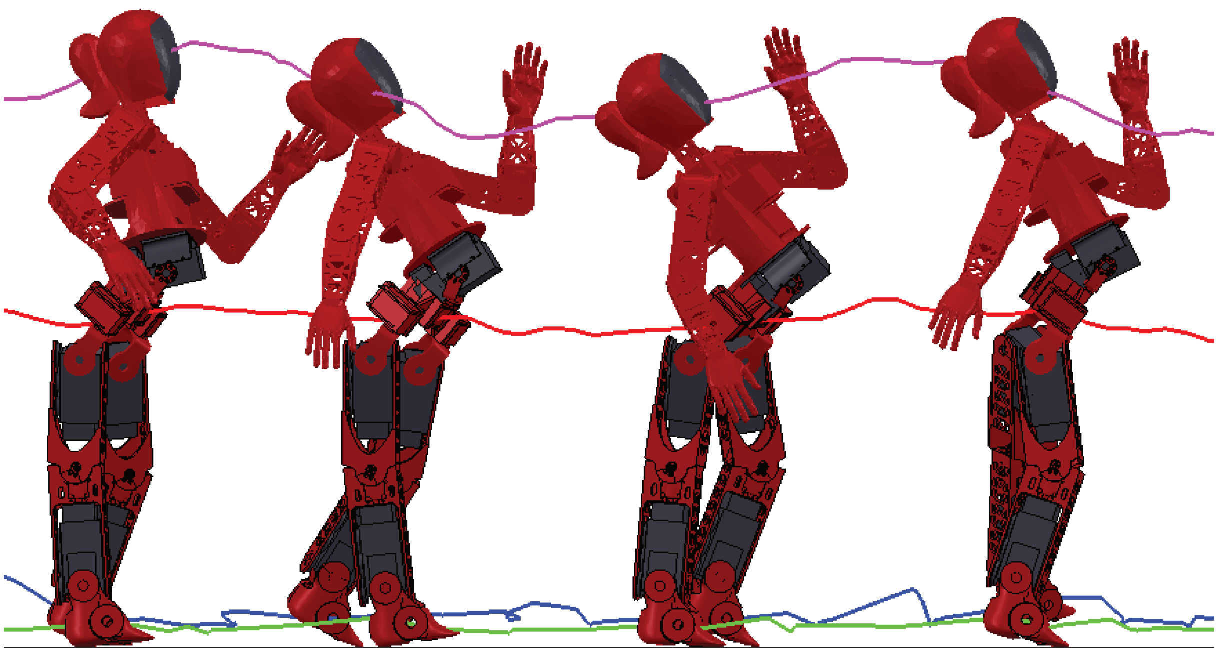

Figure 6.10: Time-lapse of the gait produced by π_{knee}.

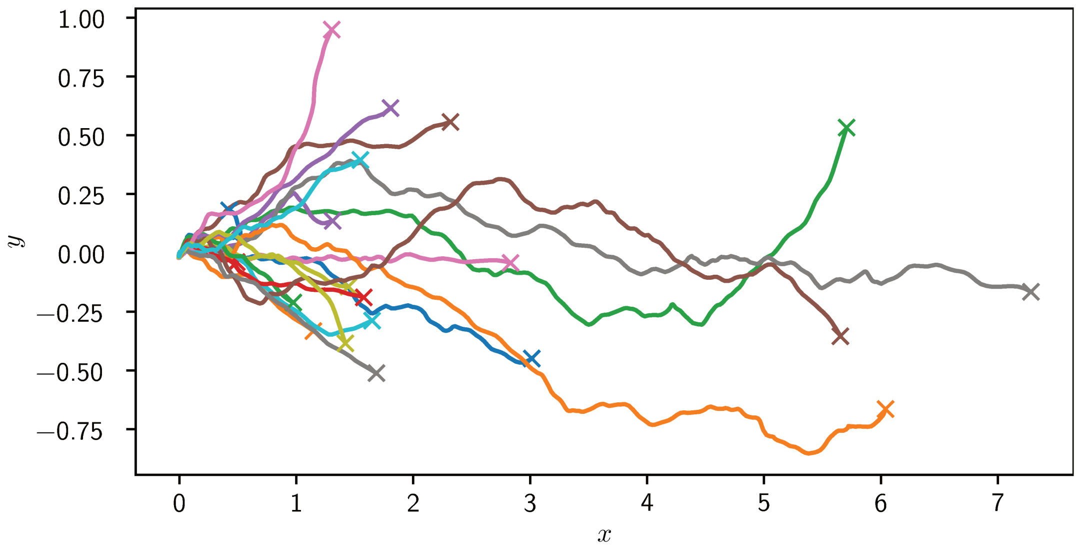

Figure 6.11: CoM X-Y position for 20 runs of π_{knee}.

Parameter	Value
$keep\_alive$	$+4.0$
$w_{progress}$	$+4.0$
w_{knee}	-1.0
β_{knee}	$+0.777$
w_{yaw}	-1.0
w_{pitch}	-1.0
w_{roll}	-1.0

Table 6.6: Reward weights used to train π_{knee}.

6.6 Quantitative Analysis

As we described in Section 4.2.2, quantitative metrics were also collected to compare the different policies. These results can be seen in Table 6.7.

Policy	**Energy**	**Feet z**	**Pitch**	**Roll**	x	y	**End**
$\pi_{progress}$	0.755 ± 0.053	0.031 ± 0.004	-0.303 ± 0.105	-0.296 ± 0.126	**32.623 ± 17.426**	-2.773 ± 1.901	**37.5%**
π_{stone}	0.725 ± 0.088	0.035 ± 0.005	**0.022 ± 0.143**	-0.150 ± 0.104	12.288 ± 12.125	**-0.012 ± 0.102**	2.5%
π_{ypr}	0.723 ± 0.047	0.029 ± 0.003	-0.247 ± 0.095	0.064 ± 0.087	28.428 ± 16.214	6.547 ± 3.858	0.0%
π_{knee}	**0.713 ± 0.088**	0.036 ± 0.010	-0.495 ± 0.221	**0.059 ± 0.162**	2.977 ± 2.514	-0.106 ± 0.501	0.0%
π_{1leg}	0.751 ± 0.100	**0.304 ± 0.091**	0.264 ± 0.348	-0.440 ± 0.407	2.347 ± 1.160	1.028 ± 0.501	0.0%
π_{back}	0.751 ± 0.053	0.039 ± 0.004	-0.093 ± 0.111	0.431 ± 0.120	24.321 ± 20.068	24.060 ± 19.665	5.0%

Table 6.7: Performance metrics averaged across 40 runs.

The energy use of all the policies are similar since this metric was not included in any reward function. Feet z measurement was dominated by π_{1leg}, a test policy specifically designed for that purpose. Both Pitch and Roll were successfully minimized in policies that included this reward signal, such as π_{ypr} and π_{knee}. $\pi_{progress}$ had the best performance on the x metric. This result was expected since $\pi_{progress}$ had virtually only that metric to optimize. The deviation on y was almost zero in π_{stone}, given that this formulation was much more constrained.

6.7 Transfer Learning Results

As discussed in Section 4.2.3, we frequently trained new policies by reusing parameters that were learned from previous ones. By using Transfer Learning this way, we could not only train our policies faster, but also iterate more quickly in promising reward functions. In all of our experiments, the use of transfer, learning did not substantially changed the final policy, only accelerated its materialization. As an example, we can see the impact of reusing the parameters learned from $\pi_{progress}$ to retrain π_{ypr} in Figure 6.12. Using transfer learning reduced the training time considerably while achieving similar results.

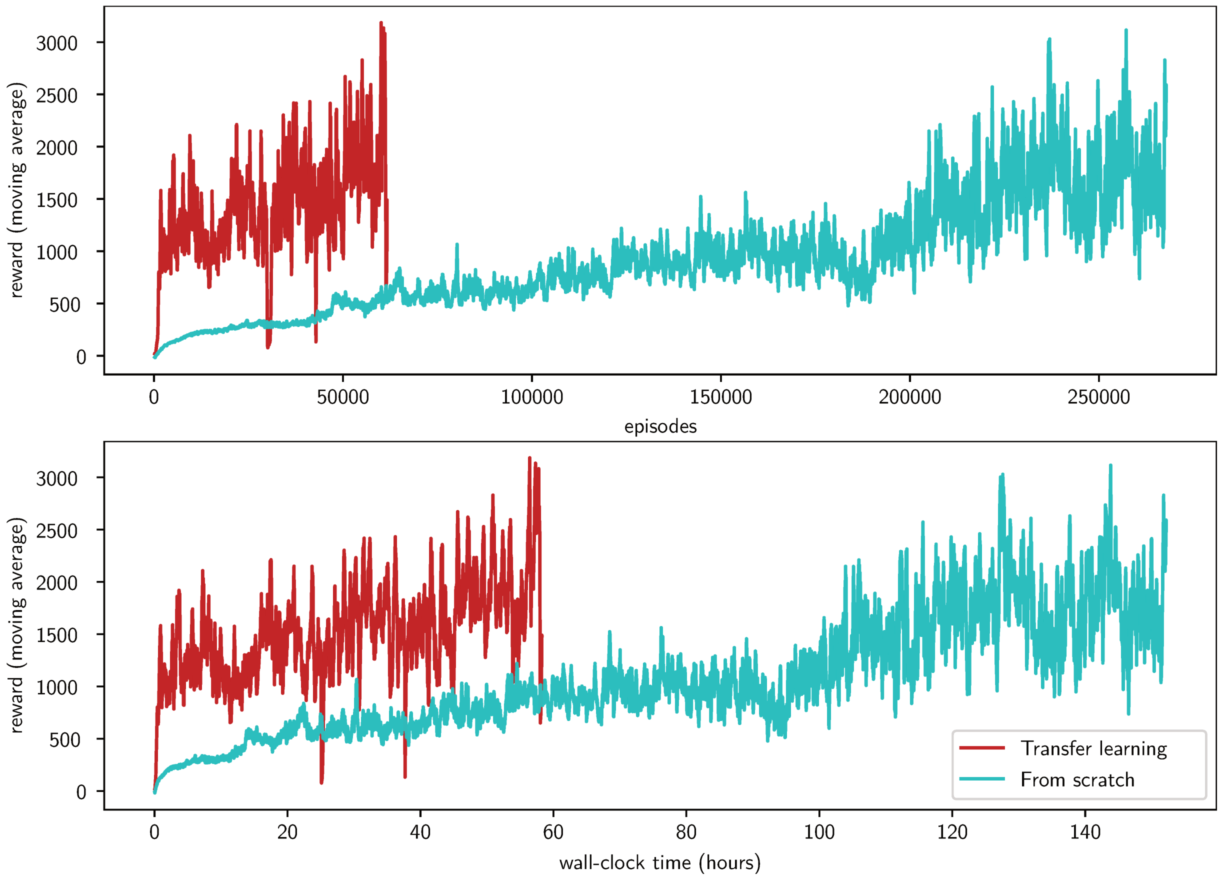

Figure 6.12: Learning π_{ypr} using the parameters trained from $\pi_{progress}$ versus learning from scratch. Reward axes are shown using a moving average of size 200.

6.8 Other experiments

Before conceiving the reward functions, observational spaces and action spaces mentioned in Chapter 5 and experimented in Chapter 6, many other experiments were conducted. In total, around 40 different RL formulations were attempted. Between $\pi_{progress}$ and π_{ypr}

alone, there were more than 20. The policies showed along Chapter 6 and summarized in Section 6.6 were chosen as a representative group to show the main ideas and consequences of different formulations while also showing a progressive increment of ideas and growing complexity in the MDP formulation.

6.9 Hypotheses discussion

Given our results, we can confirm H_1:

- H_1: DRL algorithms can be used to conceive a walking gait for Marta from scratch.

We could see that, with virtually no supervision or hand-engineering, the state-of-the-art DRL algorithm SAC [17] was able to learn a walking gait for Marta in simulation, albeit an unnatural one. We can also confirm H_2:

- H_2: Reward engineering can be used to improve discovered gaits.

Our experiments showed that careful modifications in r could shift the policy towards a more desired gait for Marta. These modifications can have their individual impacts tested and then systematically composed to have a more comprehensive, but still tractable reward function.

We could not confirm H_3:

- H_3: It is possible to transfer the policy learned in simulation to the real robot.

Despite the improvements to the walking gaits, even the best walking gaits learned in the simulation were still unusual. Although we can assess in simulation that they are stable and even tolerant to small perturbations, transferring the gait to the real robot would probably still require further refinements to avoid damage to the real robot.

Chapter 7

Conclusion and future work

In this work, we applied Deep Reinforcement Learning algorithms to learn control policies and, in particular, bipedal locomotion policies for the Marta robot. As far as we are concerned, this is the first DRL model-free policy learned for a humanoid model in a high fidelity simulator such as CoppeliaSim.

As our main contributions we can state:

- The development of an extensible and transparent framework for integrating a high fidelity simulator, CoppeliaSim, to OpenAI Gym, the standard RL toolkit;
- A model-free learned gait for Marta robot, designed without explicit supervision, under a high fidelity simulator;
- The proposition and comparison among distinct reward functions and how they affect the discovered gaits.

We proposed new RL modeling strategies for the locomotion problem and validated our models in configurable experiments in simulated environments. Our reward engineering employed novel and modular strategies for bipedal locomotion and highlighted how high-level abstractions can be translated into optimisation control tasks in an experimental and end-to-end context.

In fact, our approach does not make many assumptions about the robot model and, with the exception of step sizes in one experiment, did not use specific information from the Marta robot. This means that, in theory, a similar approach could be employed for different robot models. It also can imply that we could take advantage of Marta's unusual foot size and spherical hip joint to shape a more suitable policy.

While the learned policies appear to be unsuitable for the real robot in its current form, they provide valuable benchmarks for simulated policies and also, provide insights into the current robot dynamics design that could help in future iterations.

As for future works, the current policies could be further refined to address its current flaws and better approach more commonly expected walking gaits.

Outside of the model-free realm, imitation and transfer learning both seem good candidates for further development in locomotion tasks for biped robots like Marta.

There is also an increasing number of new techniques for better transferring simulated policies to the real robot, with and without domain randomization [38][27][62], which could be explored in future works.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. *GitHub repository*, 2018.

[3] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[4] Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[6] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2.*, pages 1023–1028. AAAI Press / The MIT Press, 1994.

[7] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2017.

[8] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2019.

[9] Xingye Da, Zhaoming Xie, David Hoeller, Byron Boots, Animashree Anandkumar, Yuke Zhu, Buck Babich, and Animesh Garg. Learning a contact-adaptive controller for robust, efficient legged locomotion, 2020.

[10] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[11] M. Freese E. Rohmer, S. P. N. Singh. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[12] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ODE and physx. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 4397–4404. IEEE, 2015.

[13] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[15] Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 42(6):1291–1307, 2012.

[16] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies, 2017.

[17] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[18] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition.* Springer series in statistics. Springer, 2009.

[19] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.

[20] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[21] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *CoRR*, abs/1901.08652, 2019.

[22] Tommi S. Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

[23] Stephen James, Marc Freese, and Andrew J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

[24] Matt Johnson, Brandon Shrewsbury, Sylvain Bertrand, Duncan Calvert, Tingfan Wu, Daniel Duran, Douglas Stephen, Nathan Mertins, John Carff, William Rifenburgh, Jesper Smith, Christopher Schmidt-Wetekam, Davide Faconti, Alex Graber-Tilton, Nicolas Eyssette, Tobias Meier, Igor Kalkov, Travis Craig, Nick Payton, Stephen McCrory, Georg Wiedebach, Brooke Layton, Peter D. Neuhaus, and Jerry E. Pratt. Team ihmc's lessons learned from the DARPA robotics challenge: Finding data in the rubble. *J. Field Robotics*, 34(2):241–261, 2017.

[25] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285, 1996.

[26] Sham Kakade. A natural policy gradient. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1531–1538. MIT Press, 2001.

[27] Manuel Kaspar, Juan David Muñoz Osorio, and Jürgen Bock. Sim2real transfer for reinforcement learning without dynamics randomization. *CoRR*, abs/2002.11635, 2020.

[28] Gavin D. Kenneally, Avik De, and Daniel E. Koditschek. Design principles for a family of direct-drive legged robots. *IEEE Robotics Autom. Lett.*, 1(2):900–907, 2016.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[30] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J. Bentley, Samuel Bernard, Guillaume Beslon, David M. Bryson, Patryk Chrabaszcz, Nick Cheney, Antoine Cully, Stephane Doncieux, Fred C. Dyer, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Stephanie Forrest, Antoine Frénoy, Christian Gagné, Leni Le Goff, Laura M. Grabowski, Babak Hodjat, Frank Hutter, Laurent Keller, Carole Knibbe, Peter Krcah, Richard E. Lenski, Hod Lipson, Robert MacCurdy, Carlos Maestre, Risto Miikkulainen, Sara Mitri, David E. Moriarty, Jean-Baptiste Mouret, Anh Nguyen, Charles Ofria, Marc Parizeau, David Parsons, Robert T. Pennock, William F. Punch, Thomas S. Ray, Marc Schoenauer, Eric Shulte, Karl Sims, Kenneth O. Stanley, François Taddei, Danesh Tarapore, Simon Thibault, Westley Weimer, Richard Watson, and Jason Yosinski. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities, 2019.

[31] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[32] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[33] Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. *CoRR*, abs/1811.09656, 2018.

[34] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[36] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012.

[37] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 278–287. Morgan Kaufmann, 1999.

[38] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[39] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[40] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. *CoRR*, abs/1712.00378, 2017.

[41] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4):41:1–41:13, 2017.

[42] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals, 2020.

[43] Paris Pennesi and Ioannis Ch. Paschalidis. Solving sensor network coverage problems by distributed asynchronous actor-critic methods. In *46th IEEE Conference on Decision and Control, CDC 2007, New Orleans, LA, USA, December 12-14, 2007*, pages 5300–5305. IEEE, 2007.

[44] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[45] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[46] Lorien Y. Pratt. Discriminability-based transfer between neural networks. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 204–211. Morgan Kaufmann, 1992.

[47] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010.

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *ArXiv e-prints*, July 2017.

[49] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[50] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[51] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054, 1998.

[52] Richard Stuart Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts Amherst, 1984. AAI8410337.

[53] Istvan Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, 2006.

[54] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots, 2018.

[55] Brendan Tidd, Nicolas Hudson, and Akansel Cosgun. Guided curriculum learning for walking over complex terrain, 2020.

[56] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012.

[57] Michel Tokic and Günther Palm. Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*, volume 7006 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2011.

[58] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: a structure for efficient numerical computation. *CoRR*, abs/1102.1523, 2011.

[59] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2094–2100. AAAI Press, 2016.

[60] C.J.C.H. Watkins. Learning from delayed rewards. ph.d. thesis. *Cambridge University*, 1989.

[61] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan W. Hurst, and Michiel van de Panne. Iterative reinforcement learning based design of dynamic locomotion skills for cassie. *CoRR*, abs/1903.09537, 2019.

[62] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan W. Hurst, and Michiel van de Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 317–329. PMLR, 2019.

[63] Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. ALLSTEPS: curriculum-driven learning of stepping stone skills. *CoRR*, abs/2005.04323, 2020.

[64] A. P. Yelnik, S. Tasseel Ponche, C. Andriantsifanetra, C. Provost, A. Calvalido, and P. Rougier. Walking with eyes closed is easier than walking with eyes open without visual cues: The Romberg task versus the goggle task. *Ann Phys Rehabil Med*, 58(6):332–335, Dec 2015.