FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO

Humanoid Low-Level Skills using Machine Learning for RoboCup

Tiago Rafael Ferreira da Silva

DISSERTATION

Integrated Master in Informatics and Computing Engineering

Supervisor: Luís Paulo Reis

Second Supervisor: Armando Jorge Sousa

July 3, 2019

Humanoid Low-Level Skills using Machine Learning for RoboCup

Tiago Rafael Ferreira da Silva

Integrated Master in Informatics and Computing Engineering

July 3, 2019

Abstract

The goal of this work is to develop a framework capable of optimising any low-level skill related to the RoboCup 3D Simulation League. This year new rules have been added where self-collisions are no longer possible. Until now, teams have taken advantage of these unrealistic behaviours in order to achieve stronger kicks and faster movements. But now that these are considered illegal, re-optimising previous behaviours with attention to the new rules has become a necessity for any team that wishes to maintain its performance.

Because humanoid movements are such high-dimensional and complicated tasks, Machine Learning approaches are rather appealing for their optimisation. Several Deep Reinforcement Learning algorithms will be reviewed in this document and the state of the art in the area will be presented. Some other Machine Learning methods will also be discussed, such as Evolution Strategies and Hierarchical Reinforcement Learning.

In order to solve the above-mentioned problems, a framework has been developed. It is generic and flexible enough to allow any low-level skill to be optimised, with out-of-the-box support for optimisations using CMA-ES or PPO algorithms. However, it allows any other optimisation technique to be used with it, and thus this framework will stay relevant in the future.

After the framework was developed, 3 behaviours were optimised: one getup, one kick and one targeted kick. The results of these optimisations are presented in this document, as well as a comparison and discussion between the optimisation techniques that were used.

Resumo

O objectivo deste trabalho é desenvolver uma *framework* capaz de optimizar qualquer *skill* de baixo nível relacionado com a Liga de Simulação 3D do RoboCup. Novas regras foram adicionadas este ano que proíbem auto-colisões. Até agora, as equipas abusaram deste tipo de comportamentos de forma a conseguir executar chutos mais fortes, ou correr mais rápido. Mas como a partir de agora isto será ilegal, as equipas que queiram continuar a ser competitivas terão de re-optimizar os comportamentos afectados.

Uma vez que movimentos humanoides são uma tarefa tão complicada, Machine Learning é uma boa opção para optimizações deste tipo. Neste documento, vários algoritmos de Deep Reinforcement Learning são analisados, e o estado da arte da área é apresentado. Além disso, outros métodos de optimização também são abordados, tais como Estratégias de Evolução ou Hierarchical Reinforcement Learning.

De modo a resolver os problemas referidos em cima, foi desenvolvida uma *framework*. Esta é genérica e flexível o suficiente de modo a permitir a optimização de qualquer *skill* de baixo nível, e tem suporte nativo para os algoritmos CMA-ES e PPO. No entanto, é também possível utilizar qualquer outro tipo de algoritmo de optimização.

Depois do desenvolvimento da *framework*, 3 *skills* foram optimizados: um *getup*, um *kick* e um *targeted kick*. Os resultados destas optimizações, assim como uma discussão e comparação entre os diferentes algoritmos usados, serão apresentados neste documento.

Acknowledgements

I would like to thank my supervisor, Professor Luís Paulo Reis of the Faculty of Engineering of the University of Porto (FEUP), who helped and supported me throughout this research, guiding me and providing valuable insights when needed.

I would also like to thank David Simões, PhD student at University of Aveiro and at FEUP, who was always available and willing to lend a hand. His help and his knowledge were absolutely vital for the development of this work.

Finally, I'd like to thank every member of the FC Portugal 3D Simulation League Team. Without each and everyone of them, this dissertation would not have been possible.

Tiago Silva

"One of the greatest accomplishment of robotics competitions
is how they force us to take our robots out of the lab
into the real world, in order to solve real problems"

Gerhard K. Kraetzschmar

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
DDPG	Deep Deterministic Policy Gradient
DDQN	Dueling Deep Q-Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ES	Evolution Strategy
HIRO	Hierarchical Reinforcement Learning with off-policy correction
HRL	Hierarchical Reinforcement Learning
I2A	Imagination-Augmented Agents
MCTS	Monte Carlo Tree Search
PPO	Proximal Policy Optimization
ML	Machine Learning
RL	Reinforcement Learning
RoboCup	Robot World-Cup Soccer
SAC	Soft Actor Critic
TD3	Twin Delayed Deep Deterministic Policy Gradient
VPG	Vanilla Policy Gradient

Chapter 1

Introduction

The Robot World-Cup Soccer, usually referred to as RoboCup, is an international initiative to promote AI and robotics research by means of soccer competitions [1]. One of its many leagues, the 3D Simulation League, makes use of a multiagent system, and thus provides an optimal place for the research of coordination, cooperation and learning of both individual and group skills [2]. For that reason, this work makes use of the RoboCup 3D Simulation League as its development and testing environment, alongside a simulated version of the NAO[1] humanoid robot. The conducted research focused primarily on the optimisation of low-level behaviours, as well as the development of a framework in order to ease and streamline future optimisations.

1.1 Context

This work is inserted within the FC Portugal 3D Simulation Team - a Portuguese team that has won several awards in the RoboCup 3D Simulation League, including the titles of World Champion in 2006 and European Champion in 2007, 2012, 2013, 2014 and 2015 [3].

Throughout the years, the team has developed multiple work related to low level skills such as running, passing, and kicking.

By making use of the work previously developed by the research group as a starting point, this dissertation further improved the existing behaviours, and adjusted them to match the new RoboCup rules.

1.2 Motivation

The first RoboCup competition was held in 1997, in Nagoya, Japan. And ever since then, a world championship has been organised every year. Joining this annual event, a symposium is also

[1] SoftBank Robotics' NAO robot: https://www.softbankrobotics.com/emea/en/nao (visited on 01/13/2019).

organised every year - a place for sharing research and results inside the RoboCup community [4]. Throughout the years, the state of the art in robotics has advanced tremendously, in part due to the annual soccer competitions. But one thing that remains the same, even now, is RoboCup's objective:

> *By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup [5].*

In order to achieve this goal, a good execution of low level skills, such as passing, kicking and dribbling, is required, which are the main focus of this dissertation.

Because humanoid robots have a large amount of degrees of freedom, they are able to move very similarly to humans. However, this poses many challenges. They must always adjust their speeds and directions, while constantly keeping and adjusting their balance [6]. Because of the complexity of this task, humanoid behaviours can not be feasibly implemented by *hard-coded* programs. Which is why Machine Learning (ML) is such an appealing alternative. By letting the machines learn by themselves, it's possible to achieve much more stable and flexible solutions, that would otherwise be impossible to implement.

1.3 Problem

Recently, new rules to constrain the movements of agents have been added to RoboCup. Until now, the body parts of the same agent could intersect with each other, without repercussion. These self-collisions are unrealistic and have since become something taken advantage of by many teams, in order to achieve stronger kicks or faster movements. However, new rules are being implemented to limit self-collisions. Re-optimising previous behaviours with attention to the new rules is then a necessity for any team that wishes to maintain its performance.

1.4 Objectives

The objectives of this dissertation are to optimise low-level skills, and develop the necessary tools to do so. Firstly, a generic framework should be created in order to allow for future optimisations. This framework should be able to handle different types of optimisation techniques, with focus on Evolution Strategies and Reinforcement Learning. Secondly, a *getup* - a behaviour whose goal is to stand up, from a laying position - and a *kick* - a behaviour whose aim is to shoot the ball - should be optimised. These behaviours were selected because they were the most severely affected by the new self-collisions rules and because both are key skills in a soccer match.

1.5 Document Structure

This document follows a typical dissertation structure, and the outline of the chapters is as follows:

- **1 Introduction** This chapter serves as contextualisation and introduction to the problem, stating the objectives and goals of the proposed solution.
- **2 Related Work** Here, several state-of-the-art approaches are reviewed in regards to Machine Learning, with focus on Deep Reinforcement Learning methods. Some details about Hierarchical Reinforcement Learning and current applications in RoboCup are also talked about.
- **3 RoboCup Environment** This chapter presents the environment used in RoboCup competitions and for the optimisations. It goes over the simulation server, the 3D viewer, the robot model and the new rules that have been added for this year's competition.
- **4 Optimisation Framework** Here, the developed framework is presented alongside its architecture. This chapter also presents how the optimisations are conducted, and introduces some key concepts used in later chapters.
- **5 Getup Optimisation** This chapter explains everything related to the optimisation of the getup behaviour, from the problem definition, to reward shaping, to results.
- **6 Kick Optimisation** Here, a kick with the goal of shooting the ball in a straight line is presented. It follows the same structure as the previous chapter.
- **7 Targeted Kick Optimisation** This chapter presents the last behaviour that was optimised during this dissertation, a kick with the aim to shoot the ball to specific targets. As with the previous sections, it explains everything related to the behaviour.
- **8 Conclusion and Future Work** Finally, this chapter closes this document and reflects upon the work that was developed throughout the last few months. It also goes over some possible improvements for the future.

Chapter 2

Related Work

Machine learning - the act of a computer system learning from data [7] and continuously improving its performance on a given task - is already present in our everyday lives. Though still unseen by many, technology supported by machine learning algorithms is an essential part of modern society [8]. From search engines optimisation [9], to detection of spam tweets [10] and phishing emails [11], to genetics manipulation [12] and many other fields, the influence of machine learning has spread tremendously in recent years.

2.1 Deep Reinforcement Learning

Deep reinforcement learning, as its name implies, was born from the combination of reinforcement learning and deep learning - two sub fields in the larger scope that is machine learning as a whole. Though still in its infancy [13], as figure 2.1 illustrates, research in this field is already capable of achieving revolutionary results, as was demonstrated by Google DeepMind[1] in recent years with their breakthroughs on the game of Go[2]. One of the few games where humans could still outperform machines [14], Go was thought to be years away from being solved. That is, until DeepMind proved the world wrong.

In 2015, the first iteration of DeepMind's Go-playing-program, AlphaGo, became the first program to ever defeat a human champion in the game of Go, accomplishing an incredible result of 5-0[3] against the European Go Champion at the time, Fan Hui, and later, in 2017, winning 3-0[4] versus the World Champion, Ke Jie [16]. The game of Go had been solved. However, there was still room for improvement. Because AlphaGo relied on supervised learning - a type of learning that requires labelled training data - it needed human action as input. Therefore, its intelligence

[1] Google DeepMind: https://deepmind.com/ (visited on 01/22/2019).
[2] Go overview: https://en.wikipedia.org/wiki/Go_(game) (visited on 01/22/2019).
[3] AlphaGo versus Fan Hui: https://en.wikipedia.org/wiki/AlphaGo_versus_Fan_Hui (visited on 01/22/2019).
[4] AlphaGo versus Ke Jie: https://en.wikipedia.org/wiki/AlphaGo_versus_Ke_Jie (visited on 01/22/2019).

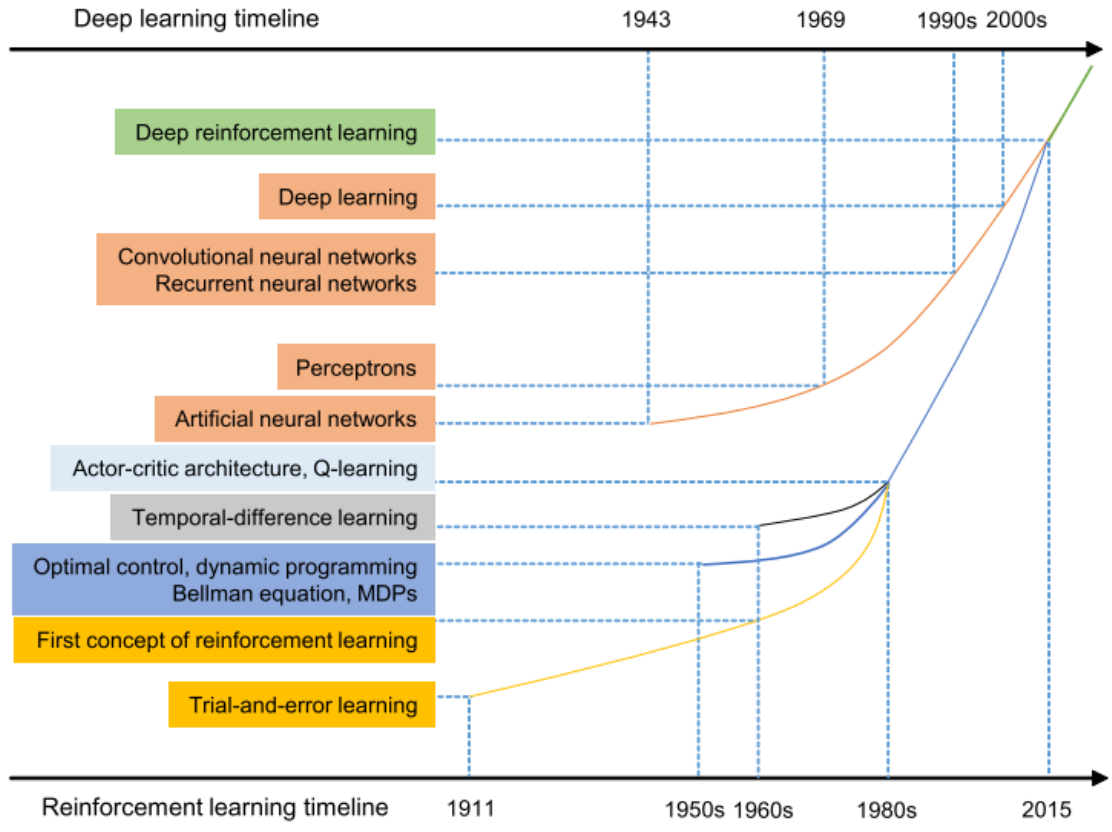


Figure 2.1: Deep Reinforcement Learning history [15].

level would always be limited by the quality of the training data it was fed. In other words, because it was learning from human examples, it would never be able to surpass human intelligence (though it could beat human champions).

This much needed improvement came in 2017, in the next iteration of the program, AlphaGoZero. This new version of AlphaGo removed its supervised learning part, and in doing so, its human part as well, thus becoming solely reliant on reinforcement learning in order to learn how to play. Though it had absolutely no training data fed to it, it was still able to learn how to play the game by itself, from scratch. But not only was it able to learn the game, it was able to learn it better than its predecessor that had relied on human knowledge. When put against AlphaGo, AlphaGoZero was able to achieve 100 wins, out of the 100 games that were played [17].

More recently, in 2018, DeepMind announced AlphaZero - a generalised version of AlphaGoZero, that not only is capable of playing Go, but also Chess and Shogi [18].

Though AlphaZero might be one of the most talked about AIs at the moment, it is merely a branch of DRL. One particular taxonomy of DRL algorithms proposed by OpenAI stands out as rather interesting, which can be seen on figure 2.2.

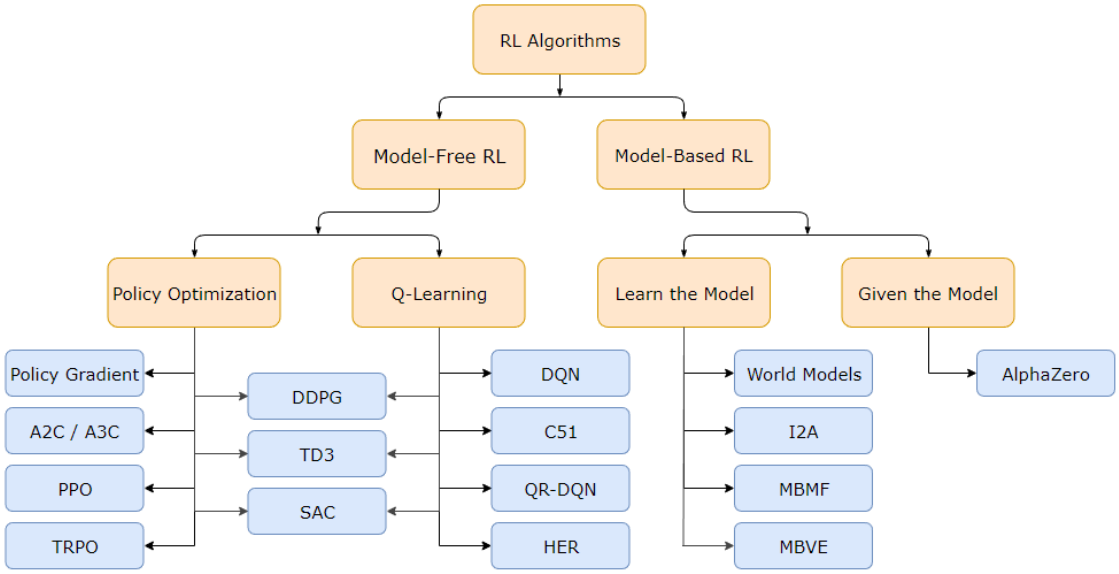


Figure 2.2: Taxonomy of Deep Reinforcement Learning Algorithms [19].

2.1.1 Model-Free

One of the most differentiating aspects between Reinforcement Learning (RL) algorithms is whether or not they have access to a model[5] of the world.

Model-Free algorithms - the ones without such knowledge - cannot anticipate what will happen if action A is taken in state S. Instead, they focus on learning the optimal policy[6], maximising the long-term cumulative reward [22], by repeatedly interacting with the environment[7] through trial-and-error. Although they are far less sample efficient, as figure 2.3 illustrates, these algorithms are much easier to implement, as often times it's not possible to give the agent a model of the environment due to the complexity of the problem.

Figure 2.3: Sample efficiency of different DRL methods [24].

As mentioned above, all of the algorithms that fall under this branch have the same goal of finding the best possible policy. There are two different ways by which this learning process occurs.

[5] An estimation of how an agent perceives the dynamics and physics of the world, allowing it to predict transitions between states.

[6] A probability distribution function mapping states to actions [20] - can be seen as the strategy employed by the agent to decide what action to perform in the current state [21].

[7] Usually a stochastic environment described as a Markov Decision Process - that is, an environment with some degree of randomness to it, where each state transition does not depend upon prior states or agent actions [23].

Q-Learning

Also known as Value-based methods, these algorithms learn to estimate the Q-value[8] of each state-action pair, leaving the policy to follow a greedy approach that maximises that value at each step [26]. This learning process is usually executed off-policy, which means the agent can reuse previously collected data. For this reason, these methods are much more sample efficient than other on-policy alternatives, which do not allow for the reuse of data.

One particular problem of these algorithms is that they do not try to directly learn the best policy. Instead they focus on learning the best action to take in each state. Thus, the policy isn't learned *per se*, but instead is inferred, which can lead to errors.

DQN One of the main algorithms that fall under this category is Deep Q-(Learning-)Network (DQN). Presented by DeepMind in 2015 [27], this agent was capable of learning how to play Atari games without any prior knowledge of them (hence it being model-free) and outperformed humans.

One of its key features is Experience Replay. As the agent is training, it will collect and store information in a buffer, which can then later be sampled through to train the network. This helps solve the instability associated with approximating Q-values using neural networks, thus guaranteeing the algorithm will converge to a solution. Furthermore, it also prevents the network from getting stuck on a local optimum, making sure the global optimum is reached [28].

DQN Improvements This algorithm was one of the first major breakthroughs in DRL and became the foundation for a lot of future work in the area. Later, in 2016, Google DeepMind published an article [29] where they showed the flaws of the original DQN algorithm, and presented an improved version of it, known as Double DQN.

Yet another improvement came in the form of Dueling Deep Q-Network (DDQN), where the Q-value is divided into two sub-values: the state value, representing the value of being in that state, and the advantage, which represents how much better or worse it is to take a specific action A in the state S versus all other possible actions in that state [30]. This decoupling of the Q-value is useful because by looking at the value of a state, it's possible to decide whether or not all possible actions in that state should be computed [31].

Another enhancement was the use of Prioritized Experience Replay, which instead of randomly sampling through the replay buffer, prioritised experiences which had a bigger impact on learning [32]. Simply put, it made it so that the most valuable experiences were used more often than the less valuable ones.

Policy Optimisation

Policy-based methods, unlike Value-based ones, directly learn the optimal policy, and because of that, are much more stable. Their downside is that this optimisation usually happens on-policy,

[8] The Q-value can be seen as the quality of the state-action pair and is constantly updated based on the immediate reward received by performing an action [25].

meaning they cannot reuse old data. As such, they are far less sample efficient, requiring a larger amount of training data.

VPG One of the very basic Policy-based algorithms is Vanilla Policy Gradient (VPG). As a REINFORCE algorithm, each action gets rewarded based on the final reward of the episode. That is, unlike DQN where the agent receives an immediate reward for every action performed, in this algorithm, the reward is only given at the very end of the experience. As such, if the experience is considered good, all actions that were executed will get a positive reinforcement, even if they were bad actions - figure 2.4 illustrates this. On the top the actual quality of each action is shown. On the bottom, the quality of each action, as perceived by the agent, is shown. Because the result of the experience was *good*, every action is averaged as *good*, even if they were *awesome* or *bad*. This is one of the major problems of Policy Gradient algorithms, and the reason they are less efficient.

Their major advantages are that they can learn stochastic policies and are capable of operating in a continues space [33], unlike DQN, which could only function in discrete environments.

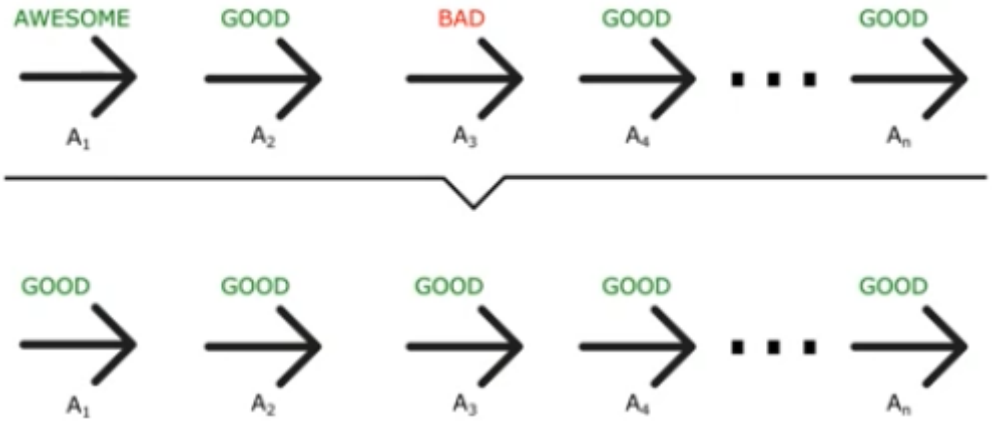


Figure 2.4: Credit assignment problem of REINFORCE algorithms [24].

A3C The Asynchronous Advantage Actor Critic (A3C) algorithm published by DeepMind in 2016 combines Value-based methods and Policy Grading methods [34]. As an *actor-critic method*, A3C does not only learn the *critic* Q-value, but also the *actor* policy. It works in both discrete and continuous spaces, making it a much more appealing option for fields where previous approaches weren't enough, such as robotics. Another big feature of this algorithm, and one of the As in its name, is its asynchronous architecture, which allows it to be dozens of times faster than DQN.

A2C The Advantage Actor Critic (A2C) algorithm was proposed by OpenAI as an improvement to A3C. Because each agent was running asynchronously, and thus updating the network also asynchronously, each agent was acting upon outdated network parameters. OpenAI solved this by adding a global coordinator to the network. In this new approach, each agent waits for all other agents to finish the current episode, and only after the coordinator has correctly updated

the network do they start executing the next iteration - each agent now having access to the same version of the network. According to OpenAI's research, the asynchronous architecture doesn't actually help to achieve better results, and when compared to A3C, their synchronous implementation is not only more cost-effective on single-GPU machines, but also faster than a CPU-only implementation [35]. Figure 2.5 illustrates the architectural difference between A3C and A2C.

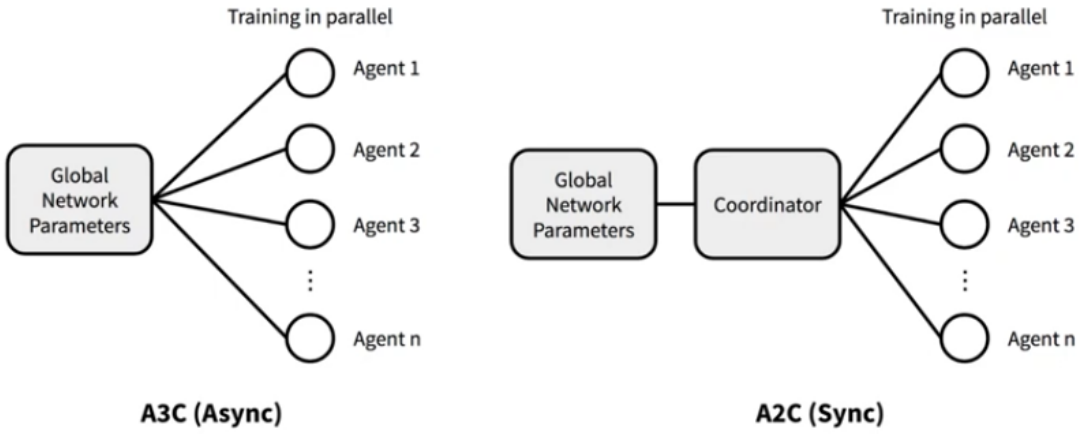


Figure 2.5: A3C and A2C high-level architectures [24].

PPO The Proximal Policy Optimization (PPO) algorithm was presented by OpenAI in 2017, focusing on ease of implementation and sample complexity [36]. At each step, it tries to calculate an update that minimises the cost function while ensuring it doesn't deviate too much from the previous policy [37]. In other words, PPO tries improve the current policy by the largest possible amount that won't negatively affect performance [38]. Recently it was able to learn how to play the esports game Dota 2[9] from scratch, beating the world champions in April 13, 2019 [39]. It was also able to train an AI policy for a humanoid agent, learning how to walk and run, while keeping its balance without falling when pushed by outside forces [37].

Q-Learning and Policy Optimisation

The two branches of Mode-free approaches mentioned above, Q-Learning and Policy Optimisation, can be combined in order to get the best out of both approaches.

DDPG Published by DeepMind, Deep Deterministic Policy Gradient (DDPG) combines the actor-critic framework with Value-based methods [40], learning simultaneously a deterministic policy[10] and a Q-function, each improving the other [19]. Unlike A3C and A2C, which optimise the policy in respect to the advantage, DDPG optimises it in regards to the Q-values [25]. It can be seen as an extension of DQN to allow it to work in continuous action spaces [41].

[9] Dota 2's Steam page: https://store.steampowered.com/app/570/Dota_2/.

[10] Unlike Policy Gradient methods, which learn stochastic policies, DDPG learns a deterministic one - following a greedy approach where the best action is chosen over the rest, similarly to DQN.

TD3 Twin Delayed Deep Deterministic Policy Gradient (TD3) is an algorithm proposed in 2018 as an improvement to DDPG [42]. Because DDPG can be thought of as an extension of DQN, it also suffers from the same problems DQN did, namely in regards to overestimation of the Q-values, which can lead to policy breaking [43]. This new algorithm solves those problems and outperforms the classic DDPG method.

SAC Yet another improvement to DDPG is the Soft Actor Critic (SAC) algorithm [44]. Unlike DDPG which optimises a deterministic policy, SAC optimises a stochastic one, combining the Maximum Entropy[11] Reinforcement Learning framework with the DDPG approach.

2.1.2 Model Based

Because they have access to a model of the world, model-based algorithms are much more sample efficient. These algorithms are capable of planning, searching through multiple possible outcomes and choosing the best one. The downside to this approach, is that often times it's not easy to construct a model. Not only that, but the models don't always reflect reality appropriately, leading the agent to lead policies that may perform well in simulation, but when extrapolated to reality hardly work. The iteration loops of model-based and model-free algorithms can be seen in figure 2.6, where model-based algorithms are represented by the outer *model learning* loop, while model-free algorithms are represented by the inner *direct RL* loop.

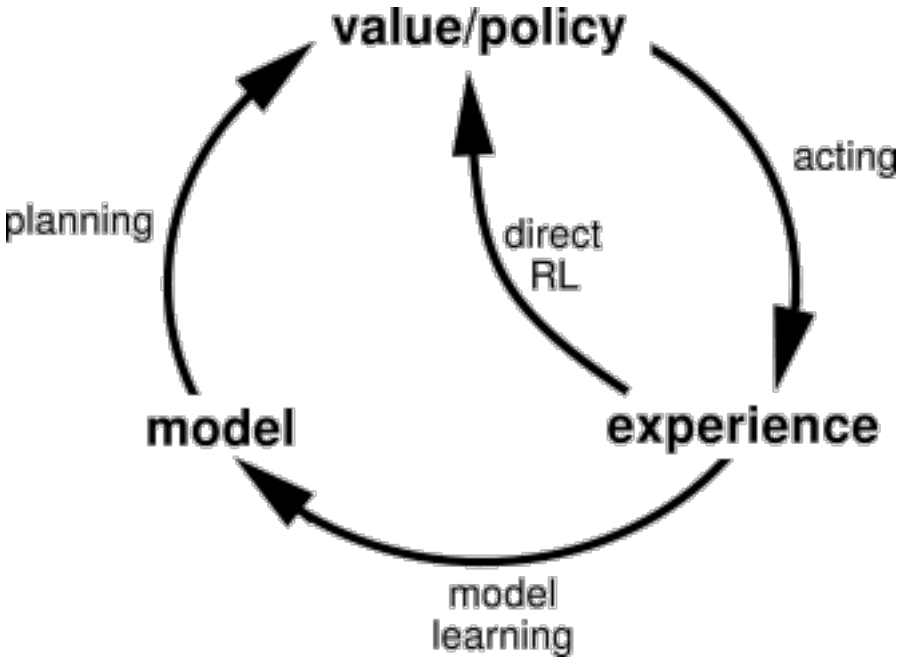


Figure 2.6: Iteration loops of model-based and model-free algorithms [25].

[11] Entropy measures how random the policy is. It can be thought of as a quantity which indicates how random something random is [45].

Given the Model

One of the most known model-based algorithms, and one that was already mentioned before, is **AlphaZero**. It is given a model of the world, and using the Monte Carlo Tree Search (MCTS) technique, evaluates the quality of the current state and plans what actions to take in the future. The neural network is the one responsible for deciding what branches of the tree to expand. This procedure is shown in figure 2.7.

Not only is AlphaZero capable of beating other AIs at Chess, Go and Shogi, it does so while searching through significantly less possible outcomes, as figure 2.8 illustrates.

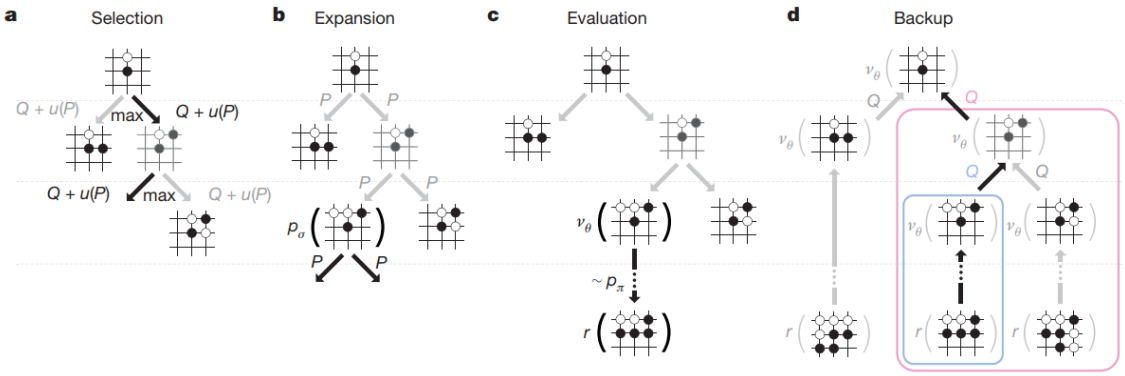


Figure 2.7: Application of MCTS in AlphaGo [16].

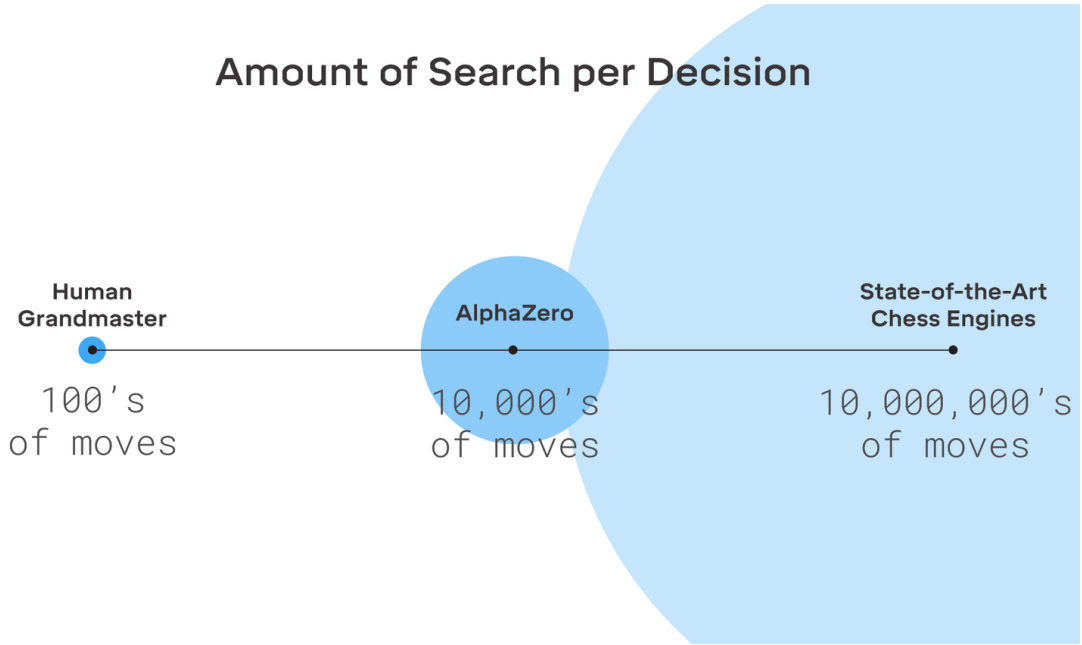


Figure 2.8: Comparison of search depth between AlphaZero and other chess engines [46].

Learn the Model

Another type of model-based methods are the ones that learn the model. Instead of being given a complete model of the world from the start, these methods learn the model as they experience the world through trial-and-error.

I2A One particularly interesting algorithm in this field is Imagination-Augmented Agents (I2A), proposed by DeepMind [47]. This algorithm combines a model-based approach with a model-free one. By embedding the agent with an imagination component (the model-based part), the agent is capable of planning and deciding on future actions. However, if the constructed model ever turns out to be incorrect, the agent learns to ignore it, following a model-free approach from then on. This solves the bias problem typically associated with model-based approaches. Because the agent is not fully dependent on the model, it can adapt to unexpected situations. On the other hand, because the agent is not solely reliant on trial-and-error, it can use its imagination to plan ahead, thus avoiding performing irreversible actions.

2.2 Hierarchical Reinforcement Learning

Another interesting field of ML is Hierarchical Reinforcement Learning (HRL). These algorithms learn from multiple policy layers. The high level policies (or controllers) are responsible for the planning and operate on larger timescales. Meanwhile, the low level controllers are responsible for outputting environmental atomic actions. This approach is rather interesting because it's similar to how humans operate. For example, the action of dribbling a soccer ball can be divided into multiple task: find the goal, plan the trajectory, move the legs, and so on. In this case, the high level policy would be responsible for the trajectory planning, and the low level policy would be responsible for moving the agent's body.

A key feature of this is that the controllers can be swapped at any time. If instead of wanting the agent to push the ball while running, we want it to push it while crawling, we could change the low level controller to a more appropriate one, and the high level controller would still work, because the skills that it learnt are not dependant on the low level policies.

HIRO Hierarchical Reinforcement Learning with off-policy correction (HIRO) is a recent approach to HRL, proposed by Google Brain in 2018 [48]. The basic idea is that the high-level policy will give goal states to the low-level policy. The low-level policy will then be rewarded based on whether or not it was able to perform actions upon the environment that successfully replicate the given state. Because it uses a variant of DDPG with off-policy training, HIRO is extremely efficient and outperforms other approaches [48].

2.3 Evolution Strategies

Evolution Strategies (ES) are an optimisation technique inspired by Darwin's natural selection. A population composed of different individuals is sampled, where those who are fitter (i.e. have

higher fitness value) are selected to form the next generation. In other words, the more successful the individual, the higher the probability of it dictating the distribution of the following generations [49].

Mathematically, they are a *black-box optimiser*. They receive a set of numbers that describe the parameters of the problem as input, and output their respective fitness.

The algorithm starts by creating a population of random individuals. Then, it calculates the fitness of each of these individuals and associates a probably to each - the higher the fitness value, the more likely for it to be *selected*. Next, the individuals that were selected are *crossed over*, creating the next generation. Finally, there's a possibility of some of these new individuals being *mutated*, becoming a different individual.

CMA-ES Covariance Matrix Adaptation Evolution Strategy (CMA-ES) reduces the time complexity of the standard evolutionary algorithm. That is, it reduces the amount of generations needed to achieve a good solution. This method is also highly parallelisable, meaning it scales extremely well with computational power [50]. Nowadays, it is one of the standard optimisation algorithms and is used in a large number of optimisation problems, including RoboCup.

2.4 Machine Learning in RoboCup

The presence of Deep Reinforcement Learning is still not heavily felt in today's RoboCup. Much of current state of the art uses Evolutionary and Genetic algorithms, and though Reinforcement Learning is sometimes found in literature, Deep Reinforcement Learning is not. Especially regarding walking and dribbling, there is not much literature to review, as it seems like the scientific community is still focused on kicking [51], [52], keepaway [53] and object detection [54], [55].

DeepLoco In the article Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning, a combination of HRL with DRL is proposed in order to learn biped locomotion skills. An actor-critic algorithm is used to train both the high-level policy and the low-level policy. The low-level policy was able to learn different walking styles, while the high-level policy was able to learn how to correctly draw trajectories, independently of the low-level controller that was in use. The agent also learned to keep itself balanced even when pushed on multiple sides by external forces, making it extremely stable. Though not directly related to RoboCup, this work was demonstrated to work when pushing a soccer ball[12], so testing its performance with the Nao robots in the 3D Simulation League environment would be a very interesting addition to the project.

Overlapping Layered Learning Published in 2018, this paper describes a new HRL approach used by the UT Austin Villa 2014 RoboCup 3D simulation team in order to learn multiple skills [56]. Using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm, they were able to optimise over 500 parameters and 19 behaviours, learning multiple low-level skills such

[12] Video examples of the different skills learned by the agent can be found on the project's homepage: https://www.cs.ubc.ca/~van/papers/2017-TOG-deepLoco/ (visited on 02/05/2019).

as walking and kicking. In the first layer, the *getting up* behaviour is learned. In the second layer, *walking* is optimised, and in the third layer, *running*. In the last layer, the *getting up* behaviour is re-optimised, and trained to assure that the agent is able to start walking / running as soon as it is standing up. This learning process improved the team's overall performance. During training, where the team played over 3000 games against other state-of-the-art teams, only one game was lost.

Walking Kicks This type of kicks are an extremely useful skill to learn. By being able to kick the ball without first coming to a full-stop, not only is it possible to perform the kick faster, thus giving your opponent's less time to react, but the kick is also more powerful. In 2017, both UT Austin Villa team and the magmaOffenburg team presented their version of this type of kicking. Using CMA-ES as the learning algorithm, magmaOffenburg team was able to create an agent that learns how to use its toes, thus increasing the performance of the kick by 30% [57]. Also using CMA-ES, UT Austin Villa team was able to significantly shorten the amount of time it takes to perform a kick - a kick that would previously take 2s to perform would now execute under 0.25s [58].

Walking Fast Yet another approach to learning that utilises CMA-ES is presented by the FC Portugal 3D Simulation team [59]. By optimising the parameters of the modelled hip height movement, the agent was able to learn both a fast forward walk as well as a fast side walk. The developed walking engine was also tested on a real-life Nao robot with successful results - the walk speed increased from the standard 0.119 m/s to 0.34 m/s. An overview of the followed approach and all of its components can be seen on figure 2.9.

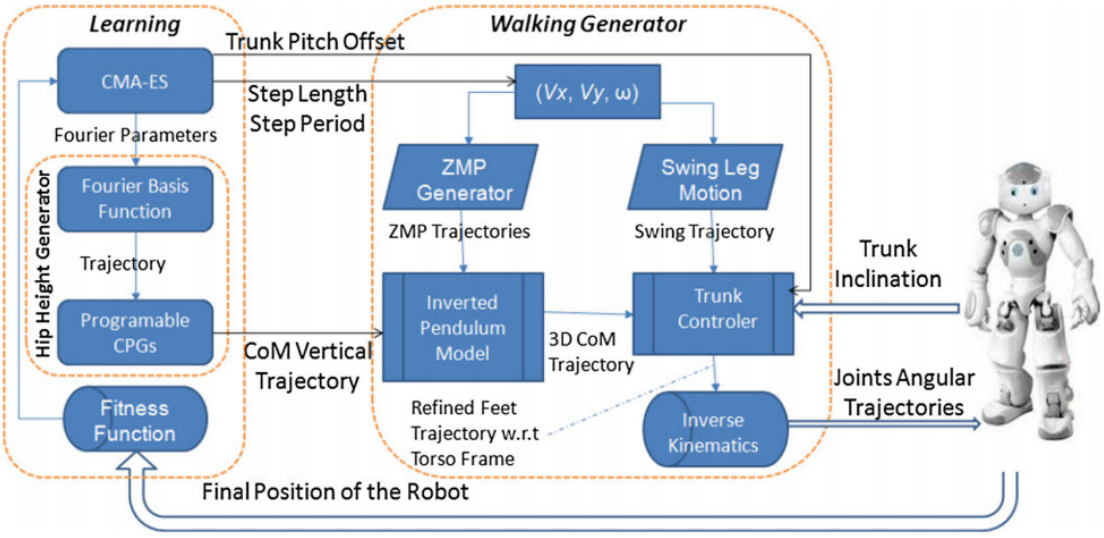


Figure 2.9: Overview of FC Portugal 3D Simulation team's approach to walking fast [59].

Running A very recent breakthrough in the area came from FC Portugal. By optimising a Neural Network with PPO, the agent was able to learn how to run. Until now, robots were not

able to run very fast. They simply *walked* faster, at a rate of approximately 1 meter per second. Through Deep Reinforcement Learning, the agent learned to run, with human-like motion, at more than double the speed of the best teams [60]. This marked a very important step for RoboCup, and will surely impact future competitions and help advance current state of the art.

2.5 Conclusion

In this chapter, several ML algorithms were reviewed, with particular focus on DRL methods. A summary of their characteristics can be found on table 2.1.

Every year new algorithms and techniques are proposed in the area, and as mentioned in the opening of this chapter, it is still a growing field with much to offer.

Especially Model-free methods (or the ones that learn the model) are interesting, as they can later be generalised and expanded to other fields outside of robotic soccer.

Algorithm	Type	Model	Policy	Action Space
DQN	DRL	Free	Off	Discrete
VPG	DRL	Free	On	Continuous
A3C / A2C	DRL	Free	On	Continuous
PPO	DRL	Free	On	Continuous
DDPG	DRL	Free	Off	Continuous
TD3	DRL	Free	Off	Continuous
SAC	DRL	Free	Off	Continuous
AlphaZero	DRL	Based	-	Discrete
I2A	DRL	Based	-	Continuous
HIRO	HRL	-	Off	Continuous

Table 2.1: Characteristics of different RL algorithms.

Chapter 3

RoboCup Environment

This chapter presents the overall environment used in the RoboCup 3D Simulation League and goes over each of its components: the physics simulator, SimSpark; the agent, NAO Robot and the 3D viewer, RoboViz.

It also covers the software and hardware setup for the RoboCup 2019 competition, as well as new rules that were recently introduced.

3.1 Simspark

SimSpark[1] is a 3D physics simulator for multi-agent systems, and is the official simulation server of the RoboCup 3D Simulation League. Unlike other specialised simulators, SimSpark is flexible and generic enough to allow for the creation and execution of non-soccer simulations, where custom simulation environments can be created using its scene description language [61].

The simulated soccer environment has the following characteristics [62]:

- Pitch: 30 by 20 meters.
- Goal: 2.1 by 0.6 meters, with a height of 0.8 meters.
- Penalty Area: 6.0 by 1.8 meters.
- Centre Circle[2]: radius of 2 meters.
- Soccer Ball: radius of 0.04 meters and mass of 26 grams.
- Markers: 1 placed in each corner and 1 in each goal post.

[1] SimSpark Repository: https://gitlab.com/robocup-sim/SimSpark (visited on 06/18/2019).

[2] Although visually represented as a circle in figure 3.1, this shape is modelled as a polygon made up of ten straight lines, and thus perceived by the agents as a decagon.

Figure 3.1 illustrates the dimensions of the soccer field, as well as the position of the markers. Because each of these markers has a unique ID and is placed in a fixed spot, by observing the field lines alongside a subset of these markers, the agents are able to locate themselves.

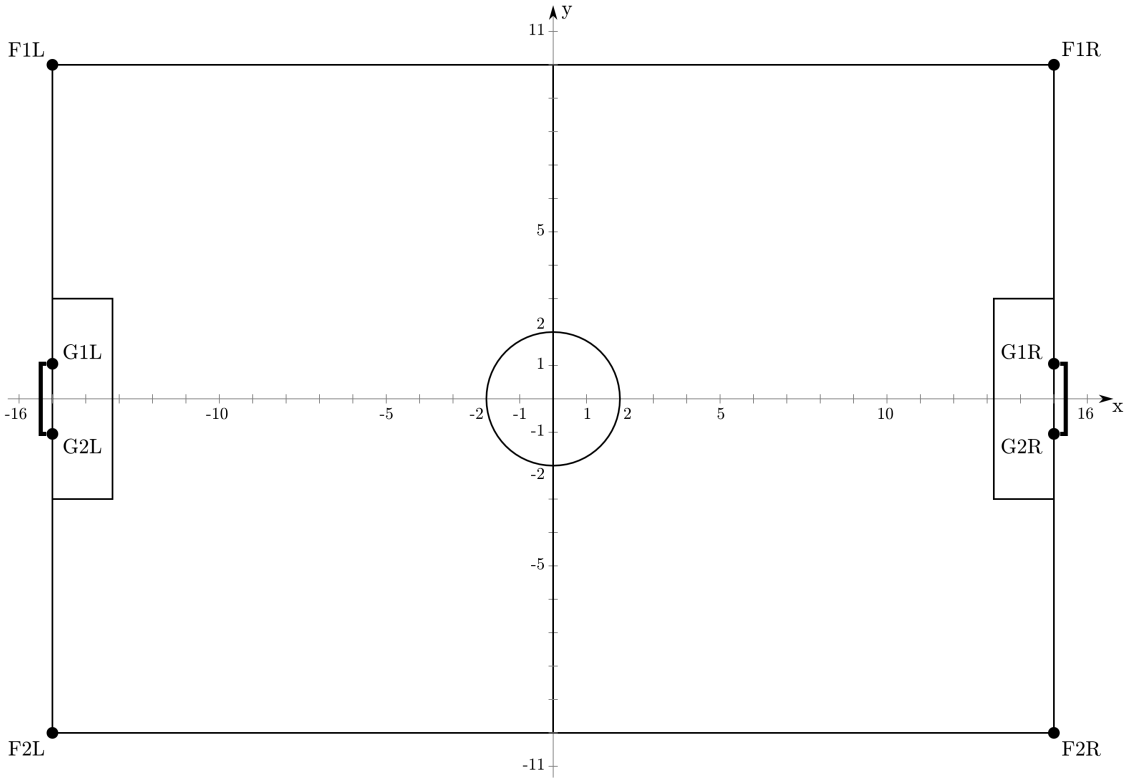


Figure 3.1: Simulated soccer field [62].

3.1.1 Competition Environment

In official competitions, each team is assigned a client in order to run the agents. Each of these agents should then connect to a proxy, which, in turn, connects to the simulation server. Finally, a monitor connects to the server in order to render the game and allow its visualisation. This setup architecture can be seen in figure 3.2.

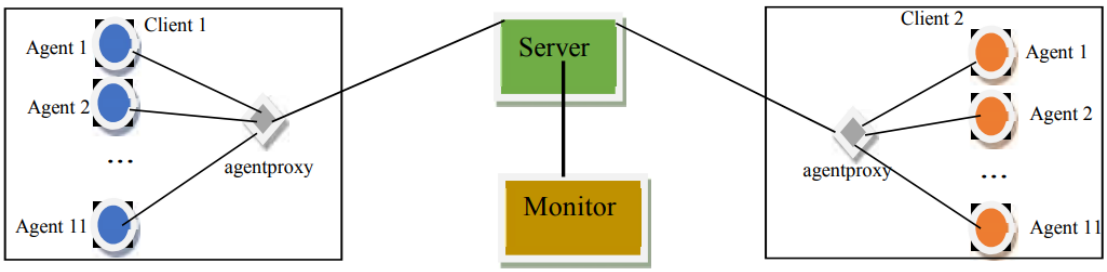


Figure 3.2: Competition environment setup of RoboCup 2019 [63].

3.1.2 Optimisation Environment

In order to increase the performance of optimisation procedures, it is necessary to slightly change some configuration files of SimSpark.

Namely, it is very important to toggle off *real-time mode*. Otherwise, it will never be possible to fully utilise the CPU.

It is also important to toggle on *agent sync mode*. This avoids wasting server cycles, forcing the server to wait for the agent commands before proceeding to the next cycle.

Finally, it is also useful to disable general soccer rules, turn off noise and allow the agent to accurately perceive its position.

Figure 3.3 shows what configuration files and settings to change in order to obtain a working optimisation environment.

```
/usr/local/share/rcssserver3d/rcssserver3d.rb
    $enableRealTimeMode = false
/usr/local/share/rcssserver3d/naosoccersim.rb
    addSoccerVar('BeamNoiseXY',0.0)
    addSoccerVar('BeamNoiseAngle',0.0)
    #gameControlServer.initControlAspect('SoccerRuleAspect')
/usr/local/share/rcssserver3d/rsg/agent/nao/naoneckhead.rsg
    (setViewCones 360 360) ;
    (setSenseMyPos true) ;
    (setSenseMyOrien true)
    (setSenseBallPos true) ;
    (addNoise false)
~/.simspark/spark.rb
    $agentSyncMode = true
```

Figure 3.3: SimSpark settings for optimisation procedures.

3.2 RoboViz

RoboViz[3] is the official monitor and visualisation tool of the RoboCup 3D Simulation League. It is actively developed and maintained by the *magmaOffenburg* team, one of usual contenders in RoboCup, and the winners of the Robotica 2019 competition, held in Portugal. Besides the 3D rendering of the environment and the agents, this tool also allows for the rendering of simple drawings, such as lines and circles. This feature is very useful when debugging or implementing new strategies, as it allows for quick visualisation of pathing trajectories, agent's perceived position of opponents and the ball, etc. Figure 3.4 shows how the interface of RoboViz looks like.

[3] RoboViz Repository: https://github.com/magmaOffenburg/RoboViz (visited on 06/18/2019).

Figure 3.4: RoboViz interface.

3.3 NAO Robot

The official agent used in RoboCup competitions is a simulated model of the NAO robot[4]. Figure 3.5 shows how both the real and simulated NAOs look like.

Figure 3.5: Side by side comparison of the real and simulated NAO robot [64].

The characteristics of the simulated NAO model are the following [64]:

- Height: 57 centimetres.
- Weight: 4.5 kilograms.
- Degrees of Freedom: 22 movable joints, illustrated on figure 3.6.

[4] SoftBank Robotics' NAO robot: https://www.softbankrobotics.com/emea/en/nao (visited on 01/13/2019).

- Gyroscopes: 1 in the torso.
- Accelerometers: 1 in the torso.
- Force Resistance Preceptors: 1 in each foot.
- Vision Preceptors: 1 in the centre of the head.
- Communication: 1 say effector and 1 hear preceptor.
- Movement: 1 hinge joint preceptor and 1 hinge joint effector for each of the 22 available joints.
- Game State: 1 gamestate preceptor used to capture the current state of the game.

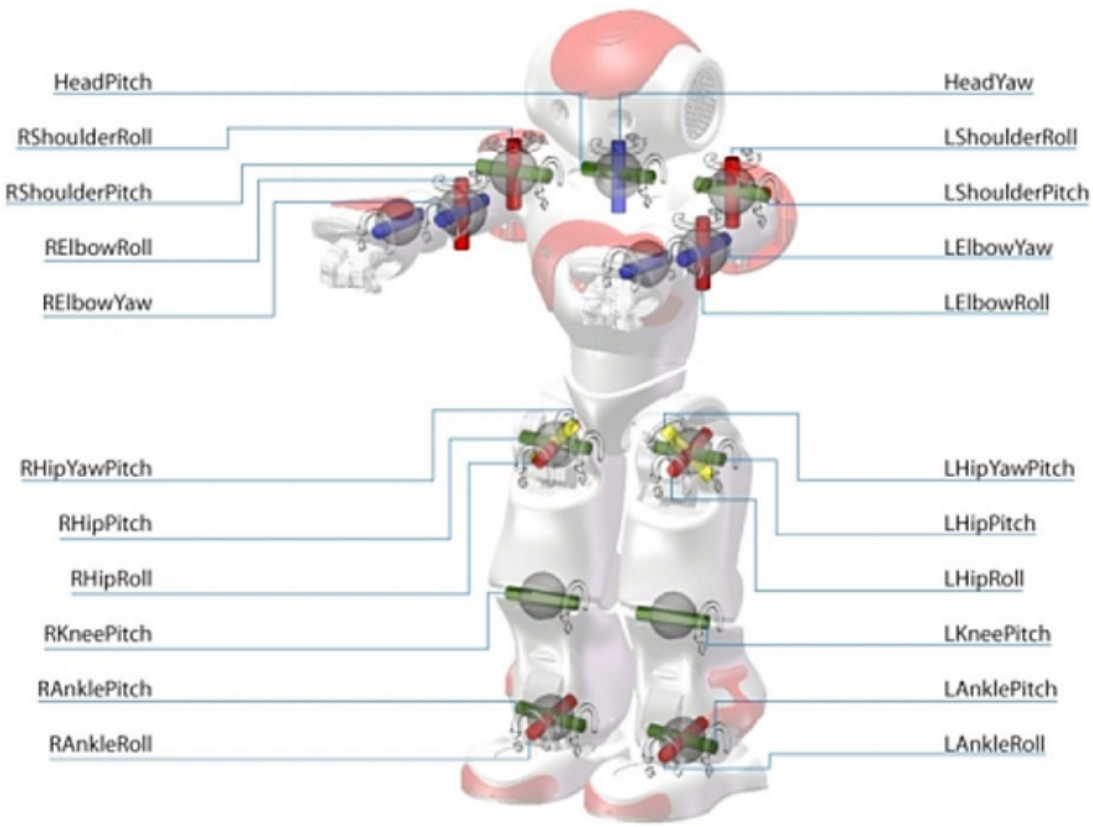


Figure 3.6: Joints and actuators of the NAO robot [60].

A more detailed illustration of the robot's dimensions can be seen on figure 3.7.

3.4 2019 Rules

The basic rules of robotic soccer are the same as the official FIFA rules, the major difference being the automatic AI referee. This prevents any subjective errors or biases, ensuring the fairness of each match.

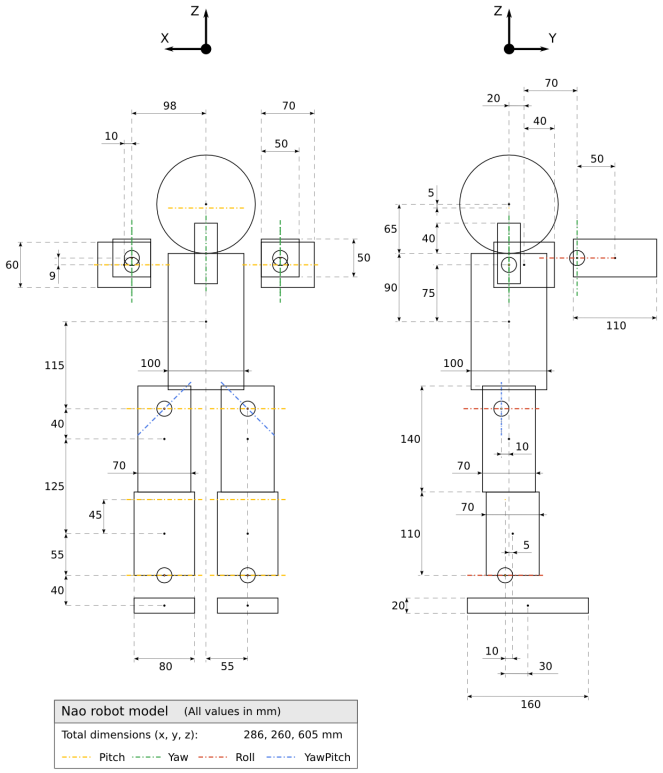


Figure 3.7: Box model of the NAO robot [64].

Nevertheless, the robotic rules are still far from the FIFA standard, and every year new rules are added in order to address this. In an effort to further close the gap between real soccer matches and the RoboCup 3D Simulation League, new rules have been added for the 2019 competition. These rules were previously tested during the Robotica 2019 competition, held in Portugal, and due to the positive feedback received from the participants, they have been included in this year's RoboCup.

3.4.1 Pass Command

This new rules allows an agent to send a request to the server, stating that it wants to pass the ball to a teammate. This prevents the opponents from getting close to the agent during a short time window, allowing it to safely execute the pass.

Up until now, matches mostly consisted of long range kicks, which resulted in a lack of strategic gameplay. By incentivising the usage of passes, and thus increasing the range of available tactics, this new rule hopes to increase importance of the strategic elements of the game, as teams capable of properly executing the pass command will surely have an advantage over their opponents.

3.4.2 Self-Collisions

Another important addition to this year's rules are the self-collisions fouls. These occur when a body part of a robot trespasses another body part of the same robot (e.g. each leg going *through* the other, as can be seen on figure 3.8), leading to largely unnatural behaviours [63].

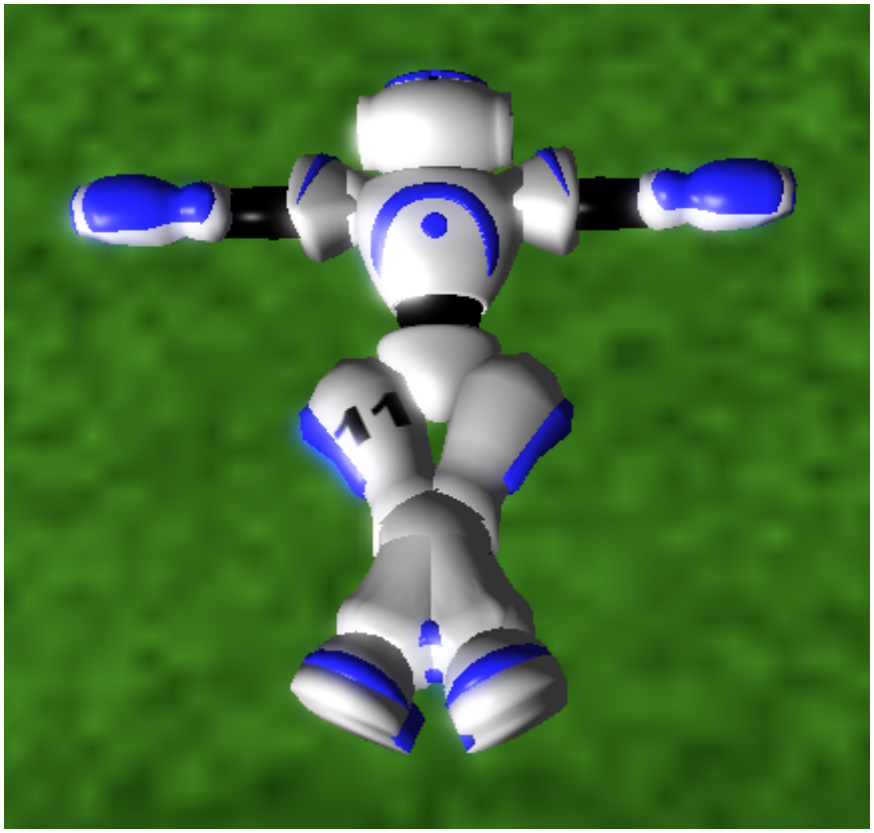


Figure 3.8: Self-collision between both legs of the robot.

When such a foul occurs, the joints involved will be temporarily disabled, meaning the agent will not be capable of moving them. In the example illustrated on figure 3.8, the agent would become unable to move both legs for a short period of time following the foul.

Also, in order to further incentivise teams to avoid self-collisions, a *challenge* will be running throughout the event, where the winner is the team with the least amount of (averaged) self-collisions per game.

Because this rule has a major impact on teams' performance (i.e. it can make entire agents' behaviours and strategies useless, forcing them to be re-optimised or even removed), a tolerance of 4 millimetres was implemented. This means that agents are allowed to have self-collisions up until that threshold without suffering the associated penalty. As such, for RoboCup 2019, only very unnatural behaviours will be penalised.

Nevertheless, the goal is to reach a point where self-collisions can no longer happen, and thus the above mentioned tolerance is likely to be lowered for future events.

3.4.3 Crouching Block

The final rule added this year consists of preventing robots from blocking the opponents by crouching or purposely falling on the ground.

This behaviour was mostly used by FC Portugal during Robotica 2019, as part of a new kick off strategy. The player performing the kick off would make a pass to a teammate located in the back, followed by the rest of the team crouching in front of the agent that performs the kick. This prevents opponents from stealing the ball, thus allowing the agent to safely prepare a powerful long kick aimed at the goal. This strategy resulted in multiple opening goals for the team during the event. Figure 3.9 contains a screenshot taken from one of the games between FC Portugal and magmaOffenburg that took place during Robotica 2019, showcasing this strategy.

Figure 3.9: Kick off strategy used by FC Portugal during the Robotica 2019 competition.

Such tactics were later deemed unfair, and so, for this year's RoboCup, a new rule was installed to prevent this. If players purposely crouch down on the ground with legs sprayed out, blocking opponents while not playing the ball, a free kick will be assigned to the opposing team. This judgement will be performed by the human referee at the sight [63].

3.5 Conclusion

This chapter presented the work environment used in this dissertation. SimSpark is the official simulation server, RoboViz the official 3D viewer and the NAO robot the official model.

In order to be able to optimise behaviours, it is necessary to do some changes to the configuration files of the server. These were explained in subsection 3.1.2.

Finally, section 3.4 explained the new rules that were added to this year's RoboCup competition. The main one being the self-collisions, as these affect a large number of behaviours, requiring their re-optimisation.

Chapter 4

Optimisation Framework

During the earlier stages of this project, focus was mainly directed towards the design and implementation of a framework that would be generic enough to match not only this dissertation's needs, but also allow future optimisations for the team.

The underlying framework was a left over artefact from previous optimisation attempts by the FC Portugal team. As such, it seemed only natural to use it as the foundation for this work. It follows the state-of-the-art reinforcement learning standard toolkit created by OpenAI, Gym[1].

Amongst the multiple improvements and bug fixes that were added to the original framework, one of the most important was its parallelisation. This allows it to take advantage of the multiple cores of a modern CPU, drastically improving optimisation times.

This chapter presents an overview of said framework and each of its components.

4.1 Deep Agent

The Deep Agent consist of an FC Portugal regular agent, stripped of its thinking module. In its place, a communication loop was implemented, where instead of thinking on its own, the agent receives orders from the *optimiser*.

The optimisation procedure is illustrated in figure 4.1. It starts with the creation of N Gym environments in parallel, each launching a SimSpark instance (presented in section 3.1) with a Deep Agent running in it. The communication between the Deep Agent and the optimiser is performed through a TCP socket. An *episode* starts with the optimiser sending a set of joint positions to the agent - the *action*. Then, the agent executes the given action and informs the optimiser about the new state of the simulation - the *observation*. Finally, the optimiser evaluates the previous actions based on the resulting observation. The cycle then repeats itself, with the optimiser sending a new set of actions to the agent. This process is running in parallel in each of the N environments, meaning there are always N episodes being executed at the same time.

[1] OpenAI Gym Documentation: https://gym.openai.com/docs/ (visited on 06/20/2019).

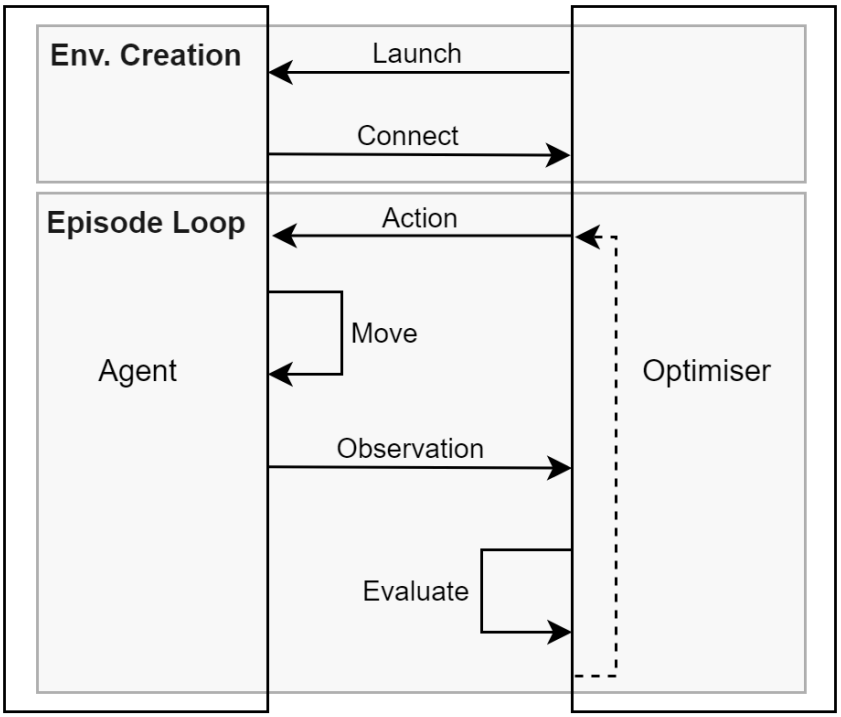


Figure 4.1: Communication between the agent and the optimiser.

4.1.1 Observation Space

The observation space is problem-specific. For example, if we're optimising a kick, we will need to observe the ball position, in order to know how far we were able to shoot it. However, if we're optimising, for example, a running behaviour, then the ball position is irrelevant.

Nevertheless, there are some characteristics of the environment that should always be observed, independently from the optimisation problem. These are:

- The state of each of the 22 joints.
- The gyroscope.
- The accelerometer.

The limits of the observations regarding the joints can be seen on table 4.1.

4.1.2 Action Space

The action space is made of the possible actions the agent can take. They can be seen as variables over which the optimiser has control. Thus, unlike the observation space, the action space is always the same for all optimisation scenarios. This is because the only controllable variables are the robot's joints[2]. Figure 4.2 shows all of the joints of the robot and their respective identifiers.

[2] In most cases, the head and neck joints are not used in optimisation problems, and instead are kept in a fixed position throughout. Therefore, for most scenarios, only 20 out of the 22 joints are optimised.

Joint	Min. Value	Max. Value
lleg1	-90	1
rleg1	-90	1
lleg2	-25	45
rleg2	-45	25
lleg3	-25	100
rleg3	-25	100
lleg4	-130	1
rleg4	-130	1
lleg5	-45	75
rleg5	-45	75
lleg6	-45	25
rleg6	-25	45
larm1	-120	120
rarm1	-120	120
larm2	-1	95
rarm2	-95	1
larm3	-120	120
rarm3	-120	120
larm4	-90	1
rarm4	-1	90

Table 4.1: Joints observation space.

4.2 Slot Behaviour Optimiser

In FC Portugal, behaviours are defined following a key-frame approach [66], [67], with each key-frame being called a *slot*. Each slot defines how the agent's joints are at a given moment. The interpolation of two key-frames and the new joint positions are calculated through the use of a sinusoidal function. Figure 4.3 shows how slot behaviours are defined, with figure 4.4 showing the resulting robot positions. This example is composed of two slots: one where the agent stands still, and one where the agent opens its arms over the course of 1 second.

In terms of optimisation, each slot consists of 21 parameters: all of the robot's joints (minus the neck and the head, as explained on subsection 4.1.2) plus the duration of the key-frame, *delta*.

Thus, the first thing to do when optimising a behaviour, is to decide how many slots the behaviour will be made of. This is important in order to know how many parameters are to be optimised. For example, for a behaviour consisting of 3 slots, there are $(3 \times 21 =)$ 63 parameters to optimise.

This type of optimisation is performed through the use of Evolution Strategies. Namely, the CMA-ES algorithm is the one used for all optimisations regarding slot behaviours.

At first, a set of solutions is randomly generated by the algorithm. Then, an episode is run for each of those samples in order to evaluate them, following the episode loop shown in figure 4.1. Because slot behaviours follow a key-frame approach, the entire set of actions to perform (i.e. all slots of the behaviour) are sent at the start of an episode. Thus, only one observation is recorded

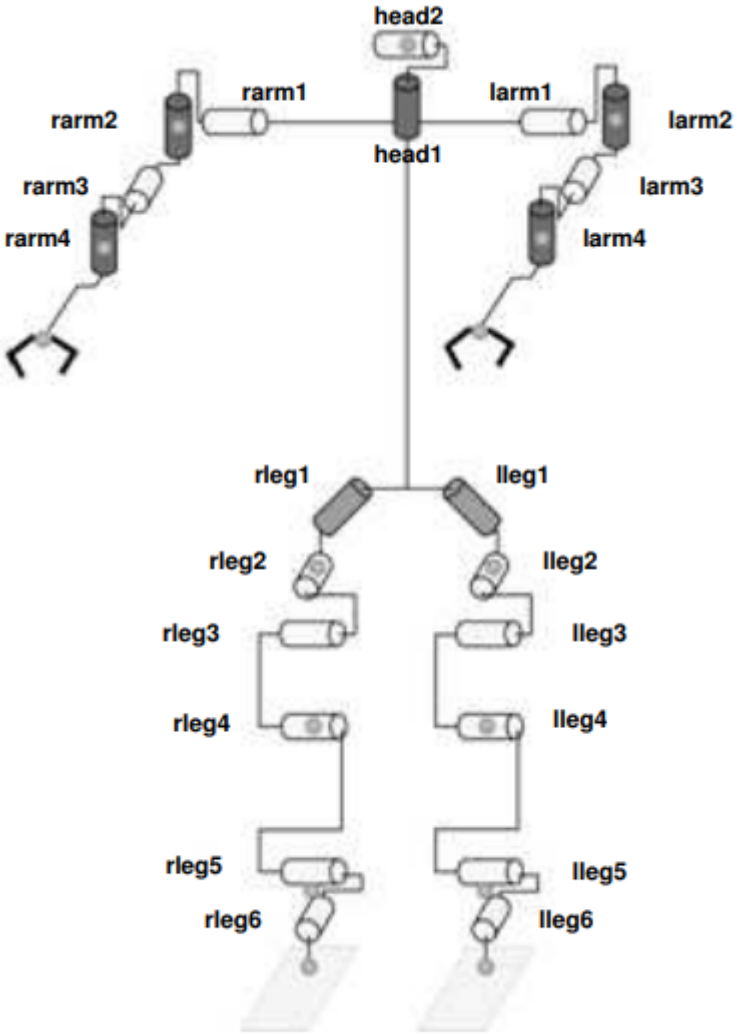


Figure 4.2: Joints of the NAO robot and their identifiers [65].

- the final state, after the behaviour has finished executing. Once this observation is received, the optimiser calculates the fitness value (also known as reward) of the current sample and proceeds to the next one. When all samples of a given generation have been evaluated, the next generation is created based on the best individuals from the previous generation. This process repeats until convergence is achieved and it is no longer possible to improve the solution.

Furthermore, due to the noisy environment, the fitness of a given sample must be averaged over a number of trials. We have assumed the reward distribution of our behaviours to be a Normal Distribution $\mathcal{N}(\theta, \sigma)$. We can then calculate the standard error of the mean $\sigma_e = \frac{\sigma}{\sqrt{n}}$, where n is the number of trials used to evaluate the sample's fitness. We now have a trade-off between accuracy of our evaluation, which increases with the amount of trials, and the speed at which each sample is evaluated.

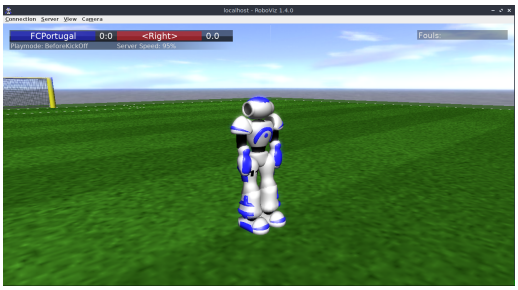
```
<behavior name="ArmsWideOpen" type="SlotBehavior">
    <slot name="Standing" delta="0.5">
        <move id="&lleg1;" angle="0"/>
        <move id="&rleg1;" angle="0"/>
        <move id="&lleg2;" angle="0"/>
        <move id="&rleg2;" angle="0"/>
        <move id="&lleg3;" angle="0"/>
        <move id="&rleg3;" angle="0"/>
        <move id="&lleg4;" angle="0"/>
        <move id="&rleg4;" angle="0"/>
        <move id="&lleg5;" angle="0"/>
        <move id="&rleg5;" angle="0"/>
        <move id="&lleg6;" angle="0"/>
        <move id="&rleg6;" angle="0"/>
        <move id="&larm1;" angle="-90"/>
        <move id="&rarm1;" angle="-90"/>
        <move id="&larm2;" angle="0"/>
        <move id="&rarm2;" angle="0"/>
        <move id="&larm3;" angle="0"/>
        <move id="&rarm3;" angle="0"/>
        <move id="&larm4;" angle="0"/>
        <move id="&rarm4;" angle="0"/>
    </slot>
```

(a) First slot, corresponding to the initial position.

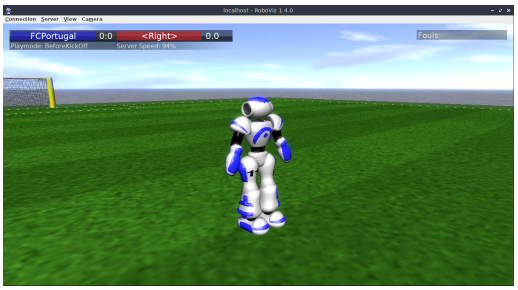
```
    <slot name="ArmsOpen" delta="1">
        <move id="&lleg1;" angle="0"/>
        <move id="&rleg1;" angle="0"/>
        <move id="&lleg2;" angle="0"/>
        <move id="&rleg2;" angle="0"/>
        <move id="&lleg3;" angle="0"/>
        <move id="&rleg3;" angle="0"/>
        <move id="&lleg4;" angle="0"/>
        <move id="&rleg4;" angle="0"/>
        <move id="&lleg5;" angle="0"/>
        <move id="&rleg5;" angle="0"/>
        <move id="&lleg6;" angle="0"/>
        <move id="&rleg6;" angle="0"/>
        <move id="&larm1;" angle="-90"/>
        <move id="&rarm1;" angle="-90"/>
        <move id="&larm2;" angle="90"/>
        <move id="&rarm2;" angle="-90"/>
        <move id="&larm3;" angle="0"/>
        <move id="&rarm3;" angle="0"/>
        <move id="&larm4;" angle="0"/>
        <move id="&rarm4;" angle="0"/>
    </slot>
</behavior>
```

(b) Second slot, corresponding to the final position.

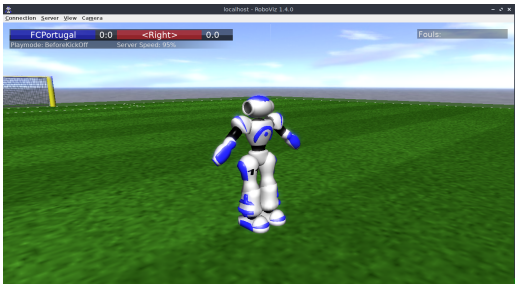
Figure 4.3: Example of a Slot Behaviour.



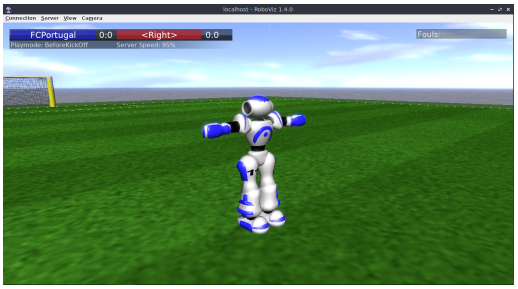
(a) Initial position, corresponding to the first slot.



(b) First in-between position, interpolated through the use of a sinusoidal function.



(c) Second in-between position, interpolated through the use of a sinusoidal function.



(d) Final position, corresponding to the second slot.

Figure 4.4: Resulting positions from the slot behaviour shown in figure 4.3.

4.3 Neural Behaviour Optimiser

Neural Behaviours follow a slightly different approach to the one explained in section 4.2. Instead of giving the agent a few key-frames and then letting it calculate the in-between positions, Neural Behaviours tell the agent how to move in every single *timestep*. Thus, Neural Networks are a perfect fit for these behaviours - hence the name, Neural Behaviours. Also, because the framework uses the Gym toolkit, any algorithm contained in the OpenAI Baselines[3] can be easily used for learning.

First, the observation (see subsection 4.1.1) recorded by the agent is given as input to the network. Based on the state of the environment, the network then outputs an action (see subsection 4.1.2), which is passed on to the agent. Once the agent has performed the given action, a new observation is recorded and once again input to the network, and the cycle repeats. After N timesteps, the network is updated with the evaluation results of the previous actions. Once convergence is achieved and it is no longer possible to improve the solution, the optimisation stops.

4.4 Conclusion

This chapter presented the framework that was developed during the early stages of this project. It makes use of the OpenAI Gym toolkit, and thus is able to run any of state-of-the-art algorithms included in OpenAI Baselines. It also allows for the parallelisation of the optimisation procedure, thus taking advantage of all the cores of a modern CPU.

The framework supports 2 types of behaviours: Slot Behaviours and Neural Behaviours. While for Slot Behaviours key-frames are optimised, for Neural Behaviours a Neural Network is used in order to calculate the best action at each timestep, according to the observation of the environment. Nevertheless, they both follow the same episode loop shown in figure 4.1.

[3] OpenAI Baselines Repository: `https://github.com/openai/baselines` (visited on 06/23/2019).

Chapter 5

Getup Optimisation

A behaviour that allows agents to stand up after having fallen, which we call a *getup*, is one of the key behaviours in humanoid robotic soccer competitions. Because soccer is a sport of constant physical contact, agents fall frequently, and even champion teams are not yet capable of reliably playing a match without falling. For example, it is quite common to see agents fall after performing a powerful kick or when trying to run fast. Standing up as soon as possible after such falls is of the utmost importance, as an agent that is lying on the ground is not capable of positively impacting the state of the game.

5.1 Problem

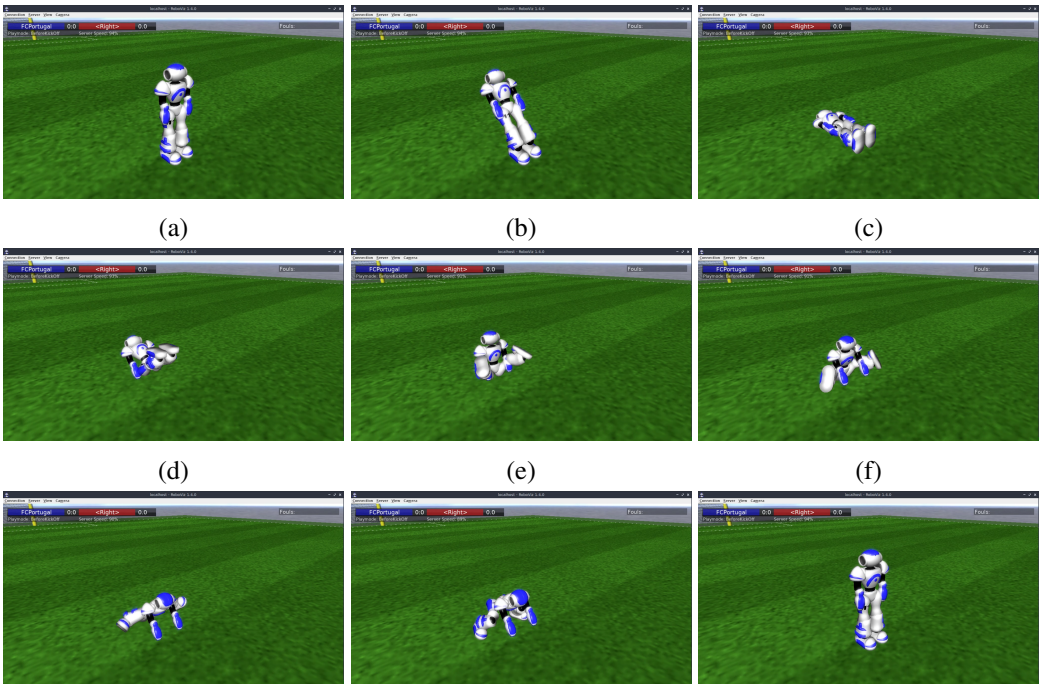
The current getup behaviours used by the FC Portugal team have a major flaw - they disregard self-collisions. This means an agent is capable of, for example, moving its arms directly through its legs, as can be seen in figure 5.1e. As stated in subsection 3.4.2, new rules are now making such movements illegal and current behaviours must be re-optimised, taking into account the new circumstances.

5.2 Observation Space

In addition to the common observations explained in section 4.1.1, for the getup scenario, it is also necessary to observe the robot's height. The agent's height is measured in meters, with a value of 0.54 when standing and 0.06 when laying on the ground. Thus, during an optimisation episode,

$$h \in [0.06, 0.54], \tag{5.1}$$

where h represents the robot's height.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.1: Original getup behaviour.

5.3 Episode Layout

The process of evaluating the parameters of a getup has the following steps:

1. The agent purposefully falls to the ground.
2. The agent tries to get up using the parameters optimised by the algorithm.
3. Once the agent has stopped moving and its position is stable, the fitness of the parameters is evaluated.

The last step is the most crucial one, and requires a good evaluation function, or the agent will not be able to learn how to stand up. Figure 5.1 shows a complete optimisation episode performed using the original getup behaviour containing self-collisions.

5.4 Reward Shaping

Usually, there is an evident reward to any optimisation problem. Using the getup scenario explained in this chapter as an example, an obvious reward function would be

$$f = \begin{cases} 0 & \text{if not standing} \\ 1 & \text{if standing} \end{cases}. \tag{5.2}$$

However, the agent would most likely fail to learn how to stand up from such a reward function. This is because the sequence of actions required for the robot to stand up is far too large for the agent to randomly guess it. Most likely, the agent would simply receive a reward of 0 over and over, indicating all actions were equally bad. This is why reward shaping is important.

Reward shaping is a technique by which the natural reward signal is augmented with additional rewards, in order to increase the progress towards a good solution [68]. In this example, a much better reward function would be

$$f = h, \tag{5.3}$$

where h represents the current height of the robot. When compared to the previous function, this one provides a lot more information about the current state, letting the agent better distinguish between good and bad actions.

5.4.1 Slot Behaviour Optimisation

During the first experiments, the fitness function was rather simple, and only took into account the final height of the agent. The fitness value f was given by

$$f = \frac{min\,(0.54, h_f)}{0.54}, \tag{5.4}$$

where h_f represents the agent's final height, and 0.54 its standing height, such that f is normalised between 0 and 1. The intuition behind this is to reward the agent based on how well it stood up from a fallen position.

However, any action taken during the behaviour would be rewarded solely based on the end result, ignoring factors like the time taken. Since speed is valuable, f was then improved by integrating time:

$$f = \begin{cases} \frac{h}{0.54} & h_f < 0.54 \\ 1 + \frac{3}{\Delta_t} & h_f \geq 0.54 \end{cases}, \tag{5.5}$$

where Δ_t represents the time taken for the agent to stand up, from the moment it fell on the ground. The new formula can be thought of as a two-part learning process. First, the agent learns how to get up successfully, no matter how long it takes. And then, it learns to do so as fast as possible. In order to make reading the results easier, the goal for Δ_t was set as 3 seconds. Thus, a fitness value $f < 1$ means the agent did not manage to get up, a value $1 < f < 2$ means the agent managed to get up but took more than three seconds, and a value $f > 2$ means the agent managed to stand up in less time.

To further penalise slow behaviours, we also set a maximum threshold for Δ_t of ten seconds, where

$$f = \begin{cases} \frac{10}{\Delta_t} & \Delta_t > 10 \\ 1 + \frac{h}{0.54} & \Delta_t \leq 10, h_f < 0.54 \\ 2 + \frac{3}{\Delta_t} & \Delta_t \leq 10, h_f \geq 0.54 \end{cases}. \tag{5.6}$$

With this, the agent's first priority is to move in a way that takes less than ten seconds to complete. However, we quickly noticed that rewarding the agent based solely on the end result was not enough for it to consistently learn how to get up. Not only did it take the agent hundreds of generations to get up successfully for the first time, but sometimes it did not even manage to get up at all. Often times, the agent would prioritise a stable final position, even if it meant it was not standing up.

In order to address this problem, we drew inspiration from Q-Learning [69], a classic reinforcement learning algorithm. In Q-Learning, future rewards are discounted by a factor γ, and in doing so, their importance is incorporated into current rewards. In our case, we wanted to give more importance to heights achieved by the agent near the end of the movement, but without completely disregarding earlier heights. With this in mind, we arrived at the current fitness function

$$f = \begin{cases} \frac{10}{\Delta_t} & \Delta_t > 10 \\ 1 + \frac{\sum_{i=0}^{n} \gamma^{n-i} \times h_{n-i}}{\sum_{i=0}^{n} \gamma^{n-i} \times 0.54} & \Delta_t \leq 10, h_f < 0.54\,, \\ 2 + \frac{3}{\Delta_t} & \Delta_t \leq 10, h_f \geq 0.54 \end{cases} \tag{5.7}$$

where n stands for the number of time-steps of the behaviour, and h_i stands for the height of the agent at time-step i. The sum $\sum_{i=0}^{n} \gamma^{n-i} \times 0.54$ represents the maximum possible reward the agent could have achieved throughout the movement, if the agent measured its standing height at every single time-step. As with the previous versions of the fitness function, this is used for normalisation purposes, in order to ease readability of the fitness values.

The height of the agent is evaluated once per server time-step, with 50 recordings per second. During testing, we found a good value for the discount factor $\gamma = 0.98$. Smaller values would give too much weight to the final height, while greater values would give too much importance to initial heights, where the agent would just learn to lift its torso, but rarely to use its legs in order to reach greater heights.

5.4.2 Neural Behaviour Optimisation

When using Neural Networks to learn a getup, the first reward function we used was the same one used with the CMA-ES optimisation, equation 5.7. Because this had worked when optimising the slots, we figured it would probably work here as well. This fitness function only rewards the agent once, at the end of the episode, which creates what is known as a *sparse reward* environment.

For comparison purposes, we decided to also test a *dense reward* environment (i.e. the agent receives a reward signal every timestep, instead of just at the end of the episode), in order to see which one would have the best results. This was accomplished with the reward function

$$f_t = \frac{min\,(0.54, h_t)}{0.54}, \tag{5.8}$$

where f_t represents the reward of timestep t, h_t the agent's height at timestep t, and 0.54 its standing height, such that f_t is normalised between 0 and 1.

Unfortunately, the agent was not able to learn from this reward function, as like with previous iterations of the slot reward function, the agent would prioritise stable positions, such as sitting. Therefore, there was a need to punish the agent for constantly staying at the same height. This was accomplished with the reward function

$$f_t = \begin{cases} -1 & h_t \leq h_{best} \\ 1 & h_t > h_{best} \\ 2 & h_t = 0.54 \end{cases}, \tag{5.9}$$

where h_t stands for the normalised height (between 0 and 1) at timestep t and h_{best} stands for the best normalised height reached so far in this episode. With this fitness function, the agent gets rewarded whenever it improves the current best height, and gets punished when it does not. This teaches the agent that achieving a stable position is not good enough, and the only way to get rewarded is to keep improving. However, not all improvements should weight the same. Improving the current best height by 2Δ is obviously better than improving it by Δ, but with this reward function, both cases would be rewarded +1.

This distinction was achieved by adding the improvement Δ to the reward function, such that f_t is now given by

$$f_t = \begin{cases} -1-\Delta & h_t \leq h_{best} \\ 1+\Delta & h_t > h_{best} \\ 2 & h_t = 0.54 \end{cases}, \tag{5.10}$$

where $\Delta = abs(h_t - h_{best})$. A fitness value $-2 \leq f \leq -1$ means the agent did not improve the current best height and thus gets punished: -1 for the punishment, plus whatever amount was left to improve, $-\Delta$. So, the further it gets from the best height, the larger the punishment. A fitness value $1 < f < 2$ meas the agent was able to beat the previous best height and thus gets rewarded: $+1$ as the base reward for improving, plus whatever amount was improved, $+\Delta$. Finally, a fitness value $f = 2$ means the agent achieved the goal and is now standing up.

5.5 Results

This section shows the results obtained from the experiments regarding the getup problem presented throughout this chapter. Results for both the Slot Behaviour and the Neural Behaviour approaches are presented, with a small discussion about each.

While the Neural Behaviour tries to optimise all of the robot's joints (except the head and the neck), the Slot Behaviour only optimises half of those. By assuming the movement is symmetrical, it allows us to cut in half the amount of parameters given to CMA-ES, which significantly improves optimisation times and success rate. This reduction of the action space was not performed when

optimising with Neural Networks, as these are far more flexible, and thus should not require such restraints.

5.5.1 Slot Behaviour Optimisation

Initially, we had assumed that a high amount of slots would allow the agent to be more flexible, as it would provide a finer control of the in-between-key-frames positions. This intuition was derived from the hand-tuned solutions that the team used until now.

However, as can be seen in Table 5.1, that is not the case. In fact, the behaviour Getup A with the smaller amount of slots, 5, outperformed Getup B, the one with more slots, 10. This may be due to an insufficient amount of trials per episode or of individuals per generation. Theoretically, optimising both scenarios with an infinite amount of trials and individuals should yield similar or better results in the behaviour with more slots. Nevertheless, Getup B took more than twice the amount of generations to achieve the same performance as Getup A, due to the extra amount of parameters to optimise, while still taking longer.

A summary of the optimisation process can be viewed at https://youtu.be/QlLCUiewJdM and https://youtu.be/dx0fhFiQygg for Getup A and B, respectively.

	Slots	Success Chance	Duration	1st Success Gen.	Best solution gen.	Total gens.
Getup A	5	>99%	1.715s	18	1506	2750
Getup B	10	>99%	1.858s	43	4837	7360

Table 5.1: Comparison of results between two optimised getup behaviours with a different amount of slots.

When compared with the current getup behaviour in use (which does not respect self-collisions rules), Getup A was able to outperform it, being 29% faster. And although we were not able to objectively measure it, it also outperformed the original behaviour in terms of consistency, as the latter often fails to stand up on a single try.

Getup A was also tested in live games, during Robotica 2019, where it performed well within expectations, and was surely one of the factors that help the team achieve 3rd place in the competition.

5.5.2 Neural Behaviour Optimisation

Both the sparse and dense reward functions (equation 5.7 and 5.10, respectively) were optimised using the PPO algorithm, with the parameters shown in table 5.2. These are the best parameters found by OpenAI, through random search, for a humanoid agent to learn how to walk [70].

Unfortunately, both approaches failed, and the agent was unable to learn how to stand up. The dense reward function was faster to converge and was able to reach higher than the sparse reward function, showing that dense rewards provide a better setting for learning - which was our initial assumption. Nevertheless, both were unable to reach the goal, even after tens of millions of

Parameter	Value
Time steps per actor batch	2048
Clipping	0.1
Entropy coefficient	0
Epochs	10
Step size	$1e^{-4}$
Batch size	64
Gamma	0.99
Lambda	0.95

Table 5.2: PPO parameters used for learning a getup behaviour.

timesteps. This might be due to the high complexity of the problem, as the robot has 20 degrees of freedom, or perhaps the hyperparameters used aren't suitable for this problem. Or maybe its both.

In any case, this is something that should be worked on further in the future, in order to hopefully achieve a successful result.

5.6 Conclusion

This chapter presented the optimisation of a *getup* - a behaviour whose goal is to stand up from a falling position. Because of the new self-collisions rules, the current getups needed to be re-optimised. Two optimisation approaches were followed: one using key-frames, and another using a Neural Network.

With the key-frame approach, the agent was able to successfully learn how to stand up. The resulting behaviour was collisions-free, and was also faster and more consistent than the original getup. This getup was then tested in live games during Robotica 2019, where it performed flawlessly according to expectations.

On the other hand, the Neural Network approach failed to achieve the goal of standing up. Two different settings were available for the neural behaviour optimisation: a *sparse reward* setting, where the reward was received only at the end of the episode; and a *dense reward* setting, where the agent received a reward at every timestep. The dense reward environment proved to be the most effective one, having achieved faster convergence and higher elevations.

Chapter 6

Kick Optimisation

Kicking the ball is one of the most important skills in soccer. Even more so in robotic soccer competitions, where dribbling skills are still under-developed, and so the main way of moving the ball is through either shoots or passes. There are many types of kicks with different purposes, but in this work, the focus was towards a kick with the aim of shooting the ball as far away as possible. Such kick can then be used to either directly shoot at the goal from far away, to make a long pass to an unmarked teammate, or simply to remove the ball from a dangerous position in our side of the field.

6.1 Problem

As with the getup explained in chapter 5, the new self-collisions rules make some of the current kicks useless. However, unlike the getup, the self-collisions problems of these kicks were very light. Therefore, the decision was made to use the current kicks as the initial solution for the optimisation process, rather than completely learning from scratch.

Because there would not be enough time to optimise all of the affected kicks, the kick selected for optimisation was one whose aim is to shoot the ball as far away as possible.

6.2 Observation Space

In addition to the common observations explained in section 4.1.1, for the kick scenario, it is also necessary to observe the robot's height and the ball's relative position[1]. The agent's height is measured in meters, with a value of 0.54 when standing and 0.06 when laying on the ground. The ball's relative position is also measured in meters, with the point $(0,0)$ being the agent's position.

[1] In order for this kick to be usable in live games, it can't rely on absolute coordinates. As such, coordinates are relative to the agent, with the X axis being defined as whatever direction the agent is facing (e.g. if the agent was facing North, the aim would be to shoot the ball as far away as possible, with a straight trajectory, in the North direction).

Thus, during an optimisation episode,

$$\begin{cases} h \in [0.06, 0.54] \\ x, y \in \mathbb{R} \end{cases}, \tag{6.1}$$

where h represents the robot's height, and x and y represent the ball's relative coordinates.

6.3 Episode Layout

The process of evaluating the parameters of a kick has the following steps:

1. The agent walks to the ball using FC Portugal's walking controllers.
2. The agent tries to shoot the ball as far away as possible.
3. Once the ball has stopped moving, the fitness of the parameters is evaluated.

The last step is the most crucial one, and requires a good evaluation function, or the agent will not be able to learn how to properly shoot the ball.

6.4 Reward Shaping

As was explained in section 5.4, reward shaping is a very important part of the optimisation process. And much like with the getup scenario, the rewards for the kick optimisation should also be shaped.

6.4.1 Slot Behaviour Optimisation

The fitness function used in the first experiments was derived from previous successful optimisations the team had conducted on other types of kicks,

$$f = {\Delta_x}^2 - {\Delta_y}^2, \tag{6.2}$$

where Δ_x and Δ_y stand for the distance the ball moved on the respective axis. Although the aim of this kick is to shoot the ball as far away as possible in the X axis, we want to do so in a straight line. Therefore, there is a need to subtract Δ_y from Δ_x. However, this gives too much weight to the deviation in the Y axis. So, in order to shift the weight towards Δ_x, the values of both Δ are squared.

However, this reward function had a major problem due to the nature of Slot Behaviours. What happened was that the agent learned to shoot the ball in the first slot of the behaviour, making all of the next slots useless. More importantly, as the agent kicked the ball as soon as possible, it was not able to learn how to gain momentum[2] in order to increase the shooting distance.

[2] In a typical kick, momentum would be gained by pulling the kicking leg back, and then quickly moving it forward in a burst of speed, striking the ball.

This was solved by rewarding the agent based not only on how far it shot the ball, but also by what slot was active when the agent's foot connected to the ball:

$$f = \begin{cases} \frac{s_b}{S} & s_b < S \\ 1 + \frac{{\Delta_x}^2 - {\Delta_y}^2}{256} & s_b \geq S \end{cases}, \tag{6.3}$$

where s_b represents the index of the slot that was active when the ball touched the agent's foot, and S represents the total amount of slots for the behaviour being optimised. This new expression can be thought of as a two-part learning process. First, the agent learns how to kick the ball using the last slot of the behaviour. And then, it learns to kick the ball, in a straight line, as far away as possible. In order to make reading the results easier, the goal for ${\Delta_x}^2 - {\Delta_y}^2$ was set as 256, which would be the case when shooting the ball 16 meters away, in a completely straight line. Thus, a fitness value $f < 1$ means the agent did not manage to kick the ball at the end of the behaviour, a value $1 < f < 2$ means the agent managed to properly kick the ball less than 16 meters, and a value $f > 2$ means the agent managed to kick the ball further away than 16 meters.

6.4.2 Neural Behaviour Optimisation

Unlike the getup scenario, where the Neural Network is in control throughout the entire episode, in the kicking scenario that is not the case. That is because once the agent has kicked the ball, there is nothing the Neural Network can output that will change the outcome of the episode. Because of this, a dense reward setting would not be suitable for this problem. As such, the optimisation of the kick was performed in a sparse reward setting, where the reward signal is only received at the end of an episode.

Thus, the reward function used in this optimisation is rather similar to the one used in the Slot Behaviour scenario:

$$f = \frac{{\Delta_x}^2 - {\Delta_y}^2}{256}. \tag{6.4}$$

6.5 Results

This section shows the results obtained from the experiments regarding the kick problem presented throughout this chapter. Results for both the Slot Behaviour and the Neural Behaviour approaches are presented, with a small discussion about each.

Because a kick movement is not symmetrical, it is not possible to reduce the amount of optimisation parameters, as was performed when optimising a getup. Therefore, both the Neural Behaviour and the Slot Behaviour optimise all of the robot's joints (except the head and neck).

6.5.1 Slot Behaviour Optimisation

As was mentioned in section 6.1, the Slot Behaviour kick was optimised using the current kick as the initial solution, with a search radius of 45º. This means that all of the joints of the optimised

behaviour are within 45º of the initial solution, so that

$$\forall j \in J, {a_j}^f \in \left[{a_j}^i - 45, {a_j}^i + 45\right], \tag{6.5}$$

where j represents a joint, J represents the set of all of the robot's joints (except head and neck), and ${a_j}^f$ and ${a_j}^i$ represent the angle of the given joint in the optimised solution and in the initial solution, respectively.

This optimisation was performed using 3 slots and 10 trials per episode. The results can be seen in table 6.1. The reason for choosing to optimise 3 slots is because that is the amount of slots current kicks - and thus the initial solution - contain.

Total Slots	Used Slots	Distance	Generations
3	2	>18m	1488

Table 6.1: Results of the optimisation of a kick through CMA-ES.

Interestingly, although we optimised 3 slots, the optimiser only used 2 of them: the first and the last. Somewhere down the optimisation process, the algorithm deemed it was better to only have 2 key-frames. Thinking about it, this does seem logical, as for the movement of a kick, only 2 key-frames should be necessary: the first to pull the leg backwards, and the second to push the leg forward.

6.5.2 Neural Behaviour Optimisation

When optimising the kick using Neural Networks, we did not use an initial solution, as the current kick was a Slot Behaviour, and thus incompatible with this approach. The optimisation used the PPO algorithm with the same parameters as the neural getup, listed in table 5.2.

Much like with the initial reward function of the Slot Behaviour approach, the Neural Network was not able to learn how to gain momentum. This means that the resulting kick was only able to shoot the ball very small distances, up to approximately 4 meters. This is the result of the agent trying to gain rewards as soon as possible, quickly learning to *push* the ball with its leg, instead of *kicking* it.

6.6 Conclusion

This chapter presented the optimisation of a *kick* with the aim of shooting the ball as far away as possible. Because of the new self-collisions rules, the current kicks needed to be re-optimised. Two optimisation approaches were followed: one using key-frames, and another using a Neural Network.

With the key-frame approach, the agent learned the kick by using the current one as the initial solution. The resulting behaviour was collisions-free, capable of shooting the ball farther than 18m. It is also simpler that current behaviours, as it consists of only 2 key-frames.

On the other hand, the Neural Network approach failed to achieve the goal. Because the agent tries to get rewards as soon as possible, it fails to learn how to gain momentum and thus is only able to learn how to *push* the ball instead of how to *kick* it.

Chapter 7

Targeted Kick

Chapter 6 explained the optimisation of a kick with the goal of shooting the ball as far away as possible. In this chapter, a different kind of a kick, a *targeted* kick, is presented.

The aim of this kick is to be able to shoot the ball to any desired position on the field, independently from the agent's position. Such a behaviour is not feasible of optimisation under genetic algorithms, and thus only the Neural Behaviour approach is discussed.

7.1 Problem

Kicking the ball is one of the most important skills in soccer. Even more so in robotic soccer competitions, where dribbling skills are still under-developed, and so the main way of moving the ball is through either shoots or passes. However, it is still hard to kick a ball to a specific position on the field, as most kicks aim to simply shoot the ball in a general direction. Using the typical Evolution Strategy approach, if we wanted to kick the ball to a target 5 meters away, and to another target 10 meters away, we'd have to learn 2 different behaviours.

However, using Neural Networks, only one behaviour would have to be learned. Such is the aim of this kick.

7.2 Observation Space

In addition to the observations required for a simple directional kick, explained in section 6.2, for the targeted kick scenario, it is also necessary to observe the target's relative position. The target is a randomly selected position on the playing field, and thus, during an optimisation episode,

$$\begin{cases} t_x \in [0, 15] \\ t_y \in [0, 10] \end{cases}, \tag{7.1}$$

where t_x and t_y represent the target's relative coordinates, in meters. Although the pitch measures 20 meter by 30 meters, the decision was made to scale down the problem and try to learn a target kick that would be able to shoot the ball in a smaller area.

7.3 Episode Layout

The process of evaluating the parameters of a kick has the following steps:

1. The agent walks to the ball using FC Portugal's walking controllers.
2. The agent tries to shoot the ball to a position as close as possible to the target.
3. Once the ball has stopped moving, the fitness of the parameters is evaluated.

7.4 Reward Shaping

As with the kick presented in chapter 6, this optimisation was performed in a sparse reward setting, where the reward signal is only received at the end of an episode. The reward function used is

$$f = \sqrt{{\Delta_x}^2 + {\Delta_y}^2}, \tag{7.2}$$

where Δ_x and Δ_y stand for the distance between the target and the final ball position, in the X and Y axis respectively. This is the typical formula for calculating the distance between 2 points in a 2-dimensional field.

7.5 Results

This behaviour was optimised using a Neural Network and PPO with the same parameters as the previous behaviours, listed in table 5.2.

Although the agent was given random targets every episode, it was not able to learn how to kick the ball towards them. In fact, what the agent learned was to always shoot the ball in a straight line in front of it. This is likely due to the randomness of the targets' positions. By always kicking the ball in its direction, the agent was able to maximise the average reward it would gain. The end result was a kick rather similar to the one explained in subsection 6.5.2: the agent learned how to *push* the ball, and not how to **kick** it.

7.6 Conclusion

This chapter presented the optimisation of a targeted kick with the aim of shooting the ball to specific positions on the field.

The optimisation of this behaviour was only performed with Neural Networks, as the key-frame approach is not suitable for this type of problem. The result was similar to the kick explained in 6.5.2, with the agent failing to accomplish the goal.

Changes to the way the reward is shaped are likely necessary, in order to address the lack of momentum and the missed targets.

Chapter 8

Conclusion and Future Work

Chapter 1 presented an overall introduction to this document, and the motivations behind the work. More importantly, it introduced the RoboCup goal of beating the world champion team by 2050, which is the ultimate goal for everyone involved.

The state of the art was reviewed in chapter 2. Deep Reinforcement Learning is still a relatively new field, and thus not many teams have applied it to RoboCup yet. Despite this, FC Portugal achieved a very recent breakthrough concerning humanoid running skills, having achieved extremely high running speeds when compared to other teams. Other optimisation techniques, such as Evolution Strategies have been widely used by teams, although mostly focus on kicking.

The environment used for the development of this work consisted of SimSpark as the simulation server, with RoboViz as the 3D viewer and the NAO robot as the model of the agent. The settings changed to allow for faster optimisation times and to reduce noise are shown in subsection 3.1.2. Chapter 3 also presented the new rules that have been added to this year's competition.

Chapter 4 went over the developed framework. This framework allows for the optimisation of low-level skills in one of two ways: through a key-frame approach or through a Neural Network.

In chapter 5, a getup was optimised. The goal was for the agent to learn how to stand up from a laying position. The key-frame approach was successful and managed to not only learn without self-collisions, but also outperform the original getup. This was tested in live games during Robotica 2019 where it performed well within expectations. On the other hand, the Neural Network approach failed to learn even after tens of millions of timesteps. The agent quickly learned how to sit, but not how to stand.

Chapter 6 explains the optimisation of a kick with the goal of shooting the ball as far away as possible. As with the getup, the key-frame approach using CMA-ES was rather successful, having learned how to properly kick. In this case, however, the optimiser was given a good initial solution, so learning was not made from scratch, unlike with the getup. As for the Neural Networks approach using PPO, although the agent managed to learn how to *push* the ball, it did not learn how to *kick* it.

Chapter 7 went over a targeted kick. Suck kick has the aim to shoot the ball to specific positions on the field, and thus should be optimised through Deep Reinforcement Learning methods on Neural Networks. As with the previous chapter, the agent failed to achieved the desired goal, having learned only to *push* the ball in a straight line in front of it.

It is clear that at this time, Neural Networks and Reinforcement Learning methods might not be the best choice for optimising simple skills, as key-frame approaches optimised using Evolution Strategies proved to be far easier to obtain good results with.

In the future, some more focus should be directed towards Deep Reinforcement Learning, because skills optimised under these methods are sure to be far more robust than key-frame behaviours. A starting point might be to try to give some more information to the Neural Networks, expanding the observation space.

References

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara, "RoboCup: A challenge problem for AI", *AI Magazine*, vol. 18, no. 1, pp. 73–85, 1997. [Online]. Available: www.doi.org/10.1609/aimag.v18i1.1276.

[2] U. Visser and H.-D. Burkhard, "RoboCup: 10 Years of Achievements and Future Challenges", *AI Magazine*, vol. 28, no. 2, pp. 115–132, 2007.

[3] L. P. Reis, N. Lau, A. Abdolmaleki, N. Shafii, R. Ferreira, A. Pereira, and D. Simões, "FC Portugal 3D Simulation Team: Team Description Paper 2017", *RoboCup Symposium*, 2017.

[4] A. Ferrein and G. Steinbauer, "20 Years of RoboCup", *KI - Künstliche Intelligenz*, vol. 30, no. 3, pp. 73–85, 2016. [Online]. Available: www.doi.org/10.1007/s13218-016-0449-5.

[5] RoboCup. (no date). *RoboCup's Objective*, [Online]. Available: https://www.robocup.org/objective (visited on 01/13/2019).

[6] N. Shafii, "Development of an optimized omnidirectional walk engine for humanoid robots", PhD thesis, Faculty of Engineering of the University of Porto, 2015.

[7] P. Domingos, "A Few Useful Things to Know about Machine Learning", *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012. [Online]. Available: www.doi.org/10.1145/2347736.2347755.

[8] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: www.doi.org/10.1038/nature14539.

[9] T. Joachims, "Optimizing Search Engines using Clickthrough Data", *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 133–142, 2002.

[10] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting Spammers on Twitter", *7th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference, CEAS 2010*, 2010.

[11] I. Fette, N. Sadeh, and A. Tomasic, "Learning to Detect Phishing Emails", *16th International World Wide Web Conference, WWW2007*, pp. 649–656, 2007. [Online]. Available: www.doi.org/10.1145/1242572.1242660.

[12] M. Depristo, E. Banks, R. Poplin, K. Garimella, J. Maguire, C. Hartl, A. Philippakis, G. Del Angel, M. Rivas, M. Hanna, A. McKenna, T. Fennell, A. Kernytsky, A. Sivachenko, K. Cibulskis, S. Gabriel, D. Altshuler, and M. Daly, "A framework for variation discovery and genotyping using next-generation dna sequencing data", *Nature Genetics*, vol. 43, no. 5, pp. 491–501, 2011. [Online]. Available: www.doi.org/10.1038/ng.806.

[13] L. Kaelbling, M. Littman, and A. Moore, "Deep Machine Learning - A New Frontier in Artificial Intelligence Research", *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13–18, 2010. [Online]. Available: www.doi.org/10.1109/MCI.2010.938364.

[14] S. Holcomb, W. Porter, S. Ault, G. Mao, and J. Wang, "Overview on DeepMind and Its AlphaGo Zero AI ", *ACM International Conference Proceeding Series*, pp. 67–71, 2018. [Online]. Available: www.doi.org/10.1145/3206157.3206174.

[15] N. D. Nguyen, T. Nguyen, and S. Nahavandi, "System Design Perspective for Human-Level Agents Using Deep Reinforcement Learning: A Survey", *IEEE Access*, vol. 5, pp. 27 091–27 102, 2017. [Online]. Available: www.doi.org/10.1109/ACCESS.2017.2777827.

[16] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search", *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: www.doi.org/10.1038/nature16961.

[17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge", *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. [Online]. Available: www.doi.org/10.1038/nature24270.

[18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play", *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: www.doi.org/10.1126/science.aar6404.

[19] OpenAI. (2018). *Kinds of RL Algorithms*, [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html (visited on 01/30/2019).

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey", *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. [Online]. Available: www.doi.org/10.1109/MSP.2017.2743240.

[21] Skymind. (no date). *A Beginner's Guide to Deep Reinforcement Learning*, [Online]. Available: https://skymind.ai/wiki/deep-reinforcement-learning (visited on 01/28/2019).

[22] T. Le, N. Vien, and T. Chung, "A Deep Hierarchical Reinforcement Learning Algorithm in Partially Observable Markov Decision Processes", *IEEE Access*, vol. 6, pp. 49 089–49 102, 2018. [Online]. Available: www.doi.org/10.1109/ACCESS.2018.2854283.

[23] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[24] L. Fridman. (2019). *MIT 6.S091: Introduction to Deep Reinforcement Learning (Deep RL)*, [Online]. Available: https://youtu.be/zR11FLZ-O9M (visited on 01/30/2019).

[25] J. Xu. (2018). *Beyond DQN/A3C: A Survey in Advanced Reinforcement Learning*, [Online]. Available: https://towardsdatascience.com/advanced-reinforcement-learning-6d769f529eb3 (visited on 01/28/2019).

[26] S. Huang. (2018). *Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)*, [Online]. Available: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287 (visited on 01/28/2019).

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: www.doi.org/10.1038/nature14236.

[28] F. Tan, P. Yan, and X. Guan, "Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10637 LNCS, pp. 475–483, 2017. [Online]. Available: www.doi.org/10.1007/978-3-319-70093-9_50.

[29] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning", *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 2094–2100, 2016.

[30] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling Network Architectures for Deep Reinforcement Learning", *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581.

[31] T. Simonini. (2018). *Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets*, [Online]. Available: https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682 (visited on 01/30/2019).

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay", *CoRR*, 2015. [Online]. Available: https://arxiv.org/abs/1511.05952.

[33] OpenAI. (2018). *Vanilla Policy Gradient*, [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/vpg.html (visited on 01/31/2019).

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning", *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783.

[35] OpenAI. (2017). *OpenAI Baselines: ACKTR & A2C*, [Online]. Available: https://blog.openai.com/baselines-acktr-a2c/ (visited on 01/28/2019).

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347.

[37] OpenAI. (2017). *Proximal Policy Optimization*, [Online]. Available: https://openai.com/blog/openai-baselines-ppo/ (visited on 07/03/2019).

[38] ——, (2018). *Proximal Policy Optimization*, [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html (visited on 07/03/2019).

[39] ——, (2019). *OpenAI Five*, [Online]. Available: https://openai.com/five/ (visited on 07/03/2019).

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", *CoRR*, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971.

[41] OpenAI. (2018). *Deep Deterministic Policy Gradient*, [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ddpg.html (visited on 01/31/2019).

[42] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods", *CoRR*, 2018. [Online]. Available: https://arxiv.org/abs/1802.09477.

[43] OpenAI. (2018). *Twin Delayed DDPG*, [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/td3.html (visited on 02/06/2019).

[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", *CoRR*, 2018. [Online]. Available: https://arxiv.org/abs/1801.01290.

[45] OpenAI. (2018). *Soft Actor-Critic*, [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/sac.html (visited on 02/06/2019).

[46] DeepMind. (2018). *AlphaZero: Shedding new light on the grand games of chess, shogi and Go*, [Online]. Available: https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/ (visited on 02/05/2019).

[47] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Silver, and D. Wierstra, "Imagination-Augmented Agents for Deep Reinforcement Learning", *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1707.06203.

[48] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-Efficient Hierarchical Reinforcement Learning", *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1805.08296.

[49] OpenAI. (2017). *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*, [Online]. Available: https://openai.com/blog/evolution-strategies/ (visited on 07/03/2019).

[50] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)", *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003. [Online]. Available: www.doi.org/10.1162/106365603321828970.

[51] M. Fahami, M. Roshanzamir, and N. Izadi, "A Reinforcement Learning Approach to Score Goals in RoboCup 3D Soccer Simulation for Nao Humanoid Robot", *2017 7th International Conference on Computer and Knowledge Engineering, ICCKE 2017*, vol. 2017-January, pp. 450–454, 2017. [Online]. Available: www.doi.org/10.1109/ICCKE.2017.8167920.

[52] A. Abdolmaleki, D. Simões, N. Lau, L. P. Reis, and G. Neumann, "Learning a humanoid kick with controlled distance", in *Robot World Cup*, Springer, 2016, pp. 45–57.

[53] A. Bai, S. Russell, and X. Chen, "Concurrent Hierarchical Reinforcement Learning for RoboCup Keepaway", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11175 LNAI, pp. 190–203, 2018. [Online]. Available: www.doi.org/10.1007/978-3-030-00308-1_16.

[54] S. Luo, H. Lu, J. Xiao, Q. Yu, and Z. Zheng, "Robot Detection and Localization Based on Deep Learning", *Proceedings - 2017 Chinese Automation Congress, CAC 2017*, vol. 2017-January, pp. 7091–7095, 2017. [Online]. Available: www.doi.org/10.1109/CAC.2017.8244056.

[55] M. Javadi, S. Azar, S. Azami, S. Ghidary, S. Sadeghnejad, and J. Baltes, "Humanoid Robot Detection Using Deep Learning: A Speed-Accuracy Tradeoff", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11175 LNAI, pp. 338–349, 2018. [Online]. Available: www.doi.org/10.1007/978-3-030-00308-1_28.

[56] "Overlapping Layered Learning", *Artificial Intelligence*, vol. 254, pp. 21–43, 2018. [Online]. Available: www.doi.org/10.1016/j.artint.2017.09.001.

[57] K. Dorer, "Learning to Use Toes in a Humanoid Robot", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11175 LNAI, pp. 168–179, 2018. [Online]. Available: www.doi.org/10.1007/978-3-030-00308-1_14.

[58] P. MacAlpine and P. Stone, "UT Austin Villa: RoboCup 2017 3D simulation league competition and technical challenges champions", in *RoboCup 2017: Robot Soccer World Cup XXI*, ser. Lecture Notes in Artificial Intelligence, C. Sammut, O. Obst, F. Tonidandel, and H. Akyama, Eds., Springer, 2018, pp. 473–85.

[59] N. Shafii, N. Lau, and L. Reis, "Learning to Walk Fast: Optimized Hip Height Movement for Simulated and Real Humanoid Robots", *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 80, no. 3-4, pp. 555–571, 2015. [Online]. Available: www.doi.org/10.1007/s10846-015-0191-5.

[60] M. Abreu, N. Lau, A. Sousa, and L. P. Reis, "Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning", *19th IEEE International Conference on Autonomous Robot Systems and Competitions (IEEE ICARSC'2019)*, 2019.

[61] S. Glaser. (2017). *SimSpark Wiki - About SimSpark*, [Online]. Available: https://gitlab.com/robocup-sim/SimSpark/wikis/About-SimSpark (visited on 06/18/2019).

[62] ——, (2017). *SimSpark Wiki - Soccer Simulation*, [Online]. Available: https://gitlab.com/robocup-sim/SimSpark/wikis/Soccer-Simulation (visited on 06/18/2019).

[63] M. Simões, N. Lau, D. Simões, and M. Palhang. (2019). *RoboCup Soccer Simulation 3D League - Rules for the 2019 Competition in Sydney, Australia (July 2nd - July 8th)*, [Online]. Available: https://ssim.robocup.org/wp-content/uploads/2019/06/Rules_RoboCupSim3D2019.pdf (visited on 06/18/2019).

[64] S. Glaser. (2017). *SimSpark Wiki - Models*, [Online]. Available: https://gitlab.com/robocup-sim/SimSpark/wikis/Models (visited on 06/18/2019).

[65] H. R. de Brito Picado, "Development of behaviors for a simulated humanoid robot", Master's thesis, University of Aveiro, 2008.

[66] H. Picado, M. Gestal, N. Lau, L. P. Reis, and A. M. Tomé, "Automatic generation of biped walk behavior using genetic algorithms", in *Bio-Inspired Systems: Computational and Ambient Intelligence*, J. Cabestany, F. Sandoval, A. Prieto, and J. M. Corchado, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 805–812, ISBN: 978-3-642-02478-8.

[67] L. Cruz, L. P. Reis, N. Lau, and A. Sousa, "Optimization approach for the development of humanoid robots' behaviors", in *Advances in Artificial Intelligence – IBERAMIA 2012*, J. Pavón, N. D. Duque-Méndez, and R. Fuentes-Fernández, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 491–500, ISBN: 978-3-642-34654-5.

[68] E. Wiewiora, "Reward shaping", in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 863–865, ISBN: 978-0-387-30164-8. [Online]. Available: www.doi.org/10.1007/978-0-387-30164-8_731.

[69] C. J. C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992. [Online]. Available: www.doi.org/10.1007/BF00992698.

[70] OpenAI. (2019). *OpenAI Baselines - PPO*, [Online]. Available: https://github.com/openai/baselines/blob/master/baselines/ppo1/run_humanoid.py (visited on 06/25/2019).