FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Humanoid Robot Kick in Motion Ability for Playing Robotic Soccer

Henrique da Silva Teixeira

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: Luís Paulo Reis

February 13, 2020

Humanoid Robot Kick in Motion Ability for Playing Robotic Soccer

Henrique da Silva Teixeira

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Approved in oral examination by the committee:

Chair: Prof. Luís Miguel Pinho de Almeida
External Examiner: Prof. Artur José Carneiro Pereira
Supervisor: Prof. Luís Paulo Gonçalves dos Reis

February 13, 2020

Abstract

Robotics and Artificial Intelligence are two deeply intertwined fields of study, currently experiencing formidable growth. To foster these developments, the RoboCup initiative is a fantastic test bed to experiment new approaches and ideas.

This dissertation is rooted in the groundwork laid by previous FCPortugal3D contributions for the RoboCup simulation 3D robotic soccer league, and seeks to design and implement a humanoid robotic kick system for situations where the robot is moving.

It employs Reinforcement Learning (RL) techniques, namely the Proximal Policy Optimization (PPO) algorithm to create fast and reliable skills. The kick was divided into 6 cases according to initial conditions and separately trained for each of the cases.

A series of kicks, both static and in motion, using two different gaits were developed. The kicks obtained show very high reliability and, when compared to state of the art kicks, displayed a very high time performance improvement. This opens the door to more dynamic games with faster kicks in the RoboCup simulation 3D league.

Resumo

A Robótica e a Inteligência Artificial são duas áreas de estudo profundamente interligadas e a experienciar um crescimento formidável na actualidade. Para encorajar esses desenvolvimentos, a iniciativa *RoboCup* é um tubo de ensaio para testar novas abordagens e ideias.

Esta dissertação encontra-se alicerçada no trabalho desenvolvido por contribuições prévias da equipa *FCPortugal3D* para a liga *RoboCup* de futebol robótico simulado 3D, e procura desenhar e implementar um sistema de pontapé robótico humanóide para situações em que o robot se encontra em movimento.

Para tal, utiliza técnicas de *Reinforcement Learning* (RL), nomeadamente o algoritmo *Proximal Policy Optimization* (PPO) para criar comportamentos rápidos e fiáveis. O pontapé foi dividido em 6 casos de acordo com as condições iniciais e treinado separadamente para cada um destes.

Uma serie de pontapés, estáticos e em movimento, utilizando dois portes diferentes foram desenvolvidos. Estes obtiveram uma fiabilidade elevada e, quando comparados aos pontapés existentes, uma melhoria substancial de performance temporal. Tal abre a porta para jogos mais dinâmicos e com pontapés mais rápidos na liga *RoboCup* de simulação 3D.

Acknowledgements

It is not an easy thing to write this part of the document. Because it feels just like yesterday when I started out as an university student; and yet here I am, and several years have passed.

For all their support and sacrifice, for their encouragement and comprehension, I would like to thank my parents, Sónia and Albano, without which I wouldn't even be able to be writing this document. I cannot thank them enough. To my sisters Inês and Gabriela, and to the rest of my family as well, for their warm words and patience with me. Lots of patience.

I would also like to thank my friends, who kept me going and laughing, for the good times they gave me throughout the years. And who dragged me out of the house to relax when the pressure was mounting. A special thanks to Gonçalo, José and Frederico, who kindly helped me with insights, with reviewing this document or just with their copious moral support.

I would also like to thank my supervising professor, Luís Paulo Reis, for his availability and guidance throughout the writing of this document, and to Tiago Silva and Miguel Abreu, two members of the *FCPortugal3D* team who, without their guidance and help, I would have been completely lost. Finally, to my university, who took care of me these years and made me feel at home.

There are many more people I would like to thank, and who I would like to name, but know that even if you are not here in writing, you are never forgotten, for you have made me who I am.

Thank you.

Henrique

"One must imagine Sisyphus happy"

Albert Camus

Contents

List of Figures

List of Tables

Abbreviations

RoboCup	Robot Soccer World Cup
ML	Machine Learning
DL	Deep Learning
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed DDPG
SAC	Soft Actor Critic
RL	Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
MDP	Markov Decision Processes
SARSA	State–action–reward–state–action
MTL	Multi-Task Learning
DQN	Deep Q-(Learning) Network
VPG	Vanilla Policy Gradient
A3C	Asynchronous Advantage Actor Critic
A2C	Advantage Actor Critic
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
MCTS	Monte Carlo Tree Search
I2A	Imagination-Augmented Agents
HIRO	HIerarchical Reinforcement learning with Off-policy correction
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
GAE	Generalized Advantage Estimator

Chapter 1

Introduction

Even the simplest of actions in robotics, such as walking or kicking an object are non-trivial, multi-dimensional problems, that still hold the potential for further improvement. Nevetheless, recent advances in both the underlying tools and raw computing power now allow us to consider situations where several of these atomic decisions are executed simultaneously.

This work was performed in the context of the dissertation of the *Mestrado Integrado em Engenharia Electrotécnica e de Computadores* (MIEEC) of FEUP. It seeks to describe the work performed and the approaches taken in it.

The aim of this chapter is to give an overview, context, motivation and objectives to the following chapters.

1.1 Context

Football is a collective sport where two teams of 11 players each seek to obtain at least one more goal than the other. In collective sports such as football, the performance of each player, and the ensuing result of the team is dependent on a hierarchy of interacting high and low level factors, from physical ability, technique and mental condition to team tactics [1, 2].

In this way, and being football one of the sports with more fans and players worldwide [3], it naturally follows that the area of study of robotics should use it as a reference point and a test tube for the development of new approaches, doubly so when dealing with humanoid robotics, and as a way to promote enthusiasm in the general audience. Therefore, the RoboCup initiative can be seen in this context, using the sport as a starting point and a way to frame and focus efforts. The long term goal of this initiative is, according to itself:

> "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." [4]

In this context, the team FCPortugal3D, a collaboration between the Universidade do Porto and the Universidade de Aveiro, participates in the RoboCup initiative every year, having won several

awards already, including the World Championship in 2006 and the European Championship in 2007, 2012, 2013, 2014 and 2015 [5]. The ample work developed throughout the years by the team provided the basis for the work in this document.

1.2 Motivation

The RoboCup initiative seeks to achieve the aforementioned goals through a series of annual global competitions, in which several teams face off in a series of different leagues. One of these is the simulated humanoid league, where 11 robots per team play a match. In complement to this, there is a symposium organized to share research and advances in the area [6, 7].

These robots follow a realistic physical model, behaving autonomously and following high level decisions so as to obtain low level behaviours. One of these is to kick the ball, in order to pass or shoot it [8].

Previous groundwork has obtained methodologies for several types of kicks (long, quick, precise). Nevertheless the current implementation still does not allow the kicks to be performed in a similar way to humans, that is, without stopping their forward motion [9, 10, 11].

This forward motion corresponds to one of several skills developed for usage in a game environment. These correspond to a sprinting and running pace [12].

With the goal of achieving the long term goals of the RoboCup initiative, there is a constant refining of the rules and the physical model implemented in the simulation, resulting in the recent removal of the tolerance to internal collisions of the robot, that is, clipping, which has invalidated some of the strategies that the team FCPortugal3D previously implemented [13].

Furthermore, it must be noted that the work here performed could only ever be hoped to be done in a simulation context, as the nature of the approaches requires hundreds of thousands of training episodes. These would prove unfeasible to be done with a physical robot, due to both time constraints and the natural wear and tear that repeated falls would entail on a expensive piece of hardware. Thus, the simulation is a natural environment to study new approaches and algorithms in a quick and inexpensive manner.

With these details in mind, the work of this thesis proves crucial to maintain the competitive edge of the team.

1.3 Objectives

The main goal of this dissertation is the design and implementation of a humanoid kick system based on deep neural networks capable of fluidly kicking a ball whilst walking, using the simulated NAO robot as a test bed for the RoboCup 3D Simulation League Environment.

Based on previous work on kicking and walking, the learning algorithms must obtain the parameters and associated movements to allow this sort of action. The implementation must allow for future refinement work by upcoming team members.

The end-goal of this dissertation is aimed at advancing the state-of-the-art while contributing to the research and performance of the FCPortugal3D team and the overarching area of study.

1.4 Dissertation Structure

This document follows the typical dissertation structure, and the outline of its chapters are as thus described:

Chapter 1 serves as an introduction to the problem, contextualizing it, providing motivations and objectives and describing the structure of the document.

Chapter 2 discusses the state of the art in Robot Learning in general, with some focus on the context of the work performed, starting with the foundations in machine learning before focusing in more specific topics and approaches of relevance to this dissertation.

Chapter 3 gives a more focused look on the training environment and tools used in the dissertation such as the simulator, visual interface, and the simulated NAO robot models, as well as contextualizing these with the RoboCup rules and their real world counterparts.

Chapter 4 describes the optimization environment tools and frameworks used in this work, such as TensorFlow, Stable Baselines and FCPGym, as well as the optimization algorithm chosen, PPO, and the previously existing skills. Furthermore it describes how these elements come together to create a training episode.

Chapter 5 presents a static kick using neural networks as a proof of concept, alongside a description of the observation and action spaces and overall neural network structure, as well as extra tools developed towards that goal.

Chapter 6 expands on the concepts of the previous chapter to create a kick in motion for the skills provided.

Chapter 7 goes over some extra work done to employ the approaches of the previous chapters to other situations where deep learning can be employed to optimize behaviours, namely, a targeted kick in motion.

Chapter 8 provides a brief conclusion to the document, reflecting upon the work performed throughout the last months and presenting some possibilities for future related work.

Chapter 2

Related Work

Before delving into more specific themes, an overview of the topic and its applications in the context of the RoboCup is necessary. In the context of this document, several approaches will be mentioned in section 2.7.

2.1 Machine Learning

Machine Learning (ML) is a term with its origins in the fifties [14], and in its essence, it is the study of algorithms and models that permit a computer to learn from data, continuously improving its performance. Or as more clearly stated:

> ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.”[15]

This is an extremely powerful concept, as it allows for machines to discover optimal, or at least optimized strategies in contexts where humans would struggle to do so, especially where the amount of data is immense and its correlations are not obvious to the human mind. Thus, we can see the pervasive impact that machine learning has had in our modern society, from search engine optimization [16], to social media [17], content distributors [18], to industrial processes optimization [19], to banking [20] and medical research [21], just to mention a few. In essence, the impacts of the subject are both tremendous and the shared conceptual underpinnings of the methods result in advances in one area that can synergize across multiple sectors of society at once.

2.1.1 Robot Learning

The area of concern of this dissertation nevertheless is Robot Learning, that is the combined application of both machine learning and robotics to obtain a robot that can learn how to perform a given task. The joining of the two disciplines brings about a series of non-trivial, real-world problems, such as a changing environments, reactivity and non-deterministic actions [22].

It is to be noted that Robot Learning in a simulation context has its own particularities, as it both solves the problem of limited training time but does so by creating a model that is not exactly the same as the real world. Still, progresses in computing over the years have helped to bridge the gap between simulation and real world in the field of robotics. Still, for the work performed in this dissertation, an emphasis on considerations relevant to the simulation context will be kept in mind.

2.2 Deep Learning

Deep Learning (DL), can be seen as a subset of the larger field of machine learning, and is described as:

> "A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification." [23]

For the purposes of this document, it is worthwhile to mention the difference between supervised, where the training data that the learner receives is labeled , and unsupervised learning where the data is unlabeled. This second type has more issues quantitatively evaluating performances, but has many useful implementations in areas such as clustering and dimensionality reduction [24].

2.3 Reinforcement Learning

Reinforcement Learning (RL) can be seen as a branch of machine learning, where the goal of the algorithm is to map situations to actions, so as to maximize (or minimize, in terms of optimization both cases can be seen as interchangeable [25]) a numerical reward signal [26].

Unlike supervised and unsupervised learning, the goal of RL is not to find structure or to fit a model, but instead to find a set of behaviours that optimize its reward function. Thus, while a model can be seen as convenient, it is always a means and not an end. This is a wholly different paradigm in the area of machine learning [27].

Although some more advanced models don't follow this, a simple RL process can be seen as a Markov Decision Process (MDP), where an Agent interacts with the environment:

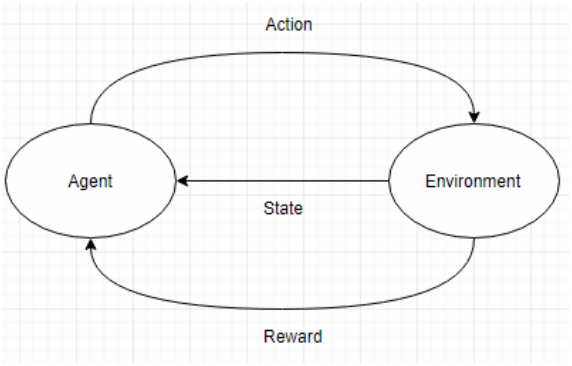

Figure 2.1: Generalized RL Scenario [28]

This is defined by:

- a state space S, with an initial s_0 state
- an action space A
- a transition function $T : S \times A \times S \rightarrow [0,1]$
- a reward function $R : S \times A \times S \rightarrow [R_{min}, R_{max}]$

2.3.1 Policy in Reinforcement Learning

Given a set of possible actions, it is not a trivial problem to define how the ideal action will be chosen by Agent in a certain state. This is defined by a policy. The role of policy in RL can be clearly seen by the step by step representation in figure 2.2 below [29]:

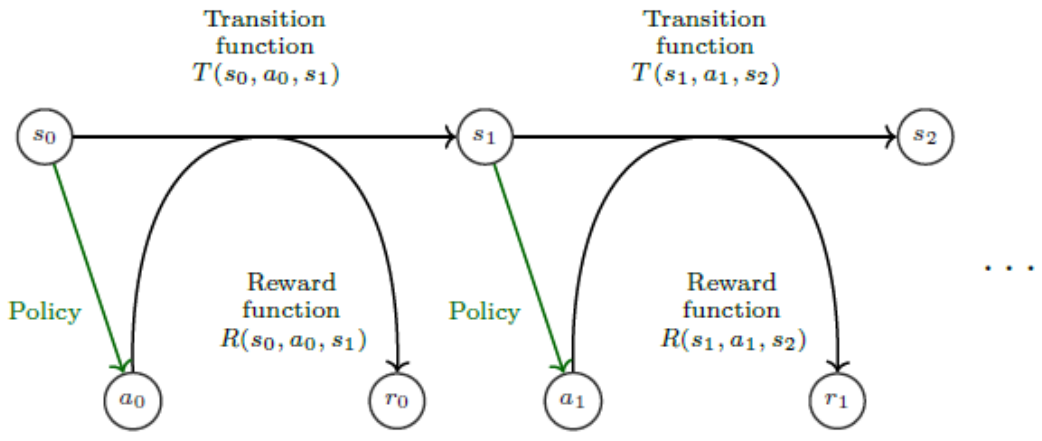

Figure 2.2: Illustration of a MDP for two time steps [27]

Thus we can see the need for a policy to allow the algorithm to choose how to act. Furthermore, we can also divide policies between deterministic, where the policy will consistently return the

same action for a given state of the observation space, and stochastic, where there is an associated probability $\pi(s,a)$ that the action a will be taken in the state s. [30]

2.3.2 On and off policy

One of the main distinctions to do in the area of RL is the one between on and off policy methods. This definition can be concisely stated as:

> "On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data."[26]

2.3.3 Value-based and Policy-based

The main difference between these two types of approaches can be seen in that Value-based methods will seek to learn a value function V^{π} or V^{*} or its generalization, the Q-function. Examples of such approaches are SARSA and Q-Learning [26, 31].

On the other hand, policy-based methods will try to directly learn or approximate optimal policy, avoiding a value function and replacing it with an arbitrary θ parameter to be maximized according to state and action [26].

2.3.4 Model free and model dependent

The algorithms given in the previous subsection are concerned with model-free approaches to ML. These eschew the model, to learn directly from the environment, through trial and error [32]. On the other hand, model-based methods seek to construct a model, an internal set of dynamic states, based on interactions with the real-world during the learning phase, unless the model is given *a priori* [33].

This model building phase usually employs model-free RL approaches. Through the model, they simulate the interactions thereafter, applying it instead of the real environment [34]. The relationships above described can be visualized in the following image:

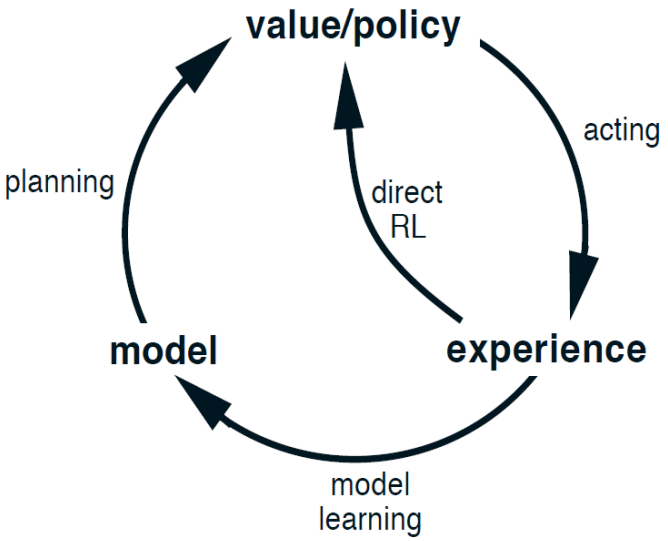

Figure 2.3: Relationships in model-free and model-based RL [26]

These have great advantages, as a model allows for a much greater sample efficiency. That is, the training data set required will be much smaller to obtain a similar level of performance, compared to a model-free approach.

The downside to this is that it comes at the cost of computational efficiency. Additionally, it requires an accurate model to train, which can be untenable for many cases. Furthermore, if the model doesn't depict reality in a satisfactory manner, the policies developed will work only in the context of the model and not real life. This is a relevant concern when translating from Simulated Football to Robotic Football in a Real-World Context [35].

2.4 Hierarchical Reinforcement Learning

For the context of this dissertation, additional concepts in ML were considered, such as Hierarchical Reinforcement Learning (HRL), which have potential to address some of the intricacies of the problem. The essence of the concept can be seen as:

> "In the same way that deep learning relies on hierarchies of features, HRL relies on hierarchies of policies." [36]

A very useful concept to understand HRL follows directly from the ideas discussed in section 2.3, and that is the concept of *options*. These can be seen as macro actions, which expand the notion of MDP to allow for actions to be taken over multiple time steps, allowing for higher level policies and goals, that coordinate lower-level ones [37, 38]. Nevertheless, it should be noted that there are several other approaches to HRL such as Hierarchical Abstract Machines [39].

2.5 Continuous and Discrete spaces

It is also to be noted that many of the earliest approaches to machine learning are limited to discrete spaces with regard to both actions and states. This is a big constraint, as most real world situations, are based out of interactions in the continuous domain. This is also true in the field of robotics [40].

Quantization of data may seem the obvious solution to this issue, but it nevertheless hits on the very common problem of the *curse of dimensionality*, especially when dealing with many-dimensional data [41].

While some of the solutions to apply ML algorithms in continuous spaces will be covered in section 2.7, the concepts covered in 2.3 can be extended to mention Semi-MDP, which can be used to cover a continuous time state space [42]. This can be represented by the image below:

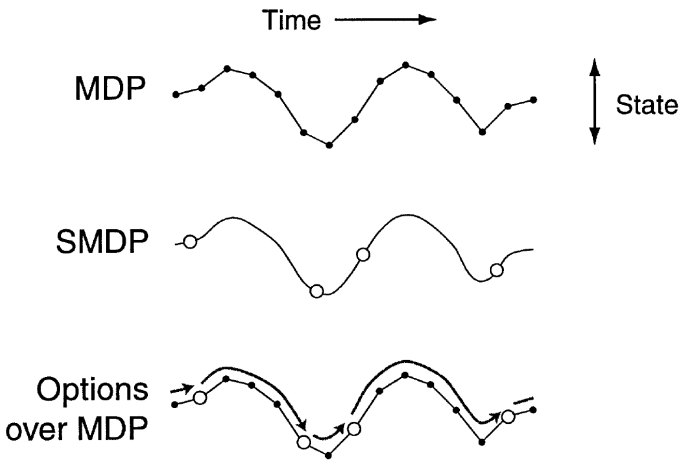

Figure 2.4: Comparison of MDP and MDP based approaches [38]

Here we can observe that the state trajectory of an MDP is constituted of individual discrete-time transitions, while an SMDP larger, continuous-time transitions. Options enable an MDP trajectory to be analyzed in either way [38].

2.6 Multi-Task Learning

Multi-Task Learning (MTL) is, in its essence an area of the field of ML that seeks to improve generalization, through the training of tasks in a parallel manner, sharing representations. In this manner, what is learned by one task can be applied to improve other related tasks [43, 44].

Several approaches in the field of MTL have been shown in the last few years that reveal great potential. Some of them are top-down, and as shown in the figure below, seek to implement a series of shared and task-specific layers to the ML algorithm, laid out in a pre-set manner.

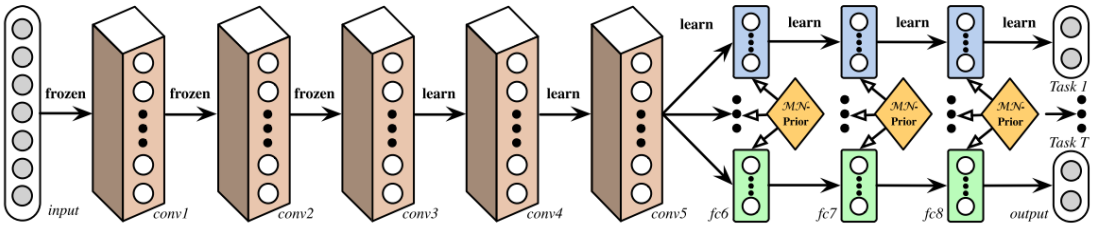

Figure 2.5: Top-down MTL through multilinear relationship networks [45]

This is a good approach for well understood problems, but can fall flat when addressing new tasks [43]. On the other hand, bottom-up models have also been used with success, with a method such as shown in the figure below:

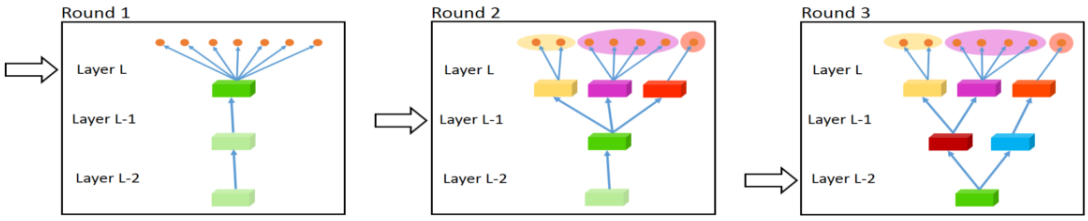

Figure 2.6: Bottom-up MTL through Fully-adaptive Feature Sharing [46]

Thus far, the methods implementing these approaches follow a greedy algorithm, which can result in a solution which is not optimal, as it seeks only a local and not a global minimum [47].

2.7 Some machine learning algorithms

The field of ML is an extremely expansive and fast developing area of study. Thus, this section does not seek to be an exhaustive and comprehensive list of machine learning algorithms, nor to provide a mathematical description of these, but to instead to serve as a quick overview, mentioning some of the most important ones, while contextualizing them in relation to the concepts discussed in the previous sections of this chapter.

2.7.1 SARSA

State–action–reward–state–action (SARSA), is probably the simplest algorithm to start describing the class of algorithms based on Q-function optimization. It is an on-policy RL algorithm for continuous state and action spaces that seeks to maximize the Q-function [48, 26].

2.7.2 Q-Learning and DQN

Q-Learning can be seen as an off-policy variation to SARSA, and is the starting point to a series of algorithms that work from its main idea [26].

Deep Q-(Learning) Network (DQN) provides an improvement through the usage of deep neural networks. This has allowed it to both train using raw visual data, but to solve part of the dimensionality problem, permitting it to be used on a continuous state space (albeit not in a continuous action space) [40, 49].

2.7.3 DDPG

Deep Deterministic Policy Gradient (DDPG) can be seen as an implementation of DQN for a continuous action space context. This is done through the simultaneous learning of an approximator to the optimal action $a^*(s)$ for the Q-function [40].

Nevertheless, DDPG is not a perfect algorithm. In fact, Q-values tend to be overestimated over time, in which cases the policy will start to exploit these errors.

2.7.4 TD3 and SAC

A very recent development to address these issues is Twin Delayed DDPG (TD3). This algorithm implements a series of modifications to DDPG. Firstly it uses two different Q-functions, always taking the smaller of the values. Furthermore it updates the policy at a slower rate than the Q-functions, and finally it adds random noise to the action.These dramatically improve performance by making it harder for errors to creep up [50, 51].

Another take on DDPG is Soft Actor Critic (SAC), which seeks bases itself on a entropy augmented objective function to provide a trade-off between exploration (maximum entropy) and exploitation (maximum return) [52]. A comparison between several state-of-the-art Q-Learning based approaches can be seen in the figure below:

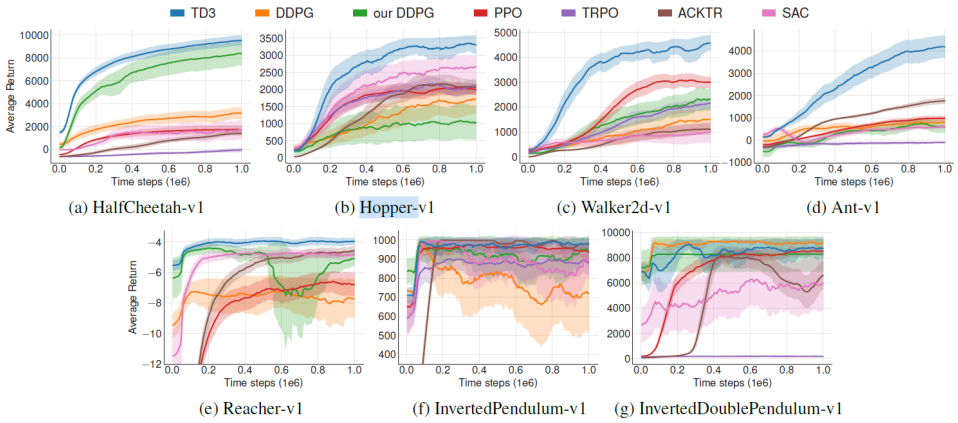

Figure 2.7: OpenAI Gym continuous control tasks learning curves. Shaded region represents half a standard deviation [52]

2.7.5 VPG

Vanilla Policy Gradient (VPG) is a simple policy-based on-policy algorithm for discrete and continuous action space. VPG updates the parameters of its policy through the usage of stochastic gradient ascent. Thus, even if one of its steps is not the best, it will evaluate according to the overall value of the set of actions. This does mean a loss of efficiency comparatively to greedy algorithms but also a possibility of avoiding being stuck in a local minimum [53, 54].

2.7.6 A3C and A2C

Asynchronous Advantage Actor Critic (A3C) and its improvement Advantage Actor Critic (A2C) are two actor-critic algorithms, that is, they merge features of both value-based and policy-based RL. In these, the actor learns optimal policy while the critic seeks to maximize the value function [55, 56]. This relationship can be seen below:

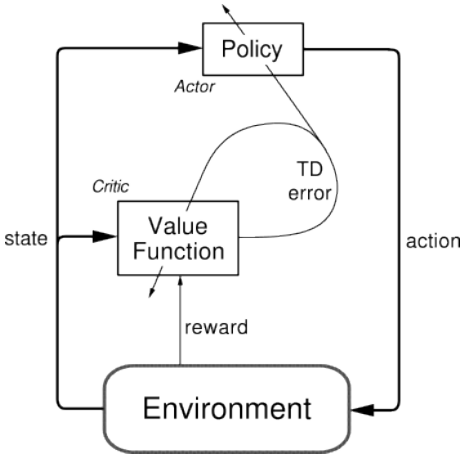

Figure 2.8: Actor-Critic architecture [26]

The goal of this competing relationship is to obtain a better outcome through this competing interaction than through either of them alone. Furthermore, a good analogy to an actor-critic model can be seen through a child and its parent. The child (actor) seeks to try out new behaviours, playing with their toys or putting their hand in the electrical socket, while the parent (critic) will observe and evaluate those actions. From this, the child grows and learns proper behaviour [57].

To understand A3C and A2C, we need to understand that the Q-function is split in two parts, the Value and Advantage functions. It is the later of these that the two algorithms utilize, hence the name.

The difference between A3C and A2C is a subtle one, hinging on synchrony. In A3C, multiple independent agents create what are different copies of the model, while in A2C each step model sends back data to the train model in a synchronous manner. This results in a great saving of computer resources comparatively to A3C with a similar performance [49, 58].

2.7.7 TRPO

Trust Region Policy Optimization (TRPO) is a method that seeks to use the concept of Trust Regions in RL. Trust region methods utilize a model function to approximate the original objective function, inside of a trust region $T_{\Delta}^{(k)}$ that can be shrunk or expanded according to the accuracy of a function $\rho^{(k)}$ of a k-dimensional dataset.

This prevents the time-steps to be either too slow, resulting in a slow rate of learning, or too fast, totally missing the minimum or maximum of the objective function. It is to be noted that trust region methods optimize for both step length and search direction simultaneously in each iteration [59].

2.7.8 PPO

Proximal Policy Optimization (PPO) is a policy-based method that seeks to strike a compromise between performance and simplicity. It is particularly easier to implement than for example TRPO in many classes of problems, such as those where visual input is important [60].

This algorithm has seen successful implementation in the area of simulated robotics, with an optimized GPU-enabled variant called PPO2, making it particularly interesting for the area of studies of this dissertation [61].

2.7.9 MCTS and I2A

Many of the most widely spoken developments in ML have come from DeepMind's work regarding AlphaZero and its predecessors such as AlphaGoZero. To understand them, the concept of Monte Carlo Tree Search (MCTS) is vital.

In many situations, such as in games like chess, the number of possible options when considering future moves grows in a exponential step with each step. Thus, it becomes computationally untenable to try and evaluate the best possible path through brute force. For this, MCTS is a heuristic approach to this problem, seeking to evaluate the smallest possible number of plays in order to achieve the best move.

In its essence MCTS achieves this through a four stepped process, that can be seen in figure 2.9. Firstly, starting from a root node, it selects a path using some evaluating *tree policy*. Once it reaches an expandable or leaf node that does not terminate the process, it will generate its child nodes. A simulation is then run from the new node(s), according to some other *default policy*. This result is then backpropagated to update the nodes [62, 63].

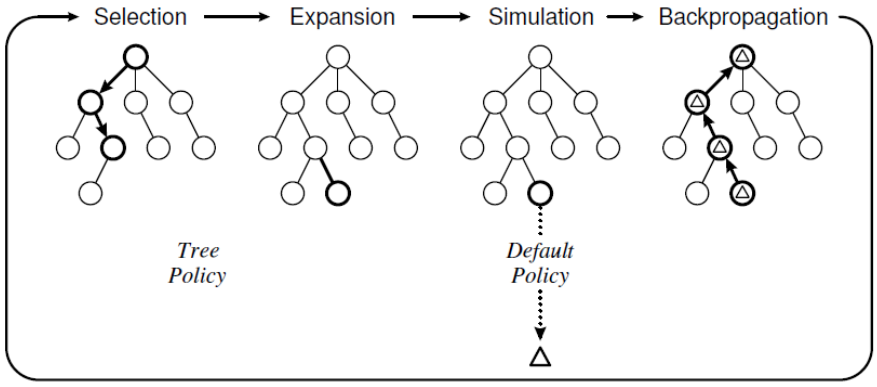

Figure 2.9: One iteration of the MCTS process [62]

Furthermore, AlphaZero represents a great advance compared to its predecessors through the use of Imagination-Augmented Agents (I2A). It combines model-free and model-based aspects to both learn a model and disregard it whenever it finds unexpected situations where it fails. Through I2A AlphaZero is capable of learning the rules of several different games unaided [64].

2.7.10 CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is an evolutionary algorithm for problems in the continuous domain, typically applied to unconstrained or bounded constraint optimization problems. CMA-ES is a second order approach for estimating a positive definite matrix, not totally unlike in result to a quasi-Newton method. It is to be noted that unlike it, CMA-ES does not require or approximate gradients, using instead a stochastic search algorithm. where the search steps are done through the recombination (mutation) of existing data points [65].

CMA-ES has an important place in this dissertation as an implementation of it was used in the work previously done by the FCPortugal3D team [9, 11], as it can be seen in the image below:

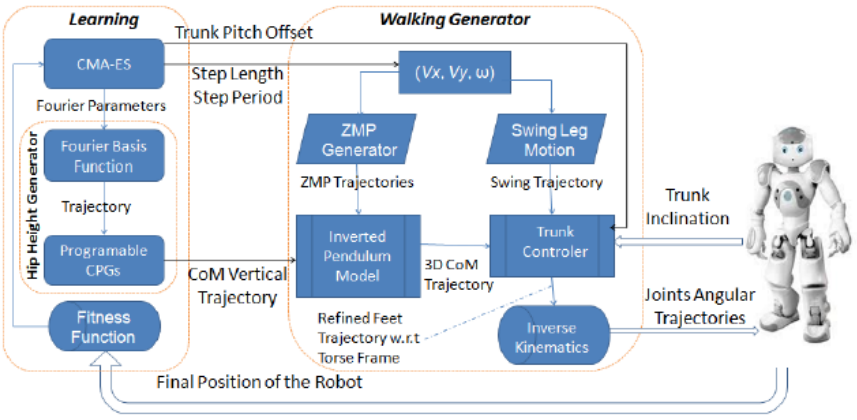

Figure 2.10: Implementation of the FCPortugal3D Omnidirectional Walk Engine [9]

2.7.11 HIRO

HIerarchical Reinforcement learning with Off-policy correction (HIRO) is a type of recent development to HRL based of an implementation of DDPG. It seeks to attribute goals low-level policies, based on those of the higher levels, whose rewards will depend on their ability to fulfill them. It has shown great potential in both performance and efficiency [50].

2.8 Previous Work in RoboCup

Throughout the early stages of this document, efforts were focused on the familiarization with the concepts related to this dissertation, as this was considered to be the key priority.

Nevertheless, several relevant documents relating to both the FCPortugal3D team, some already widely cited in this document, and others such as [66, 67, 68, 69, 70, 71, 72, 73] were studied in order to understand the current existing capabilities and approaches taken by members of the team.

Regarding other teams in the area, works such as [74, 75] were looked into, revealing an extremely dynamic environment stemming from the RoboCup competition. All of these are elements which this dissertation seeks to work upon.

2.9 Conclusions

With this chapter, a series of key concepts and algorithms in the field of machine learning were explored, with a special regard to those that could prove useful to the dissertation, or to better understand the underlying issues of creating a method through which to make a robot kick in motion.

Still, during the dissertation, the optimization algorithm overwhelmingly employed was, as further described in subsection 4.2.1, the PPO2 implementation from Stable Baselines. This is a powerful algorithm already used by the FCPortugal3D team with a great degree of success. It is to be noted that other algorithms could be swapped out due to the modular nature of the framework.

Chapter 3

Training environment

While there is a vast wealth of possible approaches to be taken while working in the area, these need to be taken in the context of the given simulation environment, the available tools and the previously developed skills, since any kick in motion skill developed must seamlessly follow from them. Therefore these can be seen as both constraints and guiding principles for the work of the following chapters.

As such, this chapter seeks to analyse the simulation environment server, SimSpark, the graphic interface (RoboViz) as well the agent (NAO robot), comparing it with its real world counterpart. Furthermore it seeks to put these in context regarding some of the relevant RoboCup rules.

3.1 SimSpark

All of this dissertation, and indeed all of the Soccer Simulation 3D League ultimately hinge on SimSpark[1]. SimSpark is a physical multiagent simulator for 3D environments built from the earlier Spark framework.

Although it was developed as part of the RoboCup initiative, and has been used in its successive iterations since 2004 as the simulation environment, it seeks to serve as a flexible platform whose uses extend well outside the scope of soccer simulation. As such, it has separated the main SimSpark repository from the RCSSServer3D soccer simulation server. Furthermore, it allows for custom environment scenarios to be created using its scene description language [76, 77].

3.1.1 Simulated Soccer Field

For the kick in motion, we need to bear in mind the RoboCup scenario, with the modifications elaborated in section 3.1.3. Thus, we deal with a simulated soccer pitch with an area roughly 35% of an official one and whose dimensions are as such [78]:

- Dimensions of 30m by 20m.

[1] SimSpark Repository https://gitlab.com/robocup-sim/SimSpark/ (visited on 02/01/2020)

- Each goal has height of 0.8m, length of 2.1m and depth of 0.6m.
- Penalty area of each goal of 6m by 1.8m.
- Center circle with a radius of 2 meters.
- Outside border of 10m in the x and y dimensions.
- Soccer ball with radius of 0.04m and mass of 26g.
- Unique Markers on the corners and goalposts.

It should be noted for the center circle, that although it is represented as such, for the vision perceptor it is recognized as a set of 10 lines forming a decagon.

Furthermore, the set of uniquely tagged markers standing on known fixed positions and height of zero, are perceived by the agent in tandem with the field lines to approximate its position on the field.

All of the above mentioned can be seen in figure 3.1 , with the aforementioned caveat of the center circle. Note the unique identifier on each of the markers.

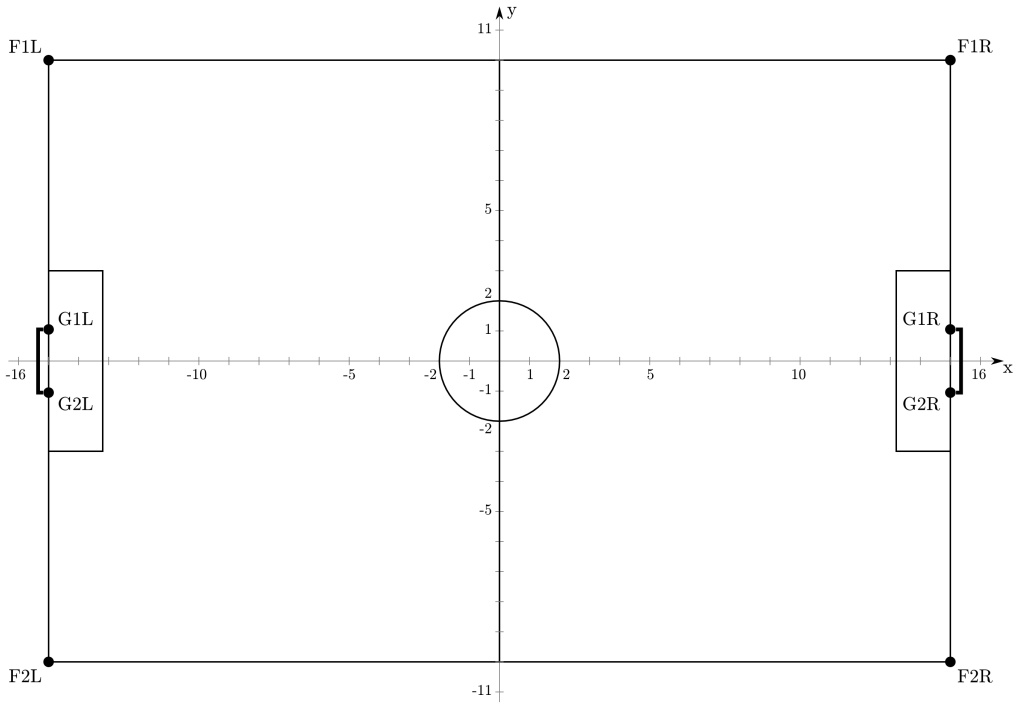

Figure 3.1: Illustration of the RoboCup Simulation Soccer Field [78]

3.1.2 SimSpark Architecture in a Competition Context

While the description of the exact inner workings of SimSpark lies outside of the scope of this document, it should be briefly mentioned how it allows for both flexibility and detail. It architecture

allows, for example, for different physics engines to be plugged through an abstract physical layer and for the in-depth simulation of physical properties of the agent, such as temperature regulation and stiffness control in each of the agents' servo motors [77, 79].

Furthermore, the overall architecture of SimSpark should be noted. While there are differences in a simulation context, these will be approached in detail in subsection 3.1.3. In this case, we will use the 2019 rules as a reference, as the full document for 2020 was not yet published at the time of writing [80].

In essence, in a competition setting, there is a SimSpark server machine and a two client machines, one for each of the teams in a match. This structure can be seen below on figure 3.2. Additionally, a further monitor computer will be connected to the server to visually display the simulation through RoboViz [81, 82].

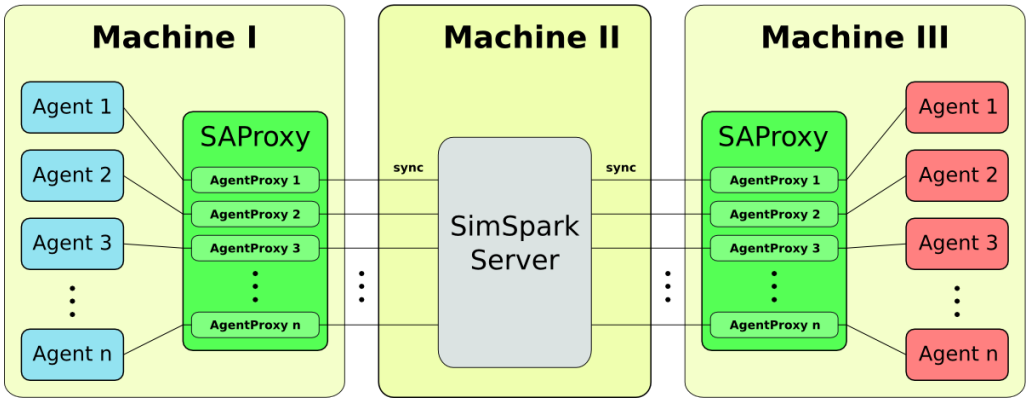

Figure 3.2: SimSpark Competition Architecture [81]

It should be noted that each of the clients have a series of agents that communicate with the server through an Agent Proxy. This proxy is divided into two threads that handle the agent's communications to and from the server. During a competition, the simulation is running in real-time mode, with all elements in sync, to ensure an ease of visualization and fairness of the game. For this effect the forwarding thread sends all messages to the agent before sleeping for 20ms. Only then it sends a "Sync" command to the server.

3.1.3 Modifications for Simulation

While the set-up above described works perfectly well for a competition scenario, this is not so the case when dealing with a simulation, since the goal at hand is to perform as many iterations as possible in the given time in order to optimize the learning speed.

Considering that all simulations are done with a single agent, thus canceling the need to worry about synchronization concerns between agents, we can disable *real-time mode* and turn *agent sync mode* on. It should be noted that even in this accelerated mode, each *sync* tick represents the same 20*ms* tick of simulation time it would in real time.

Furthermore there were several other modifications done so as to disable some of the soccer rules, remove noise from perceptions and give the agent a wider field of view than in game. The goal of these is to allow the agent to train under ideal conditions for better and quicker learning. To this effect, the following changes were made in the respective configuration files. These are in fact the same as in previous deep learning projects done by the FCPortugal3D team [83].

```
1
2     /usr/local/share/rcssserver3d/rcssserver3d.rb
3         $enableRealTimeMode = false
4     /usr/local/share/rcssserver3d/naosoccersim.rb
5         addSoccerVar('BeamNoiseXY',0.0)
6         addSoccerVar('BeamNoiseAngle',0.0)
7         #gameControlServer.initControlAspect('SoccerRuleAspect')
8     /usr/local/share/rcssserver3d/rsg/agent/nao/naoneckhead.rsg
9         (setViewCones 360 360) ;
10        (setSenseMyPos true) ;
11        (setSenseMyOrien true)
12        (setSenseBallPos true) ;
13        (addNoise false)
14    ~/.simspark/spark.rb
15        $agentSyncMode = true
```

Listing 3.1: SimSpark simulation settings

3.2 RoboViz

While it is key to have a reliable simulation environment, it serves little purpose if no visual representation can be attained. Thus, RoboViz [2] stands as the official monitor tool for the RoboCup 3D Simulation League, as a Java based fork of the original version by Justin Stoecker. It is developed and maintained by the *magmaOffenburg* team and seeks to improve from the dated, non intuitive and low performance graphics given by the native SimSpark monitor, in a way that deeply integrates with it [84, 85].

Additionally from providing a better representation of the ground truth, it allows for the graphical display of objects, such as circles and lines, advanced camera modes and log file generation. The result can be seen below in figure 3.3:

[2] Roboviz Repository https://github.com/magmaOffenburg/RoboViz (visited on 05/01/2020)

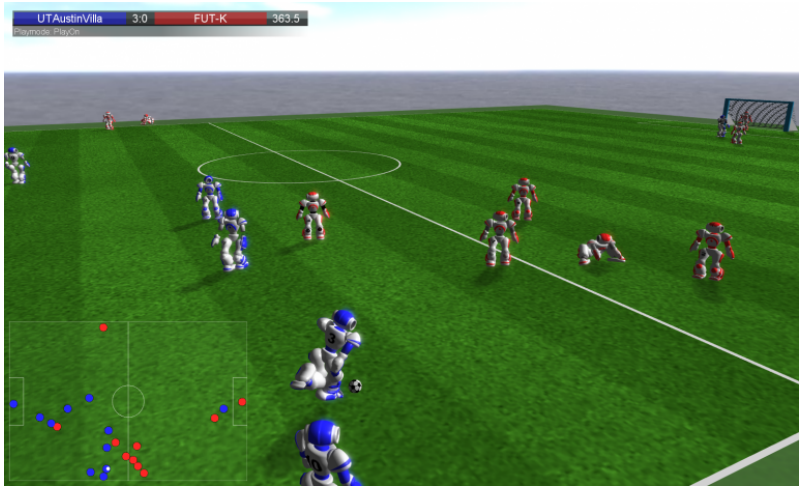

Figure 3.3: Example RoboViz visualization [86]

It should be pointed that although it entails in no behavioural difference in the simulation, visual data from the server is only updated to RoboViz on every other step.

3.3 NAO Robot

With an understanding of the platform on which the simulation occurs, we can analyse the NAO robot[3], or more concretely, the simulated model of the robot, since this is not a perfect representation of the physical one.

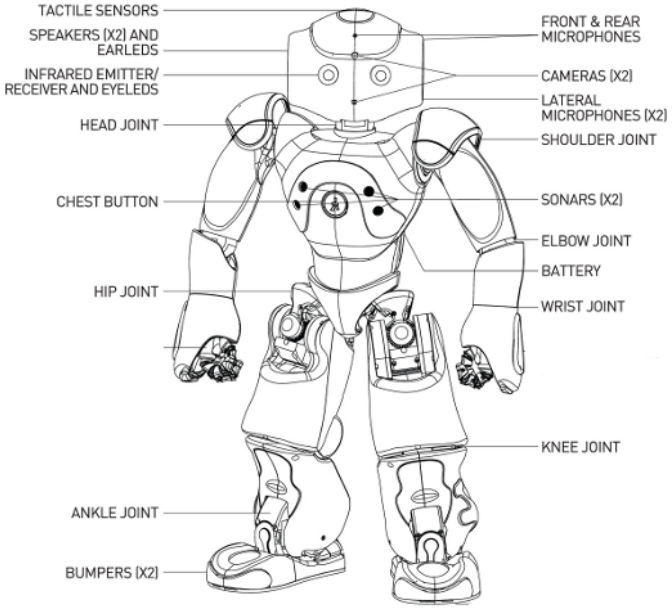

Figure 3.4: Physical NAO robot (H21 model) schematic [87]

[3]NAO Robot homepage https://www.softbankrobotics.com/emea/en/nao (visited on 11/01/2020)

The physical NAO Robot, now in its 6^{th} iteration, is a bipedal humanoid robot, developed by SoftBank Robotics (previously Aldebaran Robotics) and chosen by the RoboCup initiative as the default agent for the simulated scenarios. It is represented as a custom variety of its NAO H21 model with 22 degrees of freedom[4]. It should be noted that the RoboCup version omits the hand actuators from its simulation model.

The characteristics of the simulated robot are as follows[5] [89]:

- Height of 57cm.
- Weight of 4.5kg.
- Degrees of freedom: 22, one for each joint effector.
- Gyroscopes: 1 located on the torso.
- Accelerometers: 1 located on the torso .
- Force Resistance Perceptors: 2, one in each foot.
- Vision Perceptors: 1 in the center of the head.
- Say Effectors: 1, used for communications
- Hear Perceptors: 1, used for communications
- Joint Perceptors: 22, one for each joint effector.

3.3.1 Robot Joints

Comparing the all factors mentioned so far, we can see how the simulation tries to be a faithful model of the physical NAO robot, as can be seen through RoboViz on image 3.5 below:

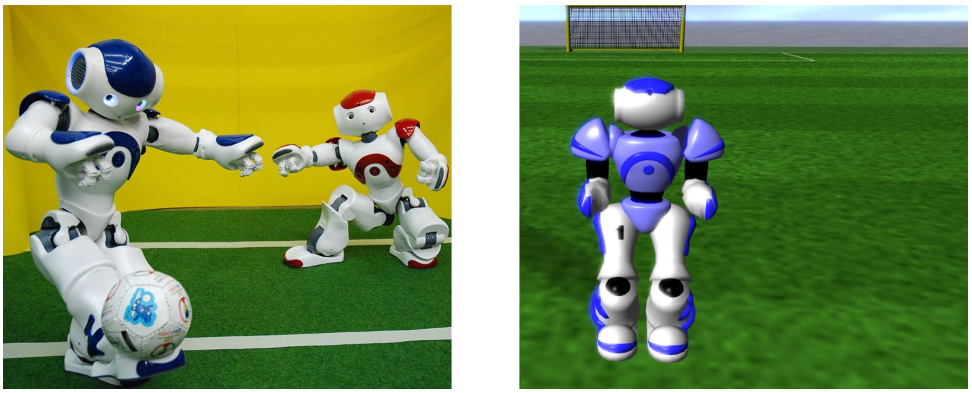

Figure 3.5: Visual comparison of real (left,[90])versus simulated (right) NAO robots

[4]The first hip joints in the physical robot share the same motor, hence the discrepancy implied in the name [88]
[5]For the default player type, see subsection 3.3.3

Nevertheless, there is a fair number of approximations taken in the simulation. First, as briefly mentioned as a footnote, the first hip joints of each leg, *LHipYawPitch* and *RHipYawPitch* act as independent joints in the model, whereas in the physical H21 model these are controlled by a single effector, resulting in a simultaneous and symmetrical movement at all times, with *LHipYawPitch* taking priority in a conflict situation [88].

As it can be seen in table 3.1 below, the chosen angles of the model correspond roughly well to its physical counterpart, with negligible rounding of values in most situations. It should be pointed that these are not the maximum and minimum angles in all conditions for the NAO H21, as there are conditional limits placed on some of the joints, depending on the angles of other joints, so as to prevent self-collisions and physical damage [91].

Simulation Model				NAO H21		
ID	Joint Name	Min	Max	Min	Max	Joint Name
0	head1	-120	120	-119.5	119.5	NeckYaw
1	head2	-45	45	-38.5	29.5	NeckPitch
2	lleg1	-90	1	-65.62	42.44	LHipYawPitch
3	rleg1	-90	1	-65.62	42.44	RHipYawPitch
4	lleg2	-25	45	-21.74	45.29	LHipRoll
5	rleg2	-25	45	-45.29	21.74	RHipRoll
6	lleg3	-25	100	-27.73	88.00	LHipPitch*
7	rleg3	-25	100	-27.73	88.00	RHipPitch*
8	lleg4	-130	1	-121.04	5.29	LKneePitch*
9	rleg4	-130	1	-121.47	5.90	RKneePitch*
10	lleg5	-45	75	-52.86	68.15	LAnklePitch*
11	rleg5	-45	75	-53.40	67.97	RAnklePitch*
12	lleg6	-45	25	-44.06	22.79	LAnkleRoll*
13	rleg6	-25	45	-22.80	44.06	RAnkleRoll*
14	larm1	-120	120	-119.5	119.5	LShoulderPitch
15	rarm1	-120	120	-119.5	119.5	RShoulderPitch
16	larm2	-1	95	-18	76	LShoulderRoll
17	rarm2	-95	1	-76	18	RShoulderRoll
18	larm3	-120	120	-119.5	119.5	LElbowYaw
19	rarm3	-120	120	-119.5	119.5	RElbowYaw
20	larm4	-90	1	-88.5	-2	LElbowRoll
21	rarm4	-1	90	2	88.5	RElbowRoll
* these joints were multiplied by -1 due to differences in chosen coordinate systems						

Table 3.1: Joint names and angle limits for the NAO robot[89, 91]

In Table 3.1, the ID and joint names on the left side correspond to how the joints are indexed in the XML file that describes the agent, and thus, in all of the functions and files pertaining to RoboCup programming. On the right, stands the joint naming convention used by SoftBank Robotics. Do note that naming conventions and indexation order of each joint change between the two and are not kept constant between all sources used in this dissertation. For the sake of clarity, the SimSpark naming on the left will be preferred except when explicitly dealing with the physical

NAO robot or when using 3^{rd} party figures.

We can observe on figure 3.6 how this table maps to the NAO robot. Additionally, see 5.1 to observe a simplified schematic with the coordinate systems in place.

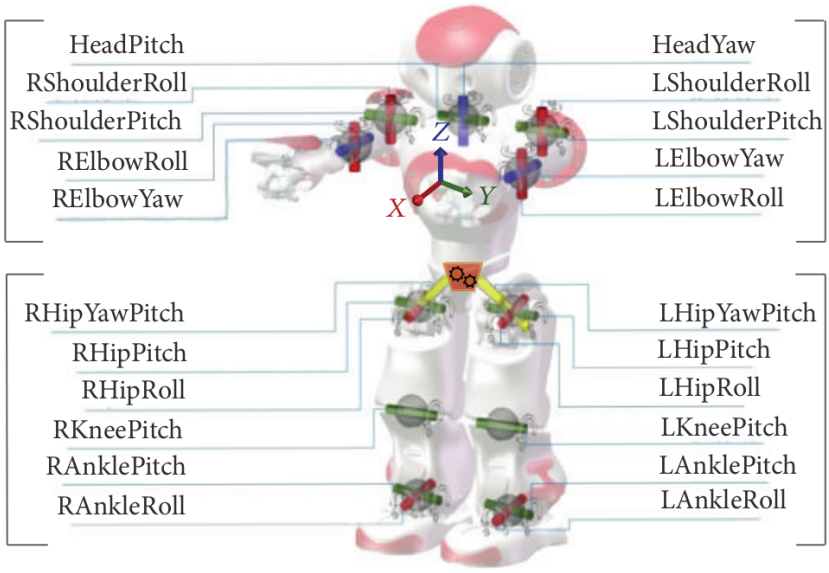

Figure 3.6: Joints mapped onto the NAO robot [92]

3.3.2 Box Model

While the approximation holds fairly well for the joints described in subsection 3.3.1 it should be mentioned that the model approximation is much coarser when regarding the actual physical shape of the agent. Visually they seem extremely similar, but the bounding volumes used for collision detection and physics simulation are in fact quite simplified.

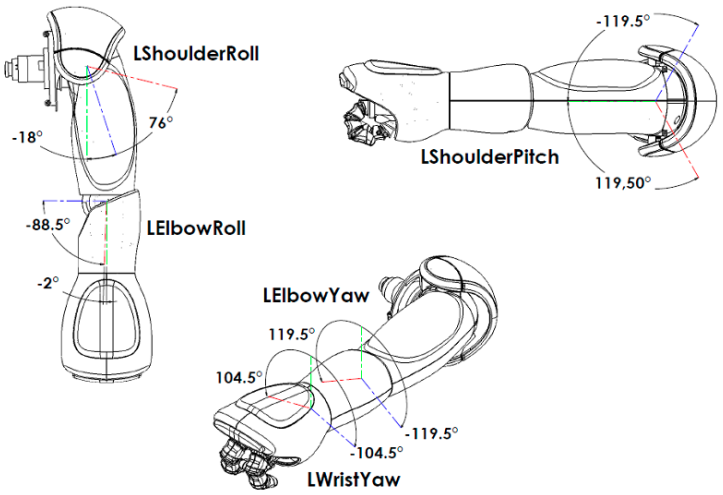

Figure 3.7: Schematic of the NAO H21 arm [91]

Taking for an example the arm, that can be seen in detail in figure 3.7 above, one can see how the arm in the real world has a very complex shape that can interact with an object in a wide variety of ways. On the other hand, the model used for RoboCup abstracts that to a set of two box shapes of equal value, with spheres to simulate the joint locations. This is extended to the rest of the robot, as the shapes are turned into a set of boxes, cylinders and spheres. The full visual description model can be seen on figure 3.8 below.

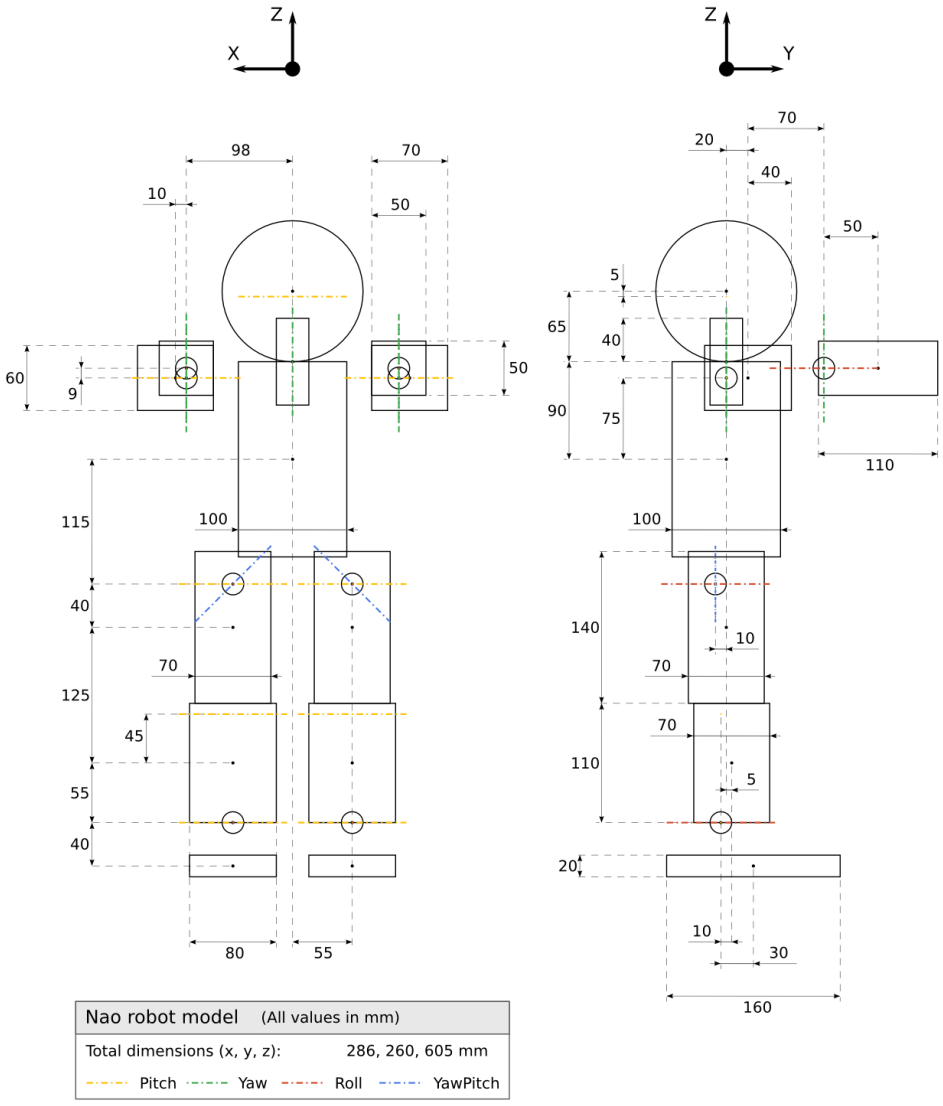

Figure 3.8: NAO robot box model for physics simulation [89]

While this model is still a far cry from the real world model of a NAO robot, and further still from that of a human player, it is a stepping stone on the way towards the goal of the RoboCup initiative.

3.3.3 Heterogeneous Players

Four further variants of the NAO robot have been developed, derived from the robot described above, as heterogeneous player types, including one with an additional joint in each foot.

According to the latest available rules, at least three player types must be used in a game situation. Nevertheless, the same general considerations of size and mass remain. For all simulations performed in this dissertation, the base *nao0* player type as described above was used [82].

3.4 Conclusion

This chapter sought to present the training environment upon which the learning occurs, based on the official RoboCup tools, with SimSpark as the training simulation server, RoboViz as the display interface and the NAO robot as the agent model, and the differences to their real life counterparts.

Furthermore, as described in section 3.1.3, in order to achieve better performance during training sessions, modifications to the server were implemented.

Chapter 4

Optimization environment: tools, agent and skills

With an understanding of the simulation environment and the robot, it becomes critical to describe how these are used alongside a set of tools to implement the goal of this dissertation, that is, the learning of further skills through the usage of RL algorithms.

This chapter seeks to present the remaining tools upon which this work is rooted, namely those that pertain directly to machine learning, some of which were developed by elements of the FCPortugal3D team during the realization of this dissertation, such as FCPGym, described in section 4.3, as well as give a description of the RL algorithm used in the following chapters.

Furthermore, this chapter seeks to describe the Deep Agent itself, and how the communication loop between the several components at hand occurs in a training scenario. Finally it introduces the two previously developed skills *run* and *sprint* that the kick in motion must flow from.

4.1 TensorFlow

TensorFlow[1] is an open-sourced, python-oriented, machine learning framework, developed by Google from its earlier *DistBelief* library. It uses dataflow graphs to represent state, computations, and the operations that change the states. It has revealed a great flexibility of its applications and usable hardware, ease of use and enormous scalability. Due to these advantages it has since gained wide adoption for academic purposes [93].

[1] TensorFlow homepage https://www.tensorflow.org/

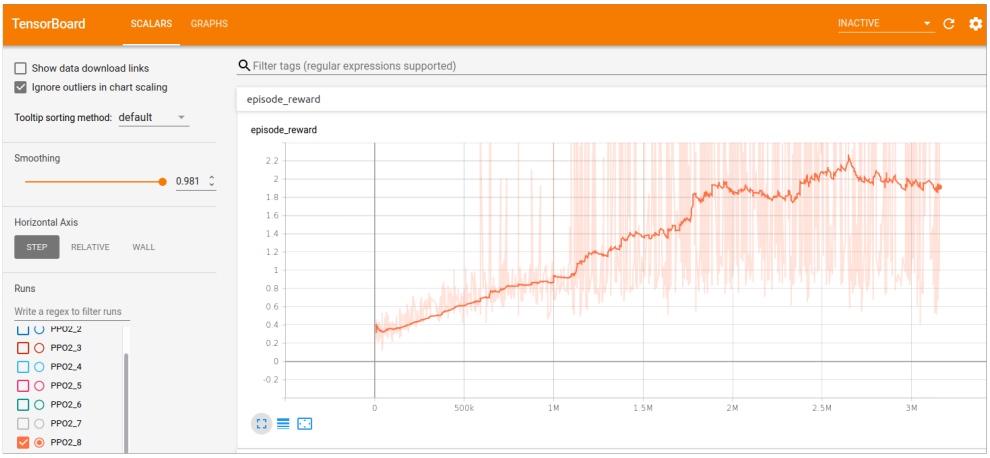

Figure 4.1: TensorBoard displaying results of a learning run

A key component to TensorFlow is TensorBoard, as this tool allows for the generated log data to be easily displayed and analysed, with regard to both rewards and other metrics such as entropy and loss, as shown of figure 4.1 above. Furthermore, detailed graphs of the inner structure of the network can be generated and interacted with ease as can be seen below in figure 4.2:

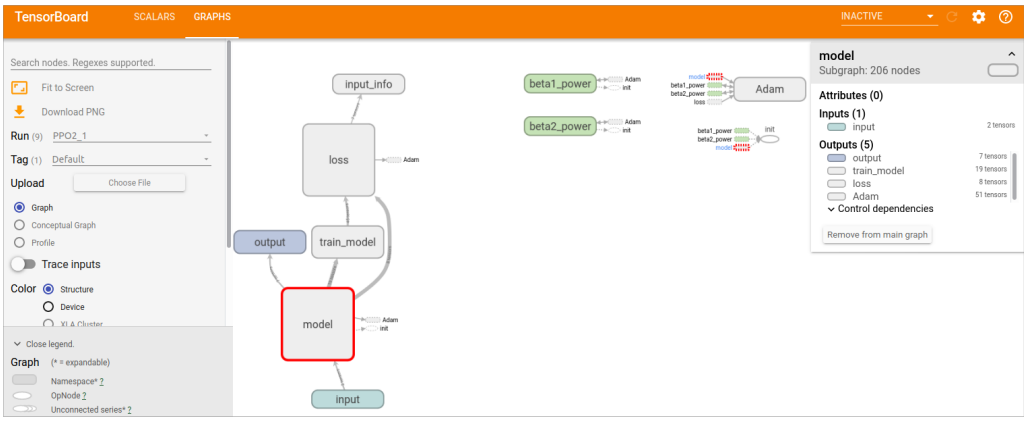

Figure 4.2: TensorBoard graph of a network used in this dissertation

Thus, TensorFlow underlies the whole dissertation, generating the underlying neural network upon which reinforcement learning algorithms operate. This was implemented through Stable

Baselines, as described in the next section.

4.2 Stable Baselines

While it is fundamental to have a solid framework for creating and performing computations on neural networks, they stand of little use without a proper Reinforcement Learning algorithm to back it up. It should be noted that such algorithms are extremely complex to create and to optimize its computational efficiency. Furthermore, the creation of one lies well outside the topic of this dissertation.

Thus, Stable Baselines[2] is a fork of OpenAI Baselines[3] that seeks to build upon it with solid documentation, TensorBoard support and several other features that make developing code much simpler. It has been used on previous RL work by the FCPortugal3D team with success and it was used to provide the RL algorithm implemented in this dissertation.

4.2.1 PPO2 in Stable Baselines

Although, many algorithms were considered beforehand to create the desired RL skills, as mentioned in chapter 2, in the end, Proximal Policy Optimization, briefly mentioned in section 2.7.8 was chosen as the preferred implementation. That was due to both its ease of implementation, proven capabilities in previous work done by the FCPortugal3D team and recommendations given by the supervising professor.

More specifically, the PPO2 implementation was chosen. The difference between PPO and PPO2 is mainly one of optimization, as PPO2 allows for greater capabilities in multiprocessing situations [94]. While a thorough description of all the intricacies of the PPO algorithm would be a task for a dissertation of its own, the main description of its working and how these values names map to the hyperparameters' in Stable Baselines is deserved. In PPO, the first step is to let the data to be sampled for a number of steps across the several threads, and then perform a number of epochs of stochastic gradient ascent on it. The objective function L_t^{PPO} for a vector of policy parameters θ is given as thus [60, 61]:

$$L_t^{PPO}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{4.1}$$

This equation can be divided in three sub-components. $L_t^{VF}(\theta)$ is a squared error loss related to the value-function, whose associated coefficient value was not tuned in this dissertation. On the other hand, S is an entropy bonus designed to encourage a greater degree of exploration, and prevent the solution to converge prematurely on a local minimum. $\hat{\mathbb{E}}_t$ is the expectation, that is, the averaged value over the sampled values.

[2] Stable Baselines repository https://github.com/hill-a/stable-baselines (visited on 16/01/2020)
[3] OpenAI Baselines repository https://github.com/openai/baselines (visited on 16/01/2020)

More relevant to understanding PPO is the first term of the equation, which can be expanded to [60]:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta))\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t]$$
$$for\ \ r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{4.2}$$

Where $r_t(\theta)$ is the ratio between the old and the new conditional probabilities of taking action a_t given the state s_t. The nature of the clip function makes it so that if a change in policy is beneficial it moves within the bounds of the trust region to prevent excessive movements that could undo the learning. Furthermore, $\hat{A}_t$ must be mentioned, as to assess this improvement, the calculation of the advantage is essential. This is done through a truncated Generalized Advantage Estimator (GAE) that is described as such [60, 95]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$
$$where\ \ \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{4.3}$$

Thus, with reward r and a value function V, the truncated version limits its horizon to the events up to T, the final step. More important are the two parameters at hand, γ, representing a discount factor for steps further ahead from the immediate moment in evaluation, and λ representing a ratio between a Temporal Difference ($\lambda = 0$) and a Monte Carlo ($\lambda = 1$) estimation approach. This is a trade-off, as Temporal Difference results in lower variance in exchange for bias and vice versa.

In the tests described in this dissertation, the algorithm is run for a preset number of steps, or until it deems it has reached a convergence situation, from which improvement is highly unlikely before stopping.

It should be noted that different implementations and frameworks have diverging names for the hyperparameters. During the following chapters of this dissertation, those used will follow the Stable Baselines PPO2 naming convention [94].

4.3 FCPGym

The FCPortugal3D team is, in its essence, a collaborative project that has been in constant improvement for many years; and in few places that is as clear as when dealing with the optimization environment and tools used throughout this dissertation.

At the start of the dissertation, a framework for implementing RL algorithms for a soccer simulation was already in place, OpenAI Gym[4]. OpenAI Gym is, in its essence, a toolkit to test and compare RL algorithms and strategies in a streamlined way, while making no assumptions on the environment used [96].

Said implementation presented both flexibility in the scope of its possibilities, and, most importantly, allowed for a great deal of scalability, permitting for multiple episodes to be ran as parallel threads for a greater usage of computational resources [83].

[4]OpenAI Gym homepage https://gym.openai.com/ (visited on 15/01/2020)

While this strategy was able to create successful training results, it left a great margin of improvement. Namely, the communication between existing *FCP agent* code (in C++) and the code pertaining to the optimization (in Python) was done through a back and forth communication through a series of TCP sockets. The passing of data worked through a series of *ad hoc* files, which required changes across multiple directories for every small change to be implemented.

Thus, during the realization of this dissertation, a new toolkit, FCPGym, was developed by Tiago Silva, a member of the FCPortugal3D team, which retained all of the advantages of the previous implementation it is based upon, while abstracting communications between the two layers through a third-party library. This has allowed to migrate all the code referring to the calculation and definition of rewards, observation and action spaces to C++, which further optimized computation while simplifying the programming process.

An important part of this toolkit is its functionality for fetching from the FCPortugal3D code a series of data points, such as object positions, joint angles and gyroscope data, as well as their maximum and minimum possible values and directly appending it to a vector to be sent back to the optimizer. This greatly streamlines the creation of new learning scenarios.

4.3.1 Vector spaces and normalization

While the action and observation spaces themselves will be addressed in detail in chapter 5, and from then on, wherever they are changed, it should be mentioned how normalization is used to improve the results. In essence, the observation space is a vector of the chosen input variables for the neural network, and these variables present a great range of magnitudes. While some can be a fraction of a unit, such as those representing foot data, others, such as those pertaining to the gyroscope can go into the hundreds. Since this variation of several orders of magnitude does not typically result in good learning data, a normalizing wrapper function *VecNormalize* was applied to the environment.

4.4 Deep Agent and learning episode

Over time, the FCPortugal3D team has developed a robust agent for playing soccer, and this makes up, with its comprehensive modeling and associated files, the basis of the Deep Agent used in learning situations. This agent has had its default decision module removed. In its stead, the loop described in figure 4.3 was placed, where the optimizer is none other than the algorithm already described in subsection 4.2.1 of this chapter.

The first step is the creation of a preset number N of parallel Gym environments, defined chiefly by the sum of the number of threads in all of the cores of the computer, so as to maximize computational power[5], each with their own associated SimSpark and agent instances.

The manner of communication between the Deep Agent and the Optimizer has already been described in section 4.3, but it should be explained how a step in the episode loop works. At the

[5]Due to inefficiencies in code, some machines do present a measurable, albeit diminishing return through the definition of multiple environments per thread, even when accounting for losses due to context switching

start of it, a set of actions is given by the optimizer to the Agent, which executes them over a time step of 20ms of simulation time. Following the taking of the action, an observation vector is returned to the optimizer that performs an evaluation of the result, before starting a new step.

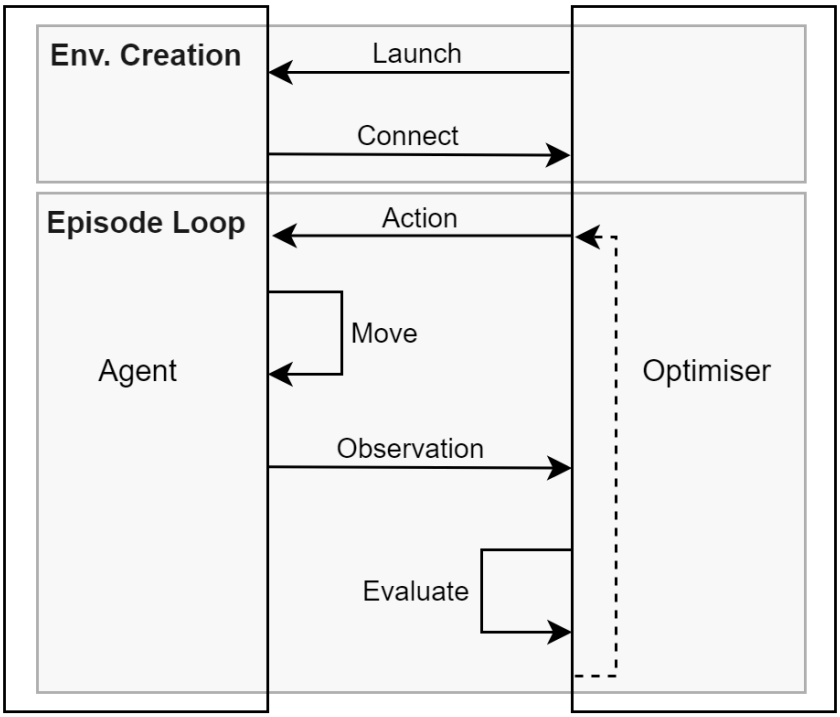

Figure 4.3: Agent - Optimizer communication in an episode [83]

The end of an episode occurs when due to one or several conditions defined in the step function, a *done* flag is triggered.

It should be noted that before the start of a learning episode, a reset period occurs. This serves to place both the robot and the ball back into their default positions and to reset the joints to their default states, without which would be impossible to provide a consistent initial state. This can be conceptually seen as the loop above where the optimizer is replaced by a set of given instructions. It should be pointed out that in this phase, there is no learning occurring.

Furthermore, since the goal of this dissertation is to create a kick in motion, the initial step of the learning must be one where the agent is already moving, thus it is here that the skills described in the section 4.5 were ran, so as to simulate the motion, which is an integral part of initial state of the learning episode.

Finally, it should be mentioned that several threads run in parallel and after a set number of steps, a stochastic gradient ascent, as described in subsection 4.2.1 is performed, after which, each of the optimizers in each of the threads is updated.

4.5 Skills

While the work so far has addressed most of the issues thus far presented, one key component remains to be described. This component is comprised of the skills that generate the motion from which the kick follows. These skills were previously developed by Miguel Abreu using a PPO algorithm [12].

In the end, two separate skills were used, which will be described in this section. It should be noted that these skills are, at the moment, undergoing constant development by members of the FCPortugal3D team, so it is possible that the behaviours and statistics used in the realization of this dissertation will be out of date, and further retraining, as defined in the next chapters may be required.

4.5.1 Run

The *Run* skill is the slowest of the two movement behaviours developed through the usage of neural networks. Like its *Sprint* counterpart, it uses two layers of 64 hidden neurons to generate a movement. Relevant data is as described in table 4.1 below, directly obtained from the developer:

Skill	Avg. linear speed	Max. linear speed	Max. rot. speed	Start time	Stop time
Run	1.41 ms^{-1}	1.51 ms^{-1}	160 $°s^{-1}$	0.9 s	[1 , 1.6]s

Table 4.1: *Run* Skill Characteristics

This behaviour does present a few advantages compared to the *Sprint* skill, such as the larger turning velocity[6]. Furthermore, its movement is more similar to that of a human, as can be seen in figure 4.4:

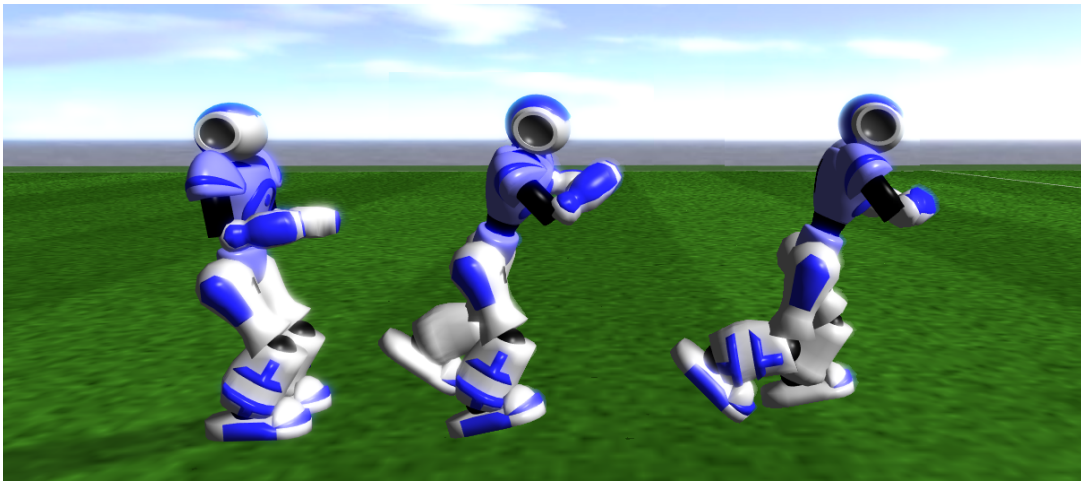

Figure 4.4: Several instants of the *Run* behaviour

[6] Although this has been seen to also induce some erratic side movements

This skill was thoroughly used in chapters 6 and 7 as the basis for the development of a kick in motion, as well as for the testing of other strategies, such as precision kicks.

4.5.2 Sprint

The *Sprint* skill is an extremely fast movement behaviour, developed in a similar fashion to the *Run* previously described, and with data as described in table 4.2 below, also directly obtained from the developer:

Skill	Avg. linear speed	Max. linear speed	Max. rot. speed	Start time	Stop time
Sprint	2.48 $\mathrm{m\,s^{-1}}$	2.62 $\mathrm{m\,s^{-1}}$	10 $^\circ\,\mathrm{s^{-1}}$	0.9 s	[1 , 1.8]s

Table 4.2: *Sprint* Skill Characteristics

Despite its faster speed contributing to more kinetic energy being transferred when kicking the ball, the *Sprint* skill does present some disadvantages. As it keeps its legs more bent, as visible in figure 4.5 below, when kicking, the movement of the leg, which moves in a pendulum-like fashion, presents a smaller distance between the hip and the foot.

Since this distance is proportional to the velocity of the foot, this results in smaller force imparted on the ball. Furthermore, the speed itself adds difficulty, as it reduces the reaction time allowed to perform a kick.

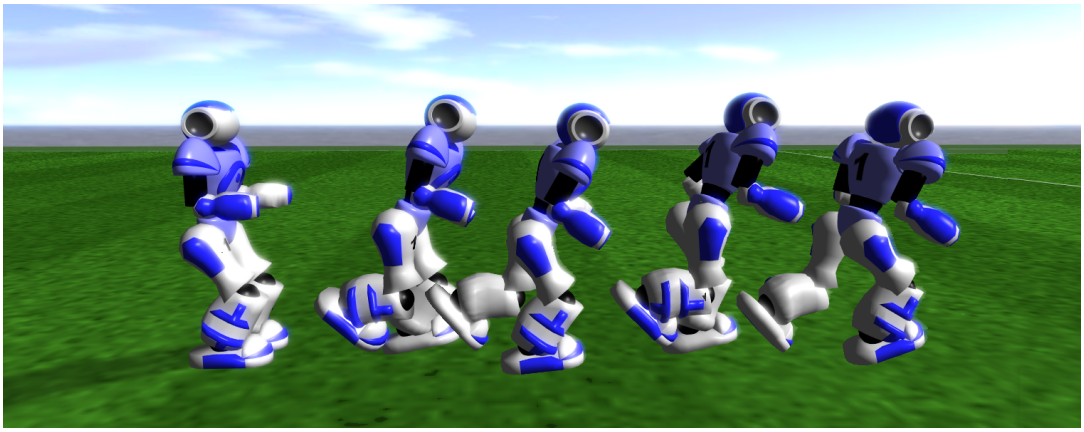

Figure 4.5: Several instants of the *Sprint* behaviour

This skill was also used in chapter 6. It should be pointed out that both skills are, despite their great capability, inaccurate, since they create a motion that tends to drift over time from the desired path. To fix this, an offset in the target location was given, so as to compensate for this movement.

4.6 Conclusion

This chapter sought to describe the tools and frameworks upon which the RL work developed in the next chapters is rooted, as well as give a more detailed description of the algorithm chosen for this dissertation. Furthermore it described the deep agent, the execution of a learning episode and the skills upon which the kick in motion must follow from.

With this knowledge, it is now possible to address the first attempts done towards the development of a kick strategy.

Chapter 5

Static Kick - A Proof of Concept

To be able make a kick in motion is a skill that can greatly improve the performance of any team, as it eliminates the time between stopping and kicking, precious time that can be used by the opposing team to take rearrange the position of their own robots or even worse, take back possession of the ball.

Starting to solve that problem however required the solving of several smaller ones, the process of which is described at length in this chapter.

5.1 Problem

Previous work had been done by the FCPortugal3D team to try and create kicking motions using Machine Learning. While approaches based on discrete key-frames, where several *slots* containing joint positions are interpolated over time with a sinusoidal function and optimized using the CMA-ES algorithm mentioned in 2.7.10, the strategies based on RL techniques using neural networks, such as those described in this dissertation or the ones used to make the skills described in section 4.5 failed to obtain satisfactory results [83].

Thus, before even trying to create a full kick in motion, the fundamental question of whether it is or not possible to create a kick had to be answered. To do so, a simplified scenario with an initially static robot was created.

5.2 Action Space

The first step was to create an action space, that is, an N-dimensional vector that encodes the degrees of control that the optimizer has available. While in theory these can map to anything chosen, such as a set of macro-actions involving whole limbs, the option was taken to map each of the outputs of the action space to an actuator of the robot.

Furthermore, the *head1* and *head2* joints were excluded, reducing the total dimensional size of the Action space to 20. This is possible since in a kick scenario, moving the head joints brings little

advantage to the agent, as can be understood from the joint schematic in figure 5.1. Thus, reducing the dimensionality of the Action Space tends to result in better results and faster convergence.

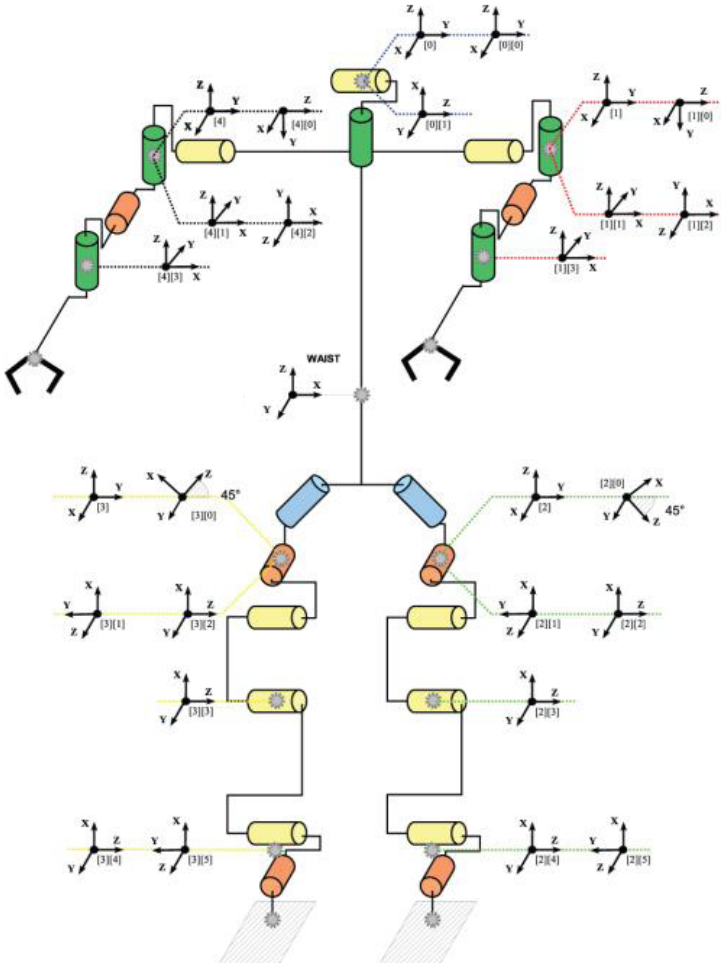

Figure 5.1: Schematic illustration of the joints in a NAO robot [89]

While the action space corresponds to a movement by the effector, there are several possible avenues on how to transform an output from the Action Space into an input for the NAO robot to react. It should be pointed that the communication to the robot is done through the function `agent::actions->moveJoint(i,vel);` , where i is the ID from table 3.1 and *vel* is an angular velocity expressed in radians per second and clipped to a maximum of $\pm 6.109 \text{rad}\,\text{s}^{-1}$, thus any choice must be bound to this.

Early attempts tried to directly relate the values of the action space to the angular velocities, using variations on a simple proportional or a complete PID controller. Nevertheless, as also verified in previous work [12], a noticeable improvement is obtained through interpreting each term of the action space as relating to an angle, and from there deriving an angular velocity to feed as an input. In the end, the desired objective angle θ^{goal}, can be obtained from the action space

output as:

$$\begin{aligned} \theta_i^{goal} &= b_i + (\frac{x_i}{k} + 1) * v_i - v_i \\ where \quad v_i &= \max(-m_i, n_i) \end{aligned} \tag{5.1}$$

Where, for a joint[1] i with an action space output of x_i, b_i is a bias term, used to make it so that if $x_i = 0$ we obtain $\theta_i^{goal} = b_i$. This implementation is further described in section 5.6. The rest of the equation is a particular case of a linear interpolation, where m_i and n_i are the minimum and maximum angles for the joint. Finally, k is a dampening constant, empirically set to $k = 5$, allowing for finer movements. Although this solution can result in invalid angles, since the robot cannot move beyond them, this is not an issue, rather, an incentive to keep applying a certain torque if advantageous.

With θ^{goal} now defined we can now obtain from it the angular velocity as:

$$\omega_i^t = (\theta_i^{goal} - \theta_i^{t-1}) * r - \omega_i^{t-1} \tag{5.2}$$

Where θ_i^{t-1} is the value of the joint, as given from the server in the previous step. Since, as can be seen in figure 4.3, the action is given before the step, the information has a one tick delay, thus requiring an ω_i^{t-1} term to account for this delay. Furthermore, both angular velocity values are clipped to their maximum values. Finally, r is an additional constant, used to convert from degrees per step to $\mathrm{rad\,s^{-1}}$ and given by $r = 50 * \pi/180$.

5.3 Observation Space

The observation space in the static kick consists in a set of 138 input variables, of which 25 are instances of differentiation between some other variable in steps $t-1$ and t. These variables are grouped in table 5.1 according to their acquisition method and data size, that is, how many variables correspond to a certain description.

By acquisition method, three types of variables should be considered:

- Raw : Data provided directly by the `FCPGym` framework, usually directly from the server.
- Generated : Data generated by the optimizer, such as the action space or the counter.
- Calculated : Where data is processed to obtain a better result.

Raw data includes the angular value from the joints' perceptor as well as their velocity and acceleration and the agent's current height. Furthermore, it gives the internal gyroscope and accelerometer data as well as the foot force data, giving force and contact values for each of the feet. These are given as multiples of three since data is expressed through a Cartesian coordinate system.

[1] Since the head joints (ID 0 and 1) are not used, in code, output i relates to joint $i+2$

It should be pointed that while it would be possible to provide the estimation of the position of the robot, as well as its rotation in reference to absolute coordinates, these are not given, so as to allow the learning algorithm to learn for relative positions instead, freeing the resulting action from part of the context for greater versatility.

Acquisition Method	**Parameter**	**Data Size**
Raw	Joint angle	22
	Joint velocity	22
	Joint acceleration	22
	Feet Force *	12
	Accelerometer *	3
	Gyroscope *	3
	Height *	1
Generated	Action Space	20
	Counter	1
Calculated	Foot Distance *	4
	Average Distance *	2
	Ball Speed	1
	* Differentiation	25
Total		138

* Values marked as such have differentiation performed on.

Table 5.1: Observation State Parameters and Details

Regarding the generated values, the action space is stored from step $t-1$ and given as an input for the neural network at step t. The counter is the internal counter that stores the current step, from 0 to max_step. Data from all the joints is gathered, since the head joints' noisy movement, although not actuated, can bring about perturbations to the robot.

Before considering the calculated values, a very important factor must be addressed, which is the data update cycle. While each tick of the physics simulation corresponds to an interval of 0.02s, and internal values of the robot, such as the angle of the joints, are updated in each of these ticks, data from the vision perception is only updated every third step. Furthermore, all this data has a one step delay, as mentioned in section 5.2. This means that some data is not updated, resulting in outdated data.

Previous work done by the FCPortugal3D team has successfully sidestepped this issue by only considering every third tick for the simulation, and performing the same action for those where data is not up to date [12, 83]. Unfortunately, given the nature of the kick, and the short time-frames it entails, this sort of granularity was not acceptable. Thus, solutions to this issue were found and described in section 5.5.

While differentiation has already been addressed, the other remaining values must be explained. *Ball Speed* is a value that corresponds to 10 times Δr, the magnitude of change in the estimated absolute position of the ball from step to step. The multiplication factor was chosen empirically, as this component is used in the reward function described in section 5.8. Since this is a noisy value, is clipped so that values under 0.05m are rounded down to zero. In steps where no server data update occurs this value is kept as is from the previous step.

Finally, average distance is the average between each of the feet distances. Each of these is given in polar coordinates, disregarding the z-axis component and estimated when no data update occurs. The method of calculating these positions is explained at length in section 5.5.

5.4 Network Shape and Hyperparameter Tuning

During the realization of this dissertation, a variety of possibilities was studied for modifying network shape, in terms of numbers of layers and of neurons in each of these layers. It should be noted that the default policy, and the one used in previous work by the FCPortugal3D team uses two layers of 64 neurons each, in both the value and the action networks, none of them shared. For the static kick described in this chapter, an improvement was found in adding a third layer of 64 neurons.

Regarding hyperparameters, these are extremely difficult to properly iterate on, requiring a great number of attempts to successfully cover the range of possibilities accurately. Thus, for the static kick ability, the option was taken to largely conform to the default parameters defined in Stable Baselines, or used in previous work by the FCPortugal3D team. The chosen values, when differing from the default parameters set by the Stable Baselines algorithm, are thus defined in table 5.2, using its variable naming convention:

Hyperparameter	**Value**	**Default Value**
n_steps	1024	128
ent_coef	0.00	0.01
noptepochs	10	4
learning_rate	2×10^{-4}	2.5×10^{-4}

Table 5.2: Modified Hyperparameters for Static Kick

5.5 Obtaining data for foot positions

For any kick, one can arguably say that the most important ability is to be able to effectively hit the ball with the foot. Thus, obtaining a reliable algorithm to measure the distance between the foot and the ball is a cornerstone of any attempt to obtain a successful kick behaviour. Additionally, this must be able to create a result even in situations where there is no data update from the server, based on previously held data.

To do so, the first step is to define a function to obtain this distance in the situation where a data update occurs. This is done through the following formula, resulting in a distance vector between the surface of the ball ($ball'$) and the tip of each foot ($foot'$):

$$\vec{P}^{t,side}_{ball',foot'} = \vec{P}^{t}_{ball,agent} + \vec{P}^{t,side}_{agent,foot} + \vec{P}^{t,side}_{correct} \tag{5.3}$$

The first component from the equation, the relative distance between the head of the agent and the ball, is trivial to obtain from the `FCPGym` framework. For the second component, some extra work is done, for even though the agent possesses a sense of proprioception, and is able to give the relative distance between the center of the head and each of the joints and body parts, it does so with regard to the agents' own relative coordinate frame, based on the rotation and inclination of the head itself. Thus the `worldState->visionTransform(Vector3f)` function was called[2] to perform an estimation of the correction of the reference frame.

Finally, the $\vec{P}^{t}_{correct}$ component serves to account for the fact that both of these two previous vectors assume that both the ball and the foot are point objects, while in fact they have a well defined hitbox, as described in 3.3.2 and 3.1.1. While the difference is seemingly small, at about 0.1m, when dealing with precisely hitting the ball, which has a radius of 0.02m, such a level of accuracy is not acceptable. The addition of this correction factor gives an offset for this and is calculated as such:

$$\vec{P}^{t}_{correct} = \begin{bmatrix} 0.1\cos\theta^{t} \\ -0.1\sin\theta^{t} \\ 0.025 \end{bmatrix} \tag{5.4}$$

Where θ^{t} is the estimate of the absolute rotation angle of the agent, as given by the `FCPGym` framework. The 0.1 component comes from the sum of the radius of the sphere with the distance between the center of the hitbox and half its length. The component for the z-coordinate was interestingly found to be of 0.025m, instead of the 0.015m expected for the difference of the heights when at rest. The resulting vector can thus be observed in figure 5.2 below:

[2] This function needs to be changed from private to public in order to access it by default

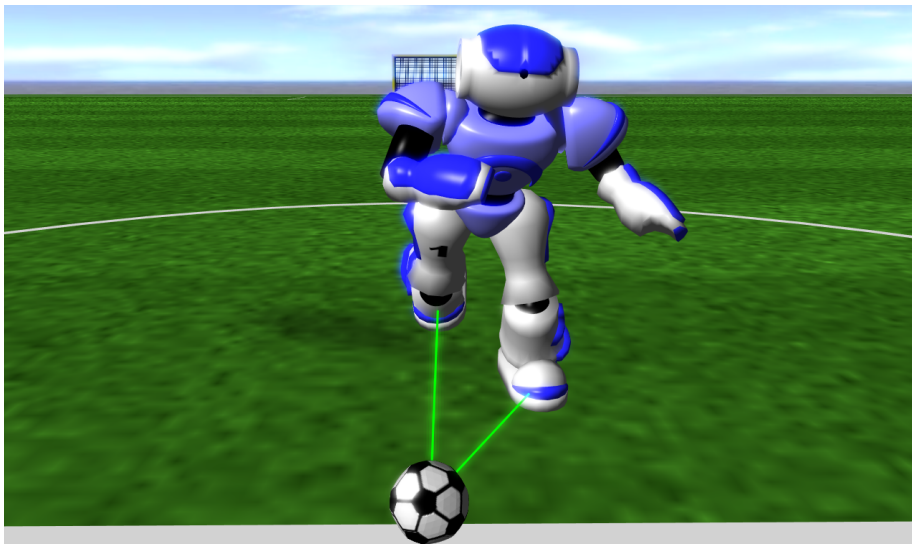

Figure 5.2: Foot to ball distance on server update step

With these angles calculated, a conversion to polar coordinates was performed, where the angle obtained is subtracted by θ^t, so as to provide a relative angle from the agent's frame of reference. It should be pointed that while magnitude was obtained from the three vector components, the relative angle uses only the xy-plane.

5.5.1 Accounting for server data

While the algorithm as described above works very well when all data from the server is updated, this same result is found to be lacking when accounting for the two thirds of the steps when it is not. It should be pointed that this is not an issue for $\vec{P}^{t,side}_{agent,foot}$, as this parameter is constantly updated. On the other hand $\vec{P}^{t}_{ball,agent}$ is only updated every third step, as previously mentioned in section 5.1, resulting in a movement that is twitchy and often simply incorrect. To account for this, an estimator algorithm was employed, replacing the aforementioned component by a term $\vec{P}^{t}_{estimate}$ whenever data is not updated from the server. Assuming that the data was last updated in step k, the algorithm is as such:

$$\vec{P}^{t}_{estimate} = \vec{P}^{k}_{ball,agent} + (v^k_P \times \frac{\Delta t}{3} + \frac{a^k_P}{2} \times \frac{\Delta t^2}{9}) \times d^t$$

$$\textit{where} \begin{cases} \Delta t = t - k \\ v^k_P = \vec{P}^{k}_{ball,agent} - \vec{P}^{k-3}_{ball,agent} \\ a^k_P = v^k_P - v^{k-3}_P \\ d^t = \begin{bmatrix} 0.75 + |\,0.5\cos\theta\,| & 0.75 + |\,0.5\sin\theta\,| & 0.66 \end{bmatrix}^T \end{cases} \tag{5.5}$$

Thus, $\vec{P}^{t}_{estimate}$ may be seen as an interpolation for intermediate steps of a simple kinematic equation for discrete times. Furthermore d^t is an empirically derived dampening factor, designed

to underestimate movement. This is beneficial, since a small jump every third step due to the update is still preferable to the oscillating movement that an overestimation would induce. Additionally θ is the same rotation of the agent as previously described in this section.

Even if it is completely impossible to say the exact error that the estimation has, since the ground truth is not available at these steps (or else this algorithm would be unnecessary), an upper bound to it can be established, by extending the estimate for a third step ($t = k+3$), beyond the required trust region of two estimation steps. Comparing the estimate to the updated value from the server, it can be ascertained with a reasonable degree of certainty that the error in $t = k+2$ will be lower.

Performing this calculation over several hundred iterations, we were able to obtain the data in table 5.3 below, for a motion on the x-axis:

	Mean Error (cm)				**Mean Absolute Error** (cm)			
	x	y	z	r	|x|	|y|	|z|	|r|
With Estimator	-1,85	-0.16	0.09	1.86	2.19	0.99	1.66	2.93
Without Estimator	-4.95	-0.29	-0.10	4.95	4.95	1.66	1.18	5.35

Table 5.3: Estimator error

Where r is the magnitude of a vector containing the three error components for the axis. For comparison, the average error between the previous and next server data was included. It was concluded that this is a very acceptable margin of error, as can be clearly visualized in figure 5.3:

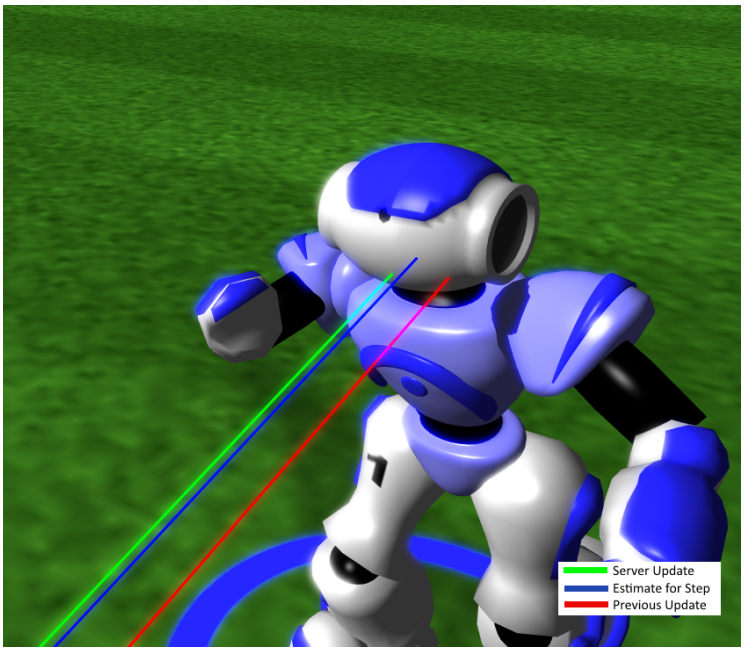

Figure 5.3: Agent to ball estimation and updated result

This further translates, for the final foot to ball distance, the value sought to be calculated, as an error as can be observed in figure 5.4 below:

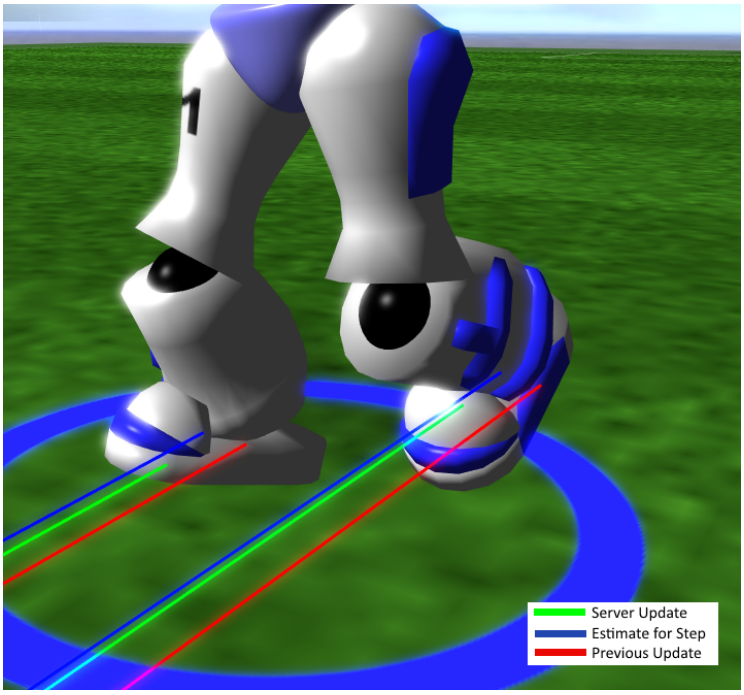

Figure 5.4: Foot to ball estimation and updated result

With all of this data, a solid estimate of the relative position of the ball to the tip of the feet can be obtained in every step, allowing for better accuracy, even more so, considering that the actual error will be smaller than the one above described.

5.6 Action Bias Vector and Initial Condition

The choice of implementing a Bias Vector emerged from the analysis of previous work by the FCPortugal3D team. In it, it was hypothesised that part of the reason for the underperforming behaviour from neural networks in previous kick attempts was due to the greedy nature of the algorithm attempting to maximize rewards in early steps, thus simply pushing the ball instead of first retracting it before swinging it back like in the pendulum-like fashion of a kick [83].

Thus, a series of bias values were chosen, so that for an action space output of $\underline{0}$ from the neural network, the robot would move towards a kicking motion with open arms and the leg performing the kick pushed back. This can be observed below in figure 5.5:

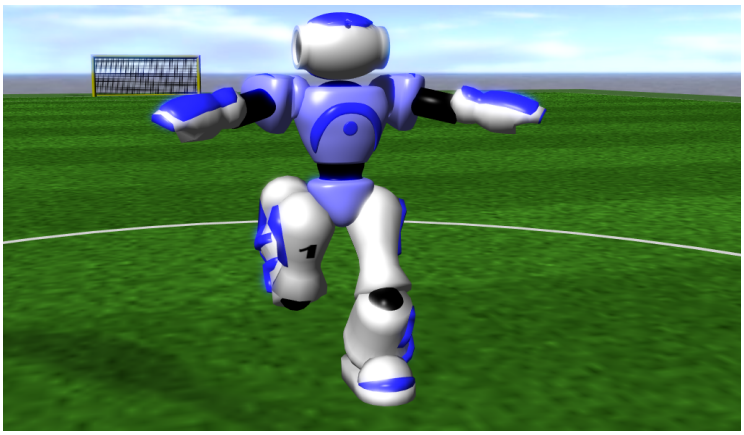

Figure 5.5: Visual representation of bias vector

In this manner, an initial incentive encourages the push back motion while the algorithm exploration deduces the forward motion, reducing the chance of the network getting stuck in a local maximum reward. This bias vector is represented in table 5.4 below, where ID and joint name are as in table 3.1. The values not represented are by default 0.

ID	**Name**	**Angle**
6	lleg3	20
7	rleg3	-25
8	lleg4	-51
9	rleg4	-130
10	lleg5	36
11	rleg5	40
14	larm1	-20
15	rarm1	-20
16	larm2	54
17	rarm2	-54

Table 5.4: Nonzero bias components for Static Kick

Furthermore, it is necessary to define the initial condition. As mentioned in section 4.4 this is done through a reset function. This function sets the angles to a predetermined set of values, as can be visualized in figure 5.6 below.

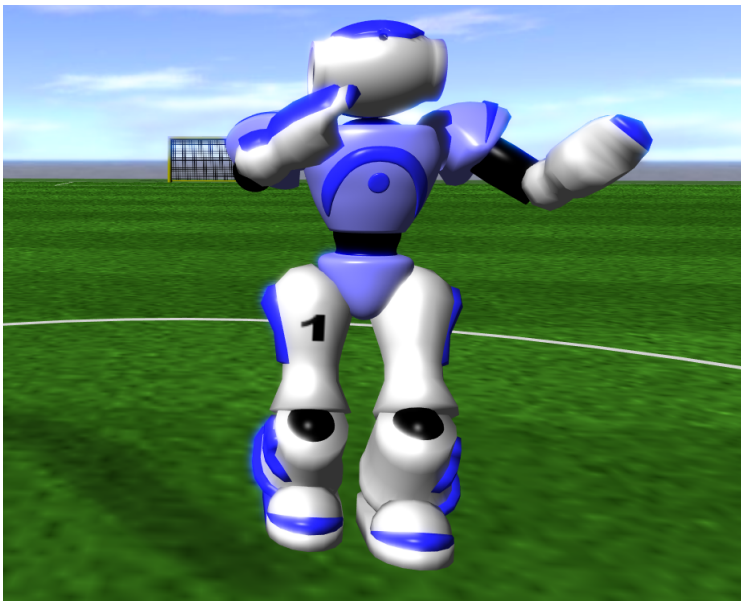

Figure 5.6: Initial conditions for the static kick

These values were chosen based on an averaging of the several of the joint angle values over several hundred steps while performing the *run* skill, as presented in table 5.5. Note that omitted values are equal to 0.

ID	Name	Angle
6	lleg3	20
7	rleg3	9
8	lleg4	-51
9	rleg4	-51
10	lleg5	36
11	rleg5	40
14	larm1	-64
15	rarm1	-20
16	larm2	54
17	rarm2	-18
18	larm3	-96
19	rarm3	35
20	larm4	-78
21	rarm4	89

Table 5.5: Initial nonzero angles for Static Kick

As it can seen in the figure 5.6, while the resetting of joints occurs, the agent is kept slightly above the ground before being allowed to fall on its designated position, and only when the agent

is firmly on the ground does the training part of the episode begin. The initial position of the ball is given by a point of absolute coordinates $P^0_{ball} = (0\ 0\ 0.02)$, and the agent position through $P^0_{agent} = (-0.2\ -0.06\ 0.5)$.

The combination of these two vector spaces results in a starting position that tends, by default, to move the joints from the initial to the bias position. It should be pointed that since the initial position is not very stable by itself, the movement, even when not using the action space inputs tends to result in the agent falling to the side.

This is actually an intended and beneficial effect, since it further emulates the inherent instability in the kick in motion situation, where, if one were to suddenly stop the movement skill, it would result in it falling over in almost every case.

5.7 Episode Layout

In the static kick, after the reset period, the optimization starts instantly, with the step cycle as described in figure 4.3 then occurring. It proceeds until the done flag is set to true. There are several conditions for this to happen, which are as follows:

- Agent height < 0.225 m
- Counter > 70
- Ball Speed > 0.05m

In the case of the ball moving more than 5 centimeters, the flag is not immediately triggered. Instead, the optimization is allowed to run for another 4 steps, since force can often still be applied, for a better final distance. Furthermore, it encourages the agent to take actions that result in it not falling over the ball, a problem often found in early attempts.

After these steps pass, the agent is transported away from the ball and a series of server side steps is done until the ball is stationary, without updating the optimizer. Only then is the final reward calculated. This is crucial, as although during the movement of the ball the action of the agent are irrelevant, thus giving more steps to the episode would only be detrimental to the intended learning, the final state of the system, as a result of the kick, is the real objective of the training.

Since that is so, it would be simple to assume that the reward function would result in a sparse reward environment. This would pose a significant problem, since a sparse reward tends to converge into a maximum result much more slowly, if at all, when comparing to a dense reward. Thus, several steps were taken to solve this issue as best as possible, as described in section 5.8.

5.8 Reward Shaping

As mentioned in the previous section, steps were taken to ensure that the effects of a sparse reward environment could be mitigated. For this, the following reward function for an episode $R^t_{episode}$ was chosen:

$$R^t_{episode} = R^t_{sparse} + \sum_{i=1}^{t-1} R^i_{dense} \tag{5.6}$$

With regard to sparse component, calculated after the done flag is triggered, it can be directly obtained from the equation

$$R^t_{sparse} = \Delta^2_x - \Delta^2_y \tag{5.7}$$

Given that the initial orientation and position are well defined in this case, one can directly obtain the reward as function of the movement on the x-axis, that is, in front, and penalize sideways motions on the y-axis. The quadratic component ensures that small improvements are progressively more rewarded, further encouraging exploration.

In further chapters, where, due to the erratic nature of the movement skills, the angle is not kept constant, a correction is applied as described in section 6.8

For the dense component of the reward, given on every step, this is given by the sum of two components as:

$$\begin{aligned} R^t_{dense} &= R^t_{foot} + BallSpeed_t \\ where \quad R^t_{foot} &= r^{t-1}_{foot} - r^t_{foot} \end{aligned} \tag{5.8}$$

With *Ball Speed* being the variable already described in section 5.1, and r_{foot} the magnitude of the vector between the foot and ball, as described in section 5.5. Since the initial position was preset, this foot distance is preset in the Static Kick thus described as relating to the right foot in particular.

The sum of this dense optimization space tends to be an order of magnitude smaller than the final component for a kick that moves the ball more than a few meters, thus becoming less and less relevant as the kick improves. It is important, in the early stages nevertheless, to guide the foot in its initial attempts at hitting the ball.

5.9 Results

The results for the training scenario described in the previous sections were taken after a training session of 30M steps over 25 hours, performed with 8 parallel threads on the personal computer of the student.

These were observed to be extremely positive, with the final results for the training scenario, evaluated for a sample of 1000 episodes, as described in the table below:

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	7	0.7	0	0
Bad kicks	24	2.4	0.82	]0 , 1.5[
Mediocre kicks	36	3.6	2.16	[1.5 , 3[
Good kicks	933	93.3	8.53	[3 , +∞[
Best kick	Distance : 12.52m			
Average time	Duration : 0.38s			

Table 5.6: Static Kick results

It should be mentioned that during the training phase results of over 13.2m were observed, although this may be seen as a statistical anomaly. Nevertheless, less then 1% of the kicks did not hit the ball, and over 93% moved the ball more than 3m. This distance was considered the threshold between an actual kick and a movement due to chance.

Taking into account these results, the static kick could already be considered a solid success. Even more so, when considering that the average time from the start of the episode to its end [3] is of only 0.38s, making it the fastest kick in the FCPortugal3D tool set. This in itself is already very advantageous for a much more dynamic game play in competitions.

The static kick can be observed in figure 5.7 below, showing several moments of a static kick episode.

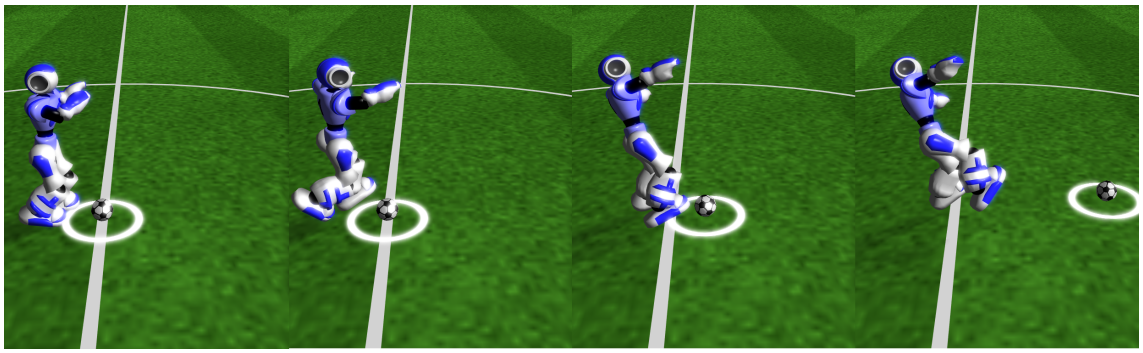

Figure 5.7: Several moments of a static kick

As expected, the neural network learned to first allow for the foot to retract, before only then starting to proceed with the forward foot movement, just as a human player would. Afterwards, the natural tendency is to fall away from the ball, avoiding the possibility that the fall may hinder the forward movement of the ball.

Additionally, taking the sampled data from table 5.6, frequency distributions of the kicks, according to their final distance and angle offset from a kick that travels straight ahead were obtained, as seen in figure 5.8

[3] when accounting for kicks that do not fail

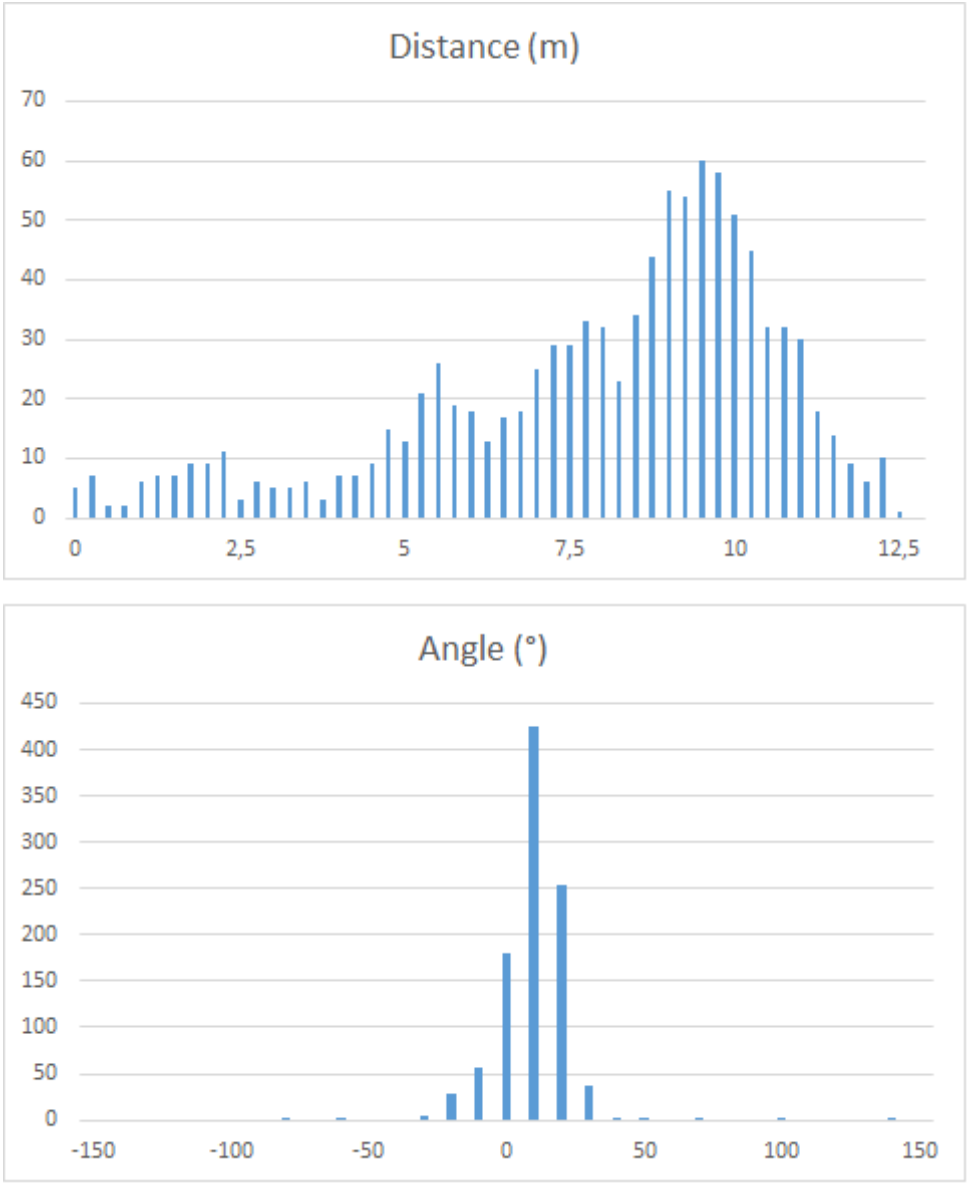

Figure 5.8: Static kick frequency distribution

There is a slight tendency for the ball to move to the left as it can be seen in the distribution. This might be explained by the small rotation in the first moments, induced by a change in the center of gravity due to the moving arms and retracting right foot, depicted in figure 5.7 above.

Finally a scatter plot was obtained, describing the final positions for a subset of 100 episodes, observable in figure 5.9 below, for starting position for the ball at the center spot of the soccer pitch.

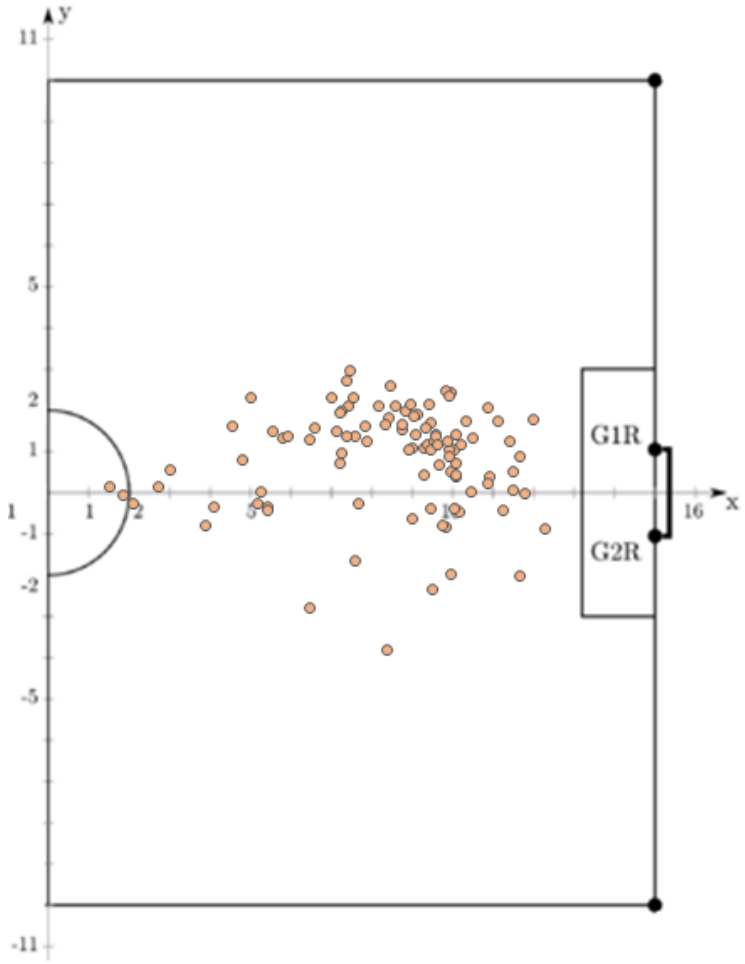

Figure 5.9: Static kick results for 100 kicks

These results allowed to conclude that, while not as powerful as some of the best kicks in the FCPortugal3D inventory, it is the fastest in the duration of its episode. Thus, the usage of neural networks to optimize static kicks was proven a viable strategy. It should be noted that it is very possible that extending the training may result in further improvements over time, but, due to timing constraints, and the fact that a static kick is only a stepping stone and a test-bed for results, the results here obtained were considered a very acceptable final result.

5.10 Conclusion

This chapter served to present the proof of concept of a static kick scenario, one where some of the dynamic concerns of the kick in motion were already at play, namely, a degree of instability of the initial situation.

To this end, a training scenario with its many components was developed as described. This structure was kept largely for the next chapters, except when thus mentioned. Additionally, a

method to reliably obtaining the relative position of the ball with regard to the feet, even in steps when the data is not fully updated from the SimSpark server was also developed.

The resulting behaviour performed quite successfully, proving the viability of the usage of these strategies for developing kick skills, and paving the way for the next chapters.

Chapter 6

Kick in Motion - A Method

To kick a ball is a natural and vital skill for any soccer team, since to do so is not simply to throw it between the goalposts. A kick can have have a huge swath of objectives, from passing the ball to another player to simply moving it to a more strategically advantageous position.

Thus, any improvement on that capability can mean the difference between victory and defeat in a match. With that in mind, developing a method to perform such a kick while moving is an asset which, if successful, can lead to substantial benefits for the FCPortugal3D team. This chapter seeks to provide a description of the changes done from the Static kick scenario to make it possible, and to present and discuss the results of the learning scenarios here described.

6.1 Problem

With the simpler question of whether it is possible to employ an RL strategy to learn a kick from a static initial position answered in chapter 5, it becomes possible to use these same tools, with modifications, to develop a set of kick behaviours for the existing movement skills.

Due to timing and computational constraints, discussed at length in sections 6.5 and 6.7, it was chosen to optimize a kick in motion technique for the maximum possible distance. The skills used for this chapter were *Run*, completely trained for all sub-scenarios described, and *Sprint*, partially trained.

6.2 Action and Observation Space

While the action space for the kick in motion, and the algorithm to convert it to a set of movements, have remained identical from that which was described in section 5.2, several changes to the observation space were done for the kick in motion so as to obtain better results. These are shown on table 6.1.

Acquisition Method	Parameter	Data Size
Raw	Joint angle	22
	Joint velocity	22
	Joint acceleration	22
	Feet Force *	12
	Accelerometer *	3
	Gyroscope *	3
	Height *	1
	Rotation *	1
Generated	Action Space	20
	Counter	1
Calculated	Foot Distance *	6
	Average Distance *	2
	Ball Speed	1
	* Differentiation	28
Total		144

* Values marked as such have differentiation performed on.

Table 6.1: Observation State for Kick in Motion

The modifications to the Observation Space add a few more variables, changing no existing ones. Namely, the estimation of the rotation given by the `FCPGym` framework and the addition of the z components left out from $\vec{P}^{t,side}_{ball',foot'}$, obtained as described in section 5.5. Note that this z-axis component value was not introduced in the average for the feet. Differentiation was also performed in these new parameters, for a total of 144 variables.

6.3 Early attempts

While many iterations of a strategy for the kick in motion were tried, both before and after the creation of the Static Kick proof of concept, the reality observed is that the neural network was proven incapable of simply handling the kick in motion by itself.

In these attempts, the agent was started a certain distance away from the ball, and the movement behaviour engaged. At some closer distance, the behaviour would be disengaged and the learning episode would start. While some attempts proved largely capable of keeping the walking movement going for a short distance after the sudden interruption, if need be, that usually came at the cost of not learning how to perform the kick itself.

The simple conclusion drawn from these trials was that episodes where the agent would only start learning in close proximity to the ball result in a training only focused on the kicking aspect, and therefore, a better final result. It follows that the idea of simply reducing the transition to a

small distance was experimented with. While a substantial improvement was noticed, the results were still largely underwhelming.

A frame-by-frame analysis of a series of kick episodes was then done, with the conclusion being that the nature of the behaviours at hand is such that, after a small number of steps, the position of the agent presents a large possible variation in both its position, rotation and joint angles. Thus, with such a wide swath of variation in the initial conditions of the observation state, it became much harder for the agent to learn a proper set of actions for each of the episodes. To mitigate this, a series of actions were taken, as described in section 6.5.

6.4 Hyperparameter Tuning and Network Shape

During the study of the kick in motion, some fine tuning was done to the hyperparameters of the neural network used. These sought to better reward behaviours that happen later in the episode, thus reducing the greediness of the algorithm. Furthermore, some velocity in convergence was traded by less variance. To this end, the set of hyperparameters, when differing from the pre-set values in Stable Baselines are given in table 6.2:

Hyperparameter	**Value**	**Default Value**
n_steps	1024	128
ent_coef	0.00	0.01
learning_rate	2×10^{-4}	2.5×10^{-4}
nminibatches	16	4
gamma	0.999	0.99

Table 6.2: Modified Hyperparameters for Kick in Motion

Additionally, a fourth hidden layer of 64 neurons was added for a better resulting performance. Although it is hard to extrapolate from the architecture of neural networks, it can be assumed that this helped to model the extra complexities and variations in initial conditions inherent in a moving kick.

6.5 Tailoring Kicks and Initial Conditions

As previously mentioned in section 6.3, using a single neural network on the *Run* and *Sprint* behaviours to create a kick in motion was an unfruitful task. Thus, a solution found to work around this issue, was to divide the problem into a set of smaller sub-problems.

In this way, taking the set of possible initial conditions, a basic criteria was set, such that an episode would only start when:

- The agent is properly oriented towards the ball (i.e., the behaviour has not set it astray).
- One of the feet is on the ground.

By setting these two basic criteria, and a tailored set of specific ones for each of the sub-problems we can obtain a number of more similar states for the initial conditions in each situation. This brings a clear trade off, for the larger the number of sub-divisions used, the better the learned behaviour will be, so will the number of neural networks needed to be trained grow, and the computation time alongside with it.

As a compromise, the problem was divided into 6 sub-kicks, 3 for each feet. The chosen criteria for each feet are given by the values in table 6.3 below:

Foot Kicking	Conditions	ID	Foot Down at t = 0	Foot Down Conditions
Left	$\theta_{left} < \theta_{right}$	(0,0)	Right	In line with ball
		(0,1)	Left	Behind ball
		(0,2)	Right	To the side and behind ball
Right	$\theta_{left} > \theta_{right}$	(1,0)	Left	In line with ball
		(1,1)	Right	Behind ball
		(1,2)	Left	To the side and behind ball
Any	$\left\lvert\frac{\vec{h}_{left,y}+\vec{h}_{right,y}}{2}\right\rvert < 0.12\text{m}$ $\lvert\alpha_{agent}\rvert < 20°$			

Table 6.3: List of Kick Sub-types and criteria

Where θ_{side} is the relative angle of the polar coordinate conversion of $\vec{P}^{t,side}_{ball',foot'}$, as described in section 5.5, and α_{agent} is the drift from the initial absolute angle for the agent, as given by the FCPGym framework.

The vector $\vec{h}_{side}$ is also derived from $\vec{P}^{t,side}_{ball',foot'}$ but a rotation matrix is applied to it for a angle ϕ to correct it for the agent's reference frame. This angle ϕ is a conservative estimate, derived from the application of an Infinite Impulse Response filter to the absolute angle for the agent. In this way $\vec{h}_{foot,y}$ is the sideways difference between the foot and the ball.

It becomes then a matter of setting the behaviour moving towards the ball, and in each time step verify whether any of the condition sets are met. In tests run, assuming that the agent does not drift due to the behaviour, in the overwhelming majority of the situations, there is at least one time step where one of the kick condition sets is obtained, which is then used as the starting situation.

The initial conditions for the learning episode can be observed in figure 6.1 and 6.2 for the *Run* and *Sprint* behaviours respectively.

6.5.1 Run Initial Conditions

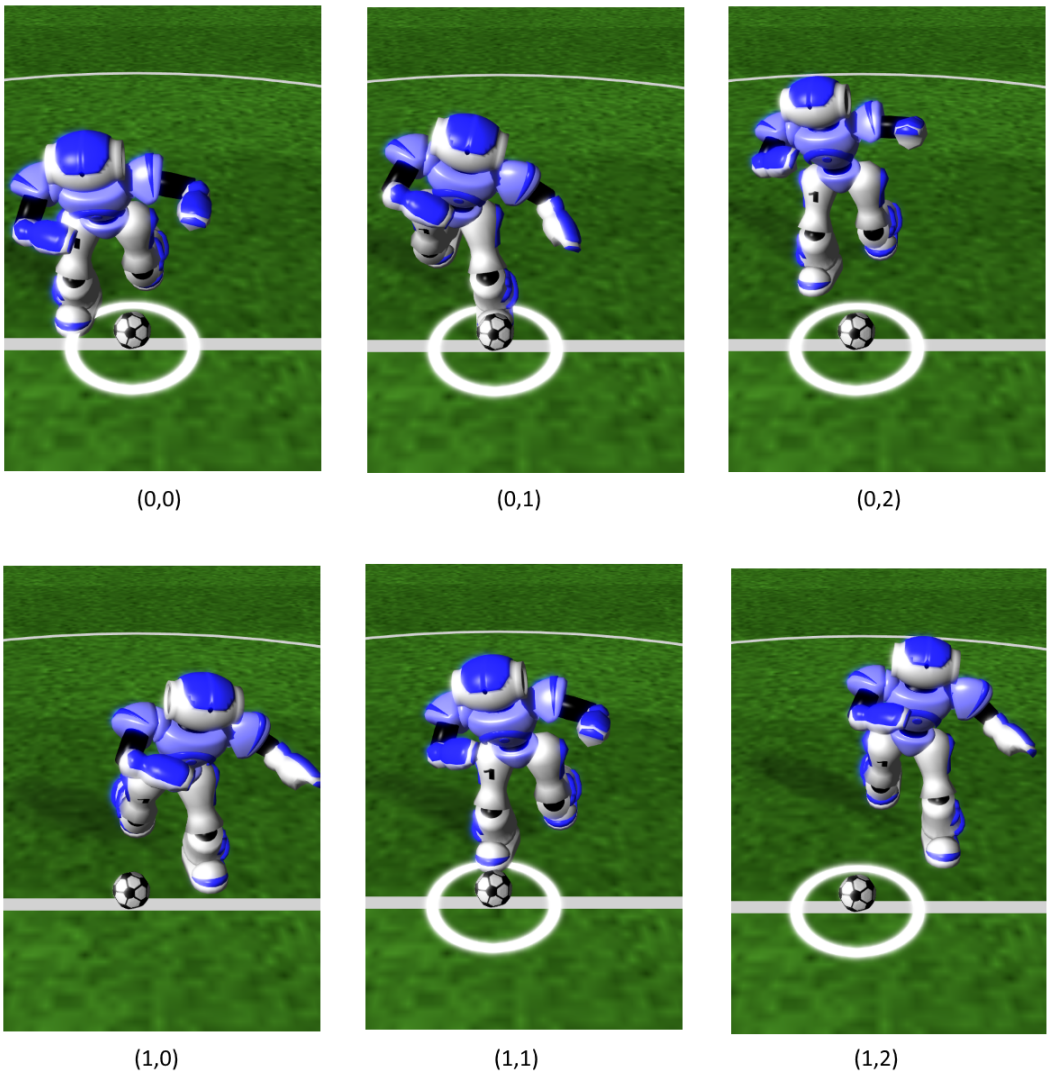

Figure 6.1: Initial conditions for the *Run* based kick

As it can be seen in the figure, the divisions chosen cover a wide range of the valid starting positions for a kick. It should be pointed that many more sub scenarios could be drawn, but for each one of them, a great deal of computational resources must be used.

Thus, this set of 6 starting conditions can be see as a good compromise between specificity and use of computational resources.

6.5.2 Sprint Initial Conditions

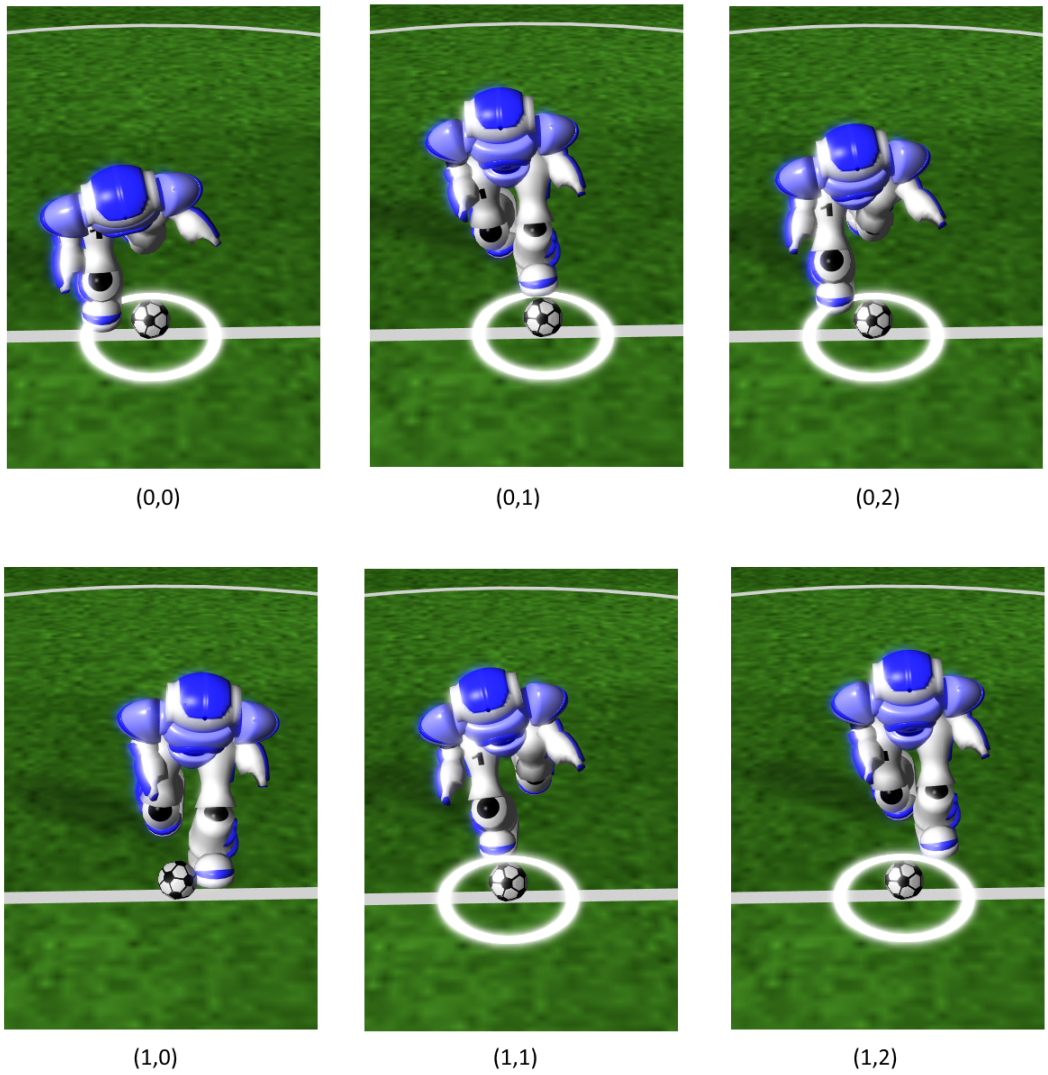

Figure 6.2: Initial conditions for the *Sprint* based kick

For the *Sprint* behaviour, not all of the sub-kicks were tested, Nevertheless, the initial behaviour can likewise be like divided as was done for the *Run* behaviour.

6.6 Action Bias Vectors

With regard to the action bias vector, the solution becomes trivially simple. Given the motion of the *Run* and *Sprint* behaviours, there are already moments in their gait that serve as good basis for the kick, with a foot pushed back and the other forward. Examples of these can be seen in figures 6.1 and 6.2 already for kicks with both feet.

A average of several of these positions was then taken and used as the Bias Vector, with the foot performing the kick always set as the foot retracted. This is less important in kicks of the type $(n,0)$ and $(n,2)$, as the joints will already be in the vicinity, but for kicks of the type $(n,1)$, it ensures that the kicking foot is retracted while encouraging the other foot forward. This is very similar to the situation described for the static kick.

6.7 Episode Layout

While the ending conditions are left unchanged, the episode layout is deeply modified in comparison to chapter 5. These changes come chiefly in the reset phase of the episode. As already mentioned in section 6.5, the kick in motion must naturally start with some motion. Thus, the agent is set at some distance away from the ball, and the behaviour is ran until a timeout or the criteria matching the type of kick being trained occur.

For the case they do not match, or the behaviour times out, the reset is contained in a wrapping function, and is itself reset, so as ensure that the initial conditions are as desired for every episode.

It should be pointed that although there is a chaotic component to the movement behaviours, for a small enough length, the initial position can be set, so that on average most kicks will be of a certain type.

This is important due to the massive amount of computational overhead inherent in a kick in motion. Since a kick in motion naturally requires movement, for every episode, dozens of extra time steps must be simulated for these behaviours. This unfortunately results in a much slower speed for the training runs when compared to a static kick.

6.8 Reward Shaping

While there are no conceptual changes in the reward shaping between a static kick and a kick in motion, a pragmatic one must be applied. For while the angle of the agent was a known constant in the static kick, in a kick in motion, an amount of drift from preset values is unavoidable. Thus, the final position of the ball, considered by the reward function is changed, so that:

$$R^t_{sparse} = {\Delta'_x}^2 - {\Delta'_y}^2$$
$$where \quad \begin{pmatrix} \Delta'_x \\ \Delta'_y \end{pmatrix} = \begin{pmatrix} cos\phi & -\sin\phi \\ \sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} \Delta_x \\ \Delta_y \end{pmatrix} \tag{6.1}$$

Here, the logic is similar to what is described in section 6.5 for obtaining $\vec{h}_{right}$, for the same ϕ, calculated only until the start of the episode. The rotation matrix accounts for the rotation of the agent at the start of the learning episode. In this way, the new position is taken as if the robot had performed a kick with the right orientation.

Do note that the corrected positions are the ones used to present the tabled results in the next section.

6.9 Results

The results described in this section correspond to a series of values obtained for the several sub-problems, as described in section 6.5. Some of these were performed in with 32 parallel threads on a server provided by the *Faculdade de Engenharia da Universidade do Porto*. Others were simultaneously trained with 8 parallel threads on the personal computer of the student.

6.9.1 Run

The *Run* behaviour was trained for the six sub-problems in question. The training runs were performed for a minimum of 20M steps and 24h each, over the course of several days.

The results obtained for each sub-problem are displayed in Appendix A for an evaluated sample of 1000 kicks each. The results displayed below on table 6.4 are the averaged out results for each of the tested sub-problems.

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	80	1.33	0	0
Bad kicks	536	8.93	0.75	]0 , 1.5[
Mediocre kicks	415	6.91	2.28	[1.5 , 3[
Good kicks	4969	82.81	5.71	[3 , +∞[
Best kick	Distance : 8,89m			
Average time	Duration : 0.33s			

Table 6.4: Results for *Run* behaviour

It should be noted that the several cases of the kick have displayed very different behaviours and results, depending on their relative position to the ball. Below in figure 6.3 is an example of a kick of type (1,0).

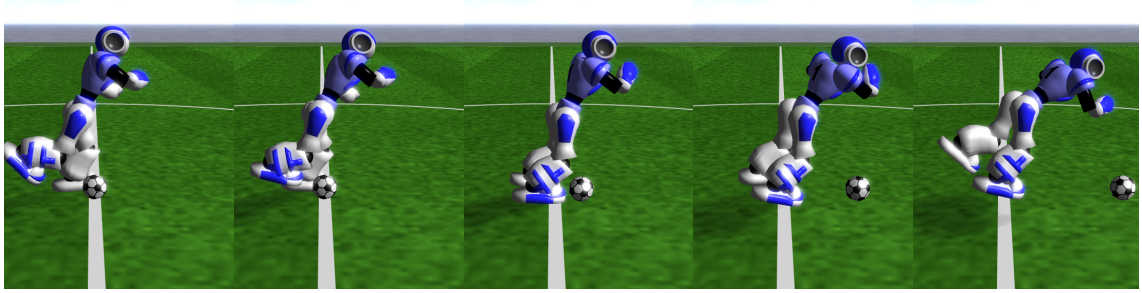

Figure 6.3: One episode of the *Run* based kick

Taking some examples, some of the best kicks were performed by the cases with the worst average performance, such as the Type (0,1), showing a large performance distribution. On the other hand, the kick of Type (1,0) has an impressive performance, with no single kick not hitting the ball

in 1000 episodes, but a considerably smaller maximum distance for a kick. When considering the overall time of the kicks, with an average of 0.33s, and going as low as 0.24s in some sub-problem cases, it becomes clear that this kick has a great deal of potential for application.

Additionally, frequency distributions were obtained for the episode data used in table 6.4, as seen in figure 6.4 below for the distance of the kick and the angle offset from a kick that travels straight ahead. The figures for each of the sub-problems are available in Appendix B.

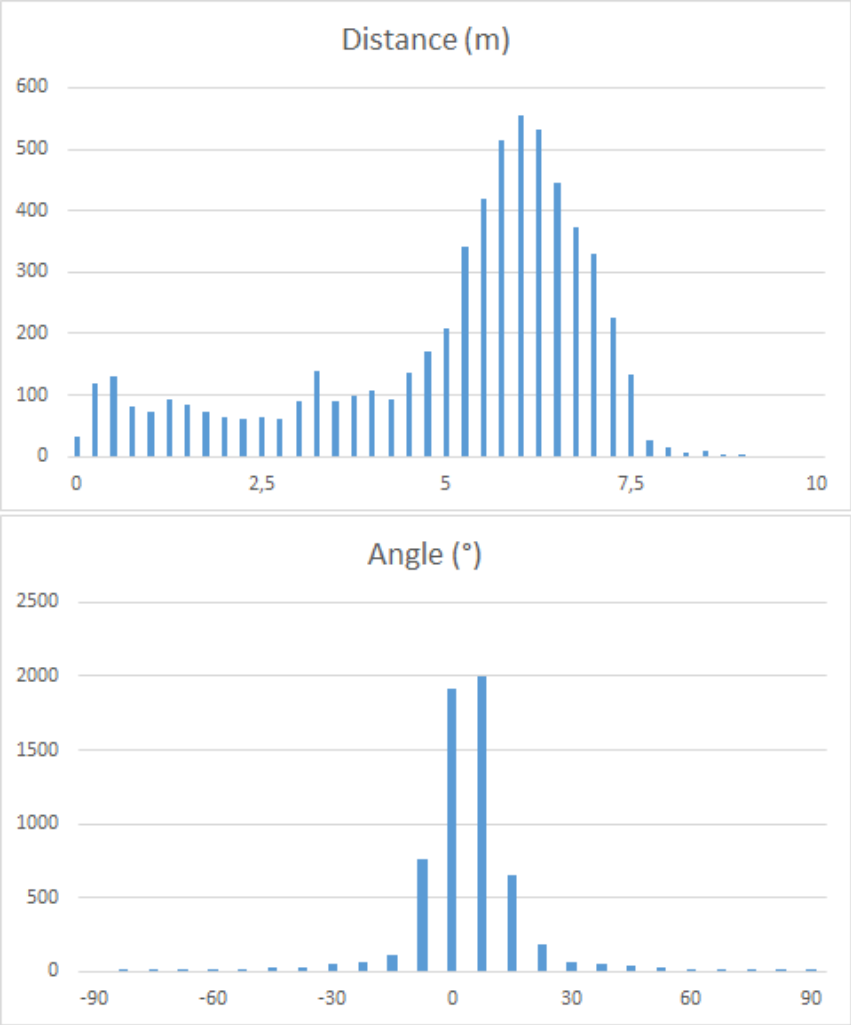

Figure 6.4: Frequency distribution for *Run* kicks

It should be noted how for the angle, the shape obtained is similar to that of a normal distribution, with a small offset already explained in section 6.8. For the distances, this is not the case, as there is a severe skew on the smaller distances. This might be due to the random nature of the initial state.

Additionally, for each of the sub-problems, 100 episodes were taken, and a scatter plot was generated, as seen in figure 6.5 below. The figures for each of the sub-problems are available in Appendix B.

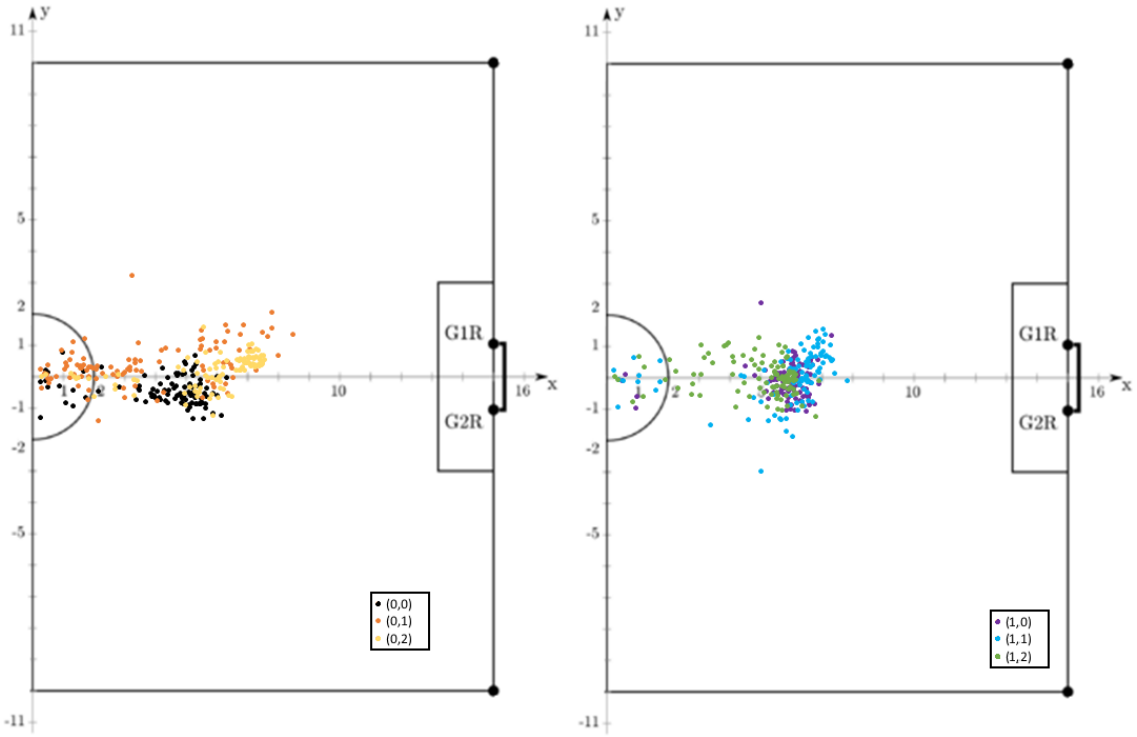

Figure 6.5: *Run* kick results on the field

6.9.2 Sprint

For the *Sprint* behaviour, only two of the six sub-problems were studied, namely $(0,0)$ and $(0,1)$. This was due to timing and computational constraints, for the calculation of every 1M steps took at least 2 hours, even when running on the server. This is due to the *Sprint* behaviour resulting in an even greater variation on the initial scenario, and consequently, a greater rate of discarded initial conditions. The results are described in table 6.5 for an average of the sub-problems, evaluated for 1000 kicks each. The complete results are available in Appendix A:

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	112	5.6	0	0
Bad kicks	134	6.7	0.70	]0 , 1.5[
Mediocre kicks	62	3.1	2.45	[1.5 , 3[
Good kicks	1692	84.6	5.27	[3 , $+\infty$[
Best kick	Distance : 7.04m			
Average time	Duration : 0.26s			

Table 6.5: Results for *Sprint* behaviour

These are extremely impressive results, especially when taking the average speed into account. It is considered that with some further fine tuning, these results may even vastly overwhelm the

performance observed for the *Run* behaviour. While not all *Sprint* based kicks are the same, a visual representation of an episode can be seen in figure 6.6 below:

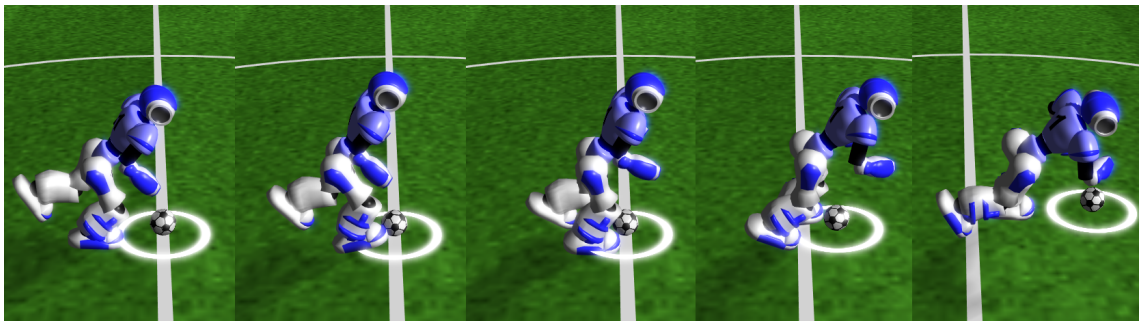

Figure 6.6: One episode of the *Sprint* based kick

Similarly as described for the *Run* skill, frequency distributions were obtained for the episode data used in table 4.2, as seen in figure 6.7 below, for the distance of the kick and the angle offset from a kick that travels straight ahead. The figures for each of the sub-problems are available in Appendix B.

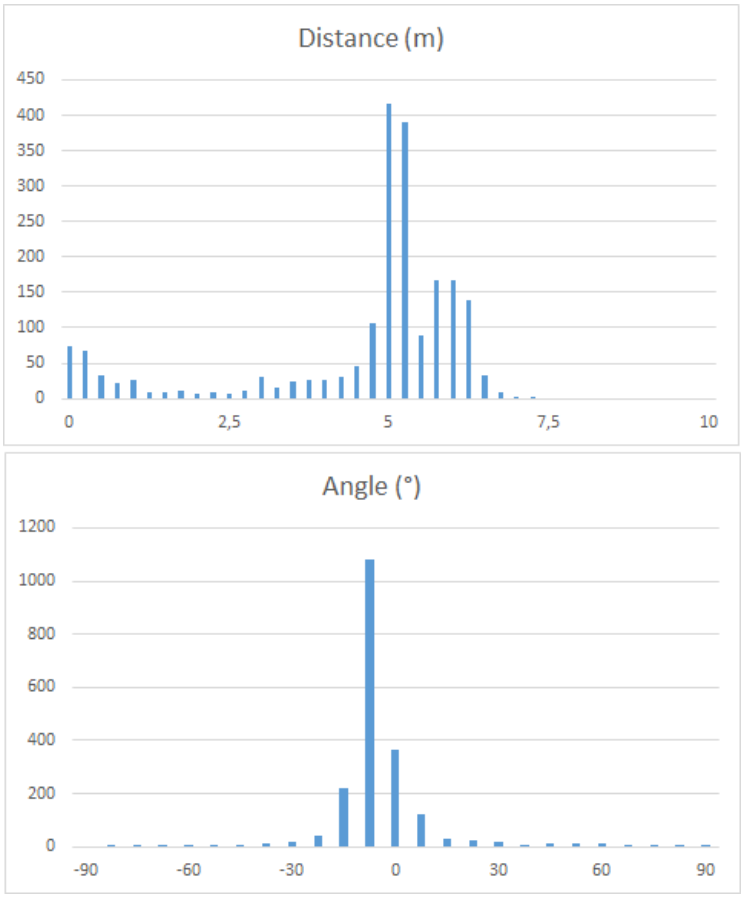

Figure 6.7: Frequency distribution for *Sprint* kicks

Additionally, for each of the sub-problems, 100 episodes were taken and a scatter plot was generated, as seen in figure 6.8 below. Individual figures for each of the sub-problems are available in Appendix B.

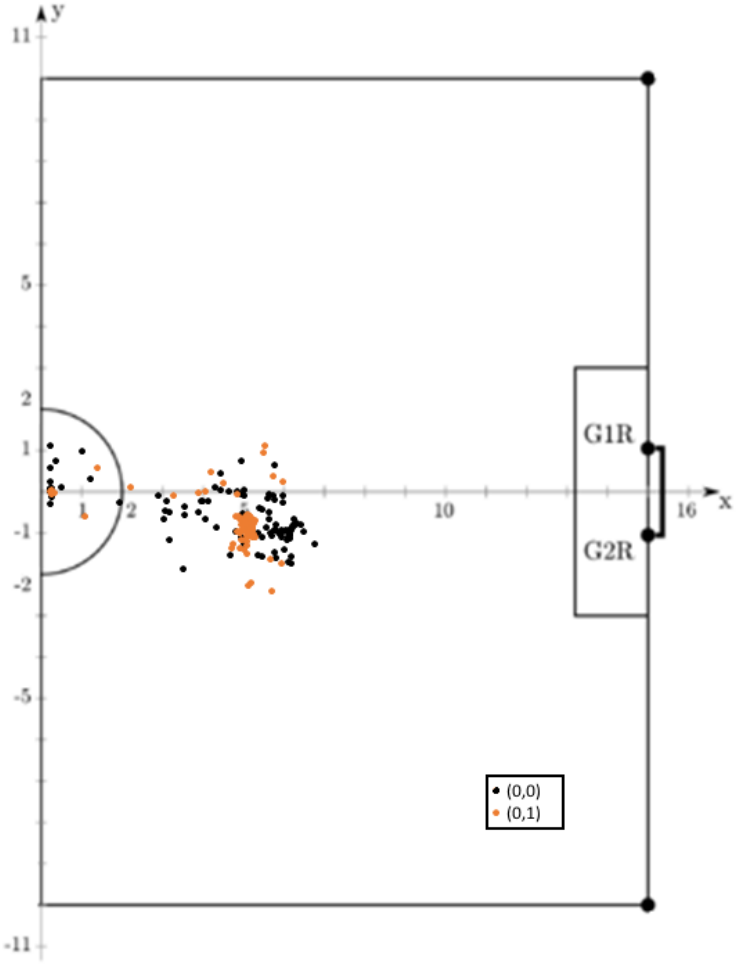

Figure 6.8: *Sprint* kick results on the field

6.10 Conclusion

In this chapter, an approach to the realization of a kick in motion was created. During this process, several modifications and adaptations were done to the kick described in chapter 5. Although the performance obtained was not good as for the static kick, the speed with which it can be done makes it an interesting kick for situations where there is a need for a fast game

The problem was decomposed into several sub-problems, which were optimized for all cases for the *Run* scenario. Regarding the *Sprint* scenario, not all of these were optimized, but enough were to prove that the concept can likewise be applied in this situation.

Chapter 7

Expanding on the Kick in Motion

Taking the lessons learned on the previous chapters it was possible to test some possibilities towards expanding the work developed in this dissertation in future paths. In this chapter, one of these, the targeted kick, was briefly studied.

7.1 Problem

While having a skill that allows for the agent to kick a ball forward is advantageous, as briefly mentioned, there is much more to it that is required to play soccer in a realistic manner. In fact, to reach the goals of the RoboCup Initiative, or more pragmatically, a better result in competitions for the FCPortugal3D team, a set of other related skills must be developed, such as a targeted kick. Such a kick can be used to pass a ball to another team member with confidence for example, and being able to do so in a moving scenario, would greatly improve the capabilities of the FCPortugal3D team.

With such a goal in mind, one of the cases for the *Run* behaviour was trained, in order to see whether that would be a feasible goal to achieve.

7.2 Neural Network

While the structure of the Neural Network was largely kept intact from the one described in chapter 6, with regard to the action space, number and size of layers and hyperparameters, it should be noted that two extra variables were added, for a total of 146 variables in the observation space, as per table 7.1 below:

Acquisition Method	Parameter	Data Size
Raw	Joint angle	22
	Joint velocity	22
	Joint acceleration	22
	Feet Force *	12
	Accelerometer *	3
	Gyroscope *	3
	Height *	1
	Rotation *	1
Generated	Action Space	20
	Counter	1
Calculated	Foot Distance *	6
	Average Distance *	2
	Target Distance	2
	Ball Speed	1
	* Differentiation	28
Total		146

* Values marked as such have differentiation performed on.

Table 7.1: Observation State for Targeted Kick

These two new variables are the distance from the ball to the intended target, in polar coordinates. Like the coordinate system for the feet described extensively in section 5.5, it should be pointed that a relative angle is calculated, adjusting for the frame of reference of the agent, and not simply using the absolute system of coordinates. This is vital for the agent to learn how to kick to any certain point regardless of its own rotation on the field.

7.3 Initial Conditions and Episode Layout

In this situation, the initial condition is based, and nearly identical to one of the ones previously described in chapter 6, using the *Run* skill. More specifically a *Run* kick of type (0,0).

The main difference thus, comes from the small change imposed by the need to create a set of target coordinates before the start of the game. These were set to random values, bounded to an angle of $\pm 45^{\circ}$ relative to the agent's starting orientation and to a magnitude between 3m and 7m. The resulting area where a target may be is thus visualized in the area shaded green in figure 7.1:

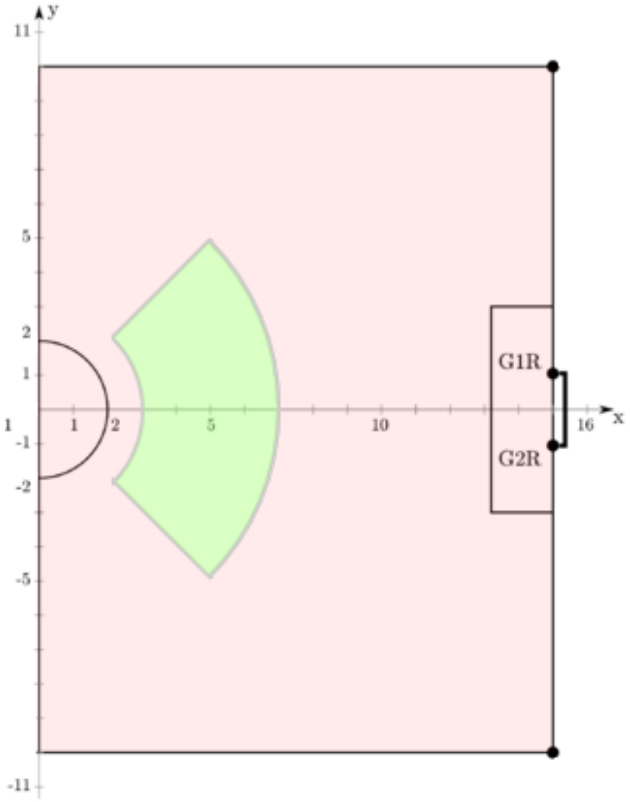

Figure 7.1: Target area bounds for kick

The layout of an episode as such is identical to that which was described in 6 for this skill, with the small aforementioned difference that a set of target coordinates for the ball are generated during the reset stage of the episode.

7.4 Reward Shaping

While the reward function for the dense component at every step was kept unchanged from previous chapters, the sparse reward at the end of each episode was modified for the new type of kick. In this way, R^t_{sparse}, may be defined as:

$$R^t_{sparse} = \left(1 - \sqrt{\frac{r^t_{target}}{r^0_{target}}}\right) \times 100 \tag{7.1}$$

Where r^t_{target} is the distance between the ball and the predetermined target at moment t. It naturally follows that r^0_{target} is the distance of the ball to the target at start. The square root term seeks to provide an extra incentive towards movement in the correct direction, by guaranteeing a steeper slope in the reward function as the ball nears the target position.

7.5 Results

The results for the training scenario described in the previous sections of this chapter were obtained after a session of over 20M steps over 25 hours, performed with 32 parallel threads on a server provided by the Faculty of Engineering of the University of Porto. The produced neural network was then evaluated for a sample of 1000 episodes, the results of which can be seen on table 7.2, where the average distance is the percentage of improvement relative to the original distance:

	Number	(%)	**Average Distance** (%)	**Range** (%)
Failed kicks	16	1.6	0	0
Bad kicks	46	4.6	-9.82	]-∞ 0[
Mediocre kicks	346	34.4	18.42	]0 33]
Good kicks	406	40.6	49.12	]33 66]
Great kicks	188	18.8	77.55	]66 100]
Best kick	Distance : 99.80%			
Average time	Duration : 0.34s			

Table 7.2: Targeted Kick results

The resulting kick can be seen in figure 7.2 below, with the yellow line representing the initial vector between the ball and the target, is very similar to a regular kick in motion, already described in chapter 6. Furthermore, an analysis of the data shows that while only a marginal number of kicks do not hit the ball (<2%), a small minority do move to a point farther away from the target than the original position, resulting in a negative reward. It should still be noted that about 94% of episodes result in a positive reward.

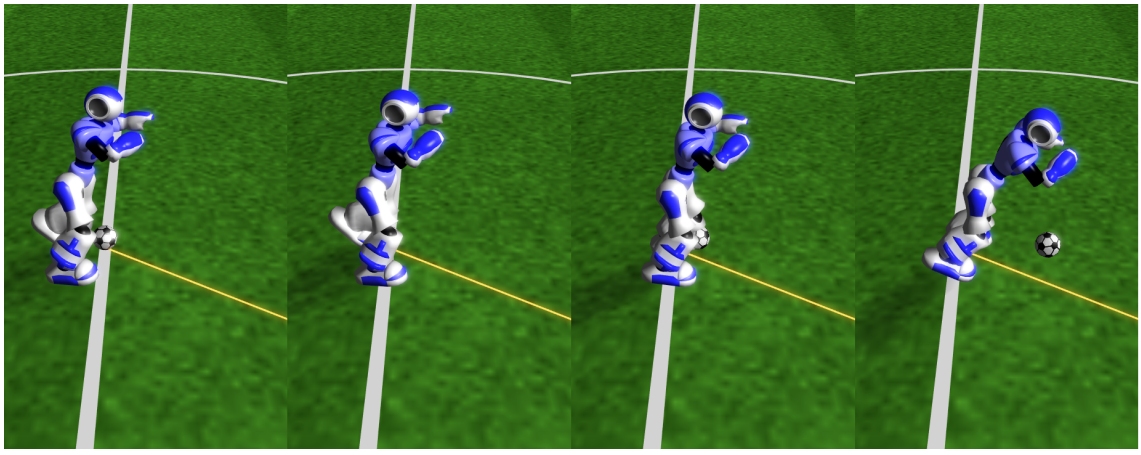

Figure 7.2: Several moments of a targeted kick

To understand if this was due to a targeted kick or a trend to push the ball forward, two series of 100 episodes were evaluated, for fixed target coordinates (4,4) and (4,-4), as seen in figure 7.3:

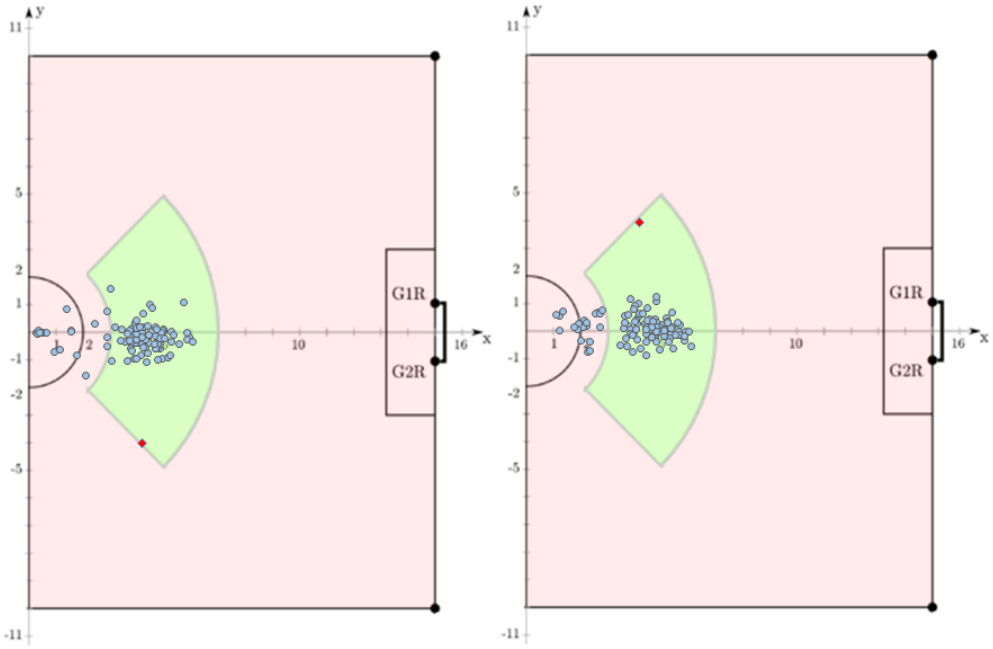

Figure 7.3: Final ball position for different targeted kicks

While the radial component of the ball seems to be in line with the desired values, the correlation between the y-axis coordinates of the balls and the targets appears to be much more tenuous. A frequency distribution, similar in nature to the ones described in previous chapters was generated for the two situations, as observable in figure 7.4 below.

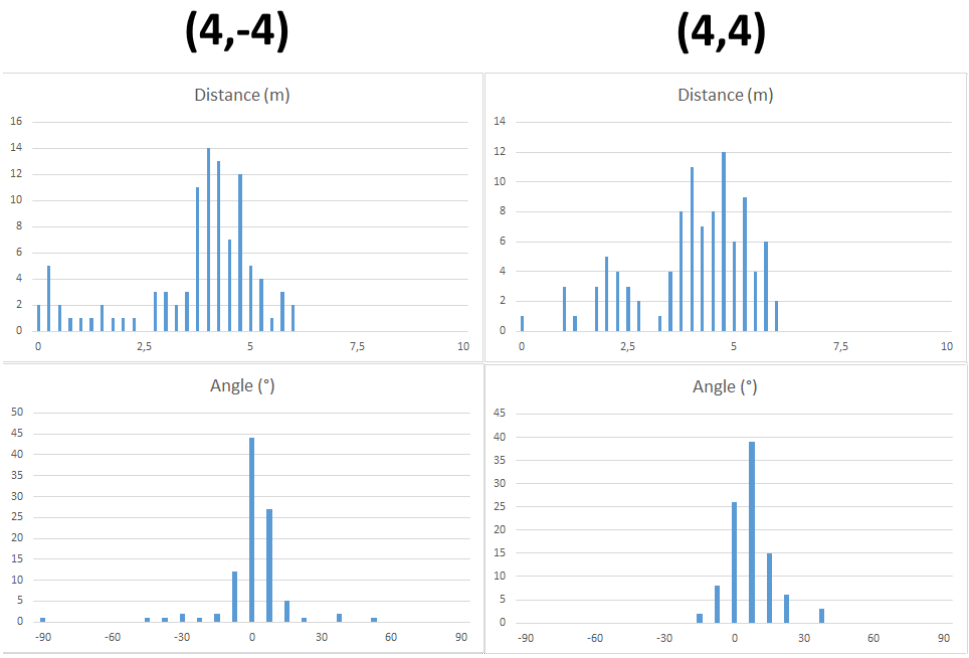

Figure 7.4: Frequency distribution of results for different targets

As it can be seen, while a distinction exists for the two situations, it is not as relevant as it could be hoped. A possible explanation to this, is that the neural network learned only to target the center of the desired area instead of the whole area, so as to maximize the reward in a stochastic manner.

A possible way to work around this situation could be to set the target position to only one of two positions on the sides of the area shaded green, so as to encourage better learning of the relationship between the angle and target position, further subdividing the problem in its nature.

7.6 Conclusion

This chapter approached some other kick in motion techniques that were briefly analysed in the realization of this dissertation, namely the precision kick in motion, presenting a limited amount of success. It nevertheless shows how the work developed in this dissertation may be expanded upon in the future.

Chapter 8

Future Work and Conclusion

8.1 Future Work

Some paths for future work seem quite clear at this point, with regards to improvements in the kick in motion. Firstly, as it was shown, for a more practical kick in motion to be developed and used in competition there is a clear need for a degree of further refinement in the sprint and run algorithms. The goal of this refinement being to ensure the regularity of the movements; for while their movement speed is remarkable, their tend to veer off course in unpredictable ways.

This results in extra noise in the observation space, creating a drop in the performance compared to the static kick. While the solutions described to overcome the problem have proved to have a degree of effectiveness, they require a much higher usage of computer resources to train several neural networks for one single behaviour. It should be pointed that work in this area is already occurring in the FCPortugal3D team as of now.

Secondly, it should be noted that the tools developed in this dissertation, such as the estimator for the foot distance, already open the doors to a variety of possible new behaviours, such as a more thorough study of the targeted kick, for both the static and the in motion scenario, serving as a way to create a better passing behaviour for example.

Another area where more work can be done is in changing the layout of the neural network, since great variations of performance were seen dependent on said factor.

More complex scenarios have also been posited, such as the development of dribbling behaviours, or the interception of a ball of an opponent getting ready to kick, but these require even further work to make them completely viable.

Finally, the static kick described in chapter 5 has proven to be remarkably fast, at just under 0.4s, while keeping a very decent performance. Thus, an extra possible avenue for work could be to further explore it for usage as a middle range kick, so as to create a more dynamic play style in a competition.

8.2 Conclusion

This document started by setting out an introduction and motivations for this work in chapter 1, in the context of the FCPortugal3D team and of the whole RoboCup competition.

It then described the state of the art in chapter 2, with a strong emphasis in Reinforcement Learning methods, as the applications in this area in the context of RoboCup are still very inchoate.

Chapter 3 served to provide an overview of the simulation environment used for the simulation, including the robot itself. On the other hand, chapter 4 provided a contextualization on the optimization side, with an overview of the tools and frameworks and algorithms upon which this dissertation rests.

In chapter 5, a proof of concept for a static kick was developed. This was very successful in creating an extremely fast kick while keeping a good kick distance.

Chapter 6, provided an analysis of the kick in motion situation, analysing the results obtained for two movement behaviours previously developed by the FCPortugal3D team. This resulted in some very attractive results for kicks with a deep focus on speed. Additionally, chapter 7 served as an extension of the concept of the kick in motion in an attempt of developing more elaborate kicks.

It stands clear by now that the application of new techniques based on Reinforcement Learning, and the methods developed in this dissertation hold a great potential for creating new skills for the FCPortugal3D team, especially when dealing with improvements in speed. This has the potential of making the play style of the team much more dynamic. It was found nevertheless, that future work is still needed to fully explore and unlock these possibilities, as previously described in this chapter.

Appendix A

Kick in Motion tables

A.1 Run Behaviour sub-problems

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	19	1.9	0	0
Bad kicks	86	8.6	0.70	]0 , 1.5[
Mediocre kicks	45	4.5	2.14	[1.5 , 3[
Good kicks	850	85.0	4.53	[3 , +∞[
Best kick	Distance : 6.36m			
Average time	Duration : 0.25s			

Table A.1: Type (0,0) kick results

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	29	2.9	0	0
Bad kicks	256	25.6	0.77	]0 , 1.5[
Mediocre kicks	200	20.0	2.23	[1.5 , 3[
Good kicks	515	51.5	5.73	[3 , +∞[
Best kick	Distance 8.89m			
Average time	Duration : 0.35s			

Table A.2: Type (0,1) kick results

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	11	1.1	0	0
Bad kicks	35	3.5	0.79	]0 , 1.5[
Mediocre kicks	50	5.0	2.47	[1.5 , 3[
Good kicks	904	90.4	6.15	[3 , +∞[
Best kick	Distance : 7.58m			
Average time	Duration : 0.40s			

Table A.3: Type (0,2) kick results

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	0	0	0	0
Bad kicks	45	4.5	0.70	]0 , 1.5[
Mediocre kicks	18	1.8	2.07	[1.5 , 3[
Good kicks	937	93.7	6.03	[3 , +∞[
Best kick	Distance : 7.27m			
Average time	Duration : 0.24s			

Table A.4: Type (1,0) kick results

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	12	1.2	0	0
Bad kicks	72	7.2	0.82	]0 , 1.5[
Mediocre kicks	21	2.1	2.16	[1.5 , 3[
Good kicks	895	89.5	6.53	[3 , +∞[
Best kick	Distance : 7.95m			
Average time	Duration : 0.34s			

Table A.5: Type (1,1) kick results

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	9	0.9	0	0
Bad kicks	42	4.2	0.73	]0 , 1.5[
Mediocre kicks	81	8.1	2.44	[1.5 , 3[
Good kicks	868	86.8	5.25	[3 , +∞[
Best kick	Distance : 7.09m			
Average time	Duration : 0.41s			

Table A.6: Type (1,2) kick results

A.2 Sprint Behaviour sub-problems

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	64	6.4	0	0
Bad kicks	84	8.4	0.71	]0 , 1.5[
Mediocre kicks	43	4.3	2.49	[1.5 , 3[
Good kicks	809	80.9	5.41	[3 , +∞[
Best kick	Distance : 6.94m			
Average time	Duration : 0.24s			

Table A.7: Type (0,0) kick results

	Number	(%)	**Average Distance** (m)	**Range** (m)
Failed kicks	48	4.8	0	0
Bad kicks	50	5.0	0.69	]0 , 1.5[
Mediocre kicks	19	1.9	2.36	[1.5 , 3[
Good kicks	883	88.3	5.15	[3 , +∞[
Best kick	Distance : 7.04m			
Average time	Duration : 0.29s			

Table A.8: Type (0,1) kick results

Appendix B

Kick in Motion figures

B.1 Run Behaviour sub-problems

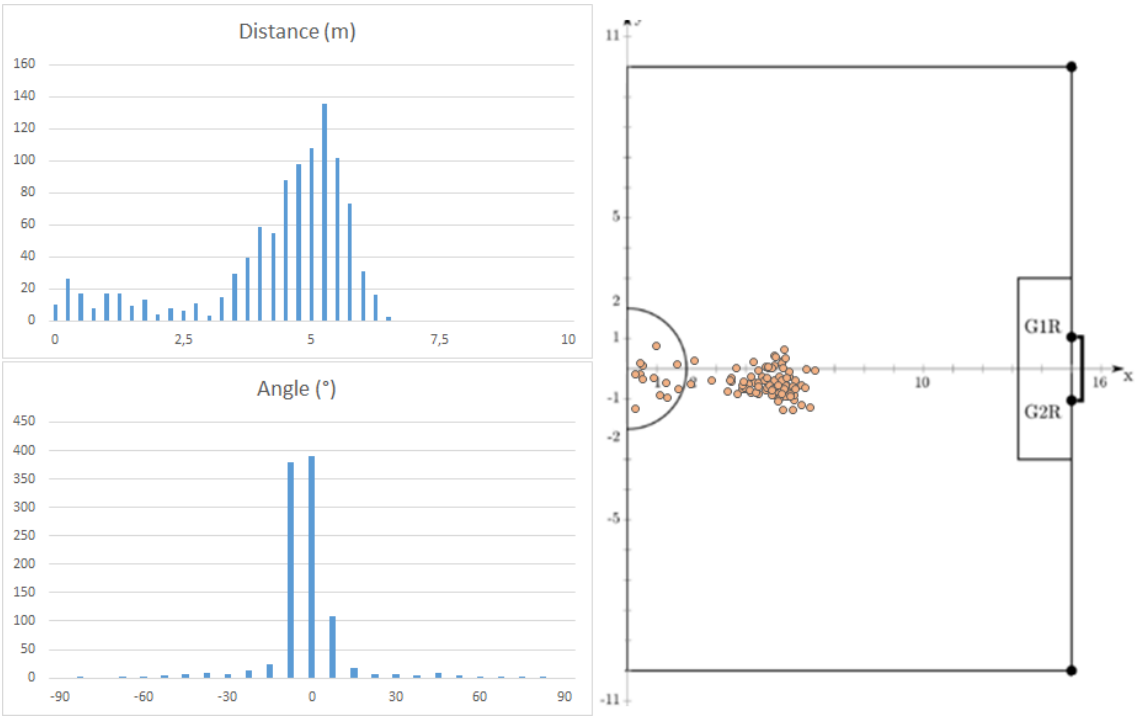

Figure B.1: Type (0,0) kick frequency distribution and scatter plot

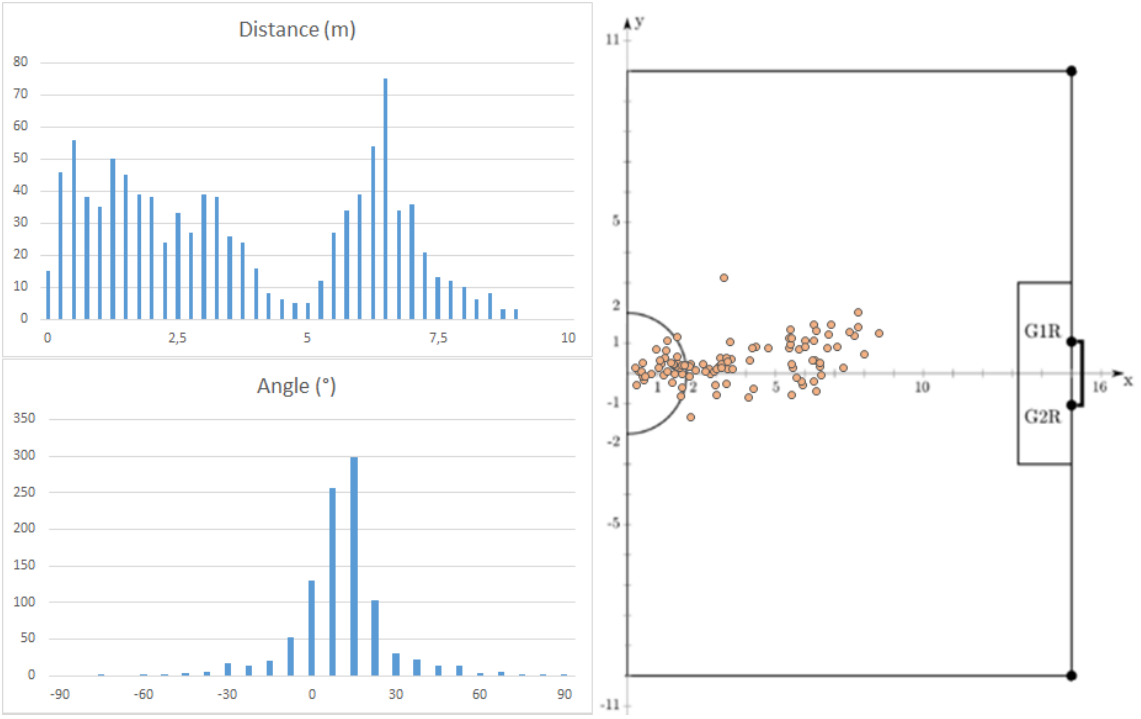

Figure B.2: Type (0,1) kick frequency distribution and scatter plot

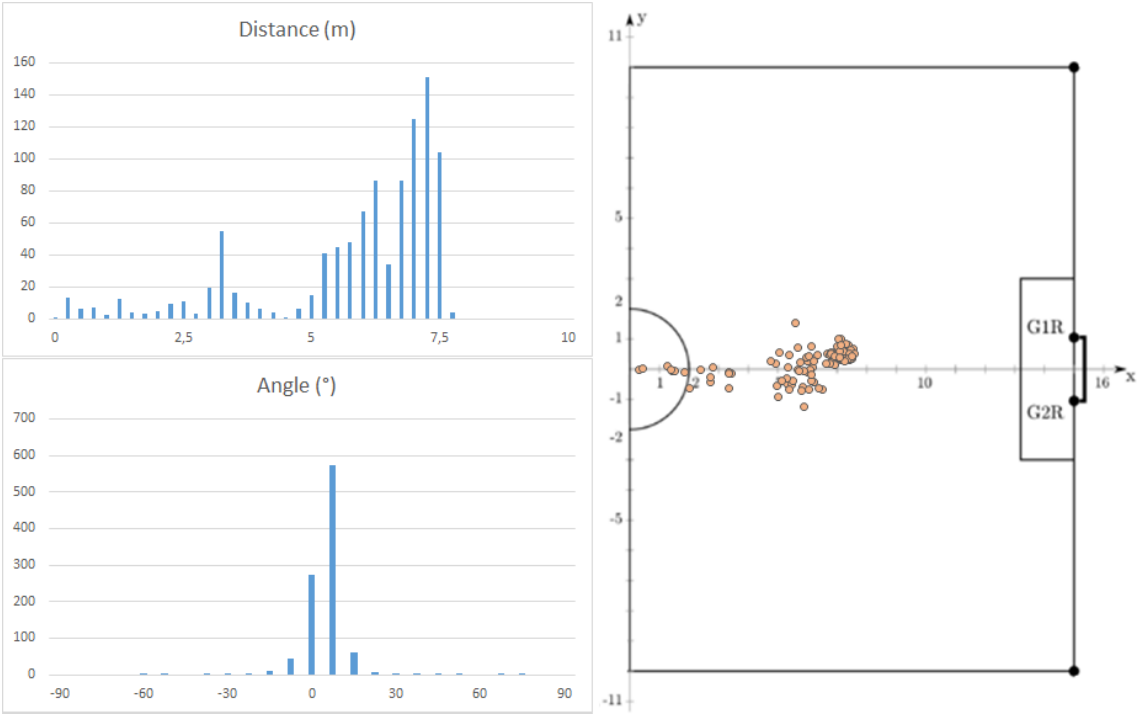

Figure B.3: Type (0,2) kick frequency distribution and scatter plot

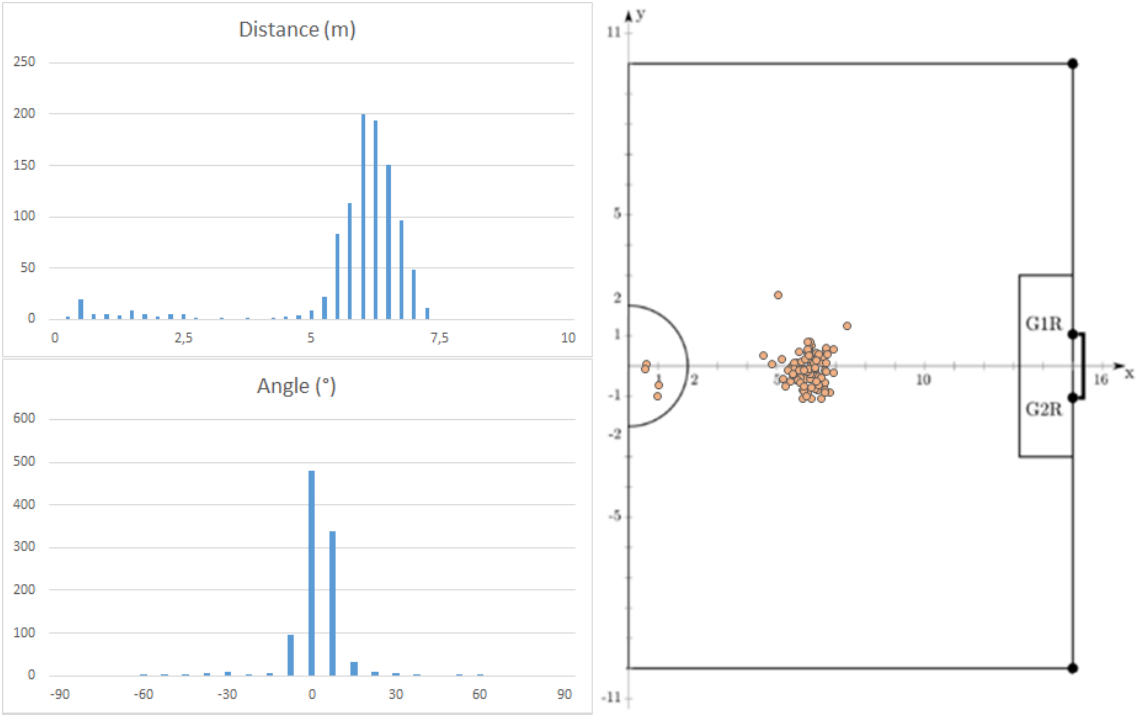

Figure B.4: Type (1,0) kick frequency distribution and scatter plot

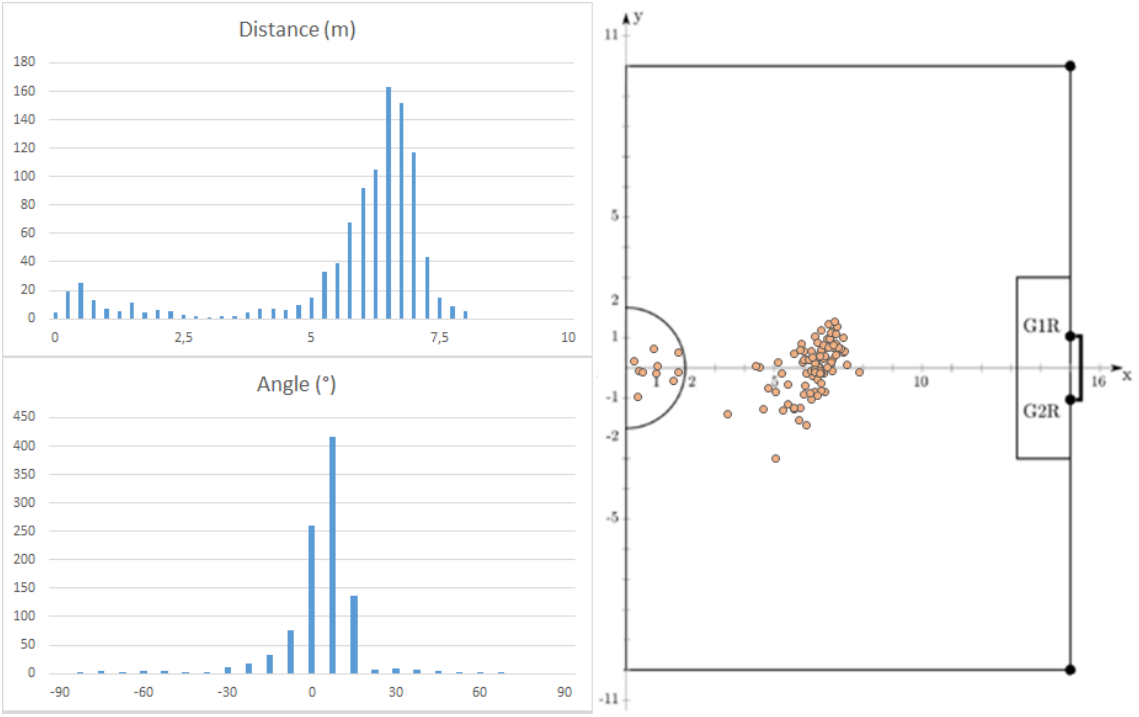

Figure B.5: Type (1,1) kick frequency distribution and scatter plot

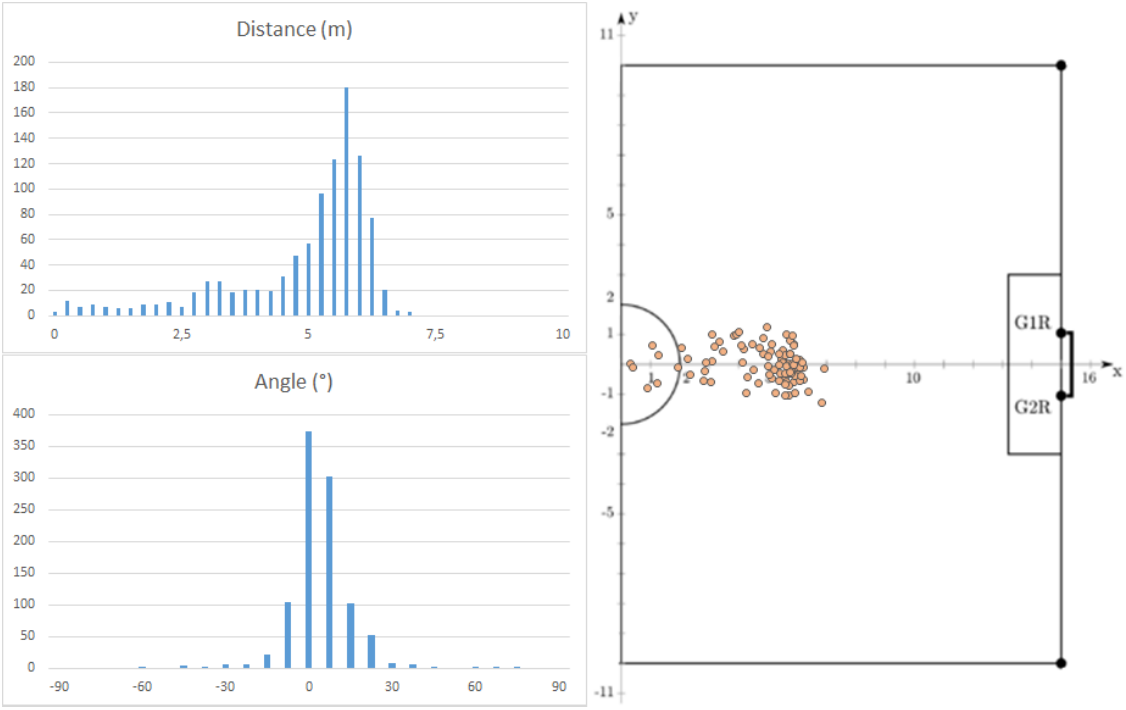

Figure B.6: Type (1,2) kick frequency distribution and scatter plot

B.2 Sprint Behaviour sub-problems

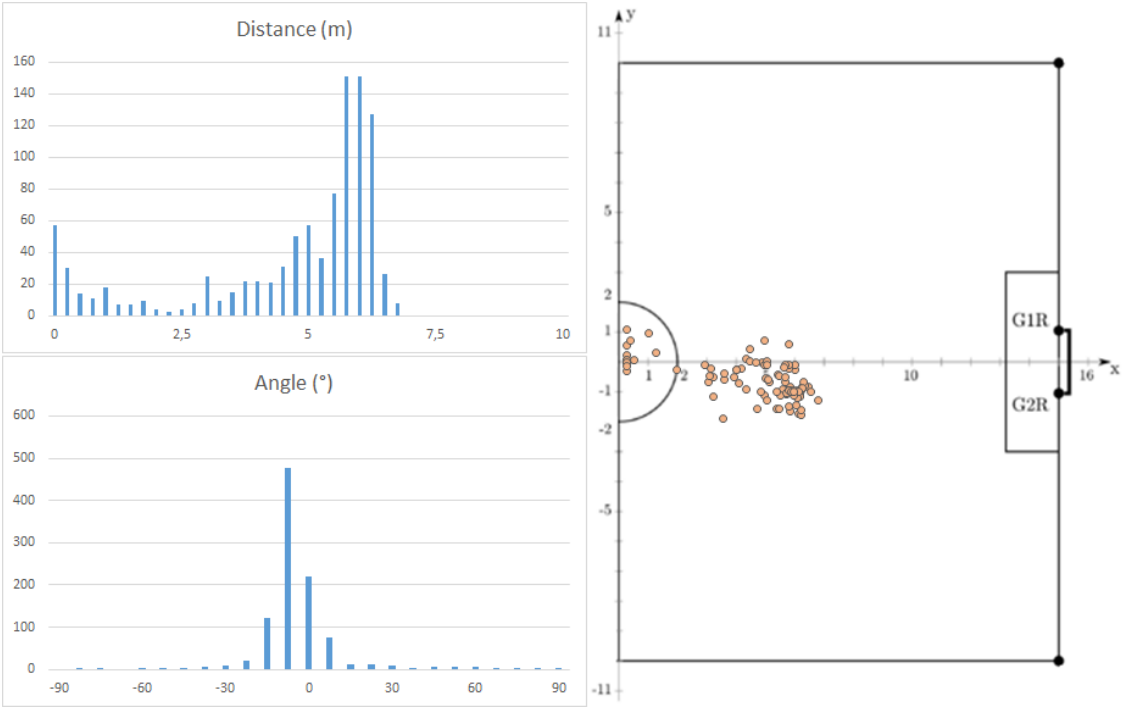

Figure B.7: Type (0,0) kick frequency distribution and scatter plot

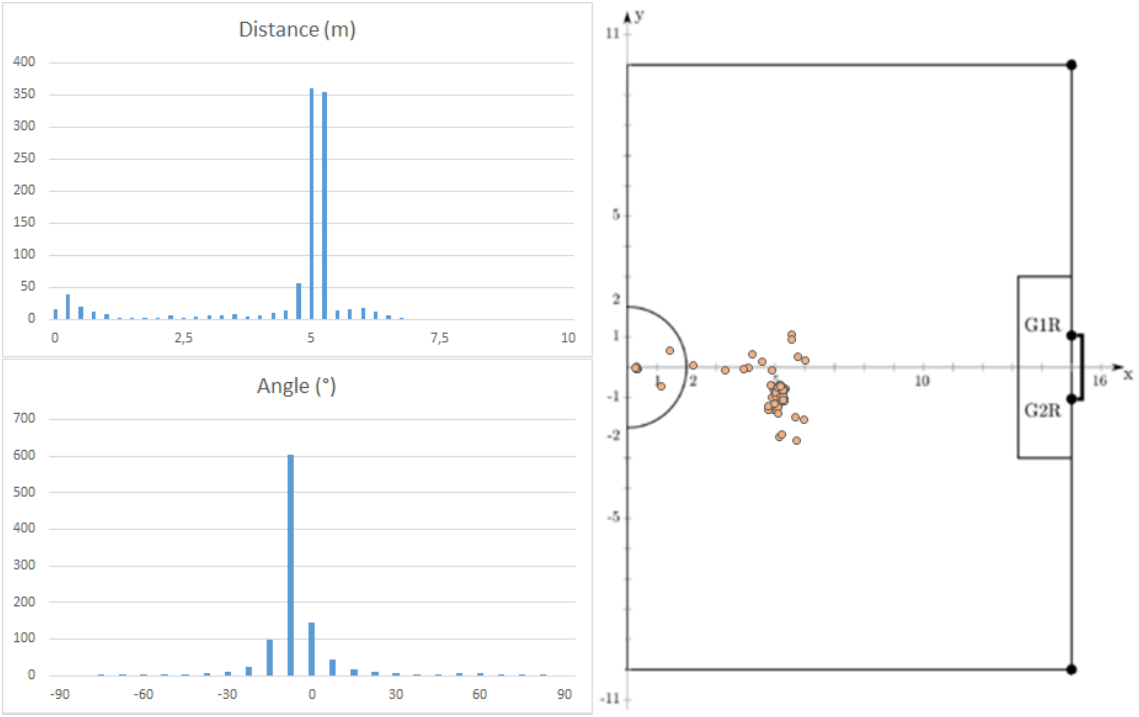

Figure B.8: Type (0,1) kick frequency distribution and scatter plot

References

[1] Jean Francis Gréhaigne, Daniel Bouthier, and Bernard David. Dynamic-system analysis of opponent relationships in collective actions in soccer. *Journal of Sports Sciences*, 15(2):137–149, 1997. Available at: www.doi.org/10.1080/026404197367416 (visited on 18/04/2019).

[2] Mike D Hughes and Roger M Bartlett. The use of performance indicators in performance analysis. *Journal of Sports Sciences*, 20(10):739–754, 2002. Available at: https://doi.org/10.1080/026404102320675602 (visited on 23/04/2019).

[3] M F Acar et al. Analysis of goals scored in the 2006 world cup. *Science and Football VI*, pages 235–242, 2009.

[4] RoboCup. RoboCup Objectives, (no date). Available at: https://www.robocup.org/objective/ (visited on 05/05/2019).

[5] Luís Paulo Reis et al. FC Portugal 2017 – 3D Simulation Team Description Paper. *RoboCup Symposium*, 2017. Available at: https://www.robocup2017.org/file/symposium/soccer_sim_3D/FCPortugal3D_TDP_2017.pdf (visited on 08/05/2019).

[6] Hiroaki Kitano et al. RoboCup: A challenge problem for AI and robotics. *AI Magazine*, 18(1):73–85, 1998. Available at: www.doi.org/10.1609/aimag.v18i1.1276. (visited on 08/05/2020).

[7] Alexander Ferrein and Gerald Steinbauer. 20 Years of RoboCup. *KI - Künstliche Intelligenz*, 30(3):225–232, 2016. Available at: https://doi.org/10.1007/s13218-016-0449-5 (visited on 06/05/2019).

[8] Abbas Abdolmaleki, Nuno Lau, Luis Paulo Reis, Jan Peters, and Gerhard Neumann. Contextual Policy Search for Linear and Nonlinear Generalization of a Humanoid Walking Controller. *Journal of Intelligent & Robotic Systems*, 83(3):393–408, 2016. Available at: https://doi.org/10.1007/s10846-016-0347-y (visited on 3/05/2019).

[9] Nima Shafii, Abbas Abdolmaleki, Nuno Lau, and Luis Paulo Reis. Development of an Omnidirectional Walk Engine for Soccer Humanoid Robots. *International Journal of Advanced Robotic Systems*, 12(12):1–14, 2015. Available at: https://doi.org/10.5772/61314 (visited on 29/04/2019).

[10] Nima Shafii, Nuno Lau, and Luis Paulo Reis. Learning to Walk Fast: Optimized Hip Height Movement for Simulated and Real Humanoid Robots. *Journal of Intelligent & Robotic Systems*, 80(3):555–571, 2015.

[11] Abbas Abdolmaleki, David Simões, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. Learning a humanoid kick with controlled distance. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9776 LNAI(July):45–57, 2017. Available at: https://doi.org/10.1007/978-3-319-68792-6_4 (visited on 30/04/2019).

[12] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. Learning to Run Faster in a Humanoid Robot Soccer Environment Through Reinforcement Learning. In Stephan Chalup, Tim Niemueller, Jackrit Suthakorn, and Mary-Anne Williams, editors, *RoboCup 2019: Robot World Cup XXIII*, pages 3–15, Cham, 2019. Springer International Publishing.

[13] Maziar Palhang, Nuno Lau, and David Simões. RoboCup Soccer Simulation 3D League - Rules for the 2019 Competition in Sydney, Australia, 2019. Available at: https://ssim.robocup.org/wp-content/uploads/2019/06/Rules_RoboCupSim3D2019.pdf (visited on 18/05/19).

[14] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM JOURNAL OF RESEARCH AND DEVELOPMENT*, pages 71–105, 1959.

[15] Tom M. Mitchel. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997. ISBN 978-0-07-042807-2 , Avaliable at: http://www-stat.stanford.edu/{~}tibs/book/preface.ps (visited on 16/05/2019).

[16] J Boyan, D Freitag, and T Joachims. A Machine Learning Architecture for Optimizing Web Search Engines. *AAAI Technical Report WS-96-06*, pages 1–8, 1996. Avaliable at: http://www.aaai.org/Papers/Workshops/1996/WS-96-06/WS96-06-001.pdf (visited on 8/05/2019).

[17] G Parvathi, N Sakthidevi, and Alphonsa Jose Veena. A Machine Learning Approach for Filtering Unwanted Comments Posted in Online Social Networks. *International Journal of Advanced Research in Computer Science & Technology (IJARCST)*, 2(2):49–53, 2014.

[18] Robert M Bell, Yehuda Koren, and Chris Volinsky. The BellKor 2008 solution to the Netflix Prize. *ATT Labs–Research Technical report November*, pages 1–10, 2008. Avaliable at https://www.netflixprize.com/assets/ProgressPrize2008_BellKor.pdf (visited on 06/06/2019).

[19] Reimund Neugebauer, Sophie Hippmann, Miriam Leis, and Martin Landherr. Industrie 4.0 - From the Perspective of Applied Research. *Procedia CIRP*, 57:2–7, 2016.

[20] K Chitra and B Subashini. Data Mining Techniques and its Applications in Banking Sector. *International Journal of Emerging Technology and . . .* , 3(8):219–226, 2013.

[21] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89–109, 2001.

[22] Jonathan H. Connell and Sridhar Mahadevan. *Robot Learning*. The Springer International Series in Engineering and Computer Science 233. Springer US, 1 edition, 1993.

[23] L. Deng and D. Yu. *Deep Learning: Methods and Applications*. Now Foundations and Trends, 2014.

[24] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 9780262018258.

[25] Sören Hohmann. *Optimization of Dynamic Systems Lecture Notes, Chapter 2 : Unconstrained Parameter Optimization*. Institute of Control Systems, Karlsruher Institut für Technologie, 2018.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[27] Vincent François-lavet et al. An Introduction to Deep Reinforcement Learning. *Foundations and trends in machine learning*, 11(3-4), 2018.

[28] J R Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.

[29] Dimitri P. Bertsekas. *Approximate dynamic programming*, volume I. Springer International Publishing AG., 1995.

[30] Tamis Achilles van der Laan. *Concolidated Deep Actor Critic Networks*. PhD thesis, TU Delft, 2015.

[31] J. Zico Kolter. Introduction to Reinforcement Learning - Lecture Notes. *Carnegie Mellon University*, 2018. Avaliable at: http://dl.acm.org/citation.cfm?id=551283 (visited on 28/05/2020).

[32] Peter Dayan and Yael Niv. Reinforcement learning: The good, the bad and the ugly. *Current Opinion in Neurobiology*, 18(2):185 – 196, 2008. Cognitive neuroscience.

[33] Jan Gläscher, Nathaniel Daw, Peter Dayan, and John P O'Doherty. States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–95, 2010.

[34] Lukasz Kaiser et al. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019.

[35] Erwan Renaudo, Benoît Girard, Raja Chatila, and Mehdi Khamassi. Respective advantages and disadvantages of model-based and model-free reinforcement learning in a robotics neuro-inspired cognitive architecture. *Procedia Computer Science*, 71:178 – 184, 2015. 6th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2015, 6-8 November Lyon, France.

[36] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. *Ieee Signal Processing Magazine, Special Issue On Deep Learning For Image Understanding (Arxiv Extended Version)*, pages 1–16, 2017.

[37] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems: Theory and Applications, 13, 41-77, 2003*, 2003.

[38] Richard S Sutton and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 1(112):181–211, 1999.

[39] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems (NIPS)*, pages 1043–1049, 1998.

[40] Timothy P. Lillicrap et al. Continuous control with deep reinforcement learning. *ICLR 2016 conference*, 2016.

[41] Naveen Venkat. *The Curse of Dimensionality: Inside Out*. PhD thesis, Dept. of CSIS, BITS Pilani, 2018.

[42] Lodewijk Kallenberg. *Markov decision processes Lecture Notes*. PhD thesis, University of Leiden, 2016.

[43] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[44] Rich Caruana. Multitask learning: A knowledge-based source of inductive bias. In *ICML*, 1993.

[45] Mingsheng Long and Jianmin Wang. Learning multiple tasks with deep relationship networks. *CoRR*, abs/1506.02117, 2015.

[46] Yongxi Lu et al. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *CoRR*, abs/1611.05377, 2016.

[47] Jørgen Bang-Jensen, Gregory Gutin, and Anders Yeo. When the greedy algorithm fails. *Discrete Optimization*, 1(2):121–127, 2004.

[48] G A. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[49] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–33, 2015.

[50] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[51] OpenAI Spinning Up. Twin Delayed DDPG, 2018. Avaliable at: https://spinningup.openai.com/en/latest/algorithms/td3.html{#}id1 (visited on 20/06/2019).

[52] Tuomas Haarnoja et al. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[53] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation Richard. *NIPS'99 Proceedings of the 12th International Conference on Neural Information Processing Systems*, 2000.

[54] John Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, University of California, Berkeley, 2016.

[55] Volodymyr Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, abs/1602.0, 2016.

[56] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. *CoRR*, pages 1–13, 2016.

[57] Sergios Karagiannakos. The idea behind Actor-Critics and how A2C and A3C improve them, 2018. Avaliable at: https://sergioskar.github.io/Actor{_}critics/ (visited on 13/06/19).

[58] OpenAI. OpenAI Baselines: ACKTR & A2C, 2017. Avaliable at: https://openai.com/blog/baselines-acktr-a2c/(visited on 15/06/19).

[59] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[61] OpenAI. Proximal Policy Optimization, 2017. Avaliable at: https://openai.com/blog/openai-baselines-ppo/ (visited on 13/01/2020).

[62] Cameron B Browne et al. A Survey of Monte Carlo Tree Search Methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES,*, 4(1):1–43, 2012.

[63] David Silver et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 10 2017.

[64] Theophane Weber et al. Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.

[65] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[66] Nuno Lau and Luis Paulo Reis. Fc portugal - high-level coordination methodologies in soccer robotics. In Pedro Lima, editor, *Robotic Soccer*, chapter 9. IntechOpen, Rijeka, 2007.

[67] Edgar Domingues et al. Humanoid behaviors: From simulation to a real robot. In *Progress in Artificial Intelligence*, pages 352–364, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[68] Rui Ferreira et al. Development of an omnidirectional kick for a nao humanoid robot. In *Advances in Artificial Intelligence – IBERAMIA 2012*, pages 571–580, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[69] Nima Shafii, Abbas Abdolmaleki, Rui Ferreira, Nuno Lau, and Luís Reis. Omnidirectional walking and active balance for soccer humanoid robot. In *Portuguese Conference on Artificial Intelligence*, volume 8154, pages 283–294, 09 2013.

[70] Abbas Abdolmaleki et al. Learning a humanoid kick with controlled distance. In *RoboCup 2016: Robot World Cup XX*, pages 45–57, Cham, 2017. Springer International Publishing.

[71] Nima Shafii, Nuno Lau, and Luís Reis. Learning to walk fast: Optimized hip height movement for simulated and real humanoid robots. *Journal of Intelligent & Robotic Systems*, 80:1–17, 02 2015.

[72] Abbas Abdolmaleki et al. Model-based relative entropy stochastic search. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3537–3545. Curran Associates, Inc., 2015.

[73] Abbas Abdolmaleki, Bob Price, Nuno Lau, Luís Reis, and Gerhard Neumann. Deriving and improving cma-es with information geometric trust regions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 07 2017.

[74] Arne Böckmann and Tim Laue. Kick motions for the NAO robot using dynamic movement primitives. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9776 LNAI(July):33–44, 2017.

[75] Judith Müller, Tim Laue, and Thomas Röfer. Kicking a ball - Modeling complex dynamic motions for humanoid robots. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6556 LNAI:109–120, 2011.

[76] Stefan Glaser. SimSpark Wiki - About SimSpark, 2017. Avaliable at: https://gitlab.com/robocup-sim/SimSpark/-/wikis/About-SimSpark (visited on 04/01/2020).

[77] Yuan Xu and Hedayat Vatankhah. SimSpark: An open source robot simulator developed by the RoboCup Community. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8371 LNAI:632–639, 2014.

[78] Stefan Glaser. SimSpark Wiki - Soccer Simulation, 2017. Avaliable at: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Soccer-Simulation (visited on 01/04/2020).

[79] Stefan Glaser. SimSpark Wiki - Abstract Physical Layer, 2017. Avaliable at: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Abstract-Physical-Layer (visited on 04/01/2020).

[80] RoboCup. RoboCup 3D Soccer Simulation Rules, 2019. Avaliable at: https://ssim.robocup.org/3d-simulation/3d-rules/ (visited on 05/01/2020).

[81] Stefan Glaser. SimSpark Wiki - Agent Proxy. Avaliable at: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Agent-Proxy (visited on 05/01/2020).

[82] Nuno Lau, Marco Simões, David Simões, and Maziar Palhang. RoboCup Soccer Simulation 3D League - Rules for the 2019 Competition in Sydney, Australia (July 2nd -July8th). Avaliable at: https://ssim.robocup.org/wp-content/uploads/2019/06/Rules{_}RoboCupSim3D2019.pdf (visited on 05/01/2020).

[83] Tiago Silva. Humanoid Low-Level Skills using Machine Learning for RoboCup. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2019.

[84] Justin Stoecker and Ubbo Visser. RoboViz: Programmable Visualization for Simulated Soccer. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 282–293, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[85] MagmaOffenburg. GitHub - Monitor and visualization tool for the RoboCup 3D Soccer Simulation League, 2019. Available at: https://github.com/magmaOffenburg/RoboViz (visited on 06/01/05).

[86] RoboCup. RoboCup Soccer Simulation League - 3D Tools and Support. Available at: https://ssim.robocup.org/3d-simulation/3d-tools/ (visited on 05/01/2020).

[87] SoftBank Robotics. NAO H21 - Aldebaran documentation. Available at: http://doc.aldebaran.com/2-1/family/nao{_}h21/index{_}h21.html{#}nao-h21 (visited on 12/01/2020).

[88] SoftBank Robotics. Effector & Chain definitions - Aldebaran Documentation. Available at: http://doc.aldebaran.com/2-1/family/robots/bodyparts.html{#}effector-chain (visited on 13/01/2020).

[89] Stefan Glaser. Sim Spark Wiki - Models, 2017. Available at: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Models (visited on 13/01/2020).

[90] Wondergy. nao soccer | Wondergy. Available at: http://wondergy.com/nao-soccer/ (visited on 16/01/2020).

[91] SoftBank Robotics. H21 Joints - Aldebaran Documentation. Available at: http://doc.aldebaran.com/2-1/family/nao{_}h21/joints{_}h21.html (visited on 14/01/2020).

[92] Alejandro Said, Ernesto Rodriguez Leal, Rogelio Soto, J. Gordillo, and Leonardo Garrido. Decoupled closed-form solution for humanoid lower limb kinematics. *Mathematical Problems in Engineering*, 2015, 03 2015.

[93] Martin Abadi et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[94] Stable Baselines. PPO2 - Stable Baselines Documentation. Available at: https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html (visited on 15/01/2020).

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

[96] OpenAI. OpenAI Gym Documentation. Available at: https://gym.openai.com/docs/ (visited on 17/01/2020).