

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FCPortugal - Machine Learning for Creating Robotic Soccer Setplays

Bruno Miguel da Silva Barbosa de Sousa



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Paulo Reis

October 31, 2021

FCPortugal - Machine Learning for Creating Robotic Soccer Setplays

Bruno Miguel da Silva Barbosa de Sousa

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Henrique Daniel de Avelar Lopes Cardoso

External Examiner: Prof. José Manuel Veiga Ribeiro Cascalho

Supervisor: Prof. Luís Paulo Gonçalves dos Reis

October 31, 2021

Abstract

RoboCup, a robotic soccer competition, was introduced as a challenge for Artificial Intelligence and Intelligent Robotics to foster research in multiple associated fields such as multi-agent collaboration. Since then, the competition has grown to now feature multiple leagues. One of them is the 3D Simulated League, where two teams of eleven simulated humanoid robots compete in a game of robotic soccer.

FCPortugal is a joint project from Faculdade de Engenharia da Universidade do Porto and Universidade de Aveiro that participates in the 3D league. In the past, during a phase where the team participated in the 2D league, the team's research focused mainly on high-level strategies for players' coordination. However, the switch to humanoid 3D robot models forced the team to shift focus to the implementation of low-level skills, such as walking, running and kicking the ball. Now that those skills have been achieved, the study of high-level strategies has been restarted. One of such strategies is using setplays, studied plans to improve chances of the team scoring goals on setpieces, moments of the game where the game's flow is stopped.

Recent developments in machine learning, namely in Deep Reinforcement Learning and Multi-Agent Deep Reinforcement Learning, provide new perspectives into how agents may independently learn to perform tasks and collaborate to achieve a common goal. These new algorithms may now be used for the robots' learning and optimization of robotic soccer setplays.

In this dissertation, we propose a framework for the creation and optimization of setplays for the FCPortugal3D team. This framework includes the definition of a new setplay optimization language, in which setplays may be defined, and a parser to translate those definitions into environments where state of the art Reinforcement Learning algorithms may be used to optimize the setplays.

In order to test the effectiveness of the framework, a simple setplay was defined, trained and tested. While the results of the tests were underwhelming, not allowing to prove that the framework can be used by itself to create and optimize setplays, the framework may still be used as a starting point for the definition of setplays that researchers can later edit to improve results.

Resumo

RoboCup, uma competição de futebol robótico, foi introduzida como um novo desafio para Inteligência Artificial e Robôs Inteligentes com o intuito de incentivar pesquisa em várias matérias associadas, como colaboração multi-agente. Desde então, a competição cresceu para conter várias ligas. Uma delas é a Liga Simulada 3D, onde duas equipas de onze robôs humanoides simulados competem em jogos de futebol robótico.

FCPortugal é um projeto conjunto da Faculdade de Engenharia da Universidade do Porto e da Universidade de Aveiro que participa na liga 3D. No passado, durante uma fase em que a equipa participava na liga 2D, o principal foco de estudo da equipa era em estratégias de alto nível para a coordenação de jogadores. Contudo, a mudança para robôs humanoides 3D forçou a equipa a mudar o foco para a implementação de habilidade baixo nível, como por exemplo andar, correr e rematar. Agora que essas habilidades foram conseguidas, o estudo de estratégias alto nível foi recommçado. Uma dessas estratégias é o uso de jogadas estudadas, planos estudados para aumentar as hipóteses da equipa marcar golos de bola parada, momentos em que o fluxo do jogo está parado.

Avanços recentes em Aprendizagem Computacional, nomeadamente em Aprendizagem por Reforço Profundo e Aprendizagem Multi-Agente por Reforço Profundo, fornecem novas perspectivas em como agentes podem aprender a executar tarefas independentemente e como podem colaborar para alcançar objetivos comuns. Estes novos algoritmos poderão agora ser usados para os robôs aprenderem e otimizarem jogadas estudadas de futebol robótico.

Nesta dissertação, propomos uma framework para a criação e optimização de jogas estudadas para a equipa FCPortugal3D. Esta framework inclui a definição de uma nova linguagem de optimização de jogadas estudadas, na qual as jogadas estudadas podem ser definidas, e um parser para traduzir essas definições em ambientes onde algoritmos de aprendizagem por reforço de última geração podem ser utilizados para optimizar as jogadas estudadas.

Para testar a eficácia da framework, foi definida, treinada e testada um jogada estudada simples. Embora os resultados dos testes tenham sido pouco impressionantes, não permitindo provar que a framework possa ser utilizada por si só para criar e optimizar jogadas estudadas, a framework pode ainda assim ser utilizada como ponto de partida para a definição de jogadas estudadas que investigadores podem posteriormente editar para melhorar os resultados.

Acknowledgements

I would first like to thank my supervisor, Professor Luís Paulo Reis, for the help and support given during the whole process of doing this dissertation, as well as thank everyone involved with the FCPortugal3D team, without which this work would not have been possible.

I also want to thank my friends for all the moments spent, either having fun or working hard, not only during this last period but over the last five years. You were one of the reasons that motivated me to get up every day and go to FEUP and were essential in keeping my spirits up during the pandemic over the last one and half years.

Finally and my biggest thank you goes to my family, which allowed me to have this amazing opportunity to continue my studies and go to university. A special thank you for the emotional support given in these last months, where you made me believe that I could complete this dissertation even when I wanted to give up. I can't put into words how much you all helped me, especially you Leonor (even if you don't yet realise it), and a thank you is not enough, but I hope you know how grateful I am for all of it.

To you all - Thank You.

Bruno

*“Playing football is very simple,
but playing simple football
is the hardest thing there is.”*

Johan Cruyff

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Objectives	2
1.4	Document Structure	2
2	Robotic Soccer	5
2.1	RoboCup	5
2.2	Soccer Simulation 3D League	6
2.2.1	Simulation	7
2.2.2	Player Model	8
2.3	FCPortugal	8
3	State of the Art on Machine Learning for Robotic Soccer	13
3.1	Machine Learning	13
3.2	Robot Learning	14
3.3	Deep Learning	15
3.4	Deep Reinforcement Learning	15
3.4.1	Model-Free RL	17
3.4.2	Model-Based RL	21
3.4.3	OpenAI Gym	22
3.5	Multi-Agent Deep Reinforcement Learning	24
3.6	Evolutionary Algorithms	25
4	Setplay Frameworks	27
4.1	Setplay	27
4.2	FCPortugal Setplay Framework	29
5	New Setplay Optimization Framework	33
5.1	Setplay language definition	33
5.2	Setplay language parser	38
6	Evaluation	43
6.1	Defined Setplay	43
6.1.1	Observation & Action Spaces	45
6.1.2	Global Rewards, Finish & Abort Conditions	45
6.1.3	Steps	45
6.2	Training	46
6.3	Results	47

6.4	Conclusion	49
7	Conclusion	53
7.1	Main Contributions	53
7.2	Future Work	54
	References	55

List of Figures

2.1	Example of the initial 3D League simulation [37]	7
2.2	3D Simulation League field with two example teams.[57]	8
2.3	Modeled NAO robot.[56]	9
2.4	NAO Model's joints, effectors and actuators.[56]	10
2.5	FCPortugal Team Strategy composed of Tactics, Formations and Player Types.[45]	11
3.1	Artificial neural network architecture[7]	15
3.2	Agent-Environment Interaction in Reinforcement Learning[59].	16
3.3	Deep Reinforcement Learning system[34].	17
3.4	Taxonomy of algorithms in modern RL[41].	18
3.5	A3C and A3C architectures[64].	19
3.6	Difference between Q Learning and Deep Q Learning[55].	20
3.7	Results from games between AlphaZero and top classical engines in the games of chess, shogi and Go[53].	23
3.8	Communication between the agent, environment and the optimiser (RL algorithm).[51]	24
3.9	Evolutionary Algorithms execution flow[62].	26
4.1	Setplay domain model[36].	29
4.2	Implemented actions[36].	30
4.3	Creation of a new setplay[36].	30
4.4	Defining a setplay in SPlanner[36].	31
4.5	Visual Representation of player actions[36]. a) direct pass, b) forward pass, c) dribble, d) hold ball, e) shoot, f) hold position, g) move to position, h) run to offside line.	31
4.6	Example of complete corner setplay[36].	32
5.1	Setplay Optimization domain model.	34
6.1	Setplay: Initial Player Positions.	46
6.2	Setplay: Stage 1.	47
6.3	Setplay: Stage 2.	48
6.4	Setplay training episode rewards.	49
6.5	Robots falling at the start of the setplay.	50
6.6	Setplay being tested against Wrightocean3D team.	51

List of Tables

6.1 Trained and FCPortugal Setplay Results 51

Abbreviations

2D	Two Dimensional
3D	Three Dimensional
A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
ADVCOM	Advanced Communication Mechanism
AI	Artificial intelligence
ANN	Artificial Neural Networks
C51	Categorical 51-Atom Deep Q-Networks
COMA	Counterfactual Multi-Agent Policy Gradients
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DPRE	Dynamic Positioning and Role Exchange
DQN	Deep Q-Networks
DRL	Deep Reinforcement Learning
EA	Evolutionary Algorithms
HER	Hindsight Experience Replay
I2A	Imagination-Augmented Agents
MADRL	Multi-Agent Deep Reinforcement Learning
MBMF	Model-Based Reinforcement Learning with Model-Free Fine-Tuning
MBVE	Model-Based Value Expansion
ML	Machine Learning
PPO	Proximal Policy Optimization
QR-DQN	Quantile Regression Deep Q-Networks
RL	Reinforcement Learning
RoboCup	Robot World Cup Initiative
SAC	Soft Actor-Critic
SBSP	Situation Based Strategic Positioning
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization

Chapter 1

Introduction

This chapter presents the context, motivation and objectives of this dissertation, along with the structure of the remaining document. Section 1.1 describes the context of this work, Section 1.2 explains the motivation that lead to this dissertation and Section 1.3 details what this works proposes to achieve. Finally, in Section 1.4 a brief overview of the structure of the document is given.

1.1 Context

FCPortugal[44] is a joint project from the University of Porto and the University of Aveiro that participates in multiple robotic competitions, such as Festival Nacional de Robótica[48] and RoboCup[14]. RoboCup, an annual competition, is considered the World Cup of robotic soccer and attracts participants from all over the world. This competition includes multiple leagues and sub-leagues. Among the Soccer leagues, in one sub-league, 3D Soccer Simulation, two teams of 11 simulated humanoid robots compete in a simulated soccer match.

Recently, improvements developed by researchers involved in FCPortugal’s team in the behaviours robots use to play the game have brought the possibility of shifting development focus from low-level behaviours to high-level tasks. This dissertation aims to build on that work, as well as other developments previously done for this and other sub-leagues, in order to improve the FCPortugal’s 3D Soccer Simulation robotic team.

1.2 Motivation

In the 3D Soccer Simulation league, the simulated robots obey realistic physics, behave autonomously, and coordinate to increase the chances of winning the game. This introduces a challenge, as good results in Real-Time Multi-Robot coordination and planning in continuous environments are tough to achieve. A soccer game does, however, have moments where the flow

of the game is stopped. These moments (setpieces) allow the team to execute setplays (preplanned moves), as the robots can position themselves on the pitch according to the plan, autonomously execute their assigned tasks following the team's plan and know exactly what the team expects of them in the multiple moments during the setpiece. If the robots are correctly taught to follow a given setplay, multiple setplays can be defined for different setpieces to increase the chances of scoring a goal.

While the process of programming a robot to execute some behaviours is currently not a hard challenge, the developed behaviours still need parameters to function correctly. Even though these parameters can be correctly set manually, they offer an opportunity to optimize the team's chance to win if they are optimized. To seize these opportunities, multiple state of the art machine learning approaches can be used. Among them, Reinforcement Learning, Deep Reinforcement Learning, Multi-Agent Deep Reinforcement Learning and Evolutionary algorithms seem the most suitable for the task. These algorithms will allow each robot to learn to maximize the usefulness of the behaviours they are asked to perform, which will lead to a better outcome of a given setplay.

1.3 Objectives

This dissertation aims to propose the optimization of robotic soccer setplays as a machine learning problem, a language for the definition of setplays and setplay optimizations and the development of a setplay optimization framework for the FCPortugal 3D Soccer Simulation team, which will allow the creation of new setplays and the optimization of robots behaviours on the setplays using a machine learning approach.

Furthermore, for the purpose of testing the aforementioned developments, a setplay was designed and implemented using the optimization framework. This optimized setplay will be tested against the FCPortugal team and against other teams that usually face this team, and the results will be compared with those obtained by the FCPortugal team in the previous competitions.

1.4 Document Structure

The remaining document is split into the following chapters. Chapter 2 provides an overview of robotic soccer, with a more in-depth look at the RoboCup competition, the Soccer Simulation 3D League and the FCPortugal project. In Chapter 3 an introduction to machine learning, deep learning and their applications for robotic soccer is given; furthermore, state of the art on deep reinforcement learning, multi-agent deep reinforcement learning and evolutionary algorithms is presented in the same chapter. Chapter 4 comprises a definition of setplays and a review of work related with setplay framework, including an analysis on a previously implemented setplay optimization framework for the FCPortugal team. Chapter 5 presents the new proposed setplay optimization framework with an in-depth look at the new setplay definition language and the setplay parser. Chapter 6 describes the evaluation done to test the developed framework as well as

some conclusions from the results obtained. Finally, Chapter 7 covers the conclusions of this dissertation, including the main contributions from this work and the future work needed to improve it.

Chapter 2

Robotic Soccer

This chapter gives background information on the context in which this work was developed, providing an explanation on multiple concepts of robotic soccer. Section 2.1 describes how the idea of robotic soccer emerged and the evolution of its biggest competition, RoboCup. The rules of the sub-league in which the FCPortugal team participates are defined in Section 2.2. Finally, Section 2.1 presents a brief history on FCPortugal's team as well as the last developments on the team.

2.1 RoboCup

RoboCup, the Robot World Cup Initiative, was first introduced in 1995. The purpose was to propose the use of the game of soccer, where a soccer team is seen as a multi-agent system, to create a new problem for AI and Intelligent Robotics research in an effort to foster multiple fields, such as "design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor fusion"[23], which are essential for a team of robots to play a game of soccer. In such a game, each team needs to collaborate to achieve a common goal, and as the goal of both teams is the same, they must compete for it.

In the introduction of the initiative, three robotic soccer competitions were proposed: a competition with real robots, a competition of simulated robots and a competition of robots' special skills; furthermore, the rules for each competition were defined, and simulators were proposed for the simulated games. One of those simulators, "The Soccer Server", has since become the official RoboCup soccer server[12].

The first RoboCup competition was held in 1997 in Nagoya, Japan, with over 40 teams competing and over 5,000 spectators watching. Since then, RoboCup has become an annual international competition and has extended the number of leagues. Currently, there are 5 distinct competitions domains[14]:

- RoboCupSoccer, where robots play the game of soccer;

- RoboCupRescue, where robots attempt to solve search and rescue scenarios;
- RoboCup@Home, where robots' service capacities are tested in a home environment;
- RoboCupIndustrial, where industrial robots are presented;
- RoboCupJunior, with an educational purpose, where robots from young students are used.

For the RoboCupSoccer[15] competition domain, the following leagues have been created:

- Humanoid - robots from all teams must have a human-like shape and use human-like senses;
- Standard Platform - all teams must use the same robot, the NAO[47] robot;
- Middle Size - each team can design their own robots, but their sensors, size and weight are restricted;
- Small Size - robots' dimensions are further restricted;
- Simulation - all the robots and the soccer game are simulated.

Each league contains multiple sub-leagues. The Simulation League, where the FCPortugal team competes and for which the work of this dissertation will be developed, is split into two sub-leagues: the Simulation 2D league, where the game is played by two teams of simulated robots in a two-dimensional soccer field, and the Simulation 3D league, which increases the realism and complexity of the simulated matches by using humanoid robots in a three-dimensional environment. As this dissertation will work with the FCPortugal team for the Simulated 3D league, this sub-league will be further described in the Section 2.2.

The ultimate goal of the RoboCup Initiative is:

"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." [13].

2.2 Soccer Simulation 3D League

The first Soccer Simulation 3D competition was held in 2004[25]. This league was started with the intention of increasing the realism of the 2D simulated league by moving to a three-dimensional environment. The first robot player model available was a simple sphere, which, while increasing the game complexity, was not as realistic as a humanoid robot could be. Figure 2.1 shows an example of the initial simulation with the sphere agents.

Later on, for the 2007 competition, a humanoid model was finally introduced. The introduction of this model, the Fujitsu HOAP-2 robot, forced teams to focus on controlling the robot's joints and learning low-level skills such as walking, standing up, and kicking the ball. On the next year, this

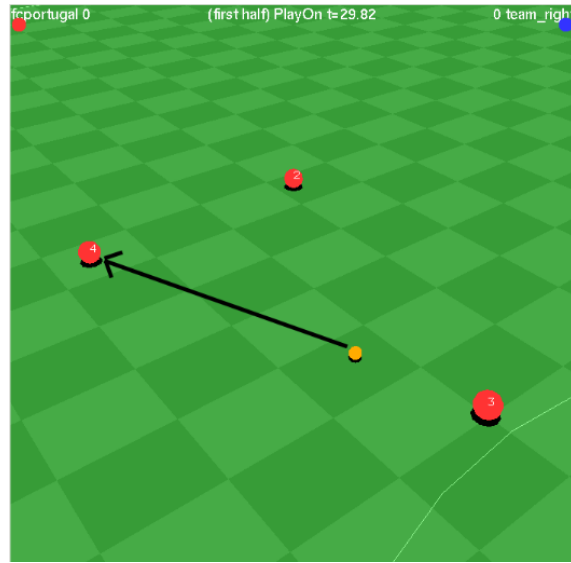


Figure 2.1: Example of the initial 3D League simulation [37]

model was replaced by a model of the NAO robot[47], the same robot used by the RoboCupSoccer Standard Platform league, which is still the robot used today.

The competition rules have been changing since the first competition, and in 2012 was the first year were the games played in the competition featured two teams of eleven players. In the last years, with the teams having learnt how to execute the low-level skills required to play the game, the research focus has shifted once again to high-level skills and multi-robot cooperation, such as setplays and tactics.

2.2.1 Simulation

The simulator used for the Soccer Simulation 3D League is SimSpark[65], which offers a simulation environment where two teams of eleven players can play each other. As previously stated, the players are models of the NAO robot. It also enforces the rules imposed in the competition ruleset[26] and allows for a human referee to intervene in specific situations. The soccer field, along with an example of two teams, can be seen in Figure 2.2.

The field's dimensions and layout are the following:

- The field is 30 by 20 meters long;
- Contains two goals which are 2.1 by 0.6 meters long and 0.8 meters tall;
- The penalty area of each goal is 3.9 by 1.8 meters long;
- The centre circle has a radius of 2 meters.

The soccer ball is a sphere with a 0.04 m radius and has a mass of 26 grams. The field has roughly 35% of the dimensions of a real soccer field. The games are played in two 5 minutes halves. The games are visualised using RoboViz[31].



Figure 2.2: 3D Simulation League field with two example teams.[57]

2.2.2 Player Model

The model used for the robot players is the NAO robot's models, which can be seen in Figure 2.3. The NAO robot is 0.57 meters tall and weighs 4.5 kilograms[56]. It contains 22 degrees of freedom, which provides robots with great mobility. The model's joints, preceptors and effector can be seen in Figure 2.4. Furthermore, it contains a gyroscope and an accelerometer to allow the robot to keep track of its axial and radial movement, a force resistance preceptor in each foot to detect contact with the ground or other robots, a restricted vision preceptor at the centre of its head, a say effector and the corresponding hear preceptor for communication, and a gamestate preceptor to keep track of the playtime and playmode.

Comparatively to older, non-humanoid models, which had easier movement, the NAO robot provides a challenge for teams to develop low-level behaviours, as the robot must be controlled by providing the robot the desired acceleration for each joint, instead of, for example, simply commanding the robot to move to a given place. While this has the downside of harder mobility and harder control of the ball, it brings the game closer to human soccer, which is the ultimate goal.

2.3 FCPortugal

FCPortugal is a joint project from Faculdade de Engenharia da Universidade do Porto and Universidade de Aveiro that participates, among other competitions, in the RoboCupSoccer Simulation League. The project started in 2000 and, in the first year, the team became RoboCup 2000 European Champion and RoboCup 2000 World Champion[44] in the RoboCupSoccer 2D Simulation

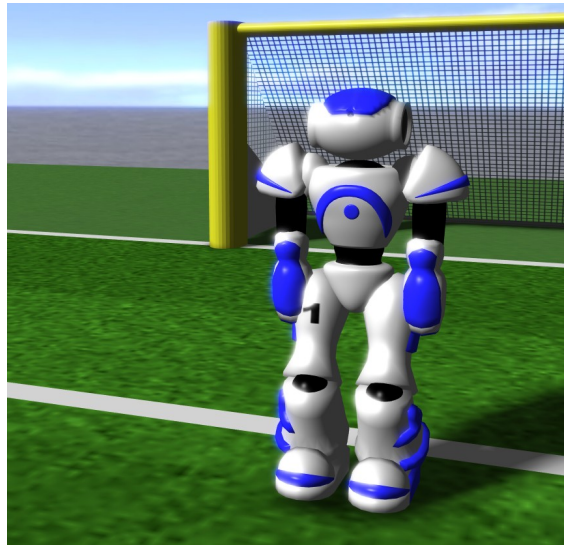


Figure 2.3: Modeled NAO robot.[56]

League. With the creation of the 3D Simulation league, the team switched to this new league. In recent years, the team has reached 4 3rd places in the RoboCupSoccer 3D Simulation League: 2013, 2015, 2016 and 2018.

Along with the team success, multiple research has been done and various low-level and high-level skills have been implemented. Initially, while the team still participated in 2D competitions, the focus was on developing high-level skills, such as tactics and setplays. From this period, the following high-level skills were implemented:

- Definition of player types[45]: although each robotic team agent is equal, their behaviours can be different just as they are in human soccer. The player types allow the team to have greater flexibility in the planning of tactics, where, for example, some players can focus on the recovery the ball from the opposing team, others can focus on covering other opposing team's robots, and other can focus on scoring; furthermore, DPRE (Dynamic Positioning and Role Exchange), allows the robots to switch roles based on their position if they deem it beneficial for the team;
- Team strategies with defined tactics, formations and player types: As seen in Figure 2.5, tactics involve the use of different formations based on the game state; formations define the desired position of each robot and its respective role;
- SBSP - Situation Based Strategic Positioning[45]: during a game, each situation is split into two different types: active situations, where the agents focus on ball possession or ball recovery behaviours, and strategic situations, when the agents are not close to the ball and can behave according to the overall strategy. In these situations, agents analyse the current tactic and formation, their position and their observation of the game state, and calculate what the best position is for them to be;

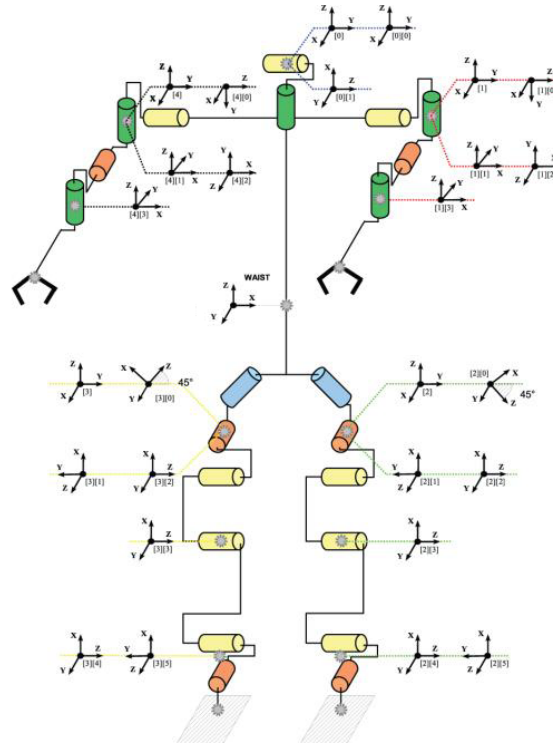


Figure 2.4: NAO Model's joints, effectors and actuators.[56]

- COACH UNILANG[46] allows the analysis of the team's games in order to extract information from the opponent teams' tactics, which then allow the team to adapt their tactics according to the tactics of their opponents;
- ADVCOM - Advanced Communication Mechanism[45]: this mechanism is responsible for the communication between the teams' agents. As the communication channel may have low bandwidth and be unreliable, only the most important information should be communicated. This information is usually to maintain the agents' world state up to date and the communication of team-wide useful events;
- PlayMaker[29] and SPlanner[9]: a framework for the definition of setplays, Playmaker, was later developed. Each setplay is defined by several steps involving one or more agents, initiation, abortion, and finishing conditions that dictate when the setplay should be started, aborted, or finished, respectively. To aid users in defining such setplays, a graphical interface, SPlanner, was later created. As this dissertation will focus on creating a setplay framework for the 3D team, the 2D framework will be discussed further in Chapter 4.

With the switch to the newly created 3D Simulation League, the focus was changed to learning low-level skills, as the process of teaching robots these skills was not trivial. In recent years, low-level skills have been developed for robots to get up[1], walk[50], run[2], and multiple ball kicks[1, 51, 61]. Furthermore, one high-level skill has been implemented for kick-off[3].

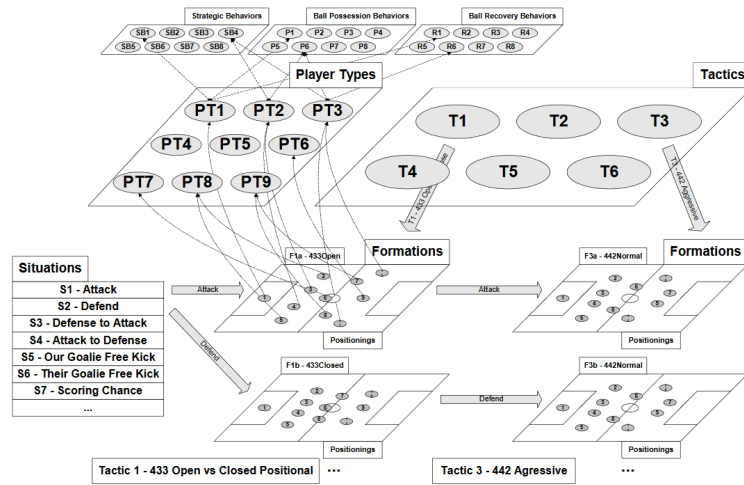


Figure 2.5: FCPortugal Team Strategy composed of Tactics, Formations and Player Types.[45]

Now that these low-level skills have been implemented, the team can start to look again into implementing high-level skills, such as tactics, formations and setplays, which leads to the proposed solution of this dissertation, creating a setplay framework for the definition and the implementation of setplays.

Chapter 3

State of the Art on Machine Learning for Robotic Soccer

This chapter presents the state of the art on machine learning techniques and algorithms available to be used for robot soccer learning and optimisation. A brief introduction is provided: to Machine Learning and its different subfields is provided in Section 3.1; to Robot Learning in Section 3.2; and to Deep Learning in Section 3.3. Section 3.4 offers an in-depth look into Deep Reinforcement Learning, the differences between model-free and model-based reinforcement learning, policy optimization and q-learning and multiple algorithms from each domain. Furthermore, it presents OpenAI Gym, a standard in Reinforcement Learning, along with FCPGym, FCPortugal's adaptation. Moving to Multi-Agent Deep Reinforcement Learning, Section 3.5 details its advancements for multi-agent environments, along with some of the algorithms developed. Finally, Section 3.6 explains Evolutionary Algorithms and how they can provide a different perspective on solving optimisation problems.

3.1 Machine Learning

Machine Learning (ML) is the study of algorithms that allow computers to learn how to perform a task through experience[35]. It is a branch of Artificial Intelligence (AI), which builds models that make predictions without being directly programmed on how to achieve those decisions. Some of the applications of machine learning are document classification, natural language processing, computer vision, fraud detection and learning to play games.

While new approaches are proposed every day, ML algorithms are often split into three main categories: supervised learning, unsupervised learning and reinforcement learning. Other learning scenarios are, for example, semi-supervised learning, on-line learning and active learning.

Supervised learning algorithms learn a mathematical function that maps the model's inputs to the desired outputs. For that reason, during the training phase, the algorithms must be fed with

ground truth examples of inputs and their respective outputs from the learning domain. Supervised learning algorithms are further split according to the type of variable they are trying to predict. If the variable is discrete (can only have a restrict number of values), classification algorithms are used, while for continuous variables, regression algorithms are used. An example application of a classification task is the classification of documents in categories, and an example of a regression task is the prediction of stock values.

On the other hand, unsupervised learning models are only provided with input data during training and are tasked with finding groups in the input data. Again, these algorithms can be split into two categories: clustering, which is the partitioning of data into different clusters/groups, and dimensional reduction, which consists in reducing the dimension of data that represents an item by creating a new representation with a smaller dimension. Clustering can be used to classify users according to their purchases, and dimensional reduction can be used on large datasets to condense item representations to help other machine learning techniques learn something from the data.

Reinforcement Learning (RL) is different as the algorithms are not provided with the input from the begging of training. In contrast with other learning techniques, in RL, the learner must interact with their environment, and perceive which actions are best for each environment state by calculating each action's reward. As these algorithms will be used in this dissertation by the robots for the task of learning setplays, they will be further explained in Section 3.4.

3.2 Robot Learning

Robot Learning is a field that intersects the robotics and machine learning fields. For robot learning, multiple machine learning approaches have been adapted to the domain, including reinforcement learning, inverse reinforcement learning and regression[43]. The interest in robot learning has surged from the fact that some tasks are nearly impossible to program robots for, not all situations can be predicted when programming and real-world scenarios are always changing, so robot learning provides a solution to train robots to adapt to non-stationary environments. Robot Learning has many applications in the robotic field. Among others, "low-level and high-level control and planning, perception and sensor fusion, as well as models of the robot and its environment"[24].

Reinforcement Learning is suited for robot learning as the ability to self-improve in respect to a reward function is essential for robots to become more autonomous[43]. In the past, three main styles of reinforcement learning are usually seen in robot learning: "model-based reinforcement learning, model-free value function approximation methods, and direct policy search"[43]. However, new advancements in machine learning, due to availability of big data and the increase in computation power, have made it so deep learning techniques can outperform hand-designed algorithms in many tasks [24]. Like other fields, the focus in robot learning has also switched to deep learning and deep reinforcement learning techniques.

3.3 Deep Learning

Deep Learning (DL) is another sub-field of machine learning, which uses artificial neural networks (ANN) to approximate non-linear functions. The typical architecture of an artificial neural network, as seen in Figure 3.1 allows models to generate any number of outputs and receive any number of inputs. The network's complexity can be altered by modifying the number of hidden layers and the number of nodes in each layer. Each node consists of a series of inputs, a weight attributed to each of those inputs, the node output, and an activation function which maps the inputs into the output. During training, the nodes' input weights are optimised using back-propagation, based on a loss function. The name Deep Learning comes from the high number of hidden layers used in ANNs, which is only possible due to recent advancements in computation power as the training phase becomes longer as new layers are introduced.

The main advantage of Deep Learning is that Deep Learning models that are composed of multiple layers can learn "representations of data with multiple levels of abstraction"[27].

They have become popular in various fields. Among them is image classification[20], where Deep Learning models can integrate high-level and low-level features in the hidden layers, as well as classifiers in the last layers. This contrasts with older image classification models, which had to extract image features by hand and then use those features to classify the image.

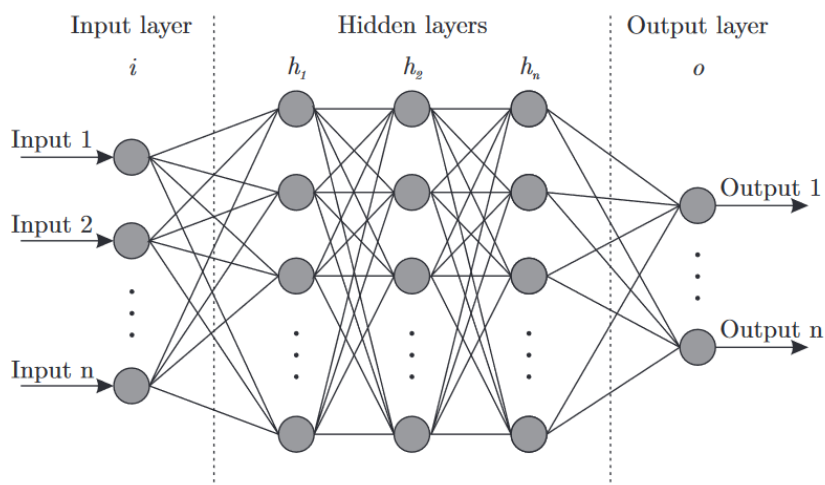


Figure 3.1: Artificial neural network architecture[7]

3.4 Deep Reinforcement Learning

Reinforcement Learning (RL) is, as previously stated, another sub-field of machine learning. In Reinforcement Learning, instead of being given inputs during the training phase, Reinforcement Learning algorithms learn by taking actions in a given environment to maximise their perceived rewards (based on their observation of the environment). An agent using RL techniques learns

”behavior through trial-and-error interactions with a dynamic environment”[22]. The cycle of analysing the environment state, taking an action and perceiving a reward can be seen in Figure 3.2.

The keys concepts in RL are agent, which is being trained to perform a given task; environment, the world where the agents reside and which the agent interacts with; observation, the perceived state of the environment by the agent (which may not be complete); action, taken by the agent to alter the agent’s and the environment’s states; and reward, which is calculated by the agent based on a value function according to their environment’s observation, and represents how good/bad the current situation is. The main goal of the agent is to maximise the cumulative reward during each training episode.

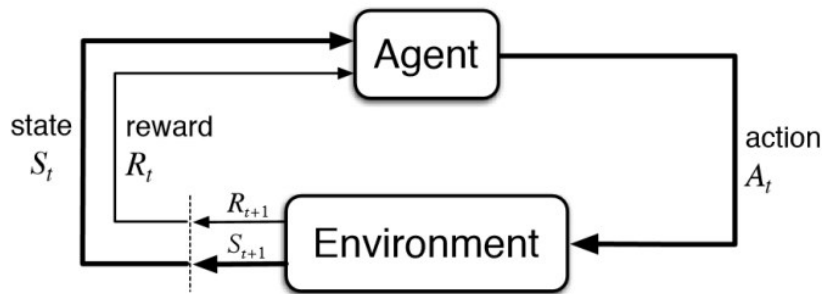


Figure 3.2: Agent-Environment Interaction in Reinforcement Learning[59].

Deep Reinforcement Learning (DRL) comes as the intersection of two machine learning fields - deep learning and reinforcement learning. With the increase in popularity of deep learning techniques in other machine learning fields, new algorithms and adaptations of older algorithms were developed to take advantages of the increase in computation power. The agents use Artificial Neural Networks to translate their input, the environment’s observation, into an output, an action to take or the expected value of each possible action. The hidden layers in the ANN allow for the agent’s actions to take into account the high-level and low-level concepts from the environment that may otherwise not be perceived by the agent. Figure 3.3 shows a representation of the use of ANNs in a deep reinforcement learning system.

Moving to Deep Reinforcement Learning algorithms, there are many different but overlapping categories. According to OpeanAI[41], DRL algorithms can be split according to their use or not of an environment model or not. Furthermore, Model-Free algorithms may be divided in Policy Optimization algorithms that learn a policy between states and actions and attempt to optimise that policy, or Q-Learning algorithms that try to optimise a function $Q(s, a)$ which maps the perceived reward of taking an action a in an environment state s . On the other hand, Model-Based algorithms can be split into algorithms which must learn a model of the environment and algorithms which are given a model. Figure 3.4 shows algorithms used in modern reinforcement learning and their division in the previously defined categories.

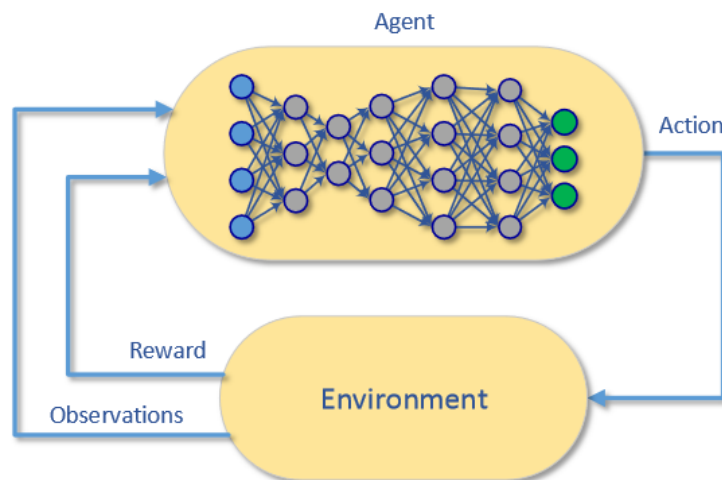


Figure 3.3: Deep Reinforcement Learning system[34].

3.4.1 Model-Free RL

In Model-Free Reinforcement Learning algorithms, the trained agent does not have access to a model of the environment. An environment model is a function that predicts transitions between states and the rewards associated with such transitions. The main downside of not using such a model is that such models allow the agent to think ahead and choose actions based on long-term perceived rewards, as agents can predict how the environment may change with their actions. While model-free algorithms can also allow agents to plan ahead, the number of training samples to do so is higher than model-based algorithms[41]. On the other hand, model-free algorithms are simpler to understand and implement, which causes them to be more popular and better developed and tested. Model-Free Reinforcement Learning algorithms can be further categorised into Policy Optimization, Q-Learning or a combination of the two.

3.4.1.1 Policy Optimization

Policy-based algorithms work by building a policy $\pi_{\theta}(a|s)$ that maps a probability for the agents to take an action a at the state s . By optimising θ , which is a parameter of the function used to map the policy (in deep reinforcement learning, corresponds to the ANN's weights), the policy is optimised for the task being trained[41]. The optimisation is done on-policy, meaning updates to the policy are only based while acting according to the latest policy. Policy-based models usually vary in how they optimise θ and how and when to update the policy.

Some examples of Policy Optimization algorithms are:

- **Policy Gradient:** Policy Gradient Algorithms attempt to directly optimized a policy by optimizing θ , according to a reward function given by equation 3.1 where $d^{\pi}(s)$ represents a stationary distribution of a Markov chain for the policy π and Q^{π} the value of executing

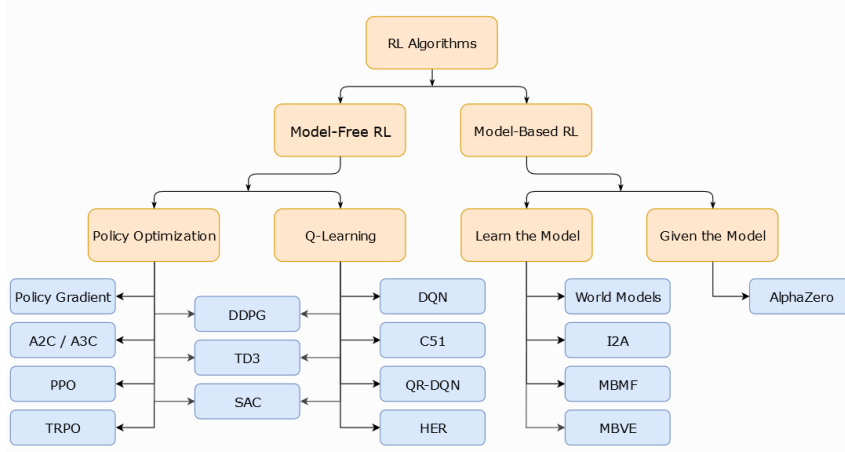


Figure 3.4: Taxonomy of algorithms in modern RL[41].

action a on state s according to policy π_θ [64].

$$J(\theta) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a) \quad (3.1)$$

Policy Gradient works by reformatting the objective function not to involve the state distribution. REINFORCE, a Monte-Carlo policy gradient algorithm uses "an estimated return by Monte-Carlo methods using episode samples to update the policy parameter θ "[64].

- A2C / A3C: Advantage Actor Critic (A2C) and Asynchronous Advantage Actor Critic (A3C) algorithms[32]: These methods are actor-critic as they consist of two models: a critic, that learns a value function either for each state or for each state-action pair and an actor which updates the policy according to the critic. A3C is designed for parallel training, as the critics learn with multiple actors running in parallel which are periodically synced. A2C is a synchronous version of A2C and was developed because, due to its parallel nature, agents in A3C could be using non-updated parameter values, which would lead to non-optimal learning. On the other hand, in A2C, a coordinator awaits the end of an iteration of all agents' training before updating the model's parameters and allowing agents to start another iteration. Figure 3.5 show the different architectures between A2C and A3C.
- TRPO: Trust Region Policy Optimization[21] was introduced to improve training stability by trying to reduce policy variation between iterations. To achieve that, it introduces a "divergence constraint on the size of policy update at each iteration"[64].
- PPO: Proximal Policy Optimization[49]. The problem with TRPO is that the introduction of the divergence constraint may make good results hard to achieve. For that reason, PPO simplifies the constraint by improving the policy as best as possible while ensuring there is not a significant change from the last policy.

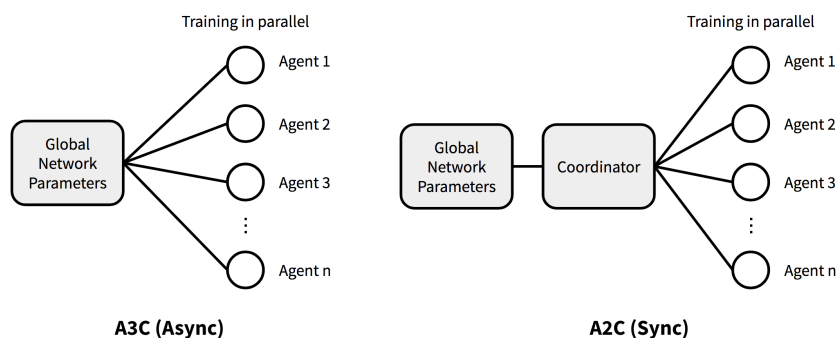


Figure 3.5: A3C and A2C architectures[64].

3.4.1.2 Q-Learning

Q-learning algorithms learn a function $Q_\theta(s, a)$ and approximate it to the optimal action-value function. This Q function maps the expected reward the agent will receive by taking action a in state s . The training is done off-policy, meaning the agent can use any previously obtained data to update the Q function parameters. Instead of building a policy for the agents to follow, this approach tries to maximise the value of the actions taken at each step using a greedy approach. At each state, the action taken by the agent is given by equation 3.2[41]

$$a(s) = \operatorname{argmax}_a Q_\theta(s, a) \quad (3.2)$$

Some examples of Q-Learning algorithms are:

- **DQN: Deep Q-Networks**[33]. Based on the classical Q-Learning algorithm which used a matrix to store all the Q values for each action-state pair, DQN replaces that matrix with a neural network. The main problem of the Q-Learning is that it needs to train in all action-state pairs, making the training phase very long, as it grows exponentially with the number of actions and states. To handle that issue, DQN use a neural network to approximate the $Q(s, a)$ function, as seen in Figure 3.6. DQN also implemented two further optimisations: Experience Replay, where instead of updating the neural network with every new observation, each observation is stored, and then only a random subset of them are used to update the neural network, and the addition of a Target Network, which is a copy of the neural network which is used to predict new actions during training instead of the original network and is not trained, being instead periodically synchronised with the original network.
- **C51: Categorical 51-Atom DQN**[6]. Categorical Deep Q-Networks replaces single numerical Q-values for any given action-state pair by a Q-value probability distribution. A numerical Q-value usually represents the mean value that the reward can take. Thus, that value may not fully represent the state-action pair. If, for example, after executing an action, the agent may, due to environment changes, get to a state with a reward of -1 or a state with a reward of 1, such state would be different from a state where the agent always has 0 rewards,

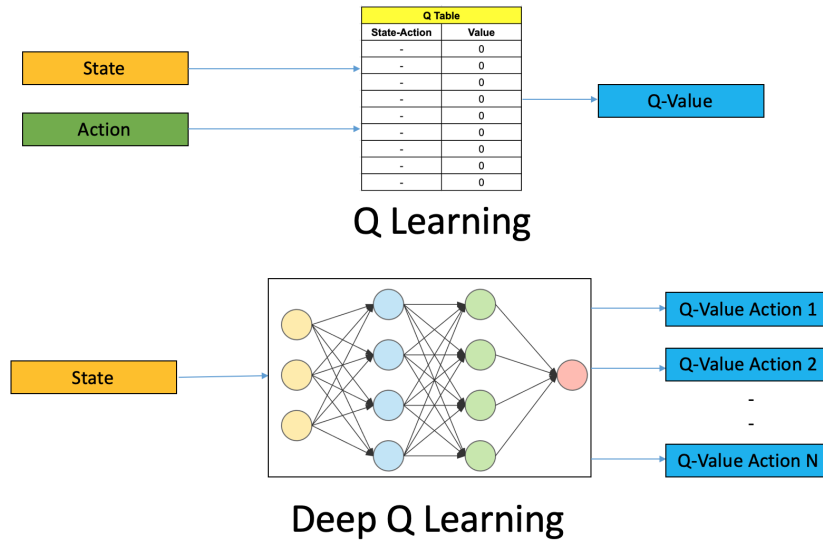


Figure 3.6: Difference between Q Learning and Deep Q Learning[55].

but the Q-value would be the same. By taking into account this distribution, the agent can make better-informed decisions, leading to better results.

- **QR-DQN: Quantile Regression DQN**[10]. Similar to C51, a distribution of the Q-Value of each action-state pair is also used. This algorithm uses a number N of quantiles to approximate the distribution of possible Q-Values.
- **HER: Hindsight Experience Replay**[4] Most problems in Reinforcement Learning only have domain rewards when a given episode ends as positive or negative. This introduces a problem, as agents may not be able to measure intermediate states. While in most problems, intermediate rewards may be given using Reward Engineering, this process requires that users understand the problem domain, which may not always be the case. To combat this problem of binary rewards, as well as sparse rewards, HER was developed. If, during an episode, an agent fails to reach a goal, instead of just updating the Q-function with such failure, there is also an update of success for the agent to reach the final state, which may later become useful. This allows the agent to learn about its environment and its task even when an episode fails.

3.4.1.3 Policy Optimization & Q-Learning

Both Policy Optimization and Q-Learning have advantages and disadvantages. The main strength of policy optimisation is that the method optimises a policy according to a policy given by the user, which tends to make the developed models more stable, while in Q-Learning, the algorithm mostly optimises the expected final reward based on a reward function, which can be unstable. However, when they work, Q-Learning algorithms tend to be faster to train, because they can always reuse past data to continuously improve the model.

To try to acquire advantages from both Policy Optimization and Q-Learning, some algorithms were developed with features from both classes:

- DDPG: Deep Deterministic Policy Gradient[28]. This algorithm learns a policy and a Q-function concurrently and uses them to improve each other. As in actor-critic models, it uses one network to propose actions (using a policy) and another to predict their value (using a Q-function). As in DQN, it uses Experience Replay to learn on a set of sample, and the concept of Target Network, but instead of using a single target network, two are used to add stability in training.
- TD3: Twin Delayed DDPG[18]. TD3 is based on DDPG. The main difference is that instead of learning a single Q-function, TD3 learns two. These two Q-function act as two distinct critics, returning different Q-values for each action proposed by the actor. Due to this, TD3 models are generally faster to train than DDPG models.
- SAC: Soft Actor-Critic[19]. SAC core idea comes from the concept of Maximum Entropy Reinforcement Learning. While most RL algorithms maximise expected rewards, in Maximum Entropy Reinforcement Learning actors also aim to maximise entropy, which means that agents should try to succeed at the given task while trying to be as random as possible. This allows models to generalise better, which gives models better results when compared to other models.

3.4.2 Model-Based RL

In Model-Based Reinforcement Learning algorithms, the agent being trained has access to a model of the environment, which allows the agent to plan ahead, and deciding which actions to take based on the scenarios and future states such actions will trigger. The planning results can then be used into building a policy[41], which decreases training time. However, having models that are true to the real-world environment is pretty rare, which means agents will also have to learn a model. If the learned model is incorrect, differences between the real-world and the model may be used by the robot to increase training rewards, and as they are not congruent with the real world, the agent will not take optimal actions when exposed to the real-world.

As previously stated, Model-Based Reinforcement Learning algorithms may be split into algorithms which must learn the model and algorithms which are given the model.

3.4.2.1 Learn the Model

Some examples of Model-Based Reinforcement Learning algorithms that learn the environment model are:

- MBMF: Model-Based RL with Model-Free Fine-Tuning[38]. MBMF uses a basic approach by not explicitly modelling a policy, but instead use planning techniques like model-predictive control to select actions[41]. Each time the agent takes an observation, it creates

a plan of all actions optimal according to its model that it needs to take in a fixed amount of steps. When the agent executes the computed plan's first action, it takes a new observation and once again creates a plan from scratch, never reusing old plans.

- MBVE: Model-Based Value Expansion[16]. To augment training data, recent Model-Free RL approaches try to build learned models to create additional training data. However, these methods rely on heuristics that limit the usage of such models. MBVE attempts to circumvent this problem by controlling model uncertainty by limiting imagination depth. MVE uses a "dynamic model to simulate the short-term horizon and Q-learning to estimate the long-term value beyond the simulation horizon"[16].
- I2A: Imagination-Augmented Agents[63]. In this algorithm, the planning model is directly joined with the learning policy, where the created plans are information encoded in the policy. Thus, the policy can choose if, how and when to use such planning, which may boost results by decreasing the planning model bias, as if such model is wrong for some environment states, the policy may learn to ignore it.

3.4.2.2 Given the Model

While there are other algorithms, the most well known Model-Based RL method with a given model is AlphaZero[52]. Given a world model, AlphaZero uses Monte Carlo Tree Search to evaluate the quality of the current state and plan which actions to take in the future. This produces better actions than the actions produced by the policy alone and may be considered "expert" analysis. These actions are then used to train the policy in producing actions closer to the ones produced by the planning algorithm (Monte Carlo Tree Search).

AlphaZero became famous as it was able to beat top computer engines in chess, shogi and Go in a short amount of training time.[53] Also impressive was that it performed better than classical engines while searching 1000x lower moves per decision. Figure 3.7 shows the results AlphaZero obtained in the three games.

3.4.3 OpenAI Gym

To develop and test new deep reinforcement learning algorithms and compare them with other already implemented algorithms, OpenAI created Gym[39], a toolkit that allows for the definition of training problems - environments - that can be used to test reinforcement learning algorithms. Furthermore, it serves as a standard on how reinforcement learning algorithm implementations should communicate with the environment to retrieve information on the environment's state, transmit to the agent the action it should take, and calculate the reward of said action. It was developed as there was no benchmark for RL algorithms, unlike supervised learning where there are benchmark datasets, and the public environments that did exist did not follow a standard and so were hard to use, as each had differences in multiple aspects: the environment definition, the observations, the

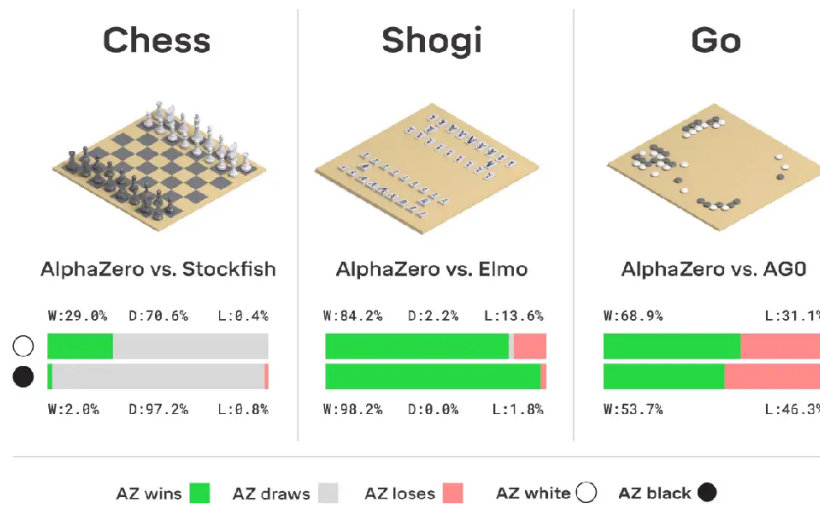


Figure 3.7: Results from games between AlphaZero and top classical engines in the games of chess, shogi and Go[53].

rewards, and the actions; because of this, it was hard to compare results from research done in different publications. Gym also provides ready to use environments[40] for classical RL problems: basic algorithm implementation, Atari games and robotic arms, among others. Along with Gym, Open AI also provide baselines[42], a set of implementations of commonly used RL algorithms, ready to be used with any Gym environment.

This toolkit allows for two different use cases:

- Researchers looking to improve or create new RL algorithms can use Gym to develop their algorithms, needing only to define some function to interact with a Gym environment. They can then use Gym environments to test their algorithms and compare them again others.
- Researches looking to optimize a problem only need to define their environment as a Gym environment and are then able to use state of the art RL algorithm implementation to aid them in the optimization problem.

To define a Gym environment or adapt one to be used by Gym, only a few steps are required:

- The definition of an initialization function which starts the environment; a reset function which resets the environment after an episode and returns an initial environment observation; and a step function, which given an action, executes said action and returns an observation of the environment after the action is taken, the reward of taking said action and whether the current training episode is finished.
- It is also required to define the action and observation spaces. The space reflects all the values that an action or an observation may take.

3.4.3.1 FCPGym

To take advantage of OpenAI Gym, the FCPortugal team has created FCPGym[51]. FCPGym works as a layer above Gym, allowing the team's researchers to create Gym environments using the code already developed for the FCPortugal robots in order to create new behaviours using Deep Reinforcement Learning. Whenever an FCPGym environment is created, SimSpark (the robotic soccer simulator) and an FCP agent are launched. This agent has all the capabilities of a regular FCP robot player but without its decision making. Instead, for each timestep, the environment's step function is called for the training algorithm to request the environment to execute a given action. The communication flow between the environment, the FCP agent and the learning algorithm can be seen in Figure 3.8.

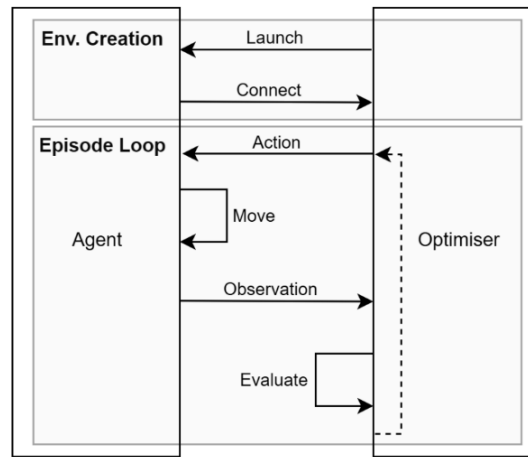


Figure 3.8: Communication between the agent, environment and the optimiser (RL algorithm).[51]

Since its creation, FCPGym has been improved to allow the parallelisation of environments, to allow multiple instances of Simspark and FCP agents to be run simultaneously to increase training speed. This advancement also allowed the training of multiple agents in the same environment. Multi-agent environments, however, still has a few restrictions that may hinder the development and training of environment with tasks for multiple agents. Of these restrictions, most can be easily circumvented. Still, one of them may pose a problem for some environments: Every agent must sync with the server simultaneously and the same amount of times, which means that, for a given environment, all agents must finish their episodes at the same time, regardless of when their task is complete.

3.5 Multi-Agent Deep Reinforcement Learning

While Deep Reinforcement Learning algorithms are well suited for single-agent tasks, they are not optimal for multi-agent systems, as learning algorithms may not converge in environments with multiple agents with goals. For this reason, the field of Multi-Agent Deep Reinforcement

Learning (MADRL) emerged to develop algorithms for the training of agents in cooperating to achieve a common goal or agent competition for a singular goal.

The main benefits of MADRL come from a speedup of training as agents can learn in parallel, experience sharing by either agents teaching each other or by imitation, and a more robust system, as if an agent fails, other can take over their tasks. The main challenges are the curse of dimensionality, goal specification, a changing goal during training as other agents change their policies, the single agent and agent group exploration-exploitation trade-off, and the need for agent coordination/communication[8].

The most relevant MADRL algorithms found in literature are[54]:

- Independent Deep Q-Networks[60]. Based on DQN, each agent is controlled by an independent Deep Q-Network. Work on this algorithm shows that it is possible to learn collaboration and competition between agents by changing the rewards given to each agent.
- COMA: Counterfactual Multi-Agent Policy Gradients[17]. Based on actor-critic algorithms, COMA uses a "centralised critic to estimate a Q-function and decentralised actors to optimise the agents' policies"[17]. This means that while the training must be centralised, agents' execution can be independent, and agents may be run using only local information. These advantages make the algorithm achieve slightly better results than other multi-agent actor-critic methods, and the agents are competitive as centrally-executed agents.
- Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments[30]. This algorithm is an extension to DDPG. Like COMA, the training must be centralised, but agents execution can be independent. Further, it uses a centralised critic, but instead of using a singular Q-function for all agents as in COMA, it uses an individual value function for each agent. An advantage of having an individual value function is that it may be used for applications where agents have to perform different tasks or may not have a common goal shared by all agents.
- Value-Decomposition Networks[58]. With Value-Decomposition Networks, agents learn a multi-agent-action value based on their own observations and the junction of all agent's value functions is used to approximate a centralised value function. Results show that such agent-based joint-action value functions can lead to superior results, especially "when combined with weight sharing, role information and information channels"[58].

3.6 Evolutionary Algorithms

Evolutionary Algorithms (EA) provide a different perspective on the robot learning paradigm. Evolutionary algorithms are not a field of machine learning, but instead a subset of Evolutionary Computations and are important optimization and search techniques[62]. Due to their flexible and robust behaviour, they can be used as a problem-solving method for many optimization problems. One such problem is the optimization of robots behaviours in a team coordination plan.

EA algorithms are inspired by biological evolution mechanisms such "as reproduction, mutation, recombination, and selection." [62]. Generally, EA follow the flow presented in Figure 3.9 and can be split into the following steps: initially, a population is created with random individuals; afterwards, multiple iterations (generations) are run. For each generation, each individual executes the task proposed and is evaluated according to a fitness function. Then, the best individuals, according to the previous evaluation, are kept and are reproduced to create new individuals for the next generation.

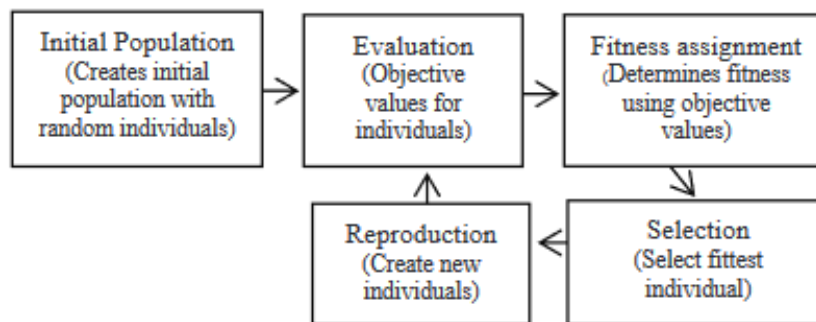


Figure 3.9: Evolutionary Algorithms execution flow[62].

EA algorithms can be split into four categories:

- Genetic Algorithm: Most popular type of EA, typically "used for machine learning, pattern recognition and optimization problems" [62], uses recombination and mutation to find the solution of a problem, which often comes as a binary representation of the genes.
- Genetic Programming: Instead of representing genes as binary numbers, genes represent sets of programming instructions, which allows for solutions to be computer programs, and their suitability for the task can be measured by fitness functions. These algorithms are typically used for boolean and arithmetic operations, as well as mathematical functions.
- Evolutionary Strategy: Often used with complex functions, genes are represented as vectors of real numbers, and mutation rates are self-adaptive. Often used in Biochemistry, Optics and Engineering design.
- Evolutionary Programming: Similar to Evolutionary Strategies, with the main difference of not having any restrictions in the genes data type. Used in Forecasting, Games and Automatic control.

The performance of EA can be improved further by the use of two extensions [62]: Memetic algorithms, which use heuristics in order to facilitate the search for the optimal solution, and distributed EA, which can be used to speed up the algorithm by offloading computation to other computers or execute computations in parallel (usually, the computation can get heavy for larger population or more complex fitness functions).

Chapter 4

Setplay Frameworks

This chapter presents already developed setplay frameworks. Section 4.1 provides a definition on setplays and the multiple types of setpieces that exist in human and robotic soccer. An overview of the previously developed setplay framework for the FCPortugal 2D team is given in Section 4.2.

4.1 Setplay

In soccer, both human and robotic soccer, setpieces are moments where the game's flow is stopped, and the ball returns to open play. Setpieces are important as many goals are directly or indirectly scored from them. The importance of such moments leads teams to create multiple specific plans - setplays - for each setpiece, as the training of such setplays leads to higher chances at goals. Teams need to train both the execution of setpieces and the defence of the setpieces from the opposing team. The training of such setpieces by players is fundamental, and some players specialize in setpieces, mainly on free-kicks and penalties. In the last years, teams have tried to capitalize on setpieces by having coaches specifically to train setplays.

In games of robotic soccer, setplays are as important as or even more important than in human soccer games. While the game of human soccer has already evolved for many years, and players and teams have elevated the playing quality, robotic soccer is still a young field, and, especially in the case of simulated 3D soccer, agents are still learning low-level skills, such as walking, running, shooting or passing the ball and coordination is difficult to achieve in real-time multi-agent environments. Setplays, as pre-planned collaborative team interactions[36], defined by series of player movements, passes and shots, provide robotic soccer teams with higher chances of scoring goals.

The following are setpieces in the game of soccer[11]:

- Kick-offs. A soccer game is usually split into two halves. At the start of each half, the ball is positioned in the centre of the field, and each team occupies their respective half of the field. One player from one of the teams (one team starts the first half, the other starts the

second) is allowed to pass the ball to one of his teammates, but direct kick-off goals are not allowed. Furthermore, whenever a team scores, the game is reset with a kick-off, where the team who conceded the goal restarts the game. While kick-offs do not directly lead to many goal chances in human soccer, they allow teams to position themselves according to their tactic and provide a chance for the team to initiate an attack quickly. On the other hand, in robotic soccer, they provide an excellent scoring chance as robots' techniques are not yet developed enough to stop a well defined and executed kick-off setplay;

- **Goal Kicks.** Goal Kicks happen when the ball gets out of play after leaving the field through one of the lines parallel to the goal (goal line), and the ball was last touched by a player of the team opposing the goal's line. Goal kicks are the setpiece with the lower scoring chance for the team taking them, but, as other setpieces, they allow the team to re-position on the field and plan an attack.
- **Throw-ins.** Throw-ins are awarded to a team when the ball leaves the pitch through one of the lines perpendicular to the goal line (side lines) and was last touched by a player from the other team and are performed on the side line. As some other setpieces, direct goals may not be scored from throw-ins. The probability a goal is scored after a throw-in is related to the distance the throw-in is from the opposing team's goal. With players' evolution, throw-ins closer to the opponent goal line are almost as effective as corners, as players can throw the ball directly into the opponent's team penalty area.
- **Corners.** Corners happen when the ball gets out of play after leaving the field through one of the goal lines, like in goal kicks, but in this case, the ball was last touched by a player from the team whose goal line the ball went over and are performed on the corner closest to where the ball left the pitch. Corners are one of the most dangerous setpieces, as they allow a team to move most players to the opposing team's penalty area and allow a player to pass the ball there, where other players can kick it into the goal. Although scoring directly from corners is allowed, such goals are hard to perform and are unlikely to happen.
- **Free-kicks.** Free-kicks are awarded when an opposing team's player does a foul outside their team's penalty area and are performed on the spot where the foul occurred. Depending on which specific foul is performed, free-kicks may be direct (where a goal may be directly scored from the free-kick) or indirect (direct goals are not allowed). As throw-ins, the danger of the free-kick to the opposing team is related which the distance to their goal. Goals from free-kicks, either direct or from plays leading indirect free-kicks, are fairly common. To make the task of scoring from direct free-kicks, the opposing team may place players forming a "wall" close to the spot where the free-kick is taken, though the distance still allows players to kick the ball over this barrier.
- **Penalties.** Penalties are the most effective setpiece, where there is a higher probability of scoring than not scoring. Penalties happen when a player does a foul on their team's penalty area, are awarded to the opposing team and are performed on the penalty spot (a specific

spot in the middle of the penalty area). Due to their importance, teams usually have a player who is effective in converting them and is also an important part of goalkeepers' training.

4.2 FCPortugal Setplay Framework

As previously stated, the main focus from the FCPortugal team on high-level strategies was when the team was still competing in the Simulated 2D League. It was during this phase that most high-level skills and tools were developed for the project. For setplays definition, a setplays library, along with PlayMaker[29], a tool that allowed users the graphic design of setplays. This tool was later reworked into SPlanner[9], which re-designed the graphical user interface and integrated the 2D Soccer Simulator within the application.

With the switch to the 3D league, during the first years, while the robot's models were only a sphere, and the movement and low-level skills were easy to develop, the setplays framework was still successfully applied to the 3D team[36]. But with the switch to humanoid robot models, and the increase in the difficulty of implementing low-level skills, such as walking, running and shooting the ball, research was moved to improve those skill, and the setplays framework was abandoned.

In the FCPortugal Setplay Framework, setplays are defined by a name, region of the field where the setplay is supposed to start, the type of setpiece and the players involved[36]. Setplays are represented as sequences of Steps. Each Step contains a time to wait before executing the step, the time the step should take (if the actual time taken reaches this time, the setpiece is aborted), a list of Participants, one or more Transitions (list of Actions that must be taken to reach another Step), and may contain a condition that must be satisfied before entering the step. Each Participant is a player from the team, to which is assigned a specific role and a region of the field. Figure 4.1 represents the diagram of the setplays domain model.

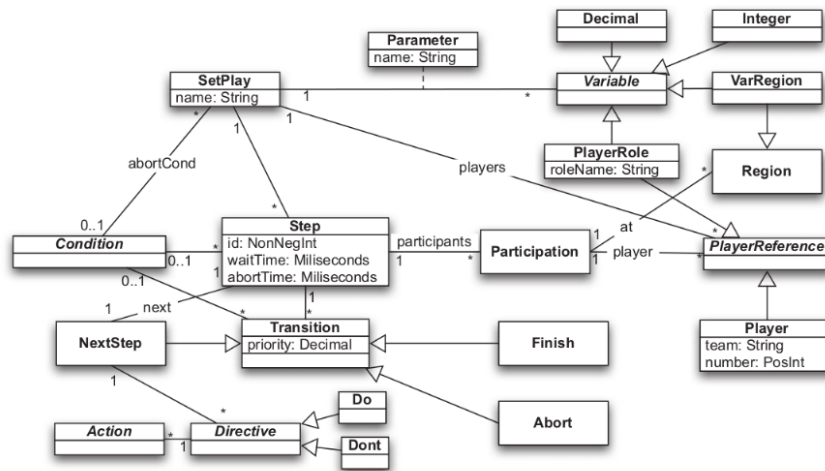


Figure 4.1: Setplay domain model[36].

The main concept in the setplays domain is Action. Actions are the concepts behind the physical movements made by the players that are executed between steps. Actions may be linked with players, both from the same and the opposing teams, with regions of the pitch, or linked directly to the ball. The actions linked to other players are: tackling an opponent, marking a passing line between opponents, passing to a teammate, and marking an opponent. The actions which refer to zones of the field are, among others, dribbling the ball to a position, moving to a position and marking a position. Other actions are holding the ball, shooting the ball and intercepting the ball. An action may also be a sequence of actions. Figure 4.2 shows the implemented actions.

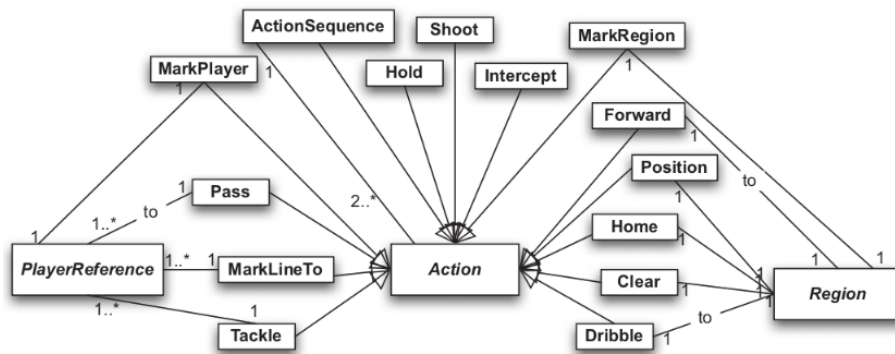


Figure 4.2: Implemented actions[36].

SPlanner, the most recent graphical interface, allow the user to graphically create new setplays or edit setplays already created. As can be seen in Figure 4.3, when creating a new setplay, the user can select the type of play (either offensive when the team has ball possession, defensive otherwise), the situation of the setplay (usually associated to which setpiece is being taken), and the position, which may be a point in the field, the whole field or one or more regions of the field.

Figure 4.3: Creation of a new setplay[36].

After creating the setplay, a screen is displayed with the setplay main information, as seen in

Figure 4.4. On the left side, the user can name the setplay and introduce a comment to explain what the setplay. Furthermore, the user can set the abort conditions and check the graph of steps. For each step, the wait and abort times can also be altered. On the right side, in the pitch, the user can add players to a step, by moving them to the pitch and assign actions by clicking in each one of them. The actions available to assign to players are direct pass, forward pass, dribble, hold ball, shoot, hold position, move to position and run to offside line[36]. The visual representation of each action can be seen in Figure 4.5



Figure 4.4: Defining a setplay in SPlanner[36].

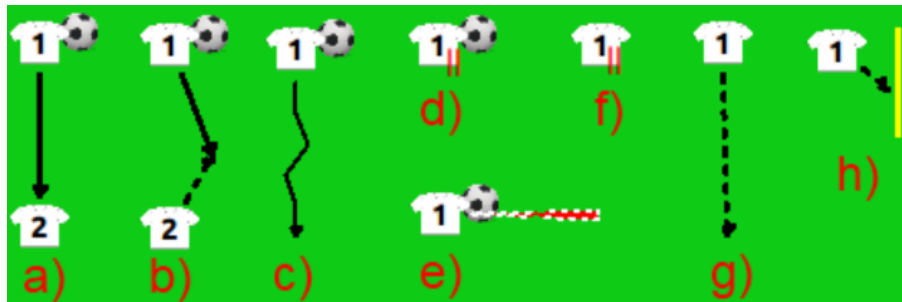


Figure 4.5: Visual Representation of player actions[36]. a) direct pass, b) forward pass, c) dribble, d) hold ball, e) shoot, f) hold position, g) move to position, h) run to offside line.

Figure 4.6 displays three steps from a complete corner setplay involving three players. On the first step, player 6 takes the kick and passes the ball to player 7, while player 8 run to the opponent's team penalty area. On the second step, player 7 passes the ball to player 8, who shoots to the goal on the third and final step. Using SPlanner, the user can create complex setplays which can lead to higher chances of scoring and lower chances of conceding goals. To evaluate and test the implemented setplays, it is possible to run a 2D simulation of the setplay.

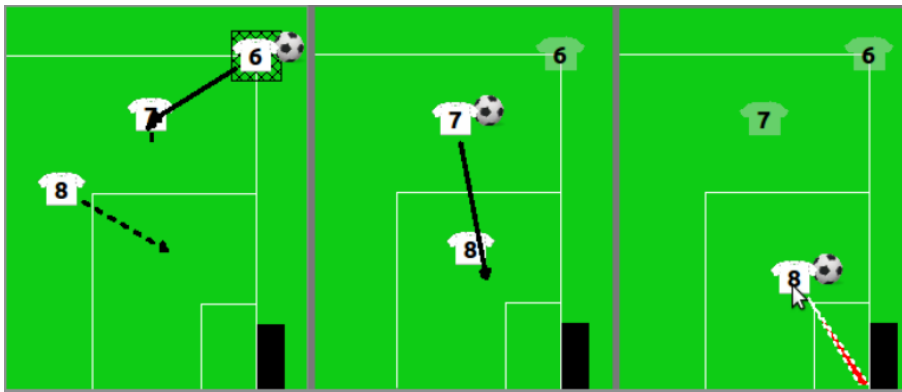


Figure 4.6: Example of complete corner setplay[36].

Chapter 5

New Setplay Optimization Framework

This chapter presents the new framework developed during this dissertation, which was based on the old framework for the FCPortugal 2D team previously presented.

Changes were made for the framework to work with the FCPortugal 3D team and to take advantage of the new tools developed for the team, such as FCPGym. Furthermore, the language was defined not only as a setplay definition language but also as a setplay optimization language definition, as the primary intent of the new setplay framework is not only to allow researchers the definition of setplays for the FCPortugal 3D team but also their optimization using machine learning along with FCPGym.

Section 5.1 presents the new setplay language definition. In Section 5.2, the parser developed to translate the definition of a setplay into a trainable FCPGym environment is explained.

5.1 Setplay language definition

The setplay language definition was adapted by removing, modifying or adding elements. Multiple reasons led to this modification: some aspects that worked for the 2D team don't work for the 3D team, the way the team works changed, and the language has been extended to allow for the optimization of the setplay.

The optimizations proposed by the framework do not imply that the robots, individually or collectively, learn to change the behaviours defined in the setplay to improve it. While that would be the ideal scenario, that level of advancement is not yet possible. Instead, the proposal is to maximize the efficiency of the setplay by learning to optimize the parameters used in the behaviours previously learned. Most behaviours used by the FCPortugal 3D team have parameters to define how they should be executed, whether it is the position to where the robot should move, the position to where it should kick the ball, or if the robot should try to avoid obstacles, among others. These parameters influence the efficacy of the behaviours, and while it is possible to set them by

- Policy Network: The neural network that will be used as a policy network for the robot to learn and then use to translate observations into actions;
 - Timesteps: The number of timesteps that the setplay will be trained. Each time step corresponds to a timestep in the robotic soccer simulator (by default, 0.02 seconds);
 - Actions Space: List of all action spaces used by the setplay. While for training robot's behaviours, the number of action spaces is the number of available robot's joints, and each action space usually corresponds to the minimum and maximum acceleration that a joint can take at each timestep, the setplay framework uses them differently. The action spaces are used to optimize the parameters of the behaviours used on the setplay. Their minimum and maximum values can be set when defining each action space and can then later be rescaled according to the need of each behaviour's parameters.
- Ball: The position of the ball in the field. This may be either a set position (X and Y coordinates) or a rectangular region (defined by two X coordinates and two Y coordinates) where the ball will be randomly positioned each episode. While some setpieces require the ball to be in a specific spot, the framework allows the ball to be placed anywhere on the field to add flexibility to the setplays. This decision also led to the decision of removing the type of setpiece that the setplay is used in. Once again, this gives the setplay flexibility but comes with the drawback of not enforcing the rules each setpiece has (for example, on a team's kickoff, no player may move to the other half until a team's player touches the ball);
- Players: A list of players that participate in the setplay. For each player, their player model number is required. There are currently five different types of players NAO models, numbered from 0 to 4, that can be used, with each having different benefits and drawbacks. Also, some behaviours were trained only with a specific model and only work for that model. For each player, their position is also required. As for the ball, this position may be a set position or a rectangular region where the player will be randomly set for each episode;
- Rewards: A list of rewards given to all players in an episode according to a condition. Currently, there are two types of global rewards, but more can easily be incorporated in the future:
 - Goal: The robots are rewarded with a user-defined value if they can score a goal;
 - Ball Position: The robots may be positively or negatively rewarded with a user-defined value if the ball gets inside a specific region. This may be used to positively reward the robots in, for example, defensive setplays where the goal is to move the ball to the opposition's half or negatively to simulate, for example, that the ball is far away from where it was supposed to be and, in a real match, would be near opposing players;
- Finish Conditions: A list of conditions that, if any is true at a given point in time, successfully finish the setplay/episode. Currently, the same ones as the rewards (Goal and Ball Position);

- **Abort Conditions:** List of conditions that, if any is true, abort the setplay. Currently, there are three implemented conditions and another one that was thought of but not implemented:
 - **Timeout:** All setplays must have a timeout where the episode will end if it reaches a certain timestamp and has not yet finished or aborted by any other condition. This is needed to ensure all training episodes eventually finish;
 - **Goal:** Same as in Rewards;
 - **Ball Position:** Same as in Rewards;
 - **Fall:** If a robot falls, it usually is no longer important for a setplay. However, as previously mentioned, in FCPGym when using multi-agent environments, all agents must finish each training episode simultaneously. Because of this, an agent may not finish the episode because he fell while other agents are still training in that episode;
- **Steps:** As in the original FCPortugal Setplay Framework, the setplay is defined as a sequence of steps. Each step contains:
 - **Duration:** Amount of time that step takes;
 - **Actions:** List of actions that each robot should do in each step. Not all players need to be in each action. Each action is defined by the behaviour that should be executed and the uniform number of the player that will execute the action. Each action also contains:
 - * **List of Parameters:** Each behaviour needs a different number of parameters. These parameters may be a set value if the robot should not try to optimize them, or a minimum and maximum value, where the robot will try to learn the best value for the parameter based on its current observations;
 - * **Reward:** An individual reward may be given at each timestamp to the robot to reward or penalize it. This reward may be based on its distance to a specific position or based on the ball's distance to a specific position. To tune the importance of this individual reward, another attribute of the reward is a multiplier. This reward is negative by default, so the closer the robot is to its objective, the higher the reward;

For the ease of declaration and parsing, setplays are defined in XML files. Listing 5.1 shows an example of the definition of a setplay.

```

1 <?xml version="1.0"?>
2 <Setplay>
3   <Name>Name</Name>
4   <Description>Description</Description>
5   <TrainingParameters>
6     <Envs>1</Envs>
7     <Algorithm>Training_Algorithm</Algorithm>

```

```

8      <PolicyNetwork>Policy</PolicyNetwork>
9      <TimeSteps>1000000</TimeSteps>
10     <ActionSpace>
11         <Parameter Type="float" Min="-15" Max="15"/>
12         <Parameter Type="float" Min="-10" Max="10"/>
13     </ActionSpace>
14 </TrainingParameters>
15 <Ball>
16     <Region X1="-15" X2="15" Y1="-10" Y2="10"/>
17 </Ball>
18 <Players>
19     <Player Type="0">
20         <Pos X="0" Y="0"/>
21     </Player>
22 </Players>
23 <Rewards>
24     <Reward Type="Goal" Value="10000"/>
25     <Reward Type="BallPos" Value="-10000">
26         <Region X1="-15" X2="0" Y1="-10" Y2="10"/>
27     </Reward>
28 </Rewards>
29 <FinishConditions>
30     <Condition Type="Goal"/>
31 </FinishConditions>
32 <AbortConditions>
33     <Condition Type="BallPos">
34         <Region X1="-15" X2="0" Y1="-10" Y2="10"/>
35     </Condition>
36     <Condition Type="Timeout" Value="15"/>
37 </AbortConditions>
38 <Steps>
39     <Step Duration="10">
40         <Actions>
41             <Action Type="kick" PlayerNumber="1">
42                 <Parameter Min="0" Max="1"/>
43                 <Parameter Min="2.5" Max="5"/>
44                 <Reward Type="BallDistance" X="10" Y="10" Multiplier="0.5"/>
45             </Action>
46             <Action Type="run" PlayerNumber="2">
47                 <Parameter Value="10"/>
48                 <Parameter Value="3"/>
49                 <Reward Type="PlayerDistance" X="10" Y="10" Multiplier="0.25"/>
50             </Action>
51         </Actions>
52     </Step>
53 </Steps>
54 </Setplay>

```

Listing 5.1: Setplay Declaration

5.2 Setplay language parser

To use the setplays defined in the language previously presented, a parser is needed to translate the XML file into FCPGym environments and training scripts. For developing the parser, Python was chosen as it allows rapid development and comes with a built-in XML parser. After parsing the data from the setplay definition, the parser must generate two different pieces of code: FCPGym environments which must be written in C++ and a training script in Python. Thus, the code generation phase can be split into two parts.

For the generation of both the environment and the training script, a *templating* technique was used, where code that is shared between all possible setplays is stored in a template file. Then, when a new environment or training script is created, the code generated is placed in previously defined zones in the template file.

The process of generating code for the training script is simple, and only the following information and validation is needed:

- Name of the environment, which is the same as the name of the setplay;
- Number of environments that will be run in parallel;
- Number of players that will be used in each environment;
- Number of timesteps that the setplay will be trained for;
- The reinforcement learning algorithm to be used for training. The implementations of the algorithms used are from Stable Baselines[5] which is a fork from the original OpenAI Baselines commonly used by the FCPortugal team. The algorithm must be one implemented by Stable Baselines;
- The Policy Network is used to translate observations into actions. Once again, this network is provided by Stable Baselines, so the selected network must be supported by them.

However, the process does get trickier when generation code for the environments, as they are more complex. An environment can be split into four different parts/functions: initializing the environment, taking an observation from the environment, doing a step in the environment and resetting the environment. The function that does a step in the environment does not perform a step as defined in the setplay (which will be called stage from now on to avoid confusion) but instead a step as defined in Gym by the agent performing an action, taking a new environment observation, calculating the reward of the action performed and check if the episode is complete.

For the initialization of the environment, the following steps are done:

- The name of the setplay is encoded so that it may be used as the name of both a file and a C++ Class name;
- The types of each robot present in the setplay is declared;

- The action spaces are defined as in the setplay definition;
- The observation space is defined. It is the same for all setplays. For each observation, the following information is given as an input to the Policy Network for the network to learn to optimize the actions based on this information:
 - The uniform number of the player;
 - The current timestamp;
 - The current stage (step in the setplay declaration);
 - The position of the ball;
 - The velocity of the ball;
 - The position of the player;
- The ball and each player are moved to their initial positions, which may be set or random in a given region;
- A list of variables is initiated: the current timestamp, the current stage and the stage on the last timestamp (as some behaviours need to know if this is the first timestamp they are being executed);

The observation function is simple, as it only retrieves the information of the observations previously stated and returns it. The reset function is also straightforward as it resets the value of the current timestamps and current stage and returns the ball and the players to their original positions. The step function is where most of the code is generated and is divided as follows:

- The current timestamp is incremented, and the current stage is calculated based on the duration of each stage defined in the setplay. While in the setplay definition the duration does not contain any unit, seconds are assumed, which are converted into timesteps according to the time between each episode;
- The robot, depending on the current stage and on its uniform number, executes a behaviour and calculates the individual reward based on the reward type defined in the setplay. This is the main piece of code that is generated and can be broken down into the following:
 - First, a C++ switch is used to identify the piece of code to be executed by the robot by looking first at the current stage and then at the robot's uniform number.
 - Afterwards, the code that executes the behaviour is generated based on the type of behaviour being performed. The type of behaviour then needs to be translated into the actual code that the FCP agent will execute. For that, a map of the currently available behaviours was developed, which maps an identifier (the type that should be used in the setplay definition) of the behaviour to the code needed to execute that behaviour. Currently, there are 21 behaviours implemented (thirteen kicks and nine movement


```
19         case 2:
20             kickBehaviour(0, 0);
21             reward = -0.25 * distance(ballPos().X, ballPos().Y,
22                                     0, 0);
23             break;
24         default:
25             break;
26     }
27     break;
28 default:
29     break;
30 }
31 return reward;
32 }
```

Listing 5.2: Action Code Example

- The global reward conditions are checked. If any is true, the corresponding reward is added to the individual reward already calculated. For the Ball Position reward, the position of the ball is compared with the defined region. For the Goal reward, the position of the ball is compared to the goal's region;
- All the finish and abort conditions are checked. If any is true, the episode is terminated. The calculation done is the same as the one for the global reward, except for the Timeout condition, where the timestamp is simply compared to the timeout value;

After declaring the setplay and building both the environment and the training script, the setplay is ready to be trained.

Chapter 6

Evaluation

In order to evaluate the new setplay framework, a setplay was defined, built, trained and tested. Section 6.1 describes the defined setplay as well as the generated environment's observation and action spaces, rewards, finish and abort conditions, and explains the different steps of the setplay. In Section 6.2 the training process is described. Section 6.3 presents the results obtained by the trained setplay during the training process as well as against other teams. Finally Section 6.4 draws a conclusion from the results obtained.

6.1 Defined Setplay

In order to test the capabilities of the framework, a simple kickoff setplay was defined. The kickoff was the chosen setpiece as it is generally the one that happens the most during a game of robotic soccer and the one with higher direct scoring chances. In this kickoff, involving two players, the first passes the ball to a position near the goal while the other runs to that position and then kicks the ball into the goal. Listing 6.1 contains the definition of the setplay.

```
1 <?xml version="1.0"?>
2 <Setplay>
3   <Name>SimpleKickOff</Name>
4   <Description>One player passes the ball close to goal and another runs to the
      ball and tries to score</Description>
5   <TrainingParameters>
6     <Envs>2</Envs>
7     <Algorithm>PPO2</Algorithm>
8     <PolicyNetwork>MlpPolicy</PolicyNetwork>
9     <TimeSteps>15000000</TimeSteps>
10    <ActionSpace>
11      <Parameter Type="float" Min="-15" Max="15"/>
12      <Parameter Type="float" Min="-10" Max="10"/>
```

```

13     </ActionSpace>
14 </TrainingParameters>
15 <Ball>
16     <Pos X="0" Y="0"/>
17 </Ball>
18 <Players>
19     <Player Type="4">
20         <Pos X="-1" Y="0"/>
21     </Player>
22     <Player Type="4">
23         <Pos X="-1" Y="5"/>
24     </Player>
25 </Players>
26 <Rewards>
27     <Reward Type="Goal" Value="10000"/>
28     <Reward Type="BallPos" Value="-2500">
29         <Region X1="-15" X2="-1" Y1="-10" Y2="10"/>
30     </Reward>
31 </Rewards>
32 <FinishConditions>
33     <Condition Type="Goal"/>
34 </FinishConditions>
35 <AbortConditions>
36     <Condition Type="BallPos">
37         <Region X1="-15" X2="-1" Y1="-10" Y2="10"/>
38     </Condition>
39     <Condition Type="Timeout" Value="30"/>
40 </AbortConditions>
41 <Steps>
42     <Step Duration="10">
43         <Actions>
44             <Action Type="kick" PlayerNumber="1">
45                 <Parameter Min="9" Max="11"/>
46                 <Parameter Min="2.5" Max="3.5"/>
47                 <Reward Type="BallDistance" X="10" Y="3" Multiplier="0.5"/>
48             </Action>
49             <Action Type="run" PlayerNumber="2">
50                 <Parameter Value="10"/>
51                 <Parameter Value="3"/>
52                 <Reward Type="PlayerDistance" X="10" Y="10" Multiplier="0"/>
53             </Action>
54         </Actions>
55     </Step>
56     <Step Duration="10">
57         <Actions>
58             <Action Type="kick" PlayerNumber="2">
59                 <Parameter Min="15" Max="15.5"/>
60                 <Parameter Min="-1" Max="1"/>
61                 <Reward Type="BallDistance" X="15" Y="0" Multiplier="0.25"/>

```

```

62         </Action>
63     </Actions>
64 </Step>
65 </Steps>
66 </Setplay>

```

Listing 6.1: Defined Setplay

6.1.1 Observation & Action Spaces

The observation space of the environment generated for this setplay, as well as all others generated by the framework, is composed of: the uniform number of the robot, which ranges between 1 and the number of players; the timestamp, which ranges from 0 to the timeout value; the stage, which varies between 0 and the number of the stage; and the x, y, z coordinates of the ball, coordinates of the player and velocity (in each axis) of the ball. These are the inputs of the network being trained.

The action space contains only two actions as the behaviours used in the setplay have at most two parameters.

6.1.2 Global Rewards, Finish & Abort Conditions

As the intent of the setplay is to score a goal, the finish condition chosen was Goal. In the worst scenario, that the ball ends up in our teams' half, the setplay is aborted. Finally, each episode has a timeout of 30 seconds, which is more than the maximum time it may take for the first player to pass the ball and the second to run and shoot to goal.

As for the global rewards, the robots are collectively rewarded with a value of 10000 if they can score a goal and penalized with a reward of -2500 if the ball goes back to their team's half.

6.1.3 Steps

Initially, the ball is placed in the centre of the field (as it is on a kickoff in a game). One of the players starts near the ball, ready to pass it, and the other starts on the side of the field near the halfway line prepared to run to the place where the first player will pass the ball, as seen in Figure 6.1.

Afterwards, in the first stage, the robot close to the ball kicks it forward to a position that will be optimized (inside a user-defined region, declared as the minimum and maximum values of the kick parameters) and the other runs to the position where he expects the ball to go to. The first robot is negatively rewarded, for every timestamp of the stage, based on the distance that the ball is to the point (10,3). This is used to incentivize the robot to kick the ball somewhere near that position but still try to optimize the best kicking stop by using a bigger reward for goals. As for the second robot, because the setplay directly defines the position where it should run to, the reward multiplier is set to 0, as the parameters are not being optimized. Figure 6.2 shows an intermediate state of stage 1.

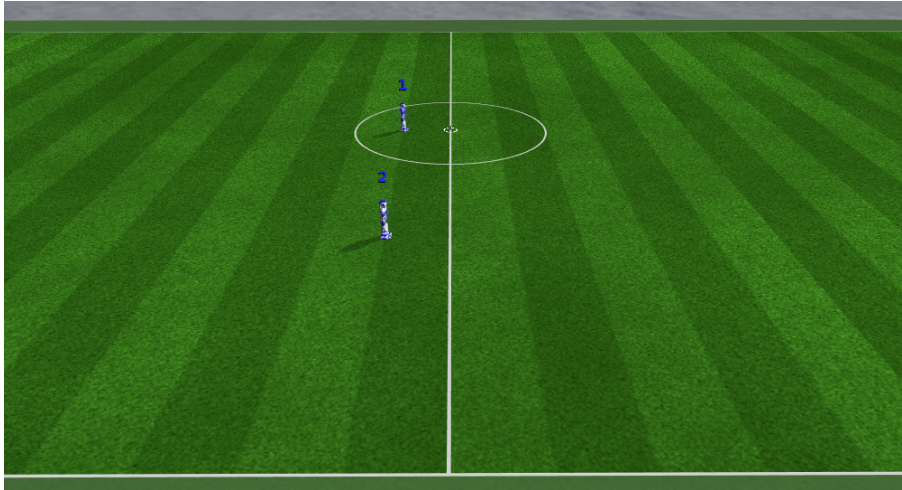


Figure 6.1: Setplay: Initial Player Positions.

In the second stage, the robot who passed the ball does not participate, and the other player tries to score by kicking the ball into the goal. The specific position in the goal that the player targets is not specified. Instead, it is optimized to be somewhere inside the goal during training. Once again, the robot is penalized by the distance from the ball’s position to the centre of the goal. This incentivizes the robot to kick the ball as fast as possible, and the global goal reward should help the robot learn the best place in the goal to aim for. Figure 6.3 displays the robot after kicking the ball in stage 2.

The setplay may end if the robots can score a goal or will eventually end when the timer reaches the defined timeout.

6.2 Training

To train the setplay, the parser was used to transform the setplay definition into a trainable FCP-Gym environment. A stable baselines[5] MlpPolicy, a neural network with two hidden layers of 64 nodes, was trained using the stable baselines’ implementation of the PPO2 reinforcement learning algorithm, a modified PPO algorithm made to take advantage of GPU capabilities. It was trained on two parallel environments for 15 million timesteps with default stable baselines PPO2 settings.

In order to train FCPGym environments, some settings of the robotic soccer simulator need to be changed. One of them allows the simulation to simulate events in the game faster than real-time speed, which helps speed up training. The other disables some of the soccer game rules, so the robot is free to train without being restricted by the simulation. While the robots must still obey normal physical restrictions, some other rules, such as moving the robot outside of the field if deemed incapacitated, aren’t enforced as they could mess up the training process.



Figure 6.2: Setplay: Stage 1.

6.3 Results

To start analyzing the results, let's first look at the training results. Figure 6.4 presents a graph with the reward of the episodes over the training process. The dark orange line represents the smoothed average reward, while the more transparent line shows the real reward obtained in each timestep.

From the graph, it is possible to see that overall the reward did not improve over time, which hints that the robots could not learn to optimize the behaviour parameters to score goals. To confirm this, a test was done where the setplay, first untrained and then with training, was run by 1000 episodes, and the number of goals scored by each was compared. The untrained setplay was able to score 165 goals, meaning it had an effectiveness rate of 16.5%. The trained setplay has slightly better results: 196 goals with a 19.6% effectiveness rate. While the results of the trained setplay are better than the untrained ones, the difference is small, and the overall results are still low given that there was no opposing team to try to stop the team from scoring.

One possible explanation for such low results was discovered during both the training process and the first tests. The robots would, sometimes, fall to the ground when executing a behaviour. This can be seen in Figure 6.5, where both robots fell at the start of the setplay. This impacted the effectiveness of the setplay, and the learning process as the robots sometimes were not able to perform the requested behaviours. This appears to happen to the first robot at the start of the setplay and to the second robot at the start and when it attempts to kick the ball into the goal.

While at least one of the robots did fall on a high number of episodes during training, if the robotic soccer simulator's setting that allows the events to happen faster than real-time was turned off, it did not occur as frequently. To test the impact of the setting, the trained setplay was once again tested for 1000 episodes while recording the number of goals scored. It managed to score



Figure 6.3: Setplay: Stage 2.

276 goals with an effectiveness rate of 27.6%. While this value presents an improvement, it is still fairly low.

For the final test of the effectiveness of the setplay, the setplay was tested against real teams that participated in the 2021 edition of RoboCup and compared with the results obtained by the FCPortugal team. While FCPGym does not directly support the addition of other teams, and so they were not able to be used for training because, among other reasons, the code for the other teams is not available, a workaround was used to run the teams binaries which connect to the robotic soccer simulator that FCPGym uses. For this workaround to work, the setting that enables the soccer rules must be turned on, once again making it impossible to use opposing teams while training the setplay. Furthermore, by turning the setting, which is active in real games, the effectiveness of the setplay may be reduced as, for example, in the kickoff, no player may pass the half-line before a member of the team touches the ball.

The binaries of the teams, as well as the 2021 results, were obtained from RoboCup archive¹. Figure 6.6 shows the setplay being tested against one of the RoboCup 2021 teams, Wrightocean3D. The results may be seen in Table 6.1. For each opposing team, the number of kickoffs that the trained setplay was tested against and the number of kickoffs FCPortugal had against them in the tournament, as well as the percentage from both which lead to a direct goal, are presented. However, against some teams, it was not possible to test the trained setplay even with the previously mentioned workaround.

The obtained results by the trained setplay are quite poor, especially when compared with the ones obtained by the FCPortugal team. Multiple reasons lead to these results:

- The defined setplay is too simple. It only uses two players and relatively simple behaviours. Comparatively, the setplay used by FCPortugal uses five players: one player kicks the ball

¹<https://archive.robocup.info/Soccer/Simulation/3D/>

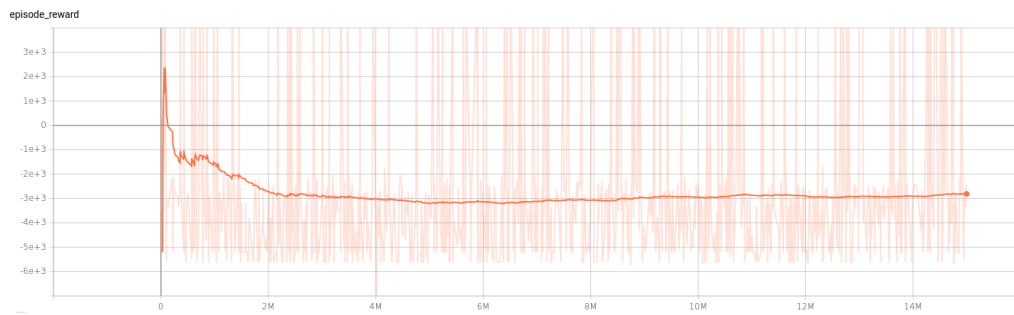


Figure 6.4: Setplay training episode rewards.

forward while four other players run fast in the direction of the ball and try to collide with it at an angle that will shoot the ball into the goal. It also uses some functionalities that are not yet present on the setplay framework: the initial position of the players and the position to where the ball is initially kicked is dynamic according to the opponents' position;

- As previously stated, the robots sometimes fall at the start of the setplay, which means that in some attempts, the setplay is unsuccessful from the start. To add to this, when the soccer rules setting is activated, the robot that is attempting to run may not pass the halfway line before the other robot has touched the ball, which is not a problem during training, but happens during an actual game which leads to the robot being reset to a backwards position if he crosses the line.

6.4 Conclusion

After taking into account the poor results obtained by the training of the defined setplay, we cannot conclude that the new setplay optimization framework can, by itself, be used to optimize robotic soccer setplays. However, the framework can be used to define setplays and translate them into FCPGym environments, which may then be changed to create and optimize better setplays. There are multiple factors that hampered the results, and that may be tackled in order to enhance them:

- The defined setplay was too simple. While this was done to facilitate the training process and provide results that could show that the optimization could improve any defined setplay, it came with the drawback that it did not achieve good results when tested against other teams. Furthermore, it leads to the conclusion that the most important part of implementing a setplay is defining a good setplay that will work against other teams, which requires experience and knowledge from the user;
- The concept of splitting the setplay into steps, while good for 2D robotic soccer, may not be the best approach for 3D robotic soccer, as robots should be able to decide to start a new behaviour without having to wait for other robots to finish a step. This is also supported by



Figure 6.5: Robots falling at the start of the setplay.

the fact that the behaviours used in the 2D league are usually simpler and less prone to error, which gives the setplay a higher degree of trust;

- The behaviours used were either not optimized enough to be used freely in a setplay in any scenario or were incorrectly used by the framework. This can be seen by the fact that the robots sometimes would fall when starting the setplay;
- The definition and training of an environment may be a process too complex to be solved by just being defined automatically and may need changes to be handmade by someone with knowledge of both the team and the league. A knowledgeable researcher can vastly improve the definition of rewards, observation and environment spaces on a setplay by setplay basis.



Figure 6.6: Setplay being tested against Wrightocean3D team.

Table 6.1: Trained and FCPortugal Setplay Results

Team	FCPortugal Goal %	FCPortugal Attempts	Trained Setplay Goal %	Trained Setplay Attempts
Apollo3D	6.67%	15	—	—
BahiaRT	0.00%	0	—	—
FCPortugal	—	—	1.00%	100
HFUTEngine2021	50.00%	4	—	—
ITAndroids	50.00%	2	—	—
KgpKubs	33.33%	3	2.00%	100
magmaOffenburg	0.00%	25	—	—
Miracle3D	50.00%	4	—	—
MIRG	0.00%	1	—	—
WITS-FC	0.00%	1	8.00%	100
Wrightocean3D	66.67%	6	5.00%	100

Chapter 7

Conclusion

In this dissertation, we explore the use of machine learning to optimize setplays for robotic soccer. After giving background on robotic soccer, examining the state of the art in machine learning for robotic soccer and analyzing the previous work done on the topic, we proposed a new setplay optimization framework that may be used to define, train and test setplays for the FCPortugal team. Furthermore, we created and optimized a simple setplay to understand if the framework could be used to define setplays that could work in real matches. Taking into account the results obtained, we could not prove that the framework, by itself, is able to optimize a setplay to be used competitively. However, it can be used as a starting point to create new setplays and translate them into reinforcement learning training environments that may then be changed to use the already developed behaviours to their maximum. We found that at this stage, the main task of creating good setplays continues to be their definition by someone knowledgeable of the team and the league. Still, this work does come up with some contributions, detailed in Section 7.1, and proposes future work, in Section 7.2, that may be developed on this topic.

7.1 Main Contributions

The main contributions of this dissertation are:

- A setplay optimization definition language that allows the definition of setplays to be optimized using reinforcement learning;
- A parser to translate setplays defined in the language mentioned above into trainable FCPGym environments that can later be edited to improve the effectiveness of training and the results obtained by the setplay;
- A small study into how the optimization may improve the effectiveness of simple setplays, including the comparison of the results obtained by an untrained and an optimized setplay, by themselves and against multiple other teams;
- An automatic mechanism to test setplays defined in FCPGym (and possibly, other FCPGym environments) against other teams that regularly compete against FCPortugal.

7.2 Future Work

Even though the framework presented in this dissertation is ready to be used, there are still multiple improvements that can be made to enhance the results obtained by its use. Among them are:

- Improvements into the definition of steps in the setplay could provide better results, as in such a complex environment, the robots should not have to wait before the next step starts in order to execute a new behaviour;
- If possible, the addition of opposing players during training of FCPGym environments could provide a good opportunity to allow the players to optimize the setplay against opponents and even learn how best to use the setplay against each specific opponent;
- A more complex and robust way to define observation and action space and to better utilize rewards will improve the optimization possible for a given setplay;
- A better study of the capabilities of the current framework by utilizing a more complex setplay;
- The addition of a graphical user interface to facilitate the process of defining setplays, in the same way one exists for the FCPortugal 2D framework.

References

- [1] A. Abdolmaleki, D. Simões, N. Lau, L. P. Reis, and G. Neumann. Learning a Humanoid Kick with Controlled Distance. In *RoboCup 2016: Robot World Cup XX*, pages 45–57. Springer International Publishing, 2017.
- [2] M. Abreu, L. P. Reis, and N. Lau. Learning to Run Faster in a Humanoid Robot Soccer Environment Through Reinforcement Learning. In *RoboCup 2019: Robot World Cup XXIII*, pages 3–15. Springer International Publishing, 2019.
- [3] P. Amaro, L. P. Reis, and A. J. Sousa. Multi-Robot Learning of High-Level Skills in RoboCup. Master’s thesis, Faculdade de Engenharia da Universidade do Porto, 2019.
- [4] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight Experience Replay, 2018.
- [5] Stable Baselines. Stable Baselines Docs. <https://stable-baselines.readthedocs.io/en/master/>, 2018. Accessed: 2021-09-13.
- [6] M. G. Bellemare, W. Dabney, and R. Munos. A Distributional Perspective on Reinforcement Learning, 2017.
- [7] F. Bre, J. Gimenez, and V. Fachinotti. Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *Energy and Buildings*, 158, 2017.
- [8] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.
- [9] J. Cravo, L. P. Reis, and F. Almeida. SPlanner. Strategy Planner: Graphical definition of soccer set-plays. Master’s thesis, Faculdade de Engenharia da Universidade do Porto, 2011.
- [10] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional Reinforcement Learning with Quantile Regression, 2017.
- [11] Premier Skills English. Football Shout-out: Set-Pieces. <https://premierskillsenglish.britishcouncil.org/skills/listen/football-shout-out-set-pieces>, 2021. Accessed: 2021-01-29.
- [12] RoboCup Federation. A Brief History of RoboCup. https://www.robocup.org/a_brief_history_of_robocup, 2016. Accessed: 2021-01-25.
- [13] RoboCup Federation. Objective. <https://www.robocup.org/objective>, 2016. Accessed: 2021-01-25.

- [14] RoboCup Federation. RoboCup Federation official website. <https://www.robocup.org/>, 2016. Accessed: 2021-01-25.
- [15] RoboCup Federation. RoboCupSoccer. <https://www.robocup.org/domains/1>, 2016. Accessed: 2021-01-25.
- [16] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning, 2018.
- [17] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual Multi-Agent Policy Gradients, 2017.
- [18] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018.
- [19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.
- [20] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [21] J. Schulman and S. Levine and P. Moritz and M. I. Jordan and P. Abbeel. Trust Region Policy Optimization, 2017.
- [22] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *J. Artif. Int. Res.*, 4(1):237–285, 1996.
- [23] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The Robot World Cup Initiative, 1995.
- [24] J. Kober. *Robot Learning*, pages 1–9. Springer London, 2019.
- [25] RoboCupSoccer Simulation League. 3D History. <https://ssim.robocup.org/3d-simulation/3d-history/>. Accessed: 2021-01-26.
- [26] RoboCupSoccer Simulation League. 3D Rules. <https://ssim.robocup.org/3d-simulation/3d-rules/>. Accessed: 2021-01-26.
- [27] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning.
- [29] R. Lopes, L. P. Reis, and N. Lau. Coordination Methodologies applied to RoboCup: a Graphical Definition of Setplays. Master’s thesis, Faculdade de Engenharia da Universidade do Porto, 2009.
- [30] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments, 2020.
- [31] magmaOffenburg. RoboViz. <https://github.com/magmaOffenburg/RoboViz>, 2020. Accessed: 2021-01-26.

- [32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.
- [33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Daan, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning, 2013.
- [34] M. Mohammadi and A. Al-Fuqaha. Enabling Cognitive Smart Cities Using Big Data and Machine Learning: Approaches and Challenges. *IEEE Communications Magazine*, 56(2):94–101, 2018.
- [35] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.
- [36] L. Mota, J. A. Fabro, L. P. Reis, and N. Lau. Collaborative Behavior in Soccer: The Setplay Free Software Framework. In *RoboCup*, 2014.
- [37] L. Mota and L. P. Reis. Setplays: Achieving coordination by the appropriate use of arbitrary pre-defined flexible plans and inter-robot communication. In *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination*, 2007.
- [38] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, 2018.
- [39] OpenAI. OpenAI Gym Documentation. <https://gym.openai.com/docs/>. Accessed: 2021-09-11.
- [40] OpenAI. OpenAI Gym Environments. <https://gym.openai.com/envs/>. Accessed: 2021-09-11.
- [41] OpenAI. Kinds of RL Algorithms. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html, 2018. Accessed: 2021-01-28.
- [42] OpenAI. Baselines. <https://github.com/openai/baselines>, 2020. Accessed: 2021-09-11.
- [43] J. Peters, R. Tedrake, N. Roy, and J. Morimoto. *Robot Learning*, pages 865–869. Springer US, 2010.
- [44] L. P. Reis. FC Portugal. <https://web.fe.up.pt/~lpreis/FCPortugal.htm>, 2006. Accessed: 2021-01-25.
- [45] L. P. Reis and N. Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 29–40, 2000.
- [46] L. P. Reis and N. Lau. Coach unilang - a standard language for coaching a (robo) soccer team. In *RoboCup 2001: Robot Soccer World Cup V*, pages 183–192. Springer Berlin Heidelberg, 2002.
- [47] SoftBank Robotics. NAO the humanoid and programmable robot. <https://www.softbankrobotics.com/emea/en/nao>, 2016. Accessed: 2021-01-25.
- [48] Festival Nacional de Robótica. Festival Nacional de Robótica 2021. <https://www.festivalnacionalrobotica.pt/>, 2021. Accessed: 2021-09-03.

- [49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.
- [50] N. Shafii, N. Lau, and L. P. 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.
- [51] T. Silva, L. P. Reis, and A. J. Sousa. Humanoid Low-Level Skills using Machine Learning for RoboCup. Master’s thesis, Faculdade de Engenharia da Universidade do Porto, 2019.
- [52] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017.
- [53] D. Silver, T. Hubert, J. Schrittwieser, and D. Hassabis. Reinforcement Learning: Deep Q Networks. <https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go>, 2018. Accessed: 2021-01-28.
- [54] D. Simões, N. Lau, and L. P. Reis. *Learning Coordination in Multi-Agent Systems*. PhD thesis, Universidade de Aveiro, 2019.
- [55] P. Singh. Reinforcement Learning: Deep Q Networks. <https://blogs.oracle.com/datascience/reinforcement-learning-deep-q-networks>, 2020. Accessed: 2021-01-28.
- [56] Stefan Glaser. Models. <https://gitlab.com/robocup-sim/SimSpark/-/wikis/Models>, 2017. Accessed: 2021-01-26.
- [57] Stefan Glaser. Soccer Simulation. <https://gitlab.com/robocup-sim/SimSpark/-/wikis/Soccer-Simulation>, 2017. Accessed: 2021-01-26.
- [58] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning, 2017.
- [59] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.
- [60] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, and R. Vicente. Multi-agent Cooperation and Competition with Deep Reinforcement Learning, 2015.
- [61] H. Teixeira, T. Silva, M. Abreu, and L. P. Reis. Humanoid Robot Kick in Motion Ability for Playing Robotic Soccer. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 34–39, 2020.
- [62] P. A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265, 2016.
- [63] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning, 2018.
- [64] L. Weng. Policy Gradient Algorithms. lilianweng.github.io/lil-log, 2018.

- [65] Y. Xu and H. Vatankhah. SimSpark: An Open Source Robot Simulator Developed by the RoboCup Community. In *RoboCup 2013: Robot World Cup XVII*, pages 632–639. Springer Berlin Heidelberg, 2014.