FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FCPortugal - Machine Learning for a Flexible Kicking Robotic Soccer Skill

Henrique Martins

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Paulo Reis

October 31, 2023

Resumo

O objetivo desta dissertação é investigar e implementar técnicas de remate eficientes e precisas compatíveis com a nova abordagem de dribles desenvolvida pela equipa FC Portugal. Devido a uma recente alteração nas regras que limitou o método de dribles mais eficaz da equipa, foi necessário reotimizar a estratégia de dribles. Para enfrentar o desafio de criar remates poderosos após o drible, foram implementadas técnicas de machine learning.

Foi utilizado o método de Otimização de Política Próxima (PPO) de machine learning para desenvolver esses remates rápidos. Além disso, vários outros algoritmos de Deep Reinforcement Learning foram examinados e comparados. A dissertação apresenta uma análise abrangente dos algoritmos de machine learning, apresentando o estado-da-arte na área.

Ao longo da pesquisa, foram estudadas 16 fases de 2 novas técnicas de dribles com o objetivo de determinar qual etapa proporcionaria a maior eficiência para um remate em movimento. Após este estudo, os remates originados nas fases mais promissoras, estudadas anteriormente, foram otimizados. Os remates desenvolvidos demonstram alta eficiência, precisão e, em comparação com os remates comummente utilizados pela equipa FC Portugal, são muito mais rápidos. Isso abre portas para melhorias no desempenho da equipa na liga de simulação 3D da RoboCup.

Abstract

The objective of this dissertation is to investigate and implement efficient and precise shooting techniques compatible with the new dribbling approach developed by the FC Portugal team. Due to a recent rule change that restricted the team's most effective dribbling method, they had to re-optimize their dribbling strategy. To address the challenge of creating powerful shots after dribbling, machine learning techniques were implemented.

The Proximal Policy Optimization (PPO) method of Reinforcement Learning was employed to develop these rapid shots. Additionally, various other Deep Reinforcement Learning algorithms were examined and compared. The dissertation provides a comprehensive analysis of Reinforcement Learning algorithms, presenting the state-of-the-art in the field.

Throughout the research, 16 phases of 2 new dribbling techniques were studied to determine which phase would offer the most efficiency for a shot in motion. Following this study, the shots originating from the most promising phases studied earlier were optimized. The developed shots demonstrate high efficiency, accuracy, and, compared to the commonly used shots by the FC Portugal team, are much faster. This opens the door to improved performances by the team in the RoboCup simulation 3D league.

Agradecimentos

Agradeço ao meu orientador, Luís Paulo Reis, pela orientação valiosa e pelo constante apoio ao longo deste trabalho. A sua dedicação e conhecimento foram fundamentais para o sucesso deste projeto.

Gostaria também de expressar minha gratidão ao Miguel Abreu, que pacientemente esclareceu todas as minhas dúvidas, forneceu assistência inestimável e orientou-me de maneira excepcional ao longo desta jornada.

À minha irmã, Gabriela, quero expressar minha profunda gratidão por sua ajuda inestimável e pelo apoio constante. Ela é verdadeiramente a melhor irmã do mundo, e seu encorajamento foi um grande impulso.

À minha mãe, ao meu pai, à minha família e aos meus amigos e à Francisca, quero expressar meu sincero agradecimento. O apoio de vocês foi fundamental durante todo o processo, e não teria conseguido sem o incentivo e o amor de vocês.

Obrigado a todos por fazerem parte desta jornada e por tornarem este trabalho possível.

Henrique Martins

"The key is in not spending time, but in investing it."
Stephen R. Covey

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

AI	Artificial Intelligence
RoboCup	Robot Soccer World Cup
ML	Machine Learning
ANNs	Artificial neural networks
RL	Reinforcement Learning
MDP	Markov decision process
SARSA	State-Action-Reward-State-Action
DQN	Deep Q-(Learning) Network
DDPG	Deep Deterministic Policy Gradients
PL	Policy Gradient
TRPO	Trust Proximal Policy Optimization
PPO	Proximal Policy Optimization
AC	Actor-Critic
A3C	Asynchronous Advantage Actor Critic
A2C	Advantage Actor critic
DFDL	Double Footed Dribble Long
LFDL	Left Footed Dribble Long

Chapter 1

Introduction

Robotics and artificial intelligence (AI) have seen remarkable advancements, fueling significant interest within the scientific community. Initiatives like RoboCup have played a crucial role in driving research in these fields.

RoboCup, a prestigious international research and education endeavor, serves as a catalyst for pushing the boundaries of robotics and AI research through soccer-related challenges. This initiative comprises a robotics competition, featuring a simulation league with a 3D sub-league where teams showcase their skills, competing in leagues and competitions, with humanoid robots engaging in simulated soccer matches. The primary aim of this league is to push the boundaries of algorithm development, empowering these virtual robots to showcase coordination, cooperation, and learning abilities similar to those of a real soccer team [7],[8].

One such enthusiastic participant in RoboCup is the FCPortugal3D team, a collaborative effort between the Universities of Porto and Aveiro. They actively engage in this prestigious global championship for robotic soccer on an annual basis.

This dissertation specifically addresses the challenge of creating, implementing, and evaluating a learning engine for robot action learning. The primary goal is to design a deep neural network-based learning engine capable of teaching a single robot the skill of "shot after dribble.'"

1.1 Context

RoboCup [7] is a prestigious international research and education initiative that pushes the boundaries of robotics and artificial intelligence research through soccer-related challenges. It was established in 1997 with the ambitious goal of creating a team of robots capable of defeating the human soccer World Cup champions by 2050 [9].

The initiative consists of several leagues, including the simulation league with a 3D sub-league. In this league, matches are held between two teams of 11 humanoid robots each. These robots act as autonomous agents and use a 3D soccer simulation alongside NAO robots to explore and demonstrate cooperation, coordination, and individual as well as collective learning of soccer skills. The primary objective is to score goals while effectively avoiding conceding them [8].

Beyond the competitive aspect, RoboCup serves as a platform for advancing the forefront of robotics and AI research, fostering international collaboration among researchers and students, and inspiring the next generation of researchers and engineers [9].

1.2 Motivation

The field of robotic soccer and the participating in RoboCup competitions motivates researchers to develop innovative approaches to enhance robot performance, provides a dynamic and challenging environment for research in machine learning and pushes the boundaries of robotics and artificial intelligence.

RoboCup's objective of creating a team of robots capable of defeating human soccer World Cup champions by 2050 serves as a driving force for continuous improvement. The evolving competition rules encourage researchers to refine their methods and algorithms, leading to more realistic and sophisticated robot behaviors. This constant evolution creates a fertile ground for exploring advanced techniques and cutting-edge approaches in robotic decision-making and coordination [9].

At the core of a robot's behavior are low-level skills, such as walking, running, shooting, and standing up. These foundational skills have reached a high level of efficiency, enabling researchers to focus on more complex implementations and behaviors. The integration of deep neural networks and learning engines opens up new possibilities for teaching robots complex skills, such as "dribble and shot," in a more adaptive and intelligent manner [10].

Additionally, the use of simulation leagues, including the 3D Simulation League, provides several advantages over real robots. It allows researchers to abstract away from hardware-related issues and focus on algorithm development and performance improvements. The ability to run numerous simulations and conduct experiments in a controlled and reproducible environment accelerates progress and facilitates the exploration of various learning approaches.

By addressing the challenge of developing a learning engine for robot action learning in the context of the RoboCup 3D Simulation League, this research aims to contribute to the advancement of robot learning, skill acquisition, and performance optimization. Moreover, the insights gained from this work have broader implications for the field of robotics, enabling the development of more intelligent and capable autonomous systems in diverse real-world applications beyond robotic soccer.

1.3 Objectives

In the 2023 RoboCup competition, significant rule changes were introduced, rendering the most commonly used dribbling behavior of the FC Portugal team ineffective. As a response, the team members collaborated to develop a new dribbling behavior to adapt to the altered rules. Furthermore, the existing versions of the team's "kick" behavior showed some limitations, although they

are quite accurate, they are performed somewhat slowly, and do not allow a fluid transition into the new dribbling behavior, having to change their speed and even stop to make the shot.

The primary objective of this dissertation is to enhance the transition from dribbling to kicking, enabling the agent to transition from the different variables of the new dribble to a kick without the need to stop or change its velocity. The performance of the kick must be in the fastest way possible, while understanding which variable of the dribble allows for a better and faster kick. This agile and continuous kicking behavior is considered a "kick in motion" after implementing the new dribble behavior. By achieving this, the agent can perform in a more human-like manner on the field, significantly improving its overall performance.

The importance of this behavior lies in its impact on both attacking and defending strategies. When attacking, stopping to perform a kick consumes valuable time, giving the defending team an opportunity to intercept the ball or reposition their agents. On the other hand, during defensive plays, a quick removal of the ball from an area of relative danger is crucial, making time improvements critical. Therefore, developing this behavior has the potential to make the play style of the FC Portugal 3D team much more dynamic, providing a substantial competitive advantage.

1.4 Thesis Structure

In addition to the introduction, this dissertation comprises 7 additional chapters:

- Chapter 2 delves into the background and fundamental aspects;
- Chapter 3 presents a review of the bibliography;
- Chapter 4 focuses on an analysis of the phases following the initial dribble after the shot to determine under what circumstances the best shot is executed;
- Chapter 5 showcases the results of optimizing the best phases identified in the analysis of the previous chapter;
- Chapter 6 examines the phases of the second dribble following the shot to understand the circumstances in which the best shot is achieved;
- Chapter 7 presents the results of optimizing the best phases identified in the analysis of the previous chapter;
- In Chapter 8, the main conclusions regarding my comprehension of the problem and future work are elucidated.

Chapter 2

Background and Fundamental Aspects

2.1 Machine Learning

To establish a foundation for the specific themes discussed in this dissertation, it is essential to provide an overview of machine learning (ML) and its relevance in the context of RoboCup. This chapter presents a concise summary of the different areas and sub-areas of AI focused on in this research.

Machine learning is a subset of artificial intelligence that involves training algorithms to recognize patterns and make predictions or decisions based on data. The process involves feeding machine learning algorithms with a dataset containing input data and their corresponding correct output. The ultimate objective of machine learning is to learn a function capable of effectively mapping the input data to the correct output [11].

In the present era, the influence of machine learning technology permeates numerous facets of contemporary society [12]. Its intricate algorithms play pivotal roles in tasks as diverse as engine otimization [13], manipulating genetic information [14], facilitating interactions on social networks [15], managing voluminous data within the banking sector [16], and even contributing to the domains of medical diagnostics and research [17]. Collectively, these remarkable strides underscore the profound significance of machine learning in shaping our lives.

2.2 Deep Learning and Neural Networks

Deep learning, a branch of machine learning, involves training artificial neural networks (ANNs) using extensive datasets to produce various outputs depending on the specific task, such as classification labels, generated text, predicted values, or autonomous decisions, across a wide range of applications [12].

These neural networks are structured with layers of interconnected nodes, taking inspiration from the organization of the human brain [18]. Artificial neural networks consist of layers of nodes, including an input layer, one or more hidden layers, and an output layer, as shown in Figure 2.1. Each node, or artificial neuron, is connected to others and is associated with a weight

and threshold. When the output of an individual node exceeds the specified threshold value, the node becomes activated, transmitting data to the next layer of the network. Conversely, if the output does not surpass the threshold, no data is forwarded to the subsequent layer. ANNs learn through training data to continually enhance their accuracy over time [19].

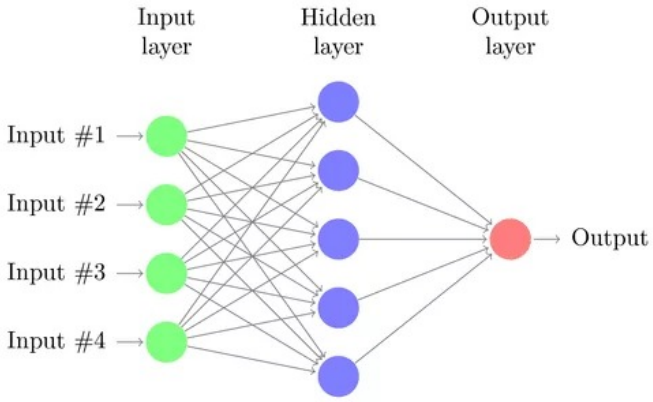

Figure 2.1: Example of an artificial neural network.

2.3 Reinforcement Learning

While deep learning implies large networks of neurons that can learn to recognize patterns in a big dataset, reinforcement learning (RL) teaches computers how to learn from experience [20].

Reinforcement learning is a form of machine learning that focuses on training an agent to make sequential decisions within an environment to maximize a reward signal. The agent learns by interacting with the environment and receiving feedback in the form of rewards or penalties for its actions [21].

In reinforcement learning, the problem is usually modeled as a Markov decision process (MDP), which consists of states, actions, transitions between states, and a reward function definition [22].

The reinforcement learning procedure, based on the MDP, is illustrated in Figure 2.2. It can be defined as follows:

- **Agent**:

  An agent is a software or physical entity capable of learning and making decisions within an environment with the primary goal of maximizing a reward signal. The agent learns through interactions with the environment, receiving feedback in the form of rewards or penalties based on its actions.

- **Environment (E)**:

The environment is the space in which the agent operates and makes decisions. It represents the task or problem that the agent is trying to solve. The environment provides the agent with the necessary information and feedback to learn and make informed decisions.

- **Action (A)**:

  An action is a decision made by the agent in a given state, which impacts the environment in some way.

- **Action Space**:

  The action space is the finite set of all possible actions that the agent can take.

- **State (S)**:

  A state is a representation of the current situation or condition of the environment.

- **State Space**:

  The state space is the set of all possible states that the agent can encounter while trying to reach the goal.

- **Reward (R)**:

  A reward is an immediate feedback given to the agent when it performs a specific action or task. The reward signal guides the learning process and helps the agent improve its decision-making over time.

- **Episode**:

  An episode refers to the entire cycle of actions and states the agent takes, starting from the initial state and ending in the terminal state.

- **Policy π(s)**:

  A policy is a function that determines the action that an agent should take in a given state. The policy is learned by the agent through interaction with the environment and is used to map states to actions to maximize the reward signal.

- **Value**:

  The value represents the expected long-term return with discount, compared to the short-term reward.

- **Value Function Vπ(s)**:

  The value function Vπ(s) specifies the value of a state, which is the total amount of reward an agent should expect to receive starting from that state, and following the policy π.

- **Discount Factor γ**:

The discount factor γ is a number between 0 and 1, which balances the importance of future rewards in defining a policy at a given state. It discounts future rewards to update the policy since they are subject to uncertainty.

- **Explore**:

  To explore all possible actions and sequences of steps that can lead to the same goal, the agent generally takes random actions. This helps the agent explore the entire environment, discover better action sequences, and maximize the reward.

- **Exploit**:

  Exploitation is the opposite of exploration, where the agent takes the best-known action given a particular state according to the learned policy.

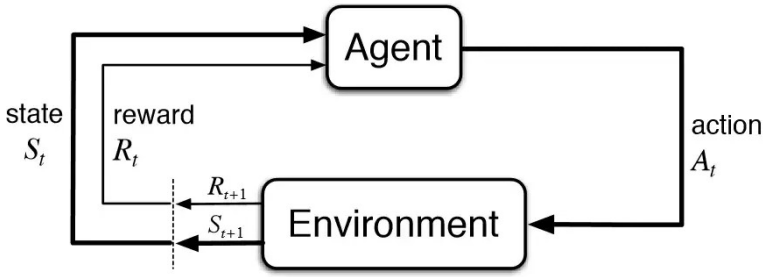

Figure 2.2: The procedure within reinforcement learning [1]

2.3.1 Discrete and Continuous Actions and States Spaces

When creating action and state spaces, we have the option to choose between discrete or continuous frameworks.

In a discrete state space, states are limited and can be listed, offering a clear and definite set of possibilities. In contrast, a continuous state space allows for an endless range of values within a specified interval, providing more adaptability and accommodating subtle variations.

Likewise, action spaces can be either discrete or continuous. Discrete action spaces consist of a finite or countably infinite set of actions, offering a straightforward and well-defined collection of choices. On the other hand, continuous action spaces cover an uncountable infinite set, granting an infinite array of possible actions within a given range. Continuous actions provide precise and detailed control, but they may demand more computational resources.

The decision to use discrete or continuous action and state spaces relies on the specific problem and the desired level of detail. While continuous spaces can present challenges and have computational consequences, they also provide enhanced flexibility, which is often crucial for accurately representing and handling complex tasks [23].

2.3.2 Deterministic and Stochastic Policies

Deterministic policies consistently select a predefined action for a particular state, whereas stochastic policies rely on the creation of action probabilities and choose the optimal action based on these probabilities for a given state [24].

Deterministic policies are simpler and computationally efficient, as they don't require calculating action probabilities. However, they lack flexibility, limiting exploration and potentially leading to suboptimal choices[24].

Stochastic policies offer flexibility by considering multiple actions for a state, allowing exploration and the discovery of better alternatives. Yet, implementing stochastic policies can be more challenging, involving the determination of action probabilities and potentially requiring additional computational resources[25]

In conclusion, the choice between deterministic and stochastic policies depends on specific requirements. Deterministic policies prioritize simplicity and efficiency, while stochastic policies offer increased flexibility and exploration potential despite increased complexity and computational demands.

2.4 Deep Reinforcement Learning

Deep reinforcement learning is the combination of reinforcement learning and deep learning that enables agents to learn and make decisions in complex environments by maximizing cumulative rewards[26]. In this paradigm, neural networks are instrumental in implementing various facets of the reinforcement learning process, including the agent's policy and value function[27].

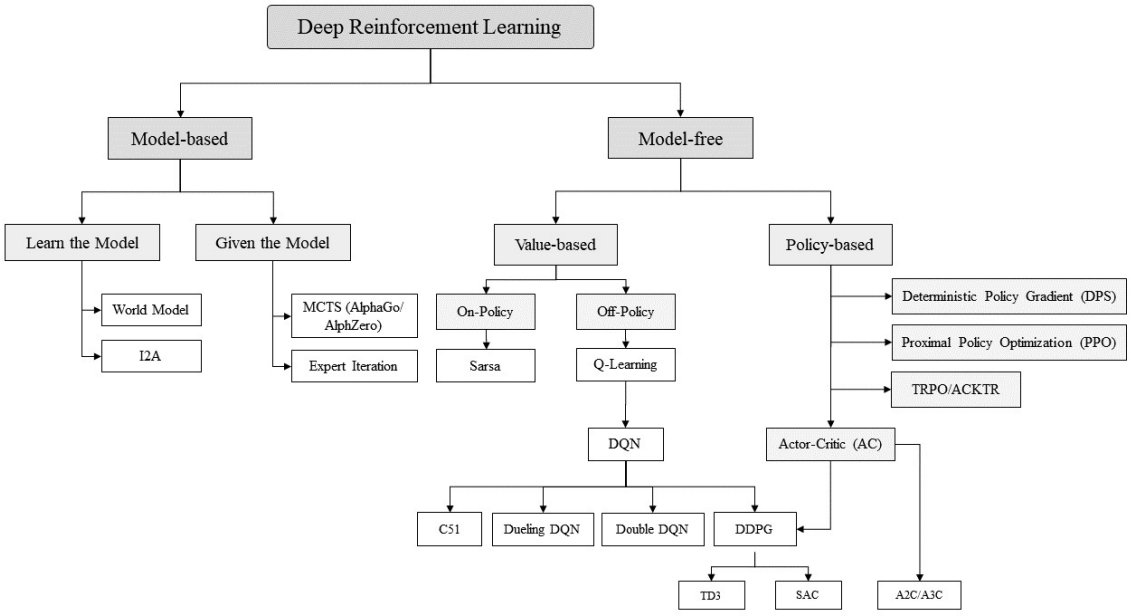

Figure 2.3: Taxonomy of RL algorithms [2]

As depicted in Figure 2.3, the taxonomy of reinforcement learning algorithms is divided into model-based methods, model-free methods, and within the model-free, it is further subdivided into value-based methods, policy-based methods, or the combination of both.

2.4.1 Model-based and Model-free

Model-based learning uses a mathematical model to represent relationships between data and predictions. It's trained on examples to find the best-fitting model for accurate future predictions.

In contrast, model-free learning doesn't rely on an explicit model. Instead, it learns from experience, adapting to new situations. It uses data to develop rules or heuristics for decision-making without depending on a specific model for predictions [1].

2.4.1.1 Model-free: Value-based and Policy-based

In value-based methods, we estimate and optimize the value function in order to achieve the best cumulative reward from a state or state-action pair [28]. We aim to discover the most rewarding actions without directly learning how to perform a task. We do this by repeatedly evaluating which actions give us the most rewards in different situations. Then, we use this information to create a strategy that selects actions with the highest expected rewards [29].

On the other hand, policy-based methods directly learn the best policy without explicitly estimating the value function [28]. Instead of estimating values, we adjust a set of rules or parameters to find the most effective way to do the task. We use special techniques to make these adjustments based on the rewards we receive, gradually getting better at the task [30].

2.4.1.2 Value-based: On and Off policies

On-policy methods aim to evaluate and refining the policy that is directly responsible for decision-making. This implies that the policy used to interact with the environment is the same one being improved. In practical terms, this approach involves the agent actively engaging with the environment to gather the needed data for policy evaluation and learning[27].

On the other hand, off-policy methods do not require that the policy employed for evaluation and improvement is identical to the one that collects data. This means that experiences gathered by other agents interacting with the environment can also be leveraged to enhance the policy [27].

2.5 Deep Reinforcement Learning Algorithms

Here are some examples of Deep Reinforcement Learning Algorithms [31, 27]:

2.5.1 State-Action-Reward-State-Action (SARSA)

SARSA is an On-policy reinforcement learning algorithm, which means it learns and improves the policy (the decision-making strategy) that it's currently using. It focuses on learning from

its own actions and interactions with the environment, making it a somewhat cautious learner. SARSA uses the Q-function to estimate how much reward an agent can get by taking a certain action "a" in a specific state "s" and then following a policy for future actions. It continuously updates its Q-values (representing the quality of the action "a" in state "s") using observed rewards and state changes, following the SARSA update rule. SARSA keeps using its current policy to decide actions and adjusts its Q-values accordingly as it learns.This algorithm is categorized as value-based and is commonly used in reinforcement learning applications [32, 1].

2.5.2 Q-Learning (QL) and Deep Q-Network (DQN)

Unlike SARSA, Q-learning doesn't necessarily follow its current policy when selecting actions during the learning process. It often explores actions that may not be optimal according to the current policy [33].

The action chosen by Q-learning is based on the maximum Q-value for the next state, regardless of whether it's consistent with the current policy. This exploration can lead to discovering better policies. Q-learning is more exploratory in nature and is willing to take risks to find the optimal policy. It's suitable for situations where the agent can afford to take suboptimal actions during learning. It can converge to the optimal policy even if the exploration policy is different from the target optimal policy [34].

In Q-learning, the Q-function is also used to estimate the expected cumulative reward, but it does so in a slightly different way compared to SARSA. Q-learning updates its Q-values based on the maximum Q-value for the next state, regardless of the action actually taken. The Q-value for a state-action pair (Q(s, a)) is updated using the Q-learning update rule[33].

A variant of Q-learning known as Deep Q-Network leverages a neural network to approximate the Q-function by taking the environment state as input, the neural network calculates estimated Q-values for all potential actions. The action with the highest expected return is then selected using these Q-values. Additionally, DQN employs a deep neural network to approximate the Q-value function in Q-learning and includes an experience replay buffer to store transition samples acquired during agent-environment interactions [27].

2.5.3 Policy Gradient (PG)

Policy gradient (Policy-based) methods belong to a category of reinforcement learning techniques that focus on optimizing parameterized policies concerning the expected return, which represents the long-term cumulative reward. This optimization process is achieved through gradient descent [27].

The fundamental concept behind policy gradient methods is to enhance the probability of actions leading to high expected return while reducing the probability of actions associated with low expected return. This is accomplished by computing the gradient of the expected return concerning the policy parameters and using this gradient to update the policy parameters in a way that maximizes the expected return.

PG methods prove to be particularly well-suited for scenarios involving high-dimensional or continuous action spaces [35].

2.5.4 Actor-Critic Methods (AC)

There exists a hybrid actor-critic approach that combines both value functions and policy search in reinforcement learning, known as actor-critic approaches [36]. In this approach, the "Critic" is responsible for estimating the value function, which can be either the action-value (Q value) or the state-value (V value). On the other hand, the "Actor" takes guidance from the Critic, typically through policy gradients, to update the policy distribution [3].

In the actor-critic framework (as depicted in figure 2.4), the actor (policy) receives a state from the environment and selects an action to perform. Meanwhile, the critic (value function) receives the state and reward resulting from the previous interaction. Using the information from this interaction, the critic calculates the TD (Temporal Difference) error and utilizes it to update both itself and the actor's policy [3].

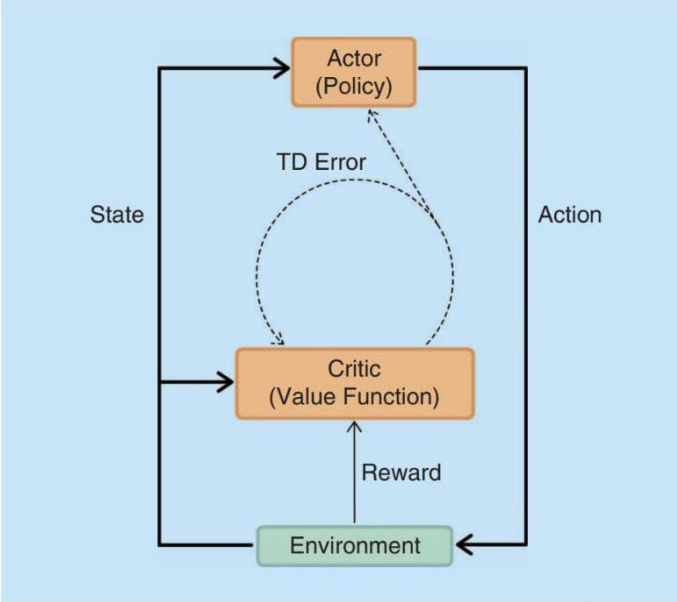

Figure 2.4: The actor-critic setup [3]

2.5.4.1 Asynchronous Advantage Actor Critic (A3C)

In A3C, multiple agent processes are run in parallel, each with its own actor and critic network. These processes communicate with each other asynchronously, meaning that they do not have to wait for one another to complete their computations before moving on to the next step. This allows for faster training and better utilization of resources [37].

One of the key features of A3C is that it uses an advantage function, which helps the agent learn more efficiently by taking into account the relative value of different actions. This is in contrast to traditional value-based methods, which only consider the overall value of a state [38].

2.5.4.2 Advantage Actor critic (A2C)

The contrast between A3C and A2C lies in their synchronization approach. In A3C, multiple independent agents create distinct copies of the model, while in A2C, each step model sends data back to the training model synchronously. This synchronous data exchange in A2C leads to significant resource savings compared to A3C, all while achieving similar performance levels [39].

2.5.5 Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradient is an algorithm which concurrently learns a Q-function and a policy [40]. It merges the DQN and actor-critic (Policy-based) approaches to update policies using deterministic policy gradients through deep learning. It introduces target networks for the actor and critic components, enabling more efficient learning from samples. However, DDPG is known for its fragility and sensitivity to hyperparameters, making it challenging to use effectively in practice [41].

2.5.6 Trust Proximal Policy Optimization (TRPO)

TRPO is a reinforcement learning method that enhances policies by taking significant steps to improve performance, while adhering to a crucial constraint on the allowable proximity between the new and old policies. This constraint is quantified using KL-Divergence, which measures the "distance" between probability distributions, although not in the conventional sense. The trust region constraint serves as a boundary to restrict the extent of policy updates. By operating within this region, TRPO avoids making drastic changes that might adversely affect performance, ensuring a stable learning process [42].

2.5.7 Proximal Policy Optimization (PPO)

PPO algorithms offer several advantages similar to TRPO, but they are more straightforward to implement, possess greater versatility, and exhibit improved sample efficiency [43]. This improvement is achieved by utilizing a distinct objective function [44].

Due to its user-friendly nature and impressive performance, PPO has become the primary choice for reinforcement learning at OpenAI, being widely adopted as the default algorithm [43].

During the course of the dissertation, the optimization algorithm that will be predominantly used is the PPO implementation of Stable Baselines. This algorithm is the primary one being used by the FCPortugal3D team and with great success. The concept of PPO is well-developed in the chapter 3.6.4.

Chapter 3

FC Portugal: Work Environment

This dissertation will use a special environment for the competition of simulated humanoid robots. Therefore, it is essential to know the technologies used such as the server (SimSpark), the interface used (RoboViz) and the type and characteristics of the agents used in these games (NAO robots). In addition, the team of FCPortugal has its own tools to facilitate the training (FCP Gym) of agents and recommends appropriate software (StableBaselines).

3.1 Robocup

Since 1997, there has been an annual competition that tests teams of humanoid robots, hosted in different countries [7], as described in 1.1. Among the many existing competitions, there is the 3D simulation competition, where the realism of the simulated environment is increased compared to other simulation competitions by adding an extra dimension [8] . In the simulated environment, there is a multi-agent system in which each agent or robot must think and act independently. This is the competition where FCPortugal competes in its events and where the work developed in this dissertation will be implemented.

3.2 SimsPark

SimSpark is a physical multi-agent simulator designed for 3D environments, utilizing the Open Dynamics Engine library [45]. It operates on the Spark application framework and has served as the official simulation platform for the RoboCup Simulation League since 2004 [46].

Rcssserver3d was specifically developed for the RoboCup simulation league. It serves as a soccer simulation server built on the SimSpark platform, offering specialized physical 3D soccer simulations. The server supports both UDP and TCP communication, enabling implementation in any programming language that supports these socket types [47].

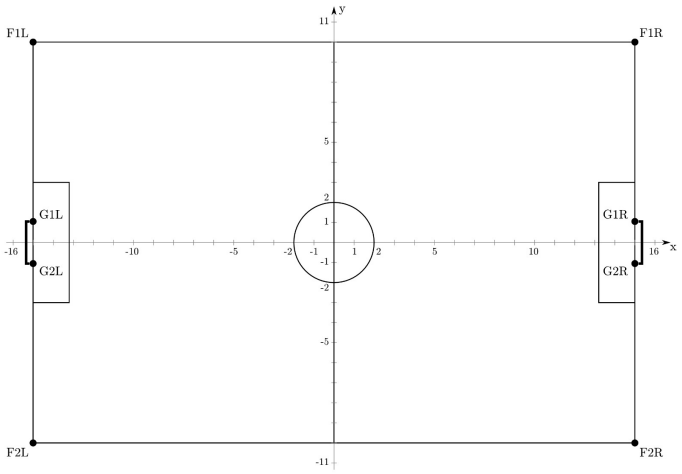

Figure 3.1: SimsPark Football field [4]

3.2.1 Simulated Field

The simulated soccer field is modeled based on a FIFA soccer field, approximately occupying 35% of its dimensions, which is about 30 by 20 meters in size. The main features of the field include:

- The goals are 2.1 by 0.6 meters, standing at a height of 0.8 meters.
- Each goal has a penalty area measuring 3.9 by 1.8 meters.
- The center circle has a radius of 2 meters.
- The soccer ball in the simulation has a radius of 0.04 meters and weighs 26 grams.
- Surrounding the soccer field is a border extending 10 meters in both the x and y dimensions.
- Unique markers are placed at each corner of the field and on the goalposts. These markers' positions are fixed and known to each agent, helping them determine their position on the soccer field by observing the field lines within their field of view.

The field is divided by two axes, with the origin point (x = 0 and y = 0) located at the center of the middle circle. The x-axis ranges from -15 to 15, and the y-axis ranges from -10 to 10, as illustrated in Figure 3.1 [4].

3.2.2 Architecture in a Competition Context

In the RoboCup competition, each team is allocated a client computer. Within each team, every agent must connect to a proxy, which, in turn, is linked to a server. A monitor is also connected to the server to provide a visual representation of the game (Figure 3.2).

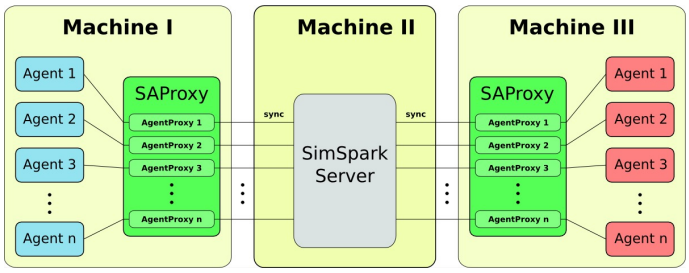

Figure 3.2: SimSpark: A Deeper Look into the Architecture [4]

The agent proxy instance consists of two threads. One thread is responsible for forwarding server perception messages and maintaining synchronization times, while the other thread relays agent action messages.

In this concept, maintaining the response time of agents is done by simply waiting 20ms after forwarding a perception message to the agent, before sending a "sync"-command back to the server in one thread, while the other thread keeps directly forwarding actions messages from the agent as explained in image 3.3.

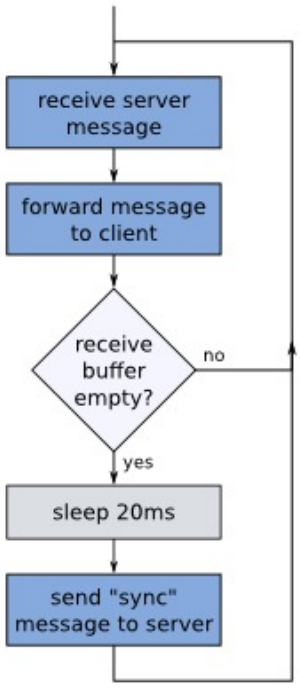

Figure 3.3: Proxy treads server thread

In order to stay synchronized with the SimSpark server, the perception messages forwarding thread always has to first forward all pending perception messages from the server, before waiting and responding with the next "sync"-command [48].

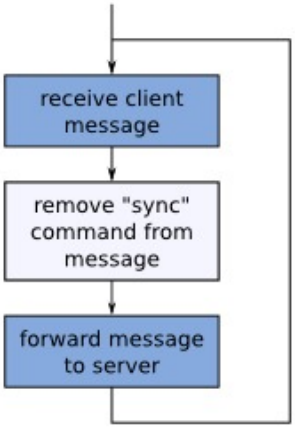

Figure 3.4: Proxy treads client thread

Additionally, any "sync"-commands found within action messages sent by the client agent must be eliminated from those messages before they are relayed to the server. This idea is straightforward, clear for both the client and server, and adaptable to the number of connections. It reinstates synchronization, which is crucial during startup, and treats all connected agents equally. The time measurement is as precise as the sleep command of the underlying platform allows. If the sleep command is deemed insufficiently precise, this approach can be straightforwardly expanded to pause the forwarding of action messages until the next "sync"-command is sent[48].

With this, agents actively interact with the simulation using effector messages, enabling them to control their movements and actions within the game environment. Conversely, monitors receive periodic updates of the simulation state for visualization and logging purposes.

3.3 Roboviz

RoboViz is an open-source program for integrating agent-driven visualizations into a real-time rendered 3D environment.It is the official display interface for the RoboCup 3D simulation league and was developed and is still maintained by the magmaOffenburg team, an active participant in the simulation competitions. It is useful to analyze the learning process of the agents [49].

RoboViz serves as a live 3D application that establishes a connection with the Simspark server through a UDP network protocol between the agents, the server and roboviz . It further connects with agents via a user-friendly drawing interface. The application operates solely through network-based communication to handle and process data from ongoing simulations [50].

In Figure 3.5, you can observe a screenshot of the real-time three-dimensional image generated by Roboviz.

Figure 3.5: RoboViz Snapshot: Real-time Soccer Simulation Visualization

3.4 NAO Robots

The NAO Robot, depicted in Fig. 3.6, is an advanced humanoid robot developed by SoftBank Robotics and serves as the foundational unit for simulated scenarios in the RoboCup initiative.

Initially, the simulation league only featured a spherical agent model. However, in 2006, a simple model of the Fujitsu HOAP-2 robot was introduced, marking the debut of humanoid models in the competition. This transition shifted the focus of the 3D simulation competition from developing strategic soccer-playing behaviors to delving into the low-level control of humanoid robots and creating fundamental behaviors like walking, kicking, turning, and standing up, among others [8].

Currently, Each NAO robot exhibits specific characteristics that may vary from team to team, such as maximum running speed. Standing at approximately 57cm tall and weighing around 4.5kg, the simulated NAO's biped architecture with 22 degrees of freedom provides exceptional mobility.

Figure 3.6: Simulated NAO

3.4.1 Equipment

The NAO robot model is equipped with an array of preceptors and effectors that furnish a wide-ranging information base for agent development [51], [6]:

- A gyroscope and accelerometer in its torso monitor its radial and axial movement in three-dimensional space.

- Each foot is equipped with a force resistance preceptor to detect contact with the ground or other objects.
- For visual detection, a restricted vision preceptor is located at the center of its head.
- Communication capabilities include a hear preceptor and a "say" effector.
- The gamestate preceptor conveys information about the actual play time and play mode.
- The position of each joint is represented by a hinge joint preceptor and can be manipulated through the corresponding hinge joint effector. The arrangement and relative orientation of the joints are depicted in Fig. 3.7.

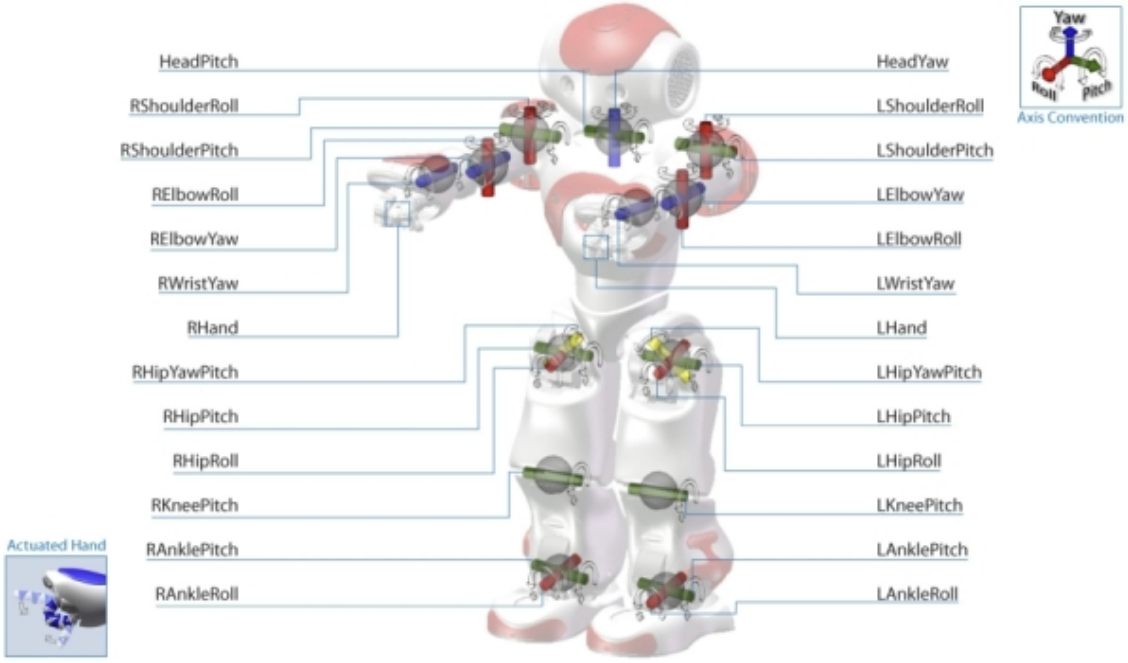

Figure 3.7: All available joints name, arrangement and their relative orientation [5]

3.4.2 Joint informations

With 22 angles of freedom the robot joints can rotate about the 3 axes x, y z. The name given to this three rotational degrees of freedom used to describe the orientation of a rigid body are pitch, roll, and yaw.

- Pitch refers to the rotation around the x-axis (or longitudinal axis) of the body, i.e., its angle relative to the horizon.
- Roll refers to the rotation around the y-axis (or lateral axis) of the body, i.e., its angle of tilt to the right or left.

- Yaw refers to the rotation around the z-axis (or vertical axis) of the body, i.e., its angle of deviation from the reference heading direction.

The name applied to the joints will refer to their body part and the type of possible rotation. A complete list of all available joints of the NAO robot, the minimal and maximum angle limit, and their corresponding identifiers is listed in the table 3.1 below.

ID	Joint Name	Min	Max
0	Neck Yaw	-120	120
1	Neck Pitch	-45	45
2	Left Hip YawPitch	-90	1
3	Right Hip YawPitch	-90	1
4	Left Hip Roll	-25	45
5	Right Hip Roll	-25	45
6	Left Hip Pitch	-25	100
7	Right Hip Pitch	-25	100
8	Left Knee Pitch	-130	1
9	Right Knee Pitch	-130	1
10	Left Foot Pitch	-45	75

ID	Joint Name	Min	Max
11	Right Foot Pitch	-45	75
12	Left Foot Roll	-45	25
13	Right Foot Roll	-25	45
14	Left Shoulder Pitch	-120	120
15	Right Shoulder Pitch	-120	120
16	Left Shoulder Yaw	-1	95
17	Right Shoulder Yaw	-95	1
18	Left Arm Roll	-120	120
19	Right Arm Roll	-120	120
20	Left Arm Pitch	-90	1
21	Right Arm Yaw	-1	90

Table 3.1: Maximum angular variations and their respective joint names [5, 6]

3.4.3 Agent Types

There are five distinct types of humanoid robots, each equipped with either 22 or 24 joints, and exhibiting characteristic physical differences. These five robot versions have been derived from the NAO Robot mentioned earlier.

In accordance with the latest tournament regulations, each team participating will be required to use three different robot types. Moreover, there is a limitation that no more than seven robots of each type can be utilized during the competition. This rule aims to promote diversity and fairness among the teams, ensuring that the gameplay remains challenging and dynamic [52].

3.5 2023 Competition Rules

Every year, the rules of the game undergo changes to enhance the simulation and make the agents' actions appear increasingly realistic. One such new rule has inspired the FCPortugal team to develop innovative dribbling techniques, becoming a significant motivation for the research conducted in this dissertation.

During a game, a player is now considered to be holding the ball if: the ball is within a distance of BallHoldRadius meters (the radius in meters from a player's center where the ball is considered held) from them, and if there are no opponents closer to the ball.

If the player holds the ball beyond BallHoldMaxTime seconds (the maximum time in seconds a player can hold the ball) or if the ball travels at least BallHoldMaxDistance (the maximum

distance in meters the ball can travel while a player is holding it) from its original position with an opponent within BallHoldOppDistance meters (The minimum distance in meters an opponent must be from the ball for a holding penalty foul to be called, results in penalties. Similar rules apply to the goalkeeper.

The ball holding penalty parameters are as follows [52]:

- BallHoldRadius (0.12m): The radius in meters from a player's center where the ball is considered held.
- BallHoldMaxTime (5.0s): The maximum time in seconds a player can hold the ball.
- BallHoldGoalieMaxTime (10.0s): The maximum time in seconds a goalie can hold the ball within their own penalty area.
- BallHoldResetTime (0.5s): The time in seconds since a player stopped holding the ball before the ball holding time resets.
- BallHoldMaxDistance (1.0m): The maximum distance in meters the ball can travel while a player is holding it.
- BallHoldOppDistance (0.75m): The minimum distance in meters an opponent must be from the ball for a holding penalty foul to be called (a negative value results in a foul regardless of opponent proximity).

3.6 Training Process

3.6.1 FC Portugal's Transition to Python

After RoboCup 2021, the entire codebase of the FCPortugal team underwent a complete rewrite in Python, without reusing the previous C++ code, except for a few specific functions. The primary motivation behind this transition was to ensure compatibility with widely used data science libraries and machine learning repositories. By adopting Python as the language of choice, the team aimed to accelerate the development of new behaviors and strategies.

Despite the notable efficiency of Python computing, especially with recent hardware advancements, certain modules still rely on C++ to guarantee precise synchronization of agents with the server during gameplay. This hybrid approach enables the team to leverage the benefits of both Python's flexibility and accessibility, and C++'s low-level control and performance when necessary for critical aspects of their system [10].

3.6.2 TensorFlow

TensorFlow is an open-source Python-based machine learning framework developed by Google that lets you design, build, and train deep learning models. The name itself derives from the operations that neural networks perform on multidimensional data arrays or tensors [53].

3.6.3 Stable Baselines

Stable Baselines [54] is an invaluable tool that simplifies the utilization of various algorithms by offering common functionalities like training, saving, and loading models. This user-friendly library stems from a fork of OpenAI Baselines [55], enhancing it with additional features and improvements.

One notable feature of Stable Baselines is its capability to customize hyperparameters, particularly the policy. This allows users to define their own network structure by adjusting the number of layers and nodes, tailoring the learning process to specific needs.

In addition to its ease of use, Stable Baselines comes equipped with comprehensive documentation, which proves indispensable for developers seeking to understand and leverage the library efficiently [56].

3.6.4 Optimizing Behavior with Proximal Policy Optimization (PPO)

The optimization method chosen for its simplicity, good performance in generating high-quality behaviors, and the professor's recommendation was Proximal Policy Optimization (PPO) [43]. To implement PPO, we utilized the OpenAI baselines implementation [57], which incorporates a loss function described as :

$$L(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t], \tag{3.1}$$

where $r_t(\theta)$ is given by:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \tag{3.2}$$

- $L(\theta)$: This is a mathematical function that represents the loss we want to minimize during training, where θ corresponds to the policy parameter.
- $\hat{E}_t$: It represents an empirical expectation over time steps, essentially an average over a series of observations.
- $r_t(\theta)$: This is the odds ratio between the probabilities of taking certain actions under the current policy (θ) and an old policy θ_{old}.
- $\hat{A}_t$: It serves as an estimator of the advantage function at a given time step.
- ε: This is a hyperparameter typically set to small values like 0.1 or 0.2.

The purpose of this formula is to evaluate how well the current policy θ is performing compared to an older policy θ_{old}. It does so by considering the advantage of taking certain actions under the current policy and uses the odds ratio $r_t(\theta)$ to balance exploration and exploitation.

To maximize the objective function, this implementation enables switching between data sampling from multiple parallel sources and conducting several cycles of the Stochastic Gradient Descent method on the sampled data [57].

3.6.5 Hyperparameters

In the field of reinforcement learning, the effective deployment of algorithms like Proximal Policy Optimization (PPO) hinges heavily on the meticulous management of hyperparameters. These hyperparameters are akin to the levers and dials of a finely-tuned machine, influencing the algorithm's behavior and learning process. As we delve into the workings of PPO, it becomes evident that understanding and appropriately configuring these hyperparameters is paramount for achieving optimal training results.

PPO is developed in two steps:

- **Step 1: Data Collection and Policy Optimization**

  In the first step, PPO gathers data called "transitions." These transitions consist of sequences of states, rewards, and actions that the agent experiences during interactions with its environment. The primary objective here is to optimize the agent's policy based on the data it has collected. The gathered transitions are saved up to a certain "horizon" range, a time frame that helps manage the amount of data used for optimization.

  Once enough data is collected, PPO employs the stochastic gradient descent method, a mathematical technique for optimizing functions, to improve the policy. This optimization occurs incrementally, with small adjustments determined by a parameter called the learning rate. The stored transitions are processed in batches, allowing for efficient policy updates.

- **Step 2: Policy Update and Training**

  In the second step, PPO updates the policy over a specified number of "epochs." An epoch is a training cycle in which the agent learns from its experiences. The total number of steps is a critical parameter, setting the maximum number of interactions the agent will have with its environment during the training process. This parameter directly affects the number of updates applied to the policy.

PPO can operate on multiple environments simultaneously, which is a practical approach for accelerating training. This parallelization enhances the efficiency of the learning process.

One of the strengths of PPO is its flexibility. Researchers and practitioners can customize the policy used by selecting different algorithms and representations. For instance, the neural network architecture used in PPO is adaptable, allowing for experimentation with various designs to match the complexity of the problem and meet specific performance requirements [57, 58].

3.6.6 FCPortugal Gym

OpenAI Gym has established itself as the industry standard for evaluating reinforcement learning algorithms, providing a benchmarking tool [59]. Its widely adopted interface has paved the way for exceptional collections of RL algorithm implementations, such as OpenAI Baselines and Stable-Baselines. For our specific research objectives, we utilized Stable-Baselines.

The primary aim of this research project was to establish adaptable behaviors that could seamlessly integrate into the RoboCup 3D Soccer Simulation League without necessitating extensive post-training adjustments. To achieve this objective, we needed to create a structure that facilitated the training and sharing of reinforcement learning models. For this purpose, we opted for FCPortugal Gym, a solution that aligns seamlessly with the Gym interface.

The FCPortugal team possesses a unique tool called FCPortugal Gym, which enables continuous improvement and optimization of the team's skills and performance. It serves as a powerful framework for implementing reinforcement learning algorithms, similar to the popular OpenAI Gym [60].

3.6.7 Learning Cycle

The training cycle in FCPortugal Gym involves the use of three essential functions. Firstly, the reset() function is employed to set up the desired environment, positioning the players and the ball as required.

After using the reset() function, the observe() function comes into play, enabling the agent to make observations defined earlier, thus establishing the state space.

Subsequently, the step(action) function executes, applying the neural network's output as the agent's actions. This function serves as the crucial interaction point where changes with the environment occur. The agent's actions and interactions with the environment are guided by the reward function.

The training cycle will continue with a variation, where after performing the reset() function once, the other two functions, observe() and step(action), are alternated until the episode concludes. This process ensures that the neural network undergoes updates to enhance the reward function and improve the agent's performance.

The episode's conclusion can arise from various reasons, and these are indicated by a terminal flag within the step(action) function. This flag signs the end of the episode and marks the completion of the training cycle for that particular episode.

3.6.8 Training evaluation

During the course of the training, it is possible to assess how it's doing due to a series of factors that are analyzed in order to give us feedback. The printout in question is the training progress summary generated by the Stable Baselines library while training a reinforcement learning model with the Proximal Policy Optimization (PPO) algorithm. This printout provides various metrics and statistics that help monitor the progress and performance of the training.

This information provides insights into various aspects of the training process, including policy update stability, entropy, value function approximation, and the overall progress of optimization. It's important to monitor these metrics to ensure that the training process is proceeding as expected and that the model is improving over time.

After a predefined number of iterations, the model undergoes testing to calculate both the average reward per episode and the average episode duration. This process plays a pivotal role in monitoring the training's progress and gauging the consistent improvement of the reward function. If the reward function reaches a point where it no longer shows significant evolution, it becomes unnecessary to prolong the training. Following this assessment, a graphical representation is generated to illustrate the evolution of the average reward per episode over time.

This tool is immensely valuable for saving time because it allows us to halt the training as soon as the reward value stabilizes, rather than continuing it for a fixed number of steps.

3.6.9 Vectorized Environments

Vectorized Environments represent a method for merging multiple independent environments into a single unified entity. Instead of training a reinforcement learning (RL) agent on a single environment step by step, this approach allows for training across a multitude of environments concurrently. Consequently, the actions supplied to the environment form a vector with a dimension of 'n.' This same principle extends to observations, rewards, and signals indicating the conclusion of an episode.

3.6.10 Behavior inversion

To achieve symmetrical inversion of a behavior learned exclusively for one limb, it is advantageous to invert both the state space and the action space. This allows for the execution of the behavior with the opposite limb, enhancing the efficiency of utilizing the acquired knowledge and expediting the learning process by training only one side.

3.6.10.1 State Space Inversion

In order to achieve the inversion of the state space, specific adjustments are implemented during the data collection phase when gathering sensor information. These adjustments are designed to allow the agent to perceive the environment symmetrically while utilizing the pre-trained neural network. The resulting modifications are reflected in the changes detailed in Table 3.2.

In order for the robot to exhibit a behavior symmetrical to what it has learned, it must receive input from its own sensors inverted along the sagittal axis of the robot. This means that the coordinates of the sensors should be inverted along the vertical axis. If the base of a robot joint was initially situated at coordinates (x, y, z), after inversion, its position would become (-x, y, z), effectively mirroring the movement in relation to the sagittal axis. For instance, the position of the left hip actuator would then become the right hip actuator.

This inversion approach is commonly employed when seeking to generate symmetrical movements in robots, particularly in cases where a learned behavior needs to be replicated in a mirrored fashion.

This table illustrates the transformation applied to the state space within the algorithm. It provides a clear mapping of how each initial input value is adjusted to its inverted position. These

Initial Input value	Inverted position
Neck Yaw	-Neck Yaw
Neck Pitch	Neck Pitch
Left Hip YawPitch	Right Hip YawPitch
Right Hip YawPitch	Left Hip YawPitch
Left Hip Roll	Right Hip Roll
Right Hip Roll	Left Hip Roll
Left Hip Pitch	Right Hip Pitch
Right Hip Pitch	Left Hip Pitch
Left Knee Pitch	Right Knee Pitch
Right Knee Pitch	Left Knee Pitch
Left Foot Pitch	Right Foot Pitch

Initial Input value	Inverted position
Right Foot Pitch	Left Foot Pitch
Left Foot Roll	Right Foot Roll
Right Foot Roll	Left Foot Roll
Left Shoulder Pitch	Right Shoulder Pitch
Right Shoulder Pitch	Left Shoulder Pitch
Left Shoulder Yaw	Right Shoulder Yaw
Right Shoulder Yaw	Left Shoulder Yaw
Left Arm Roll	Right Arm Roll
Right Arm Roll	Left Arm Roll
Left Arm Pitch	Right Arm Yaw
Right Arm Yaw	Left Knee Pitch

Table 3.2: Inverse State Space Changes in Sensor Position

adjustments are crucial for achieving a symmetrical perception of the environment and play a fundamental role in the algorithm's operation. In this way, ensuring that the agent can effectively interact with the environment in a mirrored fashion, ultimately contributing to the achievement of the desired objectives within the robotic system.

3.6.10.2 Action Space Inversion

To create symmetrical behavior in a robot, it's also necessary to perform an action space inversion. This means that the actions the robot performs need to be adjusted to reflect the opposite behavior.

For example, if the robot has been trained to move its left hip as an action, to achieve symmetrical behavior, it's needed to adjust the action to move the right hip instead. In other words, it's needed to invert the actions that the robot executes so that they align with the mirrored behavior that it's desired. This inversion ensures that the robot's movements are symmetrical, mirroring actions on either side of its body along the sagittal axis.

By performing both inversions simultaneously, it expected to align the robot's perceptions with its actions consistently. This enables the robot to operate effectively within the new mirrored environment. If only the state is inverted or only the action space, the robot might encounter challenges in comprehending and interacting with the mirrored environment, potentially resulting in undesirable or non-symmetrical outcomes.

The alterations made in the action space are analogous to those in the state space, as evidenced in the table 3.2.

3.7 Low-level Skills

In this section, we delve into the fundamental skills in the realm of robotic soccer, known as low-level skills. These skills constitute the essential building blocks for effective execution of more intricate tasks on the field. Through meticulous analysis and careful development, the FCPortugal team has honed and refined these skills to maximize our robots' performance in competitive games.

The emphasis on low-level skills is instrumental in enhancing control, precision, and efficiency of our robots across a spectrum of game situations. Through the exploration of these skills, we are poised to tackle the dynamic and intricate challenges of robotic soccer, ensuring exceptional performance and a strategic approach in every match.

3.7.1 Kick

The ability to kick is crucial in soccer to score goals and win games. The kick skills are based on a hand-tuned base model, divided into two sections: backswing and forward acceleration. FCPortugal team offers two kicking skills:

- **Short Kick**: Primarily used for passes, with a controllable range between 3 and 9 meters.
- **Long Kick**: Generally used for shooting, with an average distance of 17 to 19 meters, depending on the robot type.

These kicks, despite being highly accurate, come with a significant drawback, which is the speed at which they are executed. Agents spend a considerable amount of time aligning themselves with the ball to execute this shot. This can pose issues in sporting performance, as a substantial amount of time is consumed in their execution [10].

3.7.2 Walk

The walk skill boasts omnidirectional capabilities, enabling movement in any direction. Depending on the robot type and movement direction, the walk skill maintains an average linear speed ranging from 0.70 to 0.90 m/s.

3.7.3 Dribble

The dribble skill allows the robot to push the ball forward while maintaining close control. During forward dribbling, the robot achieves an average maximum speed of 1.25 to 1.41 m/s, varying depending on the robot type. This particular dribbling technique ensures precise ball control within a maximal distance of 0.12 meters.

3.7.4 Dribble Long

The previous approach to dribbling, involving sustained ball control within a radius smaller than 0.12 meters when an opponent was nearby, has been prohibited. To adhere to this new regulation, an innovative skill known as "Dribble Long" was formulated.

This dribbling technique comprises various distinct versions, which will be presented in the next sections. The behaviors explored in this dissertation will take inspiration from the diverse versions of the new dribbling technique.

3.7.4.1 Double Footed Dribble Long (DFDL)

The DFDL behavior relies on several key indicators influencing the creation of other behaviors in this thesis.

One crucial indicator is the activation status of the left leg. When the left leg is active, it signifies that is moving, while its inactivity denotes support for the robot's movement.

During each active and inactive phase of the left leg, there are eight distinct movement phases associated with its utilization for the action. These phases play a crucial role in effectively executing the dribble long behavior.

The subsequent photographs display all conceivable phases in Figure 3.16 and 3.25:

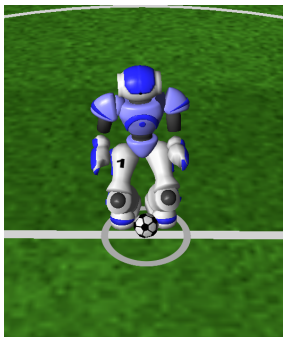

Figure 3.8: Dribbling phase (0,false) of DFDL

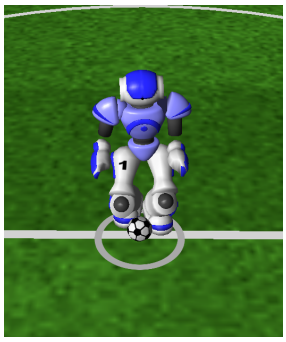

Figure 3.9: Dribbling phase (1,false) of DFDL

Figure 3.10: Dribbling phase (2,false) of DFDL

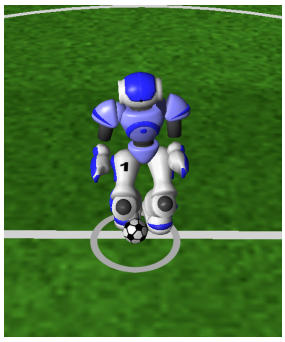

Figure 3.11: Dribbling phase (3,false) of DFDL

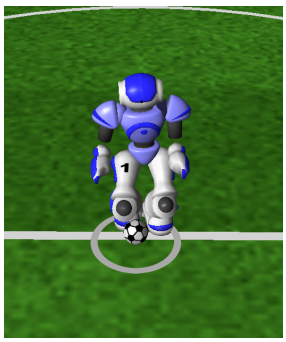

Figure 3.12: Dribbling phase (4,false) of DFDL

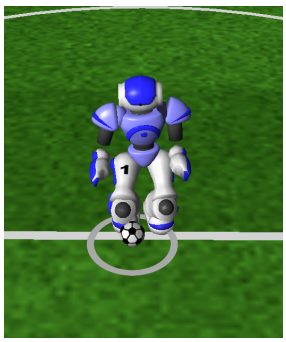

Figure 3.13: Dribbling phase (5,false) of DFDL

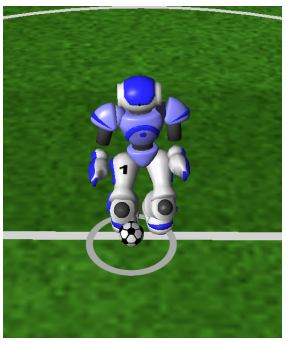

Figure 3.14: Dribbling phase (6,false) of DFDL

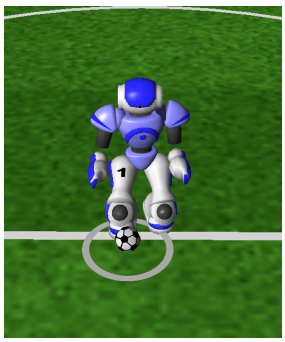

Figure 3.15: Dribbling phase (7,false) of DFDL

Figure 3.16: 8 phases of dribbling with inactive phase of the left leg of the DFDL

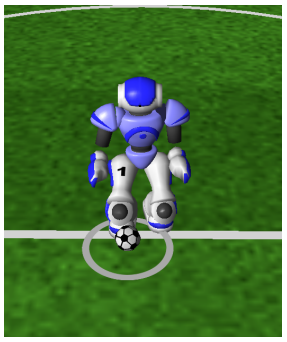

Figure 3.17: Dribbling phase (0,true) of DFDL

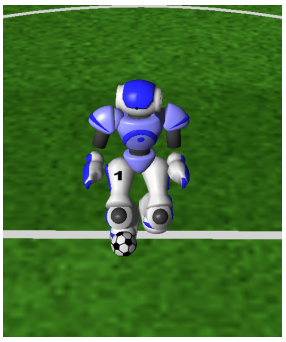

Figure 3.18: Dribbling phase (1,true) of DFDL

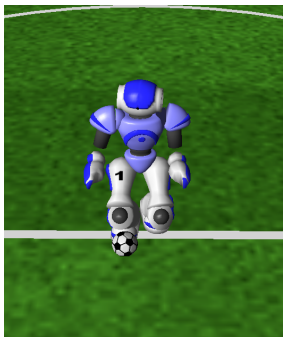

Figure 3.19: Dribbling phase (2,true) of DFDL

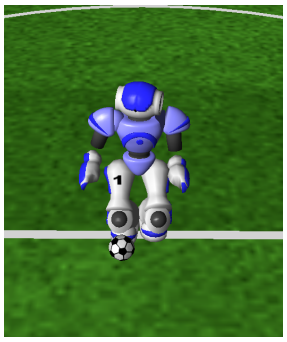

Figure 3.20: Dribbling phase (3,true) of DFDL

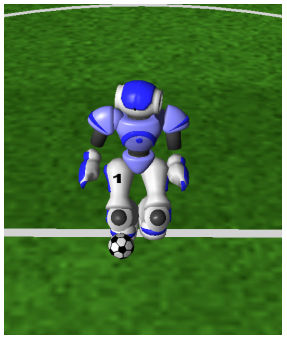

Figure 3.21: Dribbling phase (4,true) of DFDL

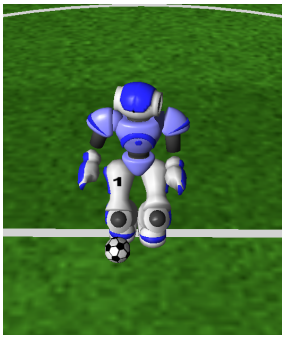

Figure 3.22: Dribbling phase (5,true) of DFDL

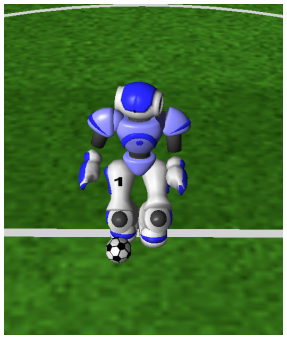

Figure 3.23: Dribbling phase (6,true) of DFDL

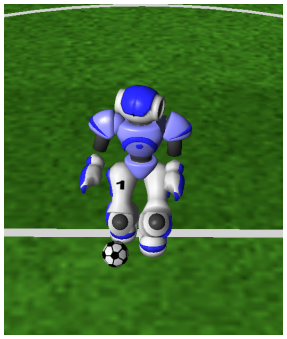

Figure 3.24: Dribbling phase (7,true) of DFDL

Figure 3.25: 8 phases of dribbling with active phase of the left leg of DFDL

The dribbling phases will now be denoted using the notation (x, y), where x represents the phase number ranging from 0 to 7, and y indicates whether the left foot is active (true) or inactive (false). An active left foot signifies that, during that particular phase of dribbling, the left leg is elevated, while the right leg is providing support.

As the name suggests, this version of DFDL employs both feet to perform the dribble.

3.7.4.2 Left Footed Dribble Long (LFDL)

This version of the LFDL behavior has a distinctive feature: it exclusively employs the left foot for dribbling. This approach offers the advantage of facilitating a smoother and more seamless transition to the rapid shot. Moreover, the distance between the ball and the robot during the dribble execution is notably greater in this LFDL, resulting in a prolonged control distance.

Similar to the preceding phase, this dribble consists of 16 distinct moments, each delineated by the phases depicted in Figure 3.34 and 3.43.

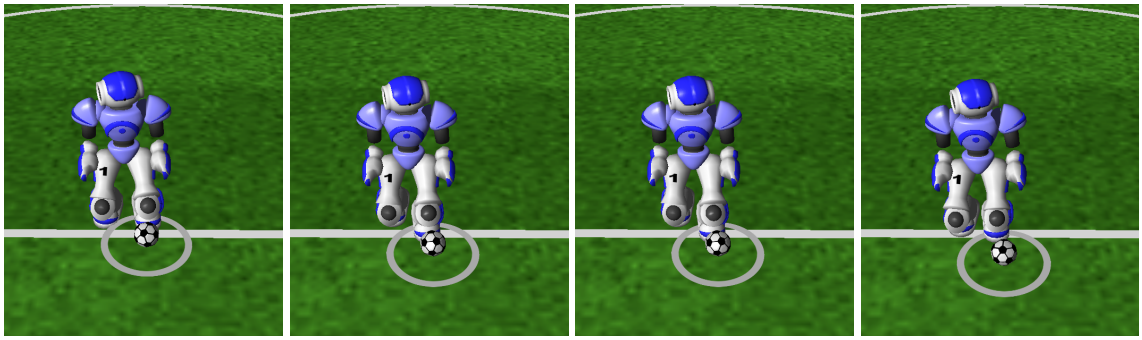

Figure 3.26: Dribbling phase (0,false) of LFDL

Figure 3.27: Dribbling phase (1,false) of LFDL

Figure 3.28: Dribbling phase (2,false) of LFDL

Figure 3.29: Dribbling phase (3,false) of LFDL

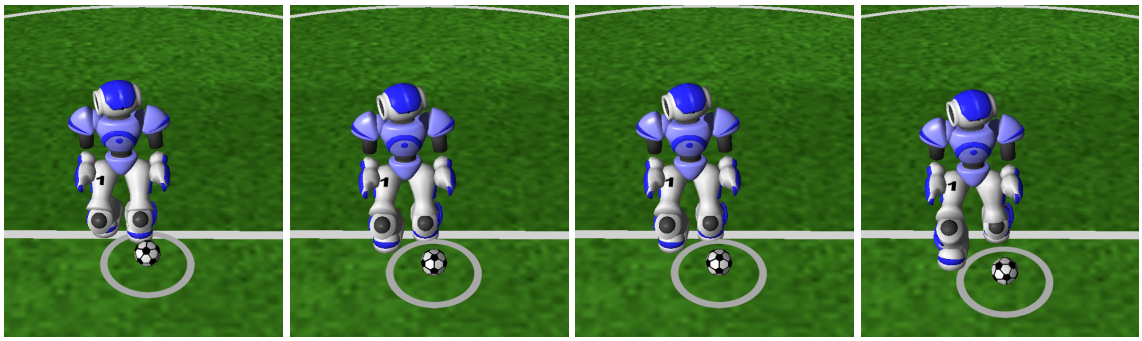

Figure 3.30: Dribbling phase (4,false) of LFDL

Figure 3.31: Dribbling phase (5,false) of LFDL

Figure 3.32: Dribbling phase (6,false) of LFDL

Figure 3.33: Dribbling phase (7,false) of LFDL

Figure 3.34: 8 phases of dribbling with inactive phase of the LFDL

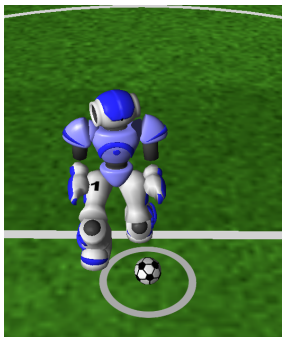

Figure 3.35: Dribbling phase (0,true) of LFDL

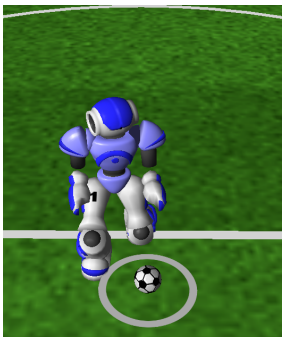

Figure 3.36: Dribbling phase (1,true) of LFDL

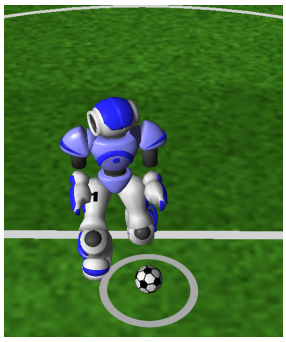

Figure 3.37: Dribbling phase (2,true) of LFDL

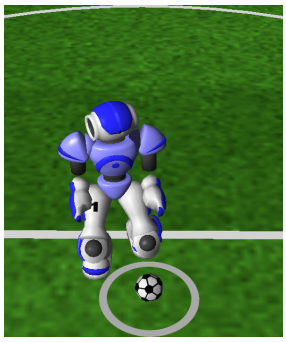

Figure 3.38: Dribbling phase (3,true) of LFDL

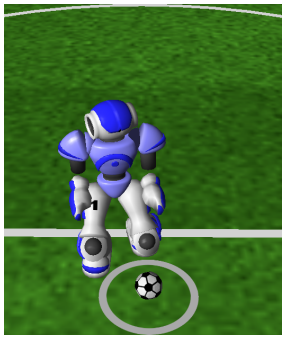

Figure 3.39: Dribbling phase (4,true) of LFDL

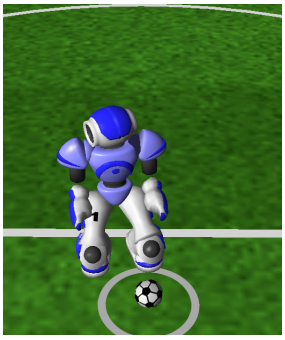

Figure 3.40: Dribbling phase (5,true) of LFDL

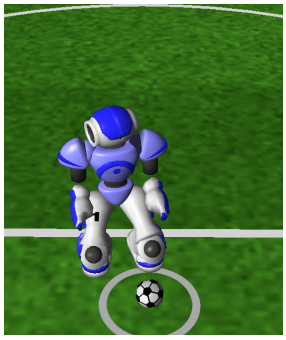

Figure 3.41: Dribbling phase (6,true) of LFDL

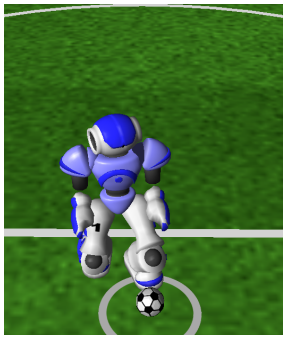

Figure 3.42: Dribbling phase (7,true) of LFDL

Figure 3.43: 8 phases of dribbling with active phase of the LFDL

The contrast in ball control distance exhibited by the robot during the 3.7.4.1 dribble and the 3.7.4.2 dribble is illustrated in figures 3.44 and 3.45. When employing the Left Footed Dribble Long, a more considerable gap between the ball and the robot throughout the dribble becomes evident.

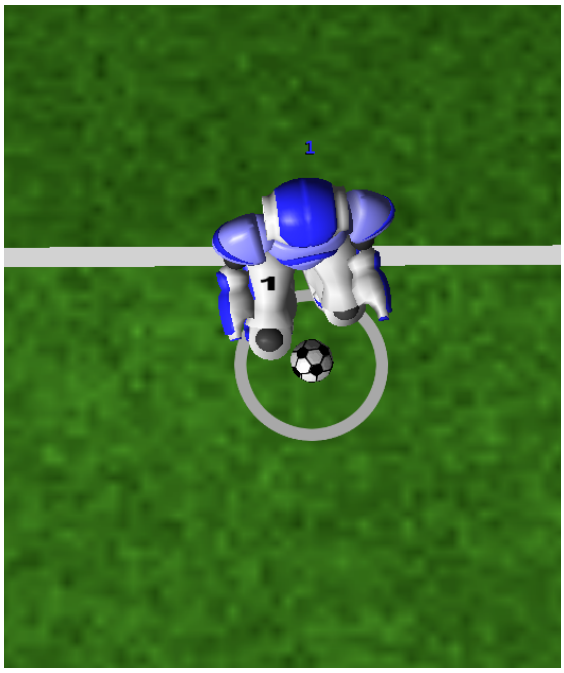

Figure 3.44: Maximum dribbling distance for DFDL

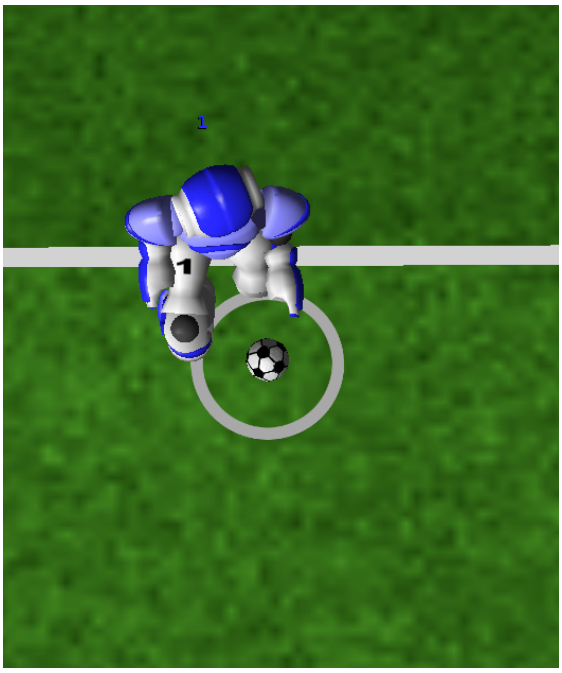

Figure 3.45: Maximum dribbling distance for LFDL

The notation used for the DFDL version will also be applied in this version (notation (x, y), where x represents the phase number ranging from 0 to 7, and y indicates whether the left foot is active (true) or inactive (false)).

3.8 Previous work contributions

Recent work, created by other members of the FCPortugal team, related to humanoid robot skills using RL were outlined and analyzed in order to help me in the understanding of the current existing capabilities and approaches used for the team.

Some of these works concentrated on refining shooting behaviors while in motion [61], [62], as well as executing controlled distance shots [63]. Additionally, the team developed behaviors like running and sprinting [64], [65]. The team's efforts also extended to creating various low-level and high-level behaviors [66], [67], [68], [69], [70], [71], [72], [73].

Chapter 4

Kick in motion after Double Footed Dribble Long

Previous efforts have laid the foundation for various kicking methodologies. However, the current implementations of FCPortugal still lack the ability to perform kicks in a manner similar to humans, without interrupting their forward motion during the Dribble behavior.

Achieving a seamless transition from the gait to a kick, without the need to halt or alter velocity, is what defines a kick in motion. This behavior not only enhances the agent's performance but also lends a more human-like quality to their on-field performance.

Consequently, the research conducted in this context holds immense potential to infuse FCPortugal 3D team's gameplay with greater dynamism, ultimately providing a notable competitive advantage.

4.1 Problem

To enhance the team's performance, the development of quick shots with speed after a dribble has been needed. The objective is to execute these shots as swiftly and fastest as possible. Currently, the team only has two types of kicks available, both of which solely allow for kicking the ball that are relatively slow in execution, requiring the player to transition to a run or walk phase after the dribble to approach the ball before taking the shot.

This foundational kick lays the groundwork for subsequent behavior refinement.

The purpose behind this training initiative was to comprehend the nuanced circumstances detailed in section 3.7.4.1 of the dribbling behavior. This comprehension serves to enhance the success rate of shooting during various phases of the dribble, alongside different activations of the left leg.

Hence, prior to establishing the viability of consistently achieving successful shots from specific positions during the variable dribbling phase, it becomes imperative to evaluate the most effective type of dribble phase for executing a shot.

Through a meticulous analysis of these variables and the identification of the optimal shot type for each phase of the dribble, the establishment of a high success rate shot becomes attainable.This investigation serves as a crucial step in developing effective behaviors for executing quick shots, contributing to the team's overall performance.

4.2 Action Space

The simulated NAO robot depicted in Figure 3.7 is equipped with actuators associated with each of its joints. In total, there are 20 actuators utilized, excluding the head pitch and yaw actuators as they are not part of the optimization objective.

To effectively control each actuator, the simulator operates based on angular speeds ranging from -7°/s to +7°/s, which are provided as continuous variables. In our approach, the agent employs an indirect control method. Instead of directly generating speed values for each actuator, the optimization algorithm produces target angles for each joint. These target angles are then used to determine the appropriate angular speeds for the actuators, enabling effective control of the robot's movements.

4.3 State Space

The state space comprises the following observations:

- Angular positions for all robot hinge joints, expressed in degrees.
- Torso angular velocity, measured by a gyroscope, presented in degrees per second.
- Torso proper acceleration relative to free fall, recorded by an accelerometer, provided in square meters per second.
- Relative position of the ball in relation to the robot's torso, calculated based on the robot's vision, presented in meters.
- Time steps since the agent began controlling the robot.

The robot's vision preceptor is subject to noise, which indirectly impacts the observed relative position of the ball as perceived by the agent. This noisy observation is used instead of a noiseless one to train the agent to handle the real-life conditions encountered during actual games.

4.4 Control Step Size

In order to accommodate the limited update frequency of visual information, the robot's perception of visual data is refreshed every 3 simulation cycles. As a result, the agent only assumes control of the robot's actions when new visual information is received, precisely once every 3

time steps. During the intermediate steps, the previously determined actions are transmitted to the robot, enabling it to continue executing the planned actions until new visual information becomes accessible. This approach guarantees that the agent's control decisions align with the updated visual input, preserving synchronization between perception and action.

4.5 Episode layout

During the initial training phase, the NAO robot begins at a distance of 0.3 meters from the center of the field and the joints are reset to an initial position, with the ball also positioned there facing it. It then initiates a DFDL, with a random duration ranging from 0 to 2 seconds, and a random orientation between -180 to 180 degrees. This randomness is introduced to create greater variability in the training process.

Once the robot completes the random time limit for dribbling and remains upright, the learning controller takes over. It commands the robot's joints every third time step, allowing for precise control over its movements.

The episode lasts approximately 10 steps and is terminated. This termination condition is specifically designed to encourage the robot to execute the shot as quickly as possible, regardless of whether it maintains its balance or falls during the process.

The objective of this kick in motion scenario is to train the agent to execute a kick immediately after the dribble without any additional movement.

4.6 Network Shape and Hyperparameter Tuning

It was opted for this thesis a neural network architecture with 2 hidden layers, each consisting of 64 neurons, following the Stable Baselines standard. With a short episode duration and concise actions, a Horizon Range of 256 steps was sufficient; a larger value wasn't justified.

To keep computation efficient, we set the minibatch size to 64, and the learning rate was chosen as 3e-4.

In parallel, we ran the training across the maximum available logical CPUs or cores in the system, performing approximately 20 million total steps.

4.7 Reward

During mid-dribbling, the agent assumes control of the robot. To mitigate challenges associated with inappropriate behaviors, such as kicking the ball sideways instead of forward, and considering the added complexity of learning this task, a simpler reward function was chosen. In this approach, the agent receives feedback based solely on the velocity the ball gains along the desired kick path.

$$r = v * (180 - \alpha)/180 \tag{4.1}$$

In the context of the training episode, the variable v represents the velocity of the ball at the end of the episode. It captures the speed at which the ball is propelled after the kick.

Additionally, the variable α denotes the difference between the absolute orientation of the ball direction and the absolute orientation of the robot. This difference is measured to ensure that a frontal shot is intended. By considering the alignment between the robot and the ball's orientation, the agent aims to execute a kick directly facing the target, optimizing the chances of a successful shot.

As evident from the graph depicted in function 4.1, the reward value exhibits a linear variation. An increase in the α value corresponds to a proportional linear decrease in the reward value, as governed by the preceding function. Similarly, a higher shot speed yields a greater reward. When the angle α reaches 90 degrees, the value will be at 50% of its total.

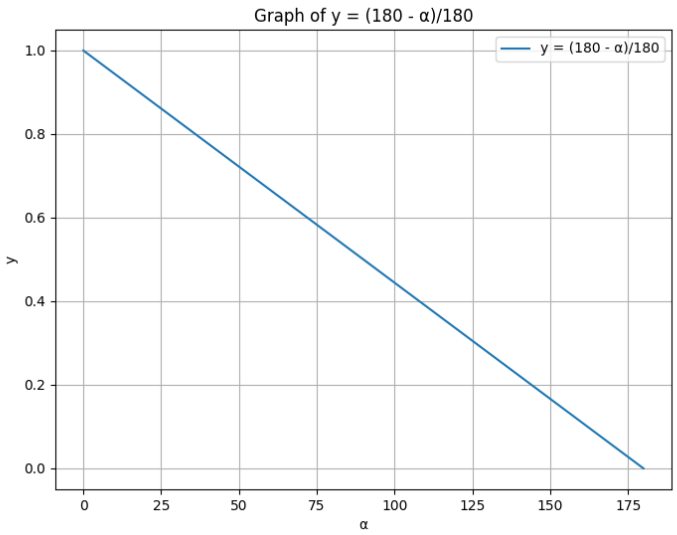

Figure 4.1: Plot of the function $y = (180 - \alpha)/180$

4.8 Results

During the testing of the initial training, where the robot performs random dribbles to various sides, it is observed that the robot successfully learns to shot in all phases of the dribble, regardless of the left foot activation. Additionally, it is observed that the robot learns to exclusively shot with the left foot. This phenomenon can be attributed to the nature of neural network algorithms, as they tend to find it easier to specialize in a specific action, in this case, shooting with one foot, rather than attempting to learn multiple variations.

As described in 3.6.8, training involves periodic testing to assess its progression. Initially, the average reward experiences a steep rise, which then stabilizes after 2 million steps. Subsequently, there are no significant fluctuations until the training process concludes at 11,345,920 steps, as depicted in figure 4.2.

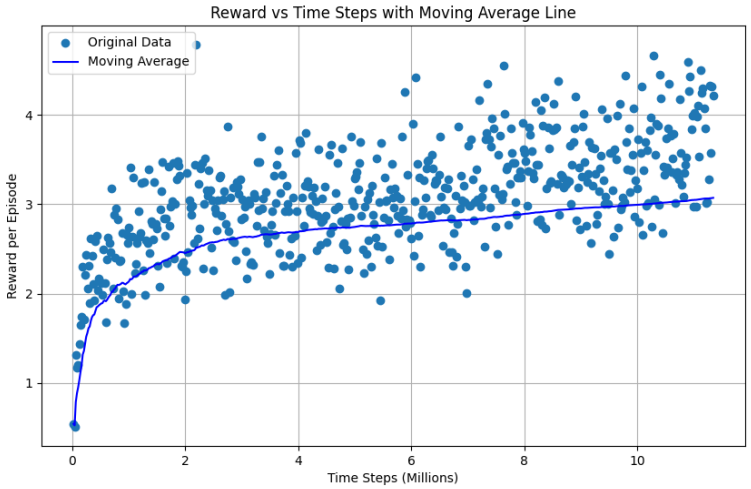

Figure 4.2: Reward Evolution Over Time Steps from first training session DFDL

The outcome of the previous training test, involving the test of over 4500 shots, is presented in figures and table below.

An analysis of the provided graphs offers insights into the frequency distribution of both achieved speeds and α angles, denominated alpha, across various dribbling phases. Initially, we present the frequency graphs concerning the speed parameter for each dribbling phase, as depicted in the figures.

Upon closer examination of the speed-related graphs, it becomes evident that the (2,true) phase, shown in Figure 4.7, can attain favorable speed values, with 4 m/s being the most frequently recorded speed. Furthermore, phases (4,true), (5,true), and (6,true), represented by Figures 4.11, 4.13, and 4.15 respectively, yielded the most noteworthy shot velocities.

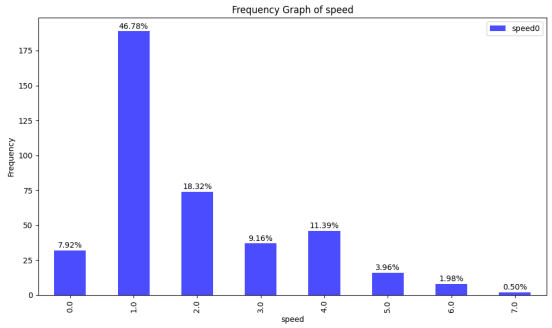

Figure 4.3: Graph of speed frequency for phase (0,true) from first training session DFDL

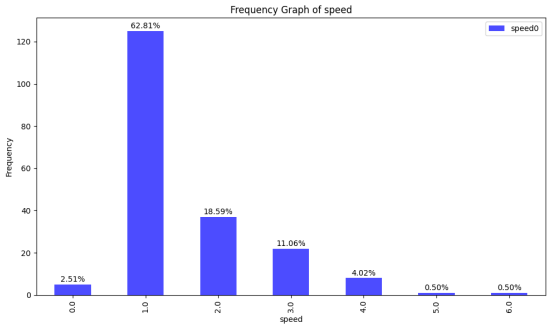

Figure 4.4: Graph of speed frequency for phase (0,false) from first training session DFDL

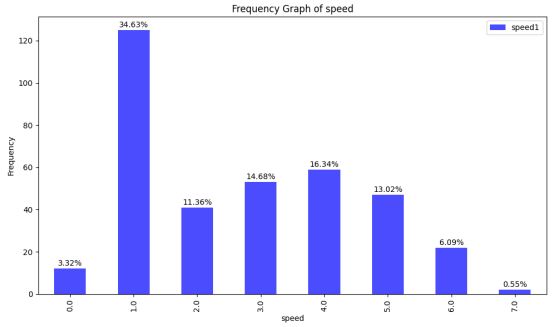

Figure 4.5: Graph of speed frequency for phase (1,true) from first training session DFDL

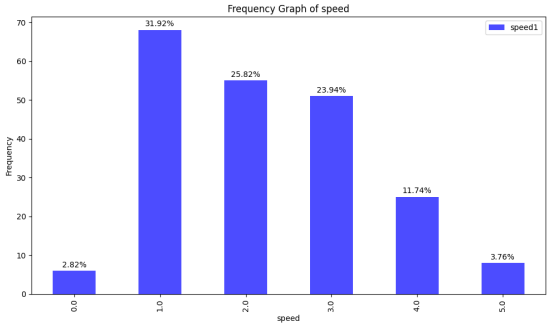

Figure 4.6: Graph of speed frequency for phase (1,false) from first training session DFDL

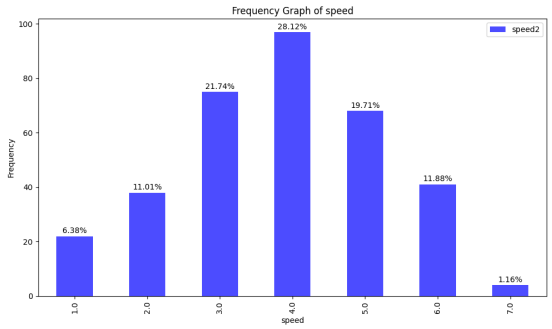

Figure 4.7: Graph of speed frequency for phase (2,true) from first training session DFDL

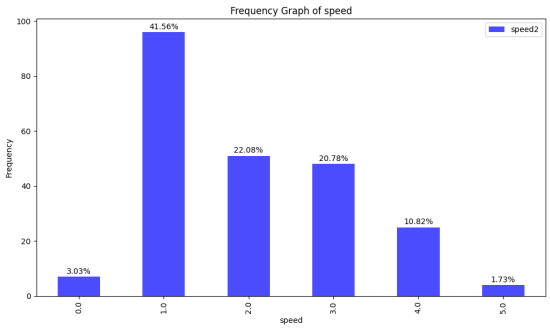

Figure 4.8: Graph of speed frequency for phase (2,false) from first training session DFDL

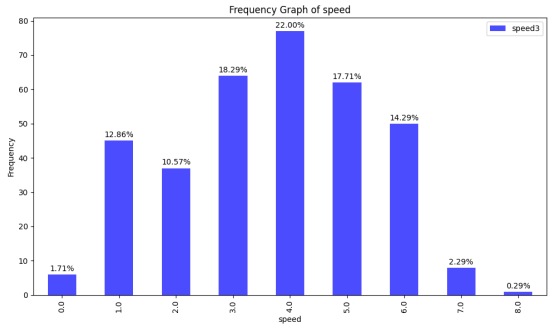

Figure 4.9: Graph of speed frequency for phase (3,true) from first training session DFDL

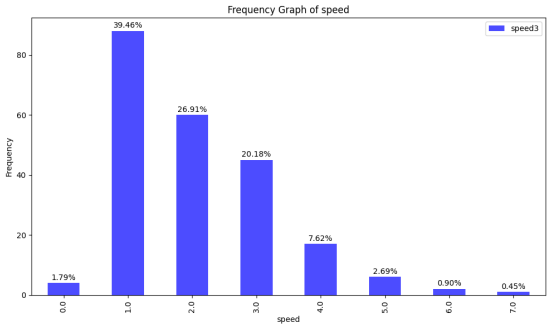

Figure 4.10: Graph of speed frequency for phase (3,false) from first training session DFDL

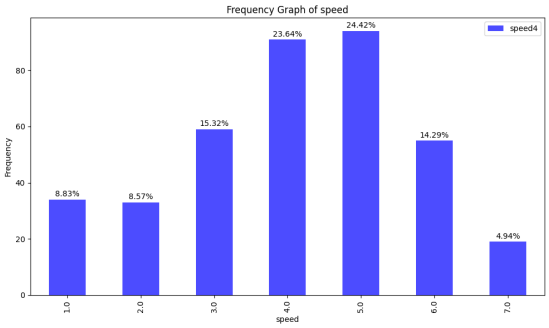

Figure 4.11: Graph of speed frequency for phase (4,true) from first training session DFDL

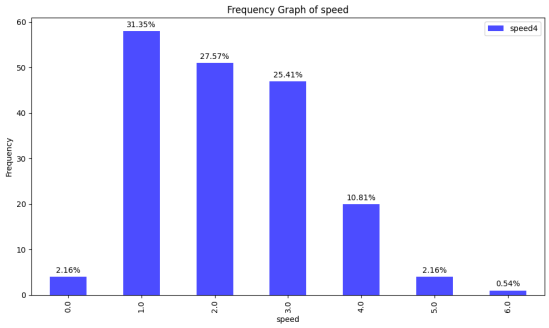

Figure 4.12: Graph of speed frequency for phase (4,false) from first training session DFDL

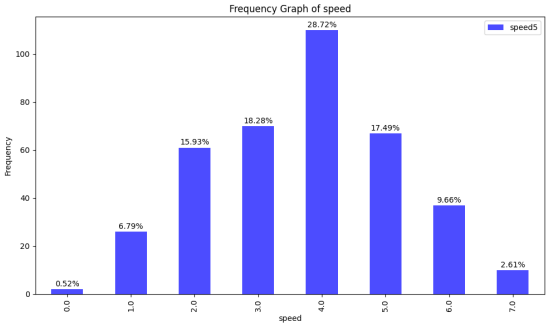

Figure 4.13: Graph of speed frequency for phase (5,true) from first training session DFDL

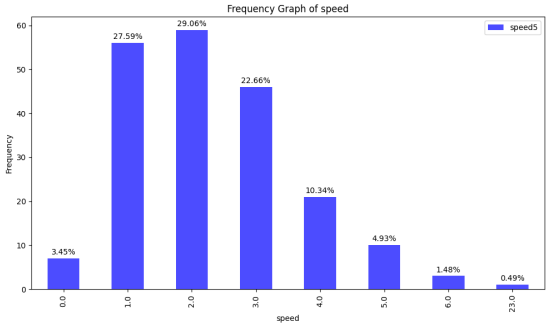

Figure 4.14: Graph of speed frequency for phase (5,false) from first training session DFDL

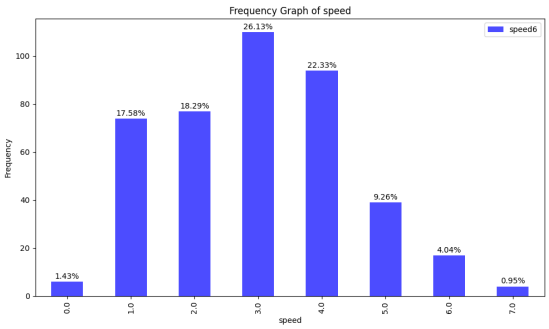

Figure 4.15: Graph of speed frequency for phase (6,true) from first training session DFDL

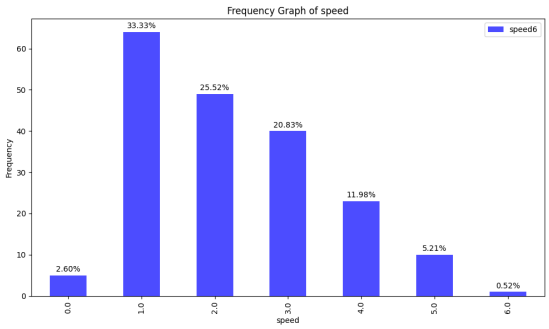

Figure 4.16: Graph of speed frequency for phase (6,false) from first training session DFDL

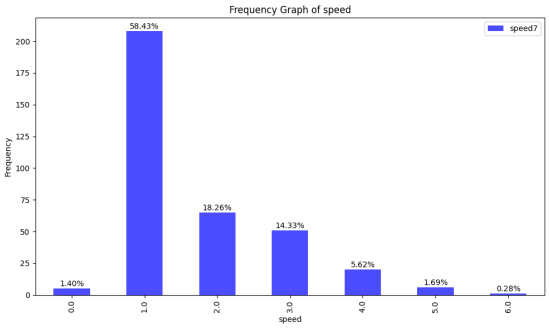

Figure 4.17: Graph of speed frequency for phase (7,true) from first training session DFDL

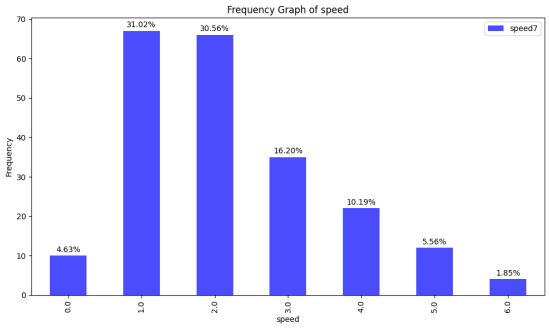

Figure 4.18: Graph of speed frequency for phase (7,false) from first training session DFDL

In terms of the alpha parameter, it is worth highlighting that the graphs associated with phases (2,true), as illustrated in Figure 4.23, as well as phases (3,false), (4,false), (5,false), and (6,false) represented by Figures 4.26, 4.28, 4.30, and 4.32 respectively, have exhibited promising outcomes. Specifically, they have demonstrated the lowest average alpha values in the shots taken during their respective phases.

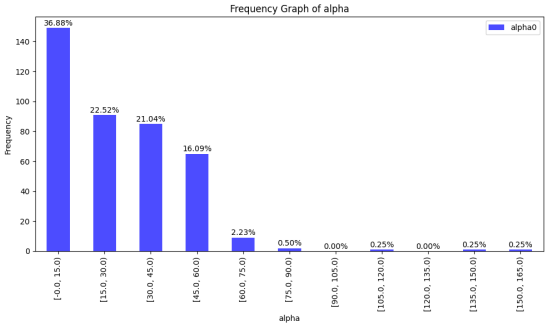

Figure 4.19: Graph of alpha frequency for phase (0,true) from first training session DFDL

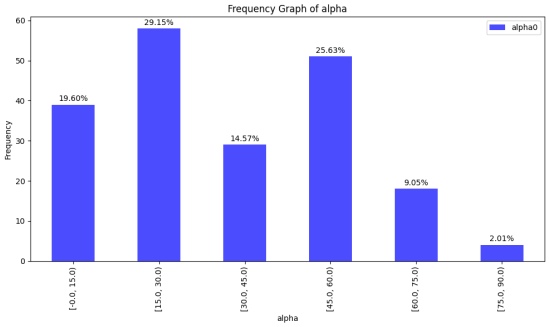

Figure 4.20: Graph of alpha frequency for phase (0,false) from first training session DFDL

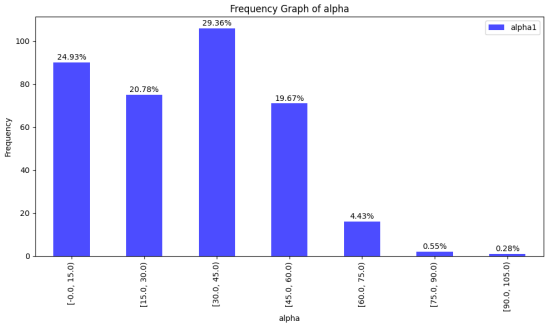

Figure 4.21: Graph of alpha frequency for phase (1,true) from first training session DFDL

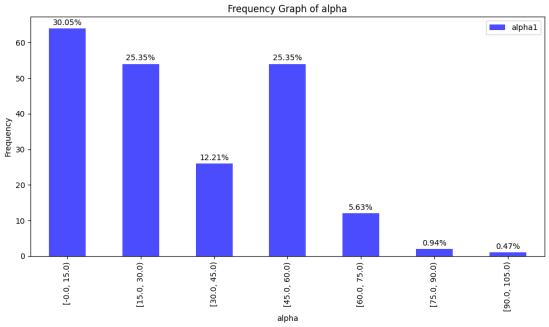

Figure 4.22: Graph of alpha frequency for phase (1,false) from first training session DFDL

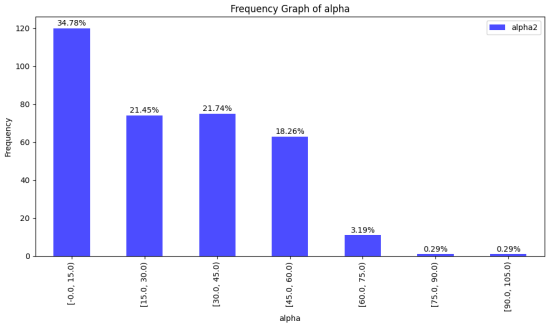

Figure 4.23: Graph of alpha frequency for phase (2,true) from first training session DFDL

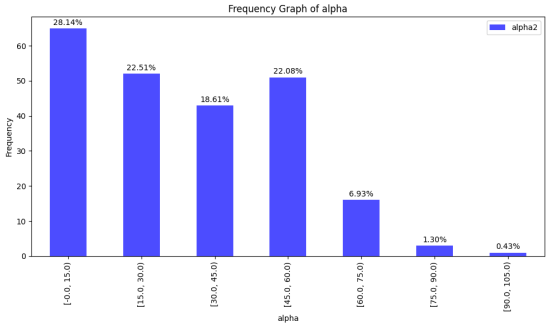

Figure 4.24: Graph of alpha frequency for phase (2,false) from first training session DFDL

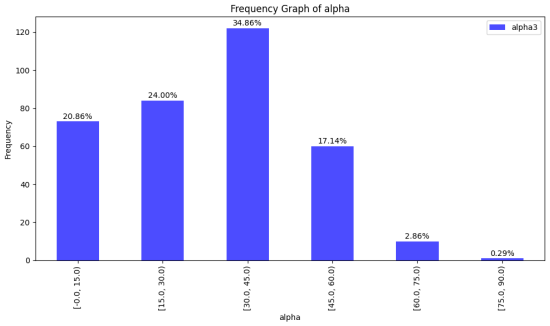

Figure 4.25: Graph of alpha frequency for phase (3,true) from first training session DFDL

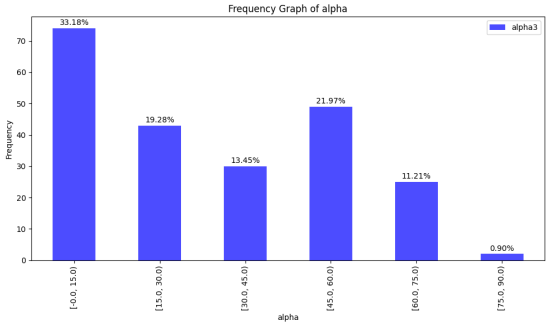

Figure 4.26: Graph of alpha frequency for phase (3,false) from first training session DFDL

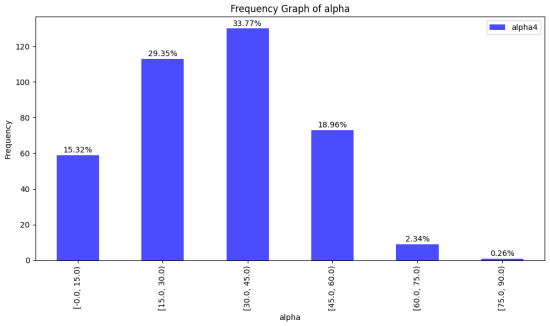

Figure 4.27: Graph of alpha frequency for phase (4,true) from first training session DFDL

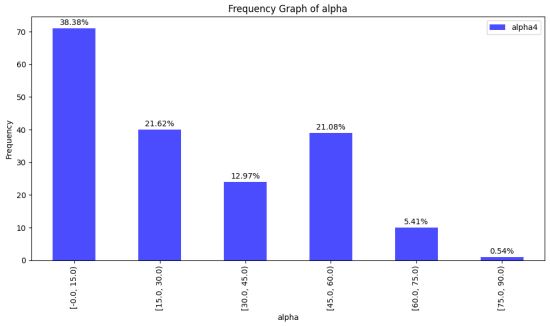

Figure 4.28: Graph of alpha frequency for phase (4,false) from first training session DFDL

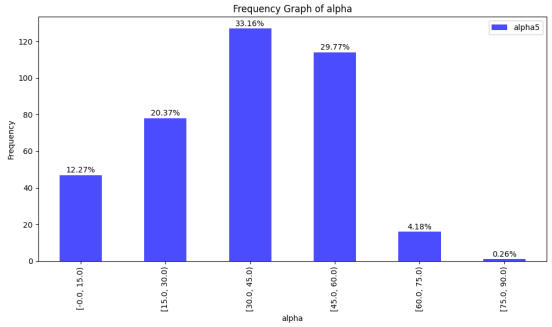

Figure 4.29: Graph of alpha frequency for phase (5,true) from first training session DFDL

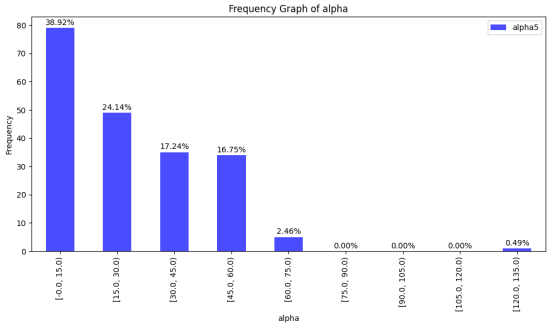

Figure 4.30: Graph of alpha frequency for phase (5,false) from first training session DFDL

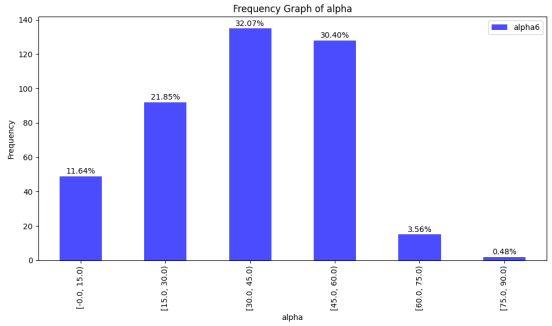

Figure 4.31: Graph of alpha frequency for phase (6,true) from first training session DFDL

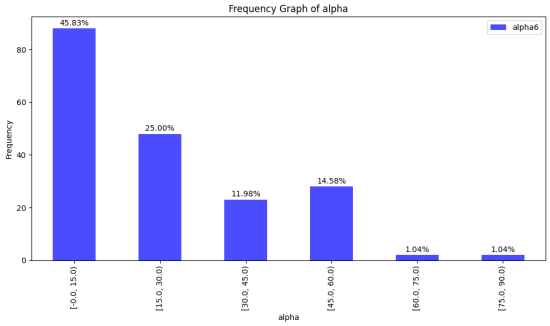

Figure 4.32: Graph of alpha frequency for phase (6,false) from first training session DFDL

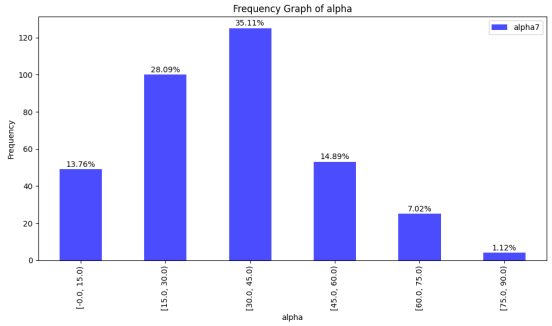

Figure 4.33: Graph of alpha frequency for phase (7,true) from first training session DFDL

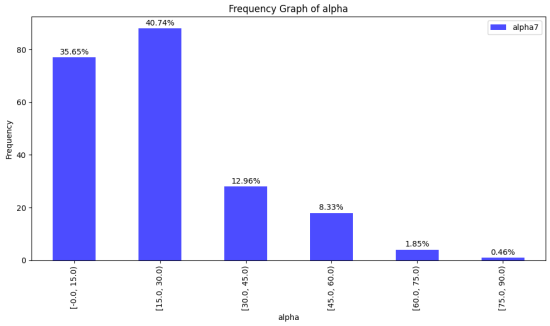

Figure 4.34: Graph of alpha frequency for phase (7,false) from first training session DFDL

Also, by analyzing the hit rates in relation to the different phases of the dribble, it becomes possible to identify which specific variable has a significant impact on the shooting accuracy, summarized in Table 4.1. We characterize a successful shot as one in which the alpha value falls within the initial range of 0º to 15º, coupled with a shot speed exceeding 2m/s.

This table includes the hit rates for each dribbling phase, categorized according to the active or inactive state of the left foot. This knowledge can then be utilized to guide subsequent training efforts, enabling the development of more refined and effective behaviors.

Dribble Phase	**Left Foot Active**	**Left Foot Inactive**
0	20.54%	7.04%
1	21.60%	21.50%
2	34.20%	17.75%
3	19.43%	21.08
4	14.28%	30.81%
5	10.93%	23.66%
6	9.98%	31.25%
7	5.34%	24.07%
Average	16.80%	29.34%

Table 4.1: DFDL general training hit rate

Additionally, all shots were executed swiftly, without needing the agent to fully utilize the available 10 time steps per episode. This rapid shot execution was a key challenge in this dissertation, and significantly outperformed the team's other shots in terms of time.

4.9 Conclusion

Based on the previous results, it's clear that phase (2,true) of dribbling, especially with the left foot active, shows the most promise for accurate shots. The speed and alpha graphs, along with the hit rate, suggest that this phase has the highest potential compared to others. Visually, this phase also looks smoother and more fluid when shooting.

Furthermore, the results regarding the success rate of phases (4,false) and (6,false) also suggest potential for further exploration and optimization. However, visually, these two phases don't allow for a fluid shot motion. This is due to the agent's left leg serving as support during these phases, requiring a quick transition to the right leg as support for the shot.

Also if focusing solely on training this specific phases of dribbling, there's a likelihood that the neural network, starting from scratch, might predominantly learn to use the right foot. This would render the values from the initial training ineffective, as the agent's learned behavior would differ from the left-footed kicking taught during that phase.

Taking these observations into consideration, the decision has been made to optimize the behavior by focusing exclusively on training specific on (2,true) phase of the dribble. By narrowing

the training scope to this phase, the aim is to further enhance the shooting performance and improve the overall results obtained.

This strategic adjustment reflects a targeted approach to training, leveraging the strengths and effectiveness observed in phase 2 of the dribble with the active left foot. By dedicating efforts to optimizing this particular phase, the goal is to achieve even greater success rates and improved shooting fluency.

Chapter 5

Kicking optimization of Dribbling phase (2,True)

5.1 Problem

Based on the analysis of the previous chapter's results, it is evident that focusing the training solely on some phases of the dribble can lead to a more efficient shot. By narrowing the agent's focus to shooting only in one phase, the variability in training is reduced, resulting in increased efficiency.

Additionally, it is worth noting that during the dribble, the left foot was consistently used for the shot, regardless of the direction of the dribble. Therefore, in this chapter, the objective is to specifically train the left foot shot when the dribble turns to the right. Following the completion of this training, the neural network can be reversed to enable the use of the right foot after a symmetrical dribble to the left, as explained in 3.6.10.

This approach allows for targeted training to enhance the effectiveness of the left foot shot in specific situations, while also providing the flexibility to adapt the learned behavior for the right foot shot when required. By optimizing the training for these scenarios, the agent can further refine its shooting capabilities and adapt to different dribbling situations with improved precision.

5.2 Action Space, State Space, Network Shape and Hyperparameter Tuning

As employed in the previous model, all joints will be used except the 2 in the head for the action space. For the State Space of this model, it will be considered the same observations as the previous models are used. For the Network Shape and Hyperparameter, we used the same presets as in the previous training.

5.3 First training

5.3.1 Episode layout

During the subsequent training sessions, we employed phase 2 with the left foot active.

The objective was to execute the shot only when the robot was in this specific phase of the dribble and turning to the right.

The upcoming episode will follow a similar pattern as the previous training runs, with one distinction: the agent will now dribble the ball while facing a random orientation ranging from -180 to 0 degrees. This adjustment compels the robot to dribble in a rightward direction before taking a shot. All other factors and parameters will remain unchanged.

5.3.2 Reward

At the outset, we utilized the same reward function as employed in the previous training phase, as referenced in the 4.1, which did produce favorable results.

5.3.3 Results of the first training

To begin with, solely employing the initial reward function, denoted as 4.1, resulted in the agent learning to prefer using their left foot, as anticipated. Once it became evident that the average reward value had peaked,as illustrated in 5.1, the training was concluded.

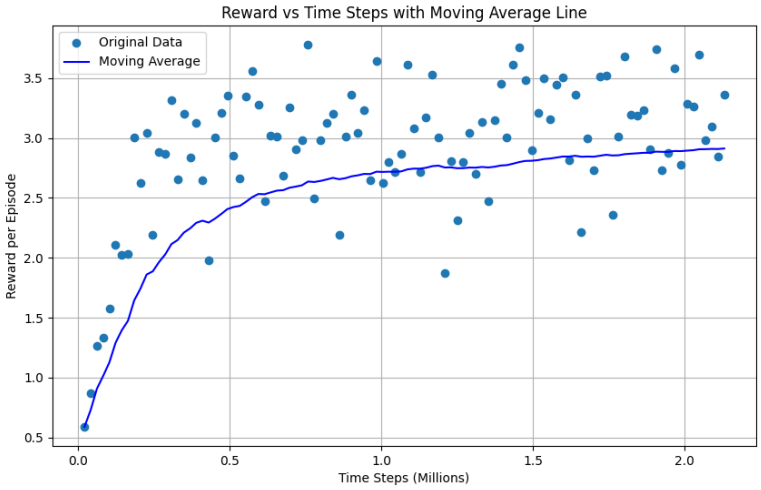

Figure 5.1: Reward Evolution Over Time Steps during second first training session of phase (2,true) of DFDL

A comprehensive test involving approximately 200 shots was conducted. This test allowed for the generation of graphs representing the frequency of shot speeds and the α angle.

The speed frequency graph, showcased some achievements in terms of shot speeds attained. The most frequently recorded speed was 4m/s, yet it was also noteworthy that shooting at 6m/s

occurred quite frequently, as evidenced in 5.2. On average, a speed of 3.836 m/s was achieved, with shots exhibiting the correct direction ($0 < \alpha < 15$) averaging 3.255 m/s.

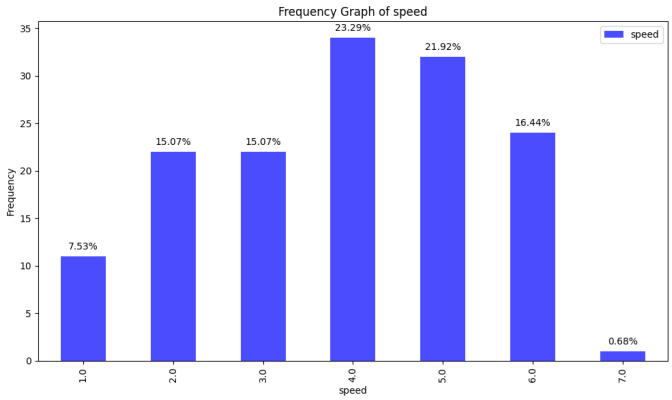

Figure 5.2: Graph of speed frequency for optimized phase (2,true) for the first training

Regarding the α angle, in figure 5.3, it's evident that only approximately 16% of the shots exhibited an acceptable direction ($0 < \alpha < 15$). This is a key factor contributing to the less-than-ideal outcomes in this chapter.

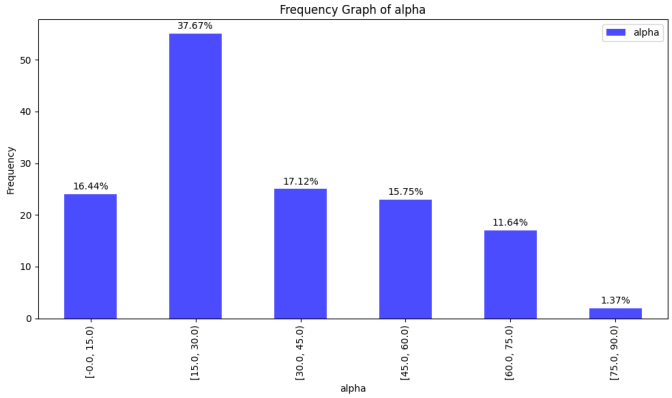

Figure 5.3: Graph of alpha frequency for optimized phase (2,true) for the first training

The hit rates for shots meeting the criteria of $\alpha < 15$ and speed > 2m/s were approximately 12.925% for the first trained shot.

5.3.4 Analysis of the first training

Upon analyzing the results presented in the previous section, it becomes evident that there is potential for optimizing the shot during the (2,true) phase of the dribble. Initially, using the first reward function, we were able to achieve high shot speeds. However, this came at the cost of accuracy, prompting us to retrain the neural network with an alternative reward function.

5.4 Second training

5.4.1 Changing Reward Function

Given the unsatisfactory performance of the α angle during the initial training phase, we decided to undergo a retraining process using a second reward function. We introduced a new reward function, which is outlined as follows:

$$r = v * 0.93^{\alpha} \tag{5.1}$$

Within the training episode's context, it continues to represent the velocity at which the ball reached its destination following the shot.

Additionally, the variable α continues to denote the difference between the absolute orientation of the ball direction and the absolute orientation of the robot.

The updated reward scheme, as visualized in the graph, enables the achievement of maximum speed when the deviation α is at 0 degrees. However, it notably diminishes, offering just half of the speed value when the deviation is as minimal as 10 degrees, as seen in 5.1. This adaptation amplifies the penalty for shots with higher α values, while incentivizing those with lower α values. Nevertheless, the overall reward value remains in proportion to the shot's speed. This exponential function provides added motivation for the agent to master challenging behaviors, with rewards rapidly escalating as the agent approaches the desired performance. Conversely, it can also be harnessed to strongly discourage undesirable behaviors.

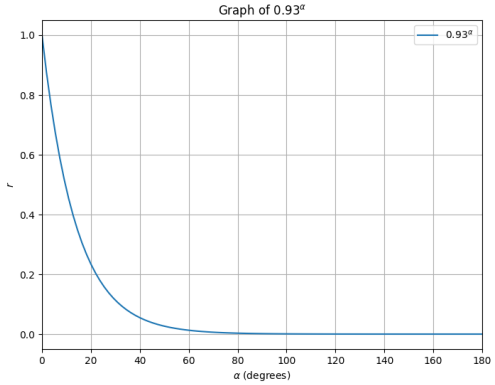

Figure 5.4: plot of the function $y = 0.93^{\alpha}$

5.4.2 Results of the second training

As illustrated in Figure 5.5, the average reward per episode rapidly increases until reaching approximately one million steps and stabilizes at a constant value.

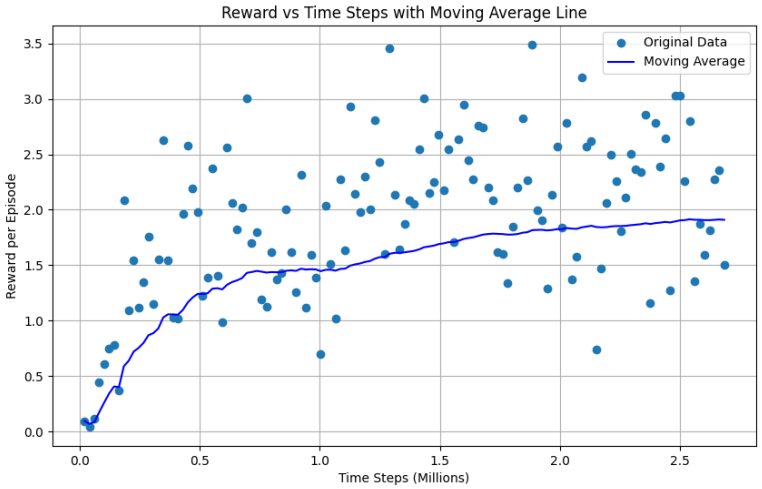

Figure 5.5: Reward Evolution Over Time Steps during second training session of phase (2,true) of DFDL

A post-training analysis enabled us to scrutinize the algorithm's best-generated model, where the agent shoots with its left foot. By subjecting this model to testing with approximately 500 shots, a statistical analysis was conducted.

To start, when examining the velocity graph as referenced in 5.6, the most frequently recorded velocity was 4 m/s. The average velocity stood at 3.608 m/s, and the average velocity for the correct direction shot ($0 < \alpha < 15$) was 4.330 m/s.

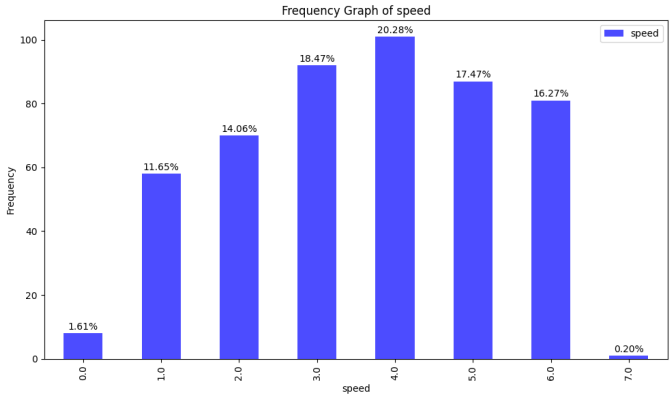

Figure 5.6: Frequency distribution of speed for the optimized phase (2, true) during the second training session.

In the second training session, there were significant improvements in the frequency distribution concerning α. Approximately 70% of the shots hit the correct direction by the end of the episode, with $\alpha < 15$, as observed in 5.7.

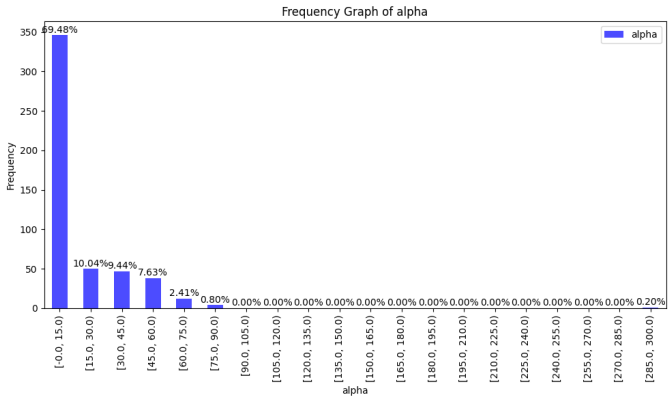

Figure 5.7: Frequency distribution of angle (α) for the optimized phase (2, true) during the second training session.

The shot accuracy rate, with $\alpha < 15$ and speed > 2 m/s, reached 64.328%

5.4.3 Analysis of the second training

In this second training session, a noticeable improvement in overall performance becomes evident. A commendable level of shot velocity has been maintained in relation to shot accuracy. Furthermore, there has been a substantial enhancement in shot precision, with the agent correctly targeting its shots approximately 70% of the time. This positive shift can be attributed to a modification in the reward function, which compels the agent to execute more precise shots in order to attain higher rewards.

One approach would involve changing the side from which the agent performs the dribble. Instead of dribbling to the right, it would dribble to the left. Although this change might not appear visually smooth, it requires the agent to use its left foot for shooting while turning to the left. The agent will continue to learn to shoot with the left foot, thanks to the inherent nature of the PPO algorithm. When training an agent, it's crucial to strike a balance between exploration (trying new actions to learn) and exploitation (using the best-known actions to maximize rewards). In this training scenario, the agent quickly realizes that using one limb yields relatively higher rewards, leading it to exploit that strategy and adhere to it, consequently neglecting the other limb. Nevertheless, over time, this strategy has the potential to yield positive results.

The second approach entails optimizing the reward function even further. This optimization could entail making the reward function more sensitive to shot accuracy, discouraging high rewards for shots with significant alpha values. By carefully adjusting the reward function, you can guide the agent toward more accurate shots.

Both approaches have their merits, and it may be advantageous to experiment with both to determine which one produces superior outcomes. Additionally, the choice of approach may depend on the specific requirements and constraints of your application and the desired behavior for the agent.

5.5 Third training

5.5.1 Changing Episode Layout

To further improve the performance of the shooting model in the (2,true) phase, a modification was introduced during training while retaining the same reward function as the previous training 5.1, which had shown promising results. This modification is a straightforward yet potentially impactful change. Instead of having the agent dribble the ball to the right, it now executes the opposite action by turning to the left. Essentially, it shifts from turning right within a range of -180 to 0 degrees to turning left within a range of 0 to 180 degrees, all while maintaining the previously mentioned dribbling duration of 0 to 2 seconds before transitioning into the shot learning phase. All other variables remain unchanged.

Although this adjustment may not appear as visually fluid, the change in dribbling direction could potentially reduce the distance between the player and the ball as they enter the shooting phase. This proximity might facilitate better shot framing and, consequently, improve overall performance.

5.5.2 Results of the third training atempt

Following this change in the dribbling direction during the initial learning phase, the agent successfully developed the ability to perform shots using its left foot.

In comparison to the previous symmetric training of turning to the right, this training was relatively longer, indicating a more extended learning curve, though not as steep. We decided to halt the training around 4 million steps when the average reward per episode stabilized, as can be seen in 5.8

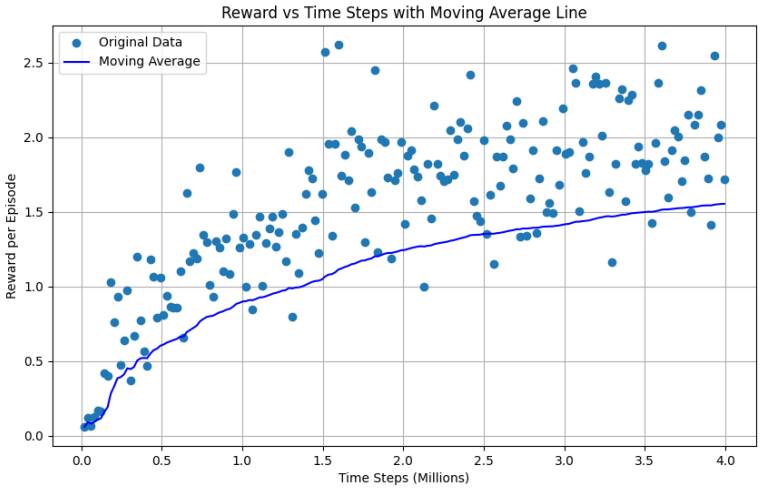

Figure 5.8: Reward Evolution Over Time Steps from third training session of phase (2,true) of DFDL

After this point, we conducted testing on approximately 450 shots.

The speed-frequency graph, as shown in Figure 5.9, displayed less favorable outcomes compared to the previous training. It revealed that the most frequently measured speed was 3 m/s. The average speed of the shots was 3.004 m/s, and the average speed of successful shots ($\alpha < 15$) was 3.530 m/s.

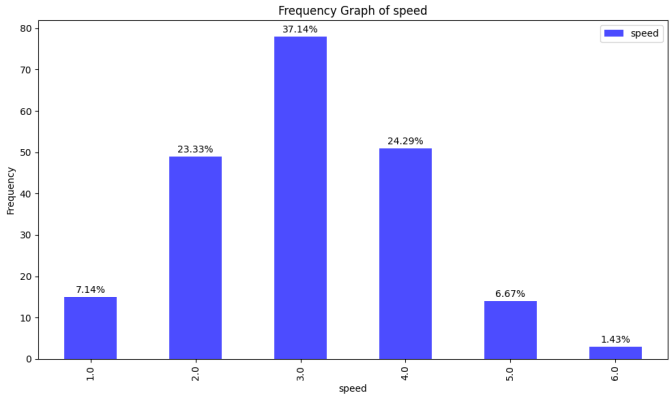

Figure 5.9: Graph of speed frequency for optimized phase (2,true) for the third training

Regarding the value of the angle α, as illustrated in Figure 5.10, a similar outcome to the previous training was achieved, with approximately 70% of the shots demonstrating an acceptable direction ($0 < \alpha < 15$).

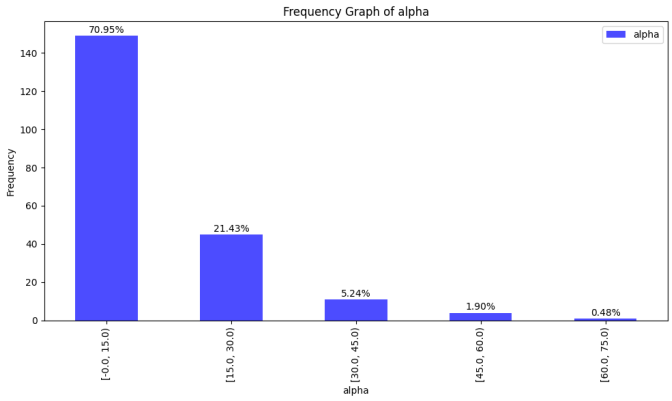

Figure 5.10: Graph of alpha frequency for optimized phase (2,true) for the third training

The success rate for shots meeting the conditions of $\alpha < 15$ and a speed exceeding 2m/s stood at approximately 67.299%.

5.5.3 Analysis of the third training

This third test served as an experiment to determine if reversing the side to which the agent turned when dribbling could influence the shot performance. It was revealed that there are no significant gains to be made by doing so, as despite the shot accuracy being relatively positive, the ball's velocity values were not optimal. This test also showed slightly higher accuracy than the previous one, but it won't be significant enough to change the initial approach of training the shot by turning to one side during dribbling, optimizing that aspect to the maximum, and then reversing the behavior to achieve it with the symmetrical limb.

5.6 Fourth training

5.6.1 Changing Reward Function

To improve the success rates in this promising shot phase, we initiated a fourth round of adjustments, in which we further modified the reward function to penalize shots that do not have an acceptable angle (α). This new function will be a variation of the one previously used, as shown in 5.1. To accomplish this, we introduced the following function:

$$r = v * 0.80^{\alpha} \tag{5.2}$$

In the context of the training episode, v remains a representation of the velocity attained by the ball upon reaching its destination post-shot.

Similar to function 5.1, the new reward setup allows us to reach the full speed value when the deviation (α) is 0. However, it decreases even more rapidly, providing only half the speed value at a deviation angle (α) of just 3 degrees, as can be observed in the graph in Figure 5.11.

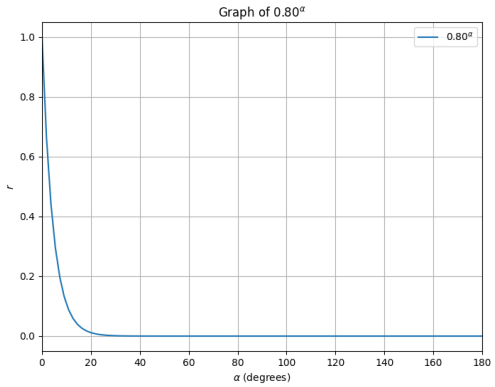

Figure 5.11: plot of the function $y = 0.80^{\alpha}$

It's worth noting that this function may give rise to a "lack of information" problem. Since the reward is the sole means by which the robot perceives its performance, and this function is highly restrictive, the agent may fail to learn actions that are critical for overall success. This could lead to suboptimal training outcomes.

To address this potential issue, we can retrain the previously trained neural network, allowing for a concept known as "Weight Initialization." This approach equips the agent with a solid foundation for learning the new task, as the network has already captured valuable insights from prior tasks. Additionally, we anticipate a significant reduction in training times.

5.6.2 Results of the fourth training atempt

Examining the graph in Figure 5.12, and despite a small initial growth, I decided to terminate the training relatively early due to the lack of substantial increase in the average reward.

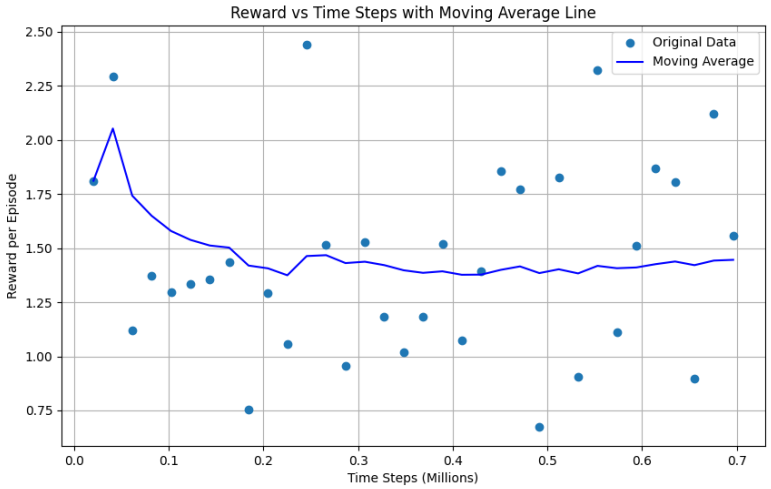

Figure 5.12: Reward Evolution Over Time Steps from fourth training session of phase (2,true) of DFDL

From the trained model, we conducted a test comprising approximately 385 shots.

A statistical examination of the velocity frequency, depicted in Figure 5.13, indicates that the 4 m/s shot is the most frequently occurring.

The average shot velocity is 3.461 m/s, and the average velocity of successful shots ($0° < \alpha < 15°$) is 4.446 m/s.

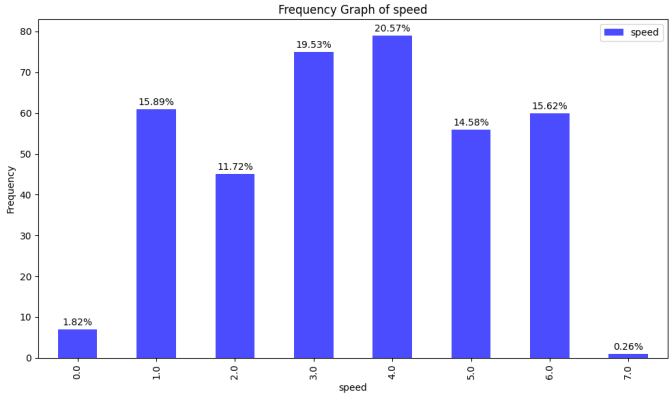

Figure 5.13: Graph of speed frequency for optimized phase (2,true) for the fourth training

Concerning the α angle value, as depicted in Figure 5.14, approximately 65% of the shots demonstrated an acceptable direction ($0° < \alpha < 15°$).

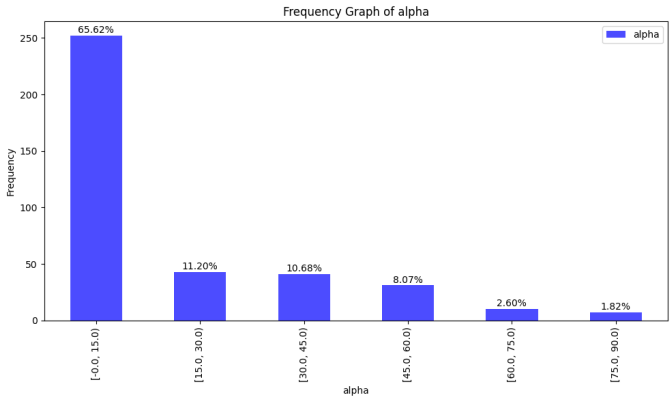

Figure 5.14: Graph of alpha frequency for optimized phase (2,true) for the fourth training

The success rate for shots meeting the conditions of $\alpha < 15°$ and a velocity exceeding 2 m/s stood at approximately 58.961%.

5.6.3 Analysis of the fourth training attempt

The results presented earlier reveal the inadequacy of this new reward function, as the agent was unable to successfully optimize the shot. It becomes evident that despite retraining a previously trained neural network, this function is poorly designed and does not provide informative and consistent feedback. Consequently, the agent encountered difficulties in determining which actions are desirable. This issue has hindered both consistent training and learning, as indicated by the significant variability and lack of growth in the average reward value per episode over time.

As for the robot's velocity, these results are consistent with the outcomes observed in the previous training, where the pre-trained neural network underwent further training.

Regarding the precision and hit rate values, there was a slight depreciation in the final output.

Additional tests were conducted using this reward function, including training a neural network from scratch, without pre-training. Various other reward functions, which were slight variations of function 5.1, were also explored. However, none of these yielded significant or noteworthy results.

5.7 Conclusion

A summary of the optimizations performed can be observed in Table 5.1, which presents, for each of the previous training sessions, the hit rate, average speed, and average speed of successful shots.

After thoroughly examining all the analyses conducted on various training variable variations, it's evident that the model developed and demonstrated in 5.4.2 has proven to be the best.

The achieved accuracy in this model makes this shot suitable for use in competitions. Considering that only 10% of shots exhibit a speed lower than 2m/s, it establishes this shot as a valuable

Training Trial	Hit Rate	Average Speed	Average Speed of Successful Shots
1	12.925%	3.836 m/s	3.255 m/s
2	64.328%	3.608 m/s	4.330 m/s
3	67.299%	3.0 m/s	3.530 m/s
4	58.961%	3.461 m/s	4.446 m/s

Table 5.1: Training Optimization Summary of DFDL

defensive option for clearing the ball from dangerous areas, particularly when combined with this dribbling phase.

In comparison to the model from training session 3, the one from training session 2 is preferable due to the higher shot speeds it achieves. Although training session 3 has a slightly better hit rate, what matters more in defensive situations is the speed at which the ball is cleared, which leads to a quicker clearance, rather than the precision of the shot.

However, it's worth noting that this shot developed in the second training attempt may not be the most accurate when preparing to execute the shot, as it only hits the target about 70% of the time. This limitation restricts its use, for example, when the robot has an open goal and it's wanted a fast shot with high confidence in scoring. Achieving a higher accuracy rate would be necessary for such scenarios. Additionally, due to the challenge of achieving great precision during this dribbling phase, it may not be possible to further optimize the shot to execute with specific directions; it will generally be directed straight ahead during this dribbling phase.

Chapter 6

Kick in motion after Left Footed Dribble Long

6.1 Problem

Continuing the investigation within the scope of this thesis, aimed at comprehending the techniques of dribbling that could enhance a more efficient shooting motion, we have now initiated training the agent for the transition between dribbling and shooting during specific phases of dribble LFDL. This approach is detailed in section 3.7.4.2.

The specific dribbling technique in question involves using only the left foot for controlling the ball, executed at a considerably greater distance from the ball compared to the previous dribbling method studied. This aspect could potentially pose challenges to the transition between dribbling and shooting during certain phases of the dribbling process.

Nonetheless, prior to drawing such conclusions, a more inclusive and comprehensive training similar to the one described in section 4 will be essential. This will aid in determining the specific phases of dribbling where the player is more likely to excel. Following an assessment of these various factors, we can ascertain whether there exists potential for optimization and practical application of this dribbling technique. Our primary focus remains on the speed at which the robot executes the shot, ensuring both shot velocity and direction are significantly accurate for competitive use.

6.2 Action Space, State Space, Network Shape and Hyperparameter Tuning

Similar to the 4 model, all joints except the two in the head are considered for the action space. The state space of this model employs the same observations as the previous models. Additionally, the network shape and hyperparameters are kept consistent with the presets used in the previous training sessions.

6.3 Episode layout

Just like in the training explained in section 4, the robot will start 0.3 meters away from the ball and dribble it for a random duration between 0 and 2 seconds. It will dribble in any direction, from -180 to 180 degrees.

After this, the training switches to teaching the robot how to shot. The agent has only 2 seconds, or 10 time steps, to shot. This short time frame is meant to make the robot shot as quickly as possible.

6.4 Reward

Similar to the previous dribbling training, we tested various functions, but we are now focusing solely on the one that had the most substantial impact. Notably, this function is in line with the equation 4.1 mentioned earlier, which demonstrated greater success in this initial training phase, where we aim to optimize future motion shots.

6.5 Results

The training was concluded once the average reward curve per episode had stabilized, which occurred after 1.75 million steps.

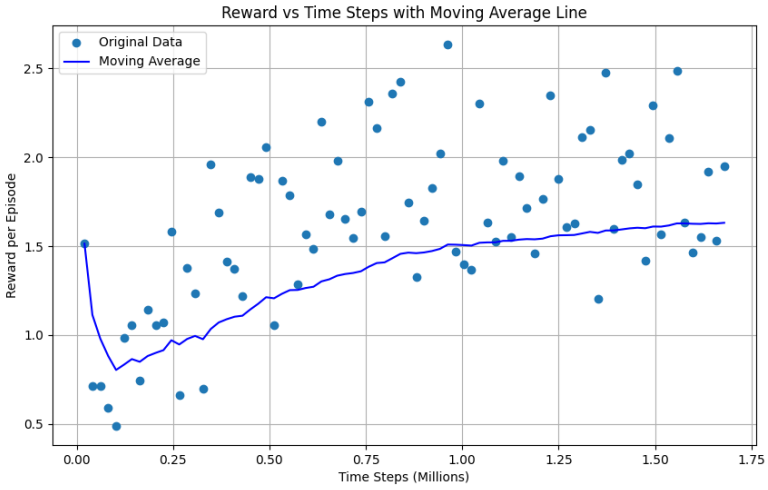

Figure 6.1: Reward Evolution Over Time Steps from first training session of phase of LFDL

In the face of utilizing solely the left foot for dribbling, the neural network successfully acquired the skill of executing shots using only the left foot. This phenomenon can be ascribed to the inherent tendencies of neural network algorithms, which find it more convenient to specialize in a

distinct action – in this context, shooting with a specific foot – rather than grappling with the complexities of learning multiple variations. However, despite this accomplishment, the utilization of left-footed shots failed to facilitate a seamless transition between dribbling and shooting during some phases of the dribbling process. This challenge arises from the player's control of the ball at an extended distance in this specific dribbling instance, coupled with the constraints imposed by the limited training duration that prevented effective shooting practice. The imposition of a time constraint is crucial in achieving one of the objectives of this thesis, which is to ensure effective and fast shooting.

The analysis encompassed the shot velocity and the α value of the ball. In this context, α signifies the disparity between the absolute orientation of the ball's trajectory and that of the robot, as previously elucidated. The dataset consists of over 1000 collected shots.

The velocity and α graphs for each of the following dribbling phases are presented below:

In terms of the velocity graphs, only the (5,true), (6,true) and (7,true) phases, seen in 6.12, 6.30 and 6.32, displayed positive outcomes. This can be attributed to the fact that during these dribbling phases, the feet are closely positioned, enabling better shot preparation, particularly for left-footed shots. The success was confined to these phases because the agent's distance from the ball upon its departure from dribbling is significantly greater compared to the previous dribbling scenario 3.7.4.1. As a result, when the feet aren't relatively close together, the robot refrains from initiating the shot. This explains the notable prevalence of shots at 1m/s in the graphs. A velocity of 1m/s was measured when the ball maintained the same speed it had when it left the dribble, and the robot was incapable of further increasing its velocity.

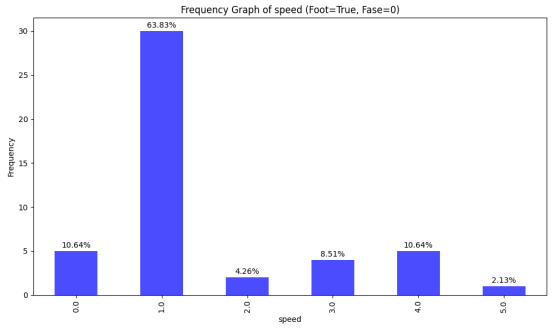

Figure 6.2: Graph of speed frequency for phase (0,true) from first training session LFDL

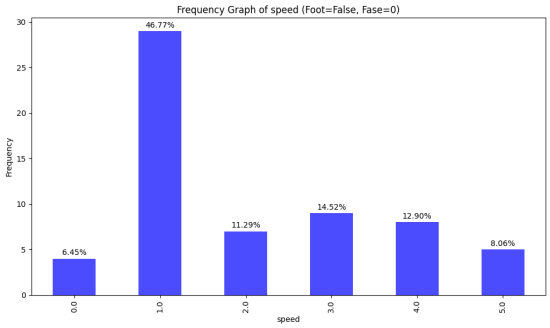

Figure 6.3: Graph of speed frequency for phase (0,false) from first training session LFDL

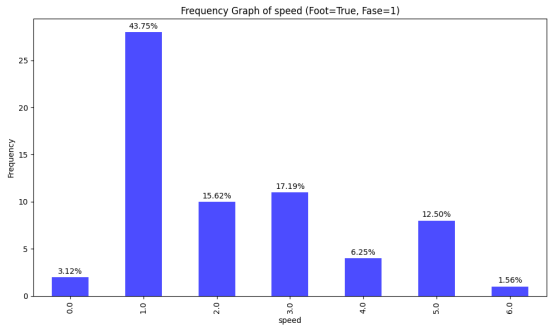

Figure 6.4: Graph of speed frequency for phase (1,true) from first training session LFDL

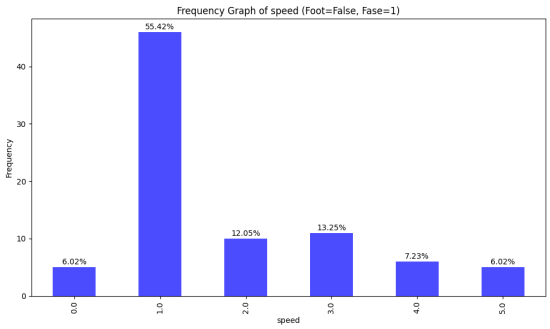

Figure 6.5: Graph of speed frequency for phase (1,false) from first training session LFDL

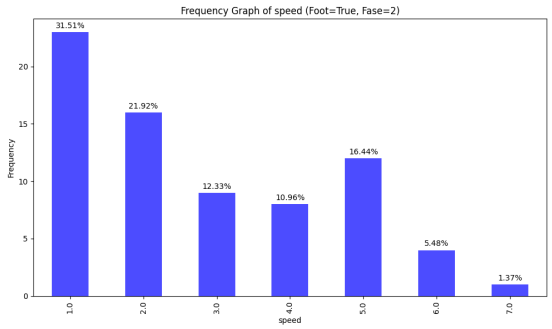

Figure 6.6: Graph of speed frequency for phase (2,true) from first training session LFDL

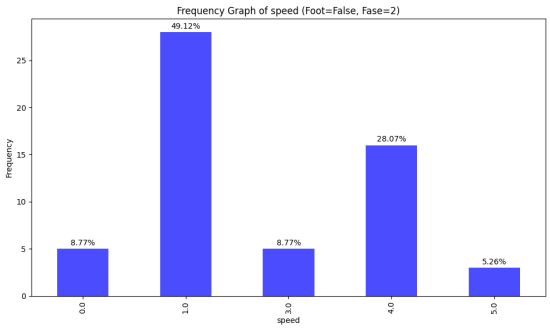

Figure 6.7: Graph of speed frequency for phase (2,false) from first training session LFDL

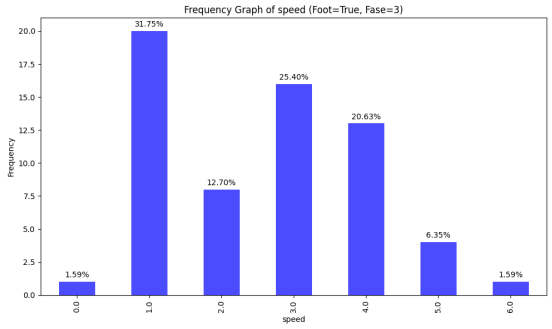

Figure 6.8: Graph of speed frequency for phase (3,true) from first training session LFDL

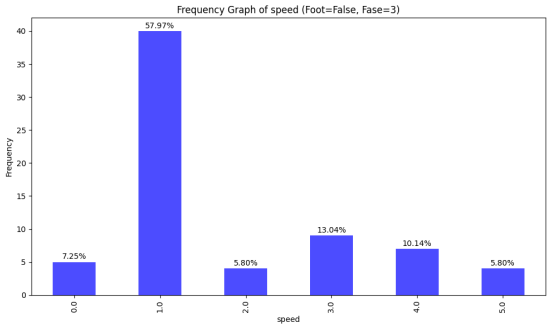

Figure 6.9: Graph of speed frequency for phase (3,false) from first training session LFDL

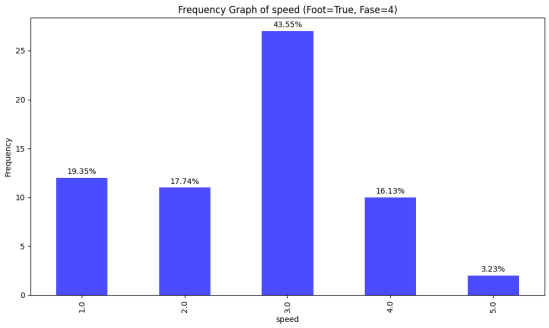

Figure 6.10: Graph of speed frequency for phase (4,true) from first training session LFDL

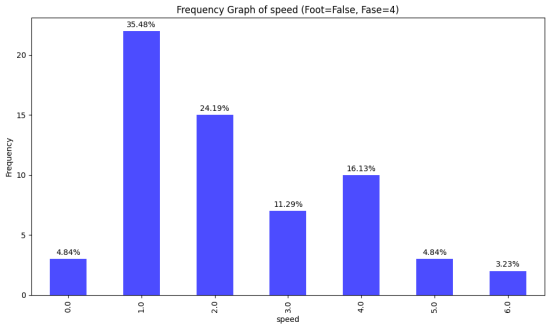

Figure 6.11: Graph of speed frequency for phase (4,false) from first training session LFDL

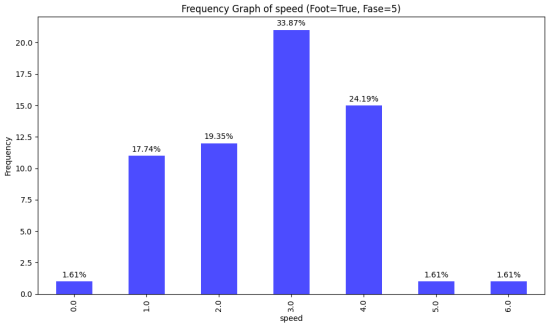

Figure 6.12: Graph of speed frequency for phase (5,true) from first training session LFDL

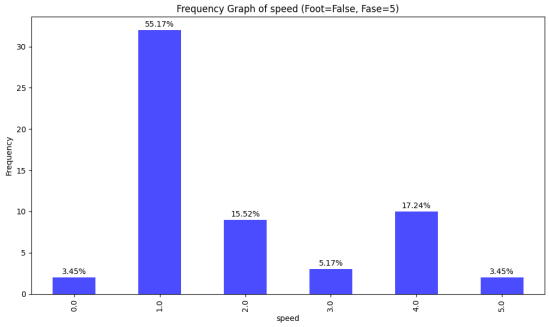

Figure 6.13: Graph of speed frequency for phase (5,false) from first training session LFDL

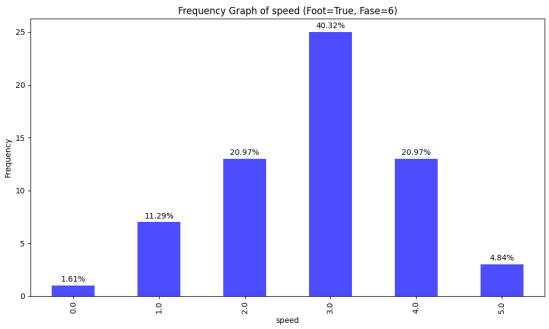

Figure 6.14: Graph of speed frequency for phase (6,true) from first training session LFDL

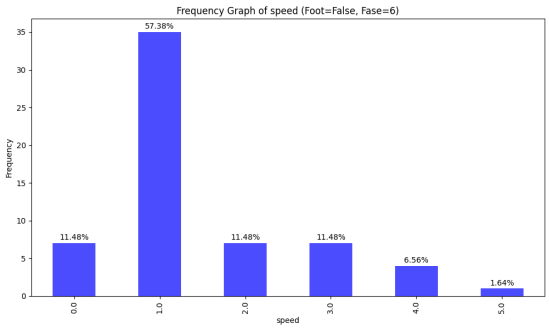

Figure 6.15: Graph of speed frequency for phase (6,false) from first training session LFDL

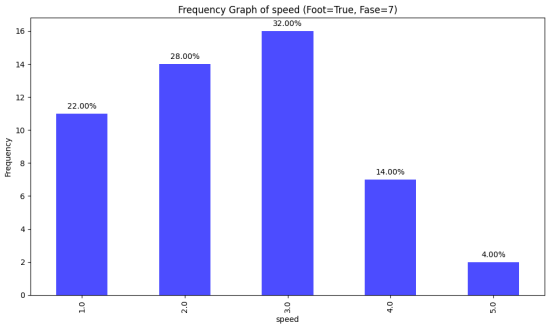

Figure 6.16: Graph of speed frequency for phase (7,true) from first training session LFDL

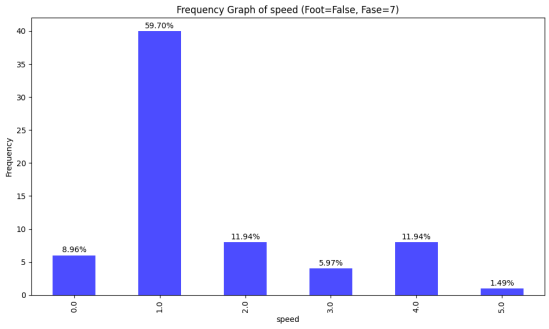

Figure 6.17: Graph of speed frequency for phase (7,false) from first training session LFDL

Moving on to the analysis of the α values, it's apparent that most of the dribbles exhibit low α values across their shots. This outcome can be attributed to the ball simply maintaining the direction set by the agent at the conclusion of the dribble. Nevertheless, the agent struggles to impart more speed to the ball, leading to missed shots.

As a result, our focus for consideration narrows down to the (5,true) and (6,true) phases. These two phases consistently display low α values for the majority of their shots, as depicted in the graphs labeled as 6.28 and 6.30.

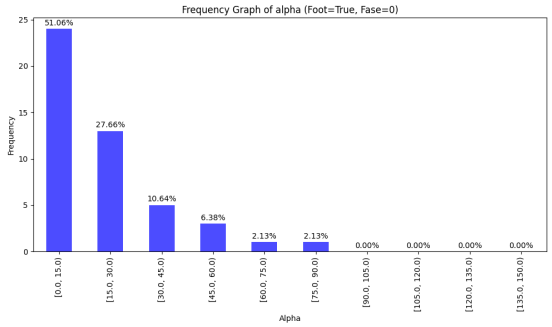

Figure 6.18: Graph of alpha frequency for phase (0,true) from first training session LFDL

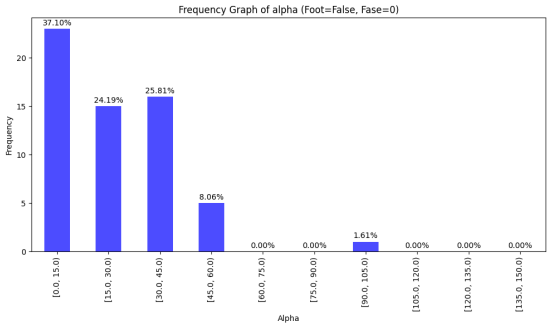

Figure 6.19: Graph of alpha frequency for phase (0,false) from first training session LFDL

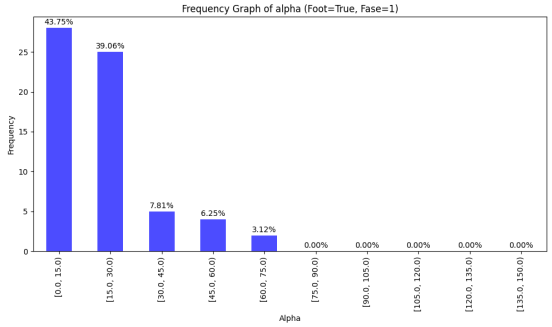

Figure 6.20: Graph of alpha frequency for phase (1,true) from first training session LFDL

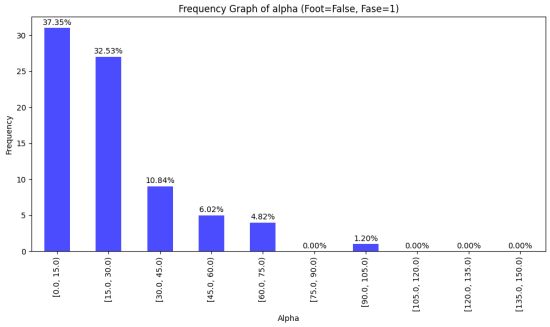

Figure 6.21: Graph of alpha frequency for phase (1,false) from first training session LFDL

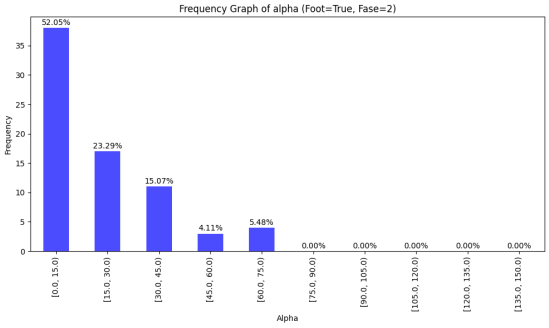

Figure 6.22: Graph of alpha frequency for phase (2,true) from first training session LFDL

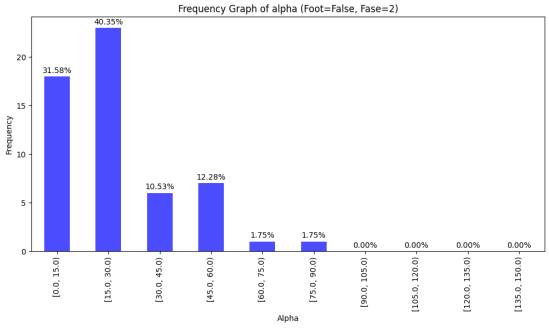

Figure 6.23: Graph of alpha frequency for phase (2,false) from first training session LFDL

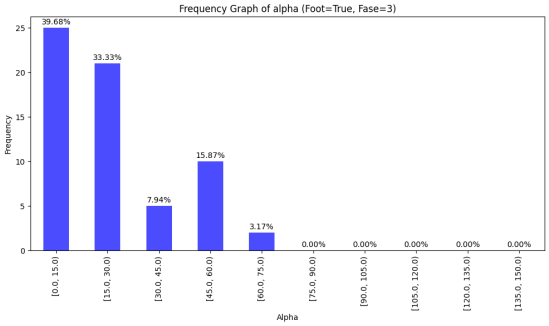

Figure 6.24: Graph of alpha frequency for phase (3,true) from first training session LFDL

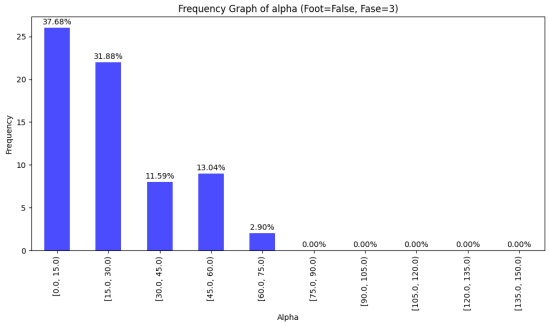

Figure 6.25: Graph of alpha frequency for phase (3,false) from first training session LFDL

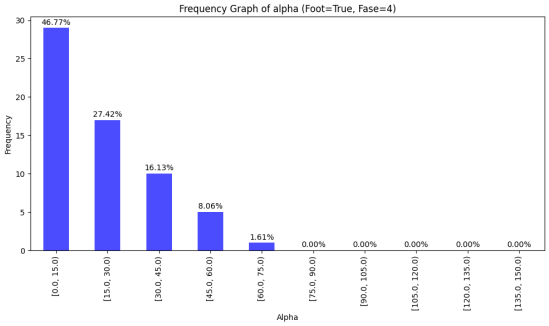

Figure 6.26: Graph of alpha frequency for phase (4,true) from first training session LFDL

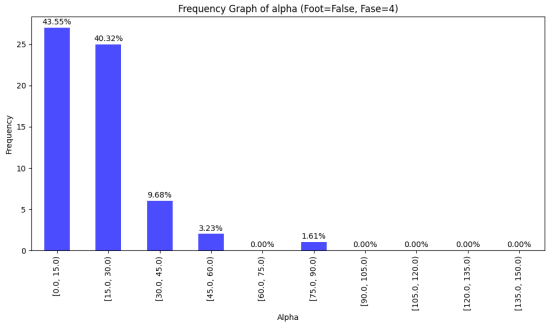

Figure 6.27: Graph of alpha frequency for phase (4,false) from first training session LFDL

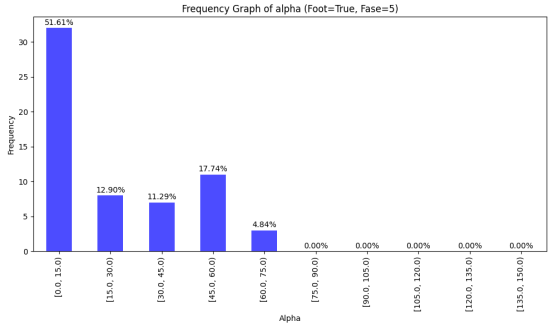

Figure 6.28: Graph of alpha frequency for phase (5,true) from first training session LFDL

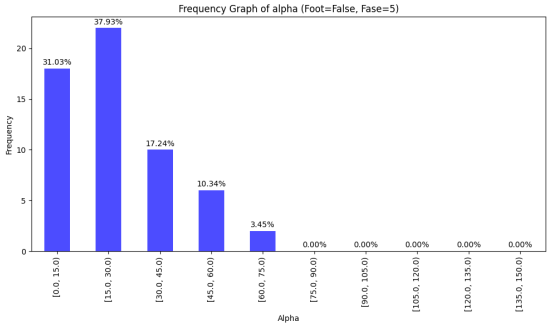

Figure 6.29: Graph of alpha frequency for phase (5,false) from first training session LFDL

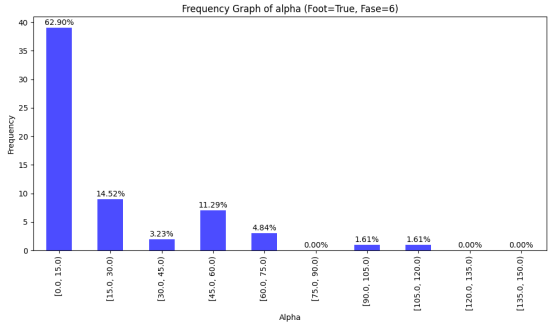

Figure 6.30: Graph of alpha frequency for phase (6,true) from first training session LFDL

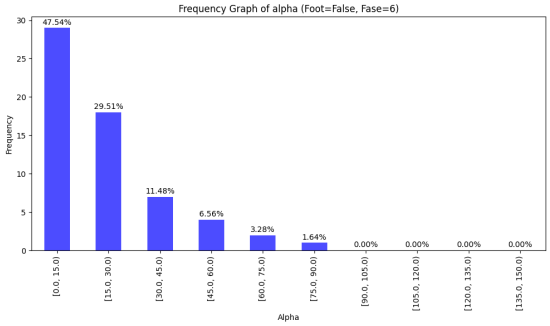

Figure 6.31: Graph of alpha frequency for phase (6,false) from first training session LFDL

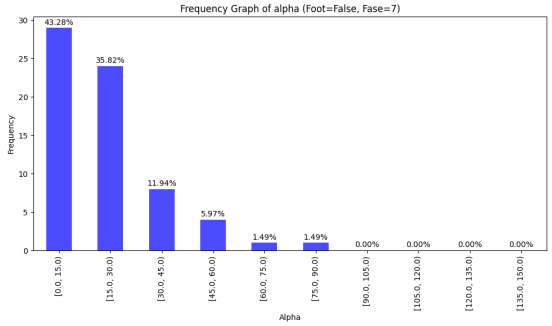

Figure 6.32: Graph of alpha frequency for phase (7,true) from first training session LFDL

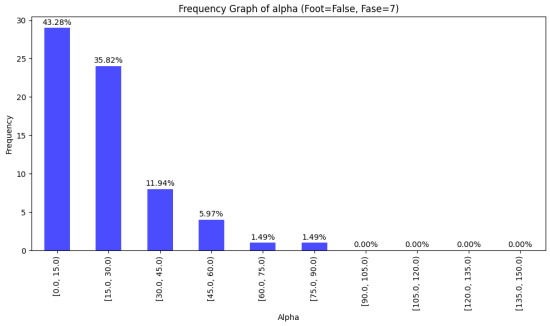

Figure 6.33: Graph of alpha frequency for phase (7,false) from first training session LFDL

After analyzing these graphs, the hit rates were also analyzed. We considered hit rates a shot at more than 2m/s and with a α less than 15º.

Analyzing them in the table 6.1, we can see that the phase with the best hit rate is the phase (6,true), as expected. In addition, phase (5,true) also had promising results.

Dribble Phase	**Left Foot Active**	**Left Foot Inactive**
0	17.021%	3.614%
1	26.563%	5.263%
2	32.877%	37.681%
3	39.683%	20.968%
4	37.097%	5.172%
5	45.161%	6.557%
6	58.730%	14.925%
7	16.0%	11.290%
Average	35.124%	13.295%

Table 6.1: LFDL general training hit rate

6.6 Conclusion

Upon analyzing the previous findings, it becomes evident that phases (5,true) and (6,true) exhibit the highest potential for future improvements in training. This is primarily because, during these phases, the robot's feet are closely positioned when initiating the shot. This unique alignment is absent in the other dribbling phases, resulting in a greater distance between the left leg and the ball. Consequently, the left leg can effectively cover the increased gap between the robot and the ball in this particular dribble phase, thereby facilitating successful shots. Based in the results of table 6.1, the decision has been made to focus on optimizing the shot execution following phase (6,true) of the LFDL in the upcoming chapter through dedicated training sessions.

Chapter 7

Kicking optimization of Dribbling phase (6,True)

7.1 Problem

Continuing with the optimization approach employed in the LFDL model, we will adopt a similar strategy of exclusively training the phase that exhibited the most potential in the prior results. This deliberate focus on a single training phase introduces heightened consistency, enabling the learning algorithm to concentrate its efforts on mastering a specific aspect of the task. This concentrated approach not only bolsters the likelihood of success but also enhances overall training efficiency.

To further streamline the training process and reduce variability, the intention is to exclusively train turning to one direction. This approach not only minimizes training time but also leverages the ability to reverse behavior, as expounded upon in Chapter 3.6.10. By employing this strategy, we aim to achieve enhanced training efficiency and proficiency in executing the desired behavior.

Upon reviewing the outcomes of the described dribbling phases of LFDL, as outlined in 6.5, the analysis highlights two particularly promising dribbling phases: (5,true) and (6,true). Given that phase (6,true) demonstrated the most encouraging results between the two, our subsequent course of action involves initiating an optimization process specifically targeting this phase.

7.2 Action Space, State Space, Network Shape and Hyperparameter Tuning

Equivalent to the preceding model, all joints except the two located in the head are included in the action space. The state space for this model utilizes identical observations as the models prior. Network configuration and hyperparameters adhere to the same presets employed in previous training sessions.

7.3 First training

7.3.1 Episode layout

To begin this optimization process, certain changes were made. In the episode layout, a modification was introduced where the agent's dribbling behavior was limited to making right turns only. These right turns were performed randomly within a range of -180 to 0 degrees and took place over a period of 0 to 2 seconds.

7.3.2 Reward

During this initial training phase, we employed the reward equation 5.1. The results observed in the previously optimized model have demonstrated its superior performance in comparison to other equations used. This function allows for a higher penalty to be applied to the reward value for high shot direction deviation values, thereby making the final trained model more precise, which aligns with one of our primary objective.

7.3.3 Results of the first attempt

After the initial training session and achieving a peak average reward, as depicted in Figure 7.1, subsequent model testing was conducted to generate a frequency graph illustrating the relationship between speed and alpha values.

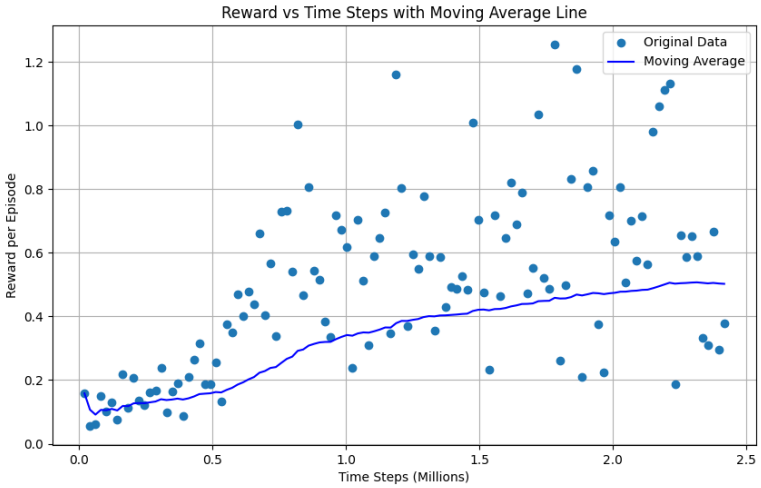

Figure 7.1: Reward Evolution Over Time Steps from first training session of phase (6,true) of LFDL

After the end of the training, a test with approximately 200 shots was conducted.

Firstly, when analyzing the velocity graph (Figure 7.2), it was observed that the most frequently recorded shot velocity during attempts was approximately 1 m/s.

The average shot velocity recorded was 1.783 m/s, while the average velocity for successful shots ($0° < \alpha < 15°$) was 3.880 m/s.

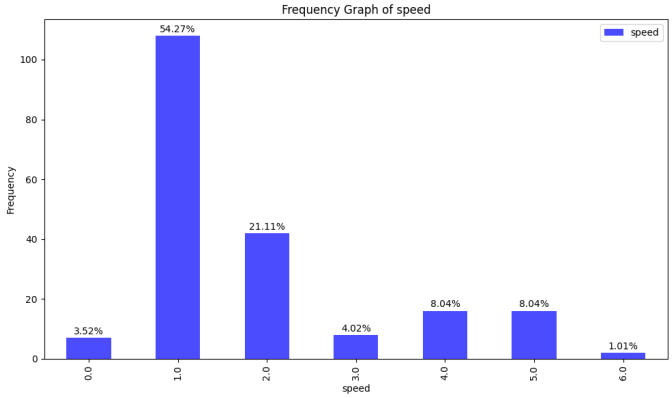

Figure 7.2: Graph of speed frequency for optimized phase (6,true) of the first training session

Approximately only 35% of the shots achieved the desired angle (α), as illustrated in Figure 7.3.

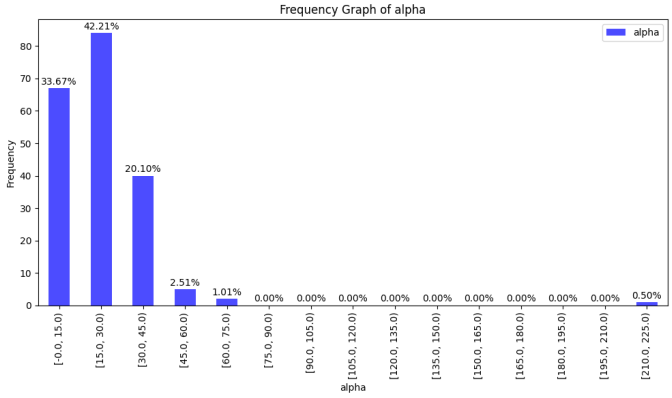

Figure 7.3: Graph of alpha frequency for optimized phase (6,true) of the first training session

The success rate for shots meeting the conditions of $\alpha < 15°$ and a velocity exceeding 2 m/s was approximately 23.5%.

7.3.4 Analysis on the first training session

From the graph examined in Figure 7.1, it was evident that the achieved results would not be optimal, primarily due to the low peak value of the average reward. Analyzing the velocity graph in Figure 7.2 revealed that in the majority of shots, the robot fails to hit the ball accurately or does so inadequately during the shooting phase. This behavior is visually confirmed.

This can be attributed to the fact that the most frequently measured velocity was 1 m/s, indicating the robot's inability to impart sufficient speed to the ball with a successful shot, thus failing

to achieve high ball velocities. Consequently, the analysis of the shooting angle becomes less meaningful due to the velocity results

Both of the above graphs clearly exhibit subpar results. Upon visual examination of the algorithm's performance, it becomes evident that when the robot executes a left turn, it encounters difficulties in positioning itself effectively for a straight-ahead shot. This results in higher alpha values and insufficient velocity, as most of the time, the outcome is unsatisfactory.

The problem at hand is a result of the rapid turn performed during dribbling, specifically the LFDL, and the extended distance that the robot keeps from the ball while executing this particular dribbling maneuver. Consequently, the robot does not have adequate time to precisely reposition itself with the ball for a successful shot, leading to an inability to close the considerable gap between the ball and the agent as they transition to the shooting phase within the limited time available.

Therefore, it becomes imperative to implement modifications to improve shot accuracy.

7.4 Second training

7.4.1 Changing Episode Layout

In the previous scenario, the distance between the robot and the ball while executing the right turn during dribbling was too significant, rendering it impossible to align for a shot within the available time frame. To enable the robot to maintain closer proximity to the ball, facilitating a swift setup for a shot, we adjusted our approach in the second training iteration.

During the LFDL, a notable difference was observed in the distance the robot kept from the ball when turning left as opposed to turning right. Specifically, when the robot turned left, it kept the ball closer.

To enhance the neural network training and improve overall results, we modified the training process to take advantage of this LFDL characteristic. Instead of executing a right turn within a range of -180 to 0 degrees, the robot now performs a left turn within a range of 0 to 180 degrees, completing the maneuver in a maximum of 2 seconds. Subsequently, it transitions to the shooting phase, using the same function (5.1) as before to achieve the aforementioned objective.

7.4.2 Results of the second attempt

After 3,500 training steps, and upon observing that the reward had reached its peak for this specific training scenario, as depicted in Fig. 7.4, it became evident that the reward value had reached a plateau. At this point, we concluded the training phase. The reward value per episode had exceeded a threshold of 2.5 reward points per episode.

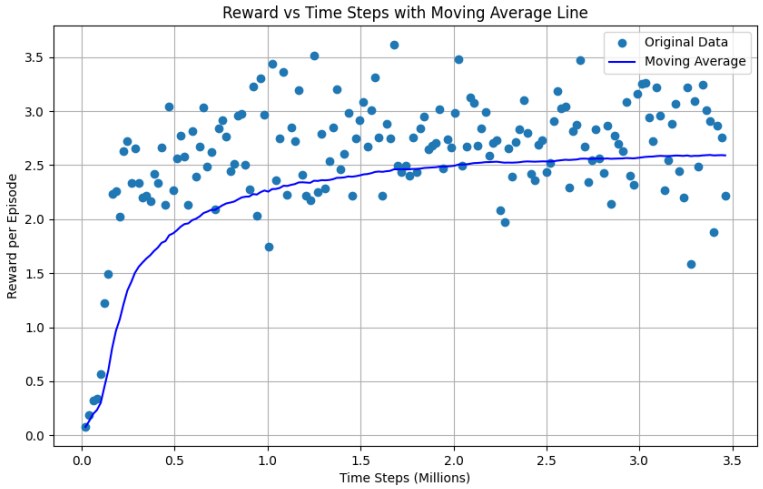

Figure 7.4: Reward Evolution Over Time Steps from second training session of phase (6,true) of LFDL

The outcomes are graphically depicted in the speed and alpha frequency graphs, following approximately 400 test shots:

The speed graph (Fig. 7.5) demonstrates promising results, with the most frequently recorded speed hovering at approximately 3m/s.

The average shot velocity recorded was 3.352 m/s, while the average velocity for successful shots ($0° < \alpha < 15°$) was 3.646 m/s.

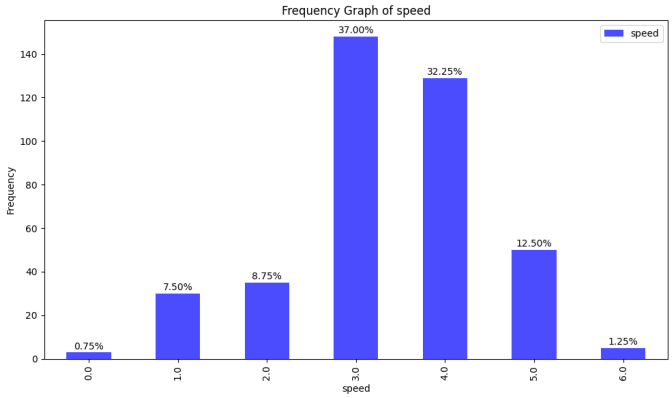

Figure 7.5: Graph of speed frequency for optimized phase (6,true) of the second training session

Furthermore, the alpha graph (Fig. 7.6) indicates an impressive success rate of roughly 95%, underscoring remarkable achievements in this aspect.

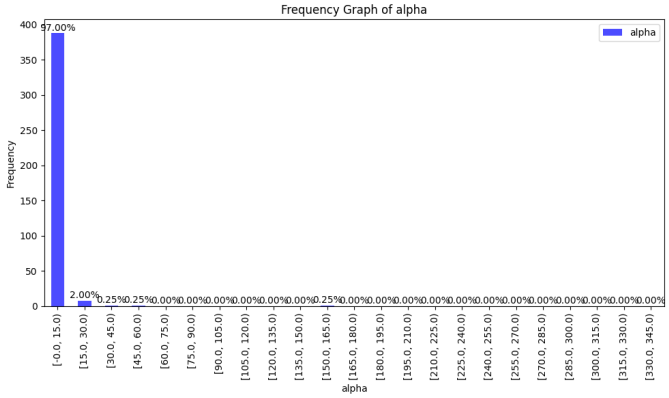

Figure 7.6: Graph of alpha frequency for optimized phase (6,true) of the second training session

Due to the low results shown in the alpha graph, we are presenting the same frequency graph but with a finer level of detail.

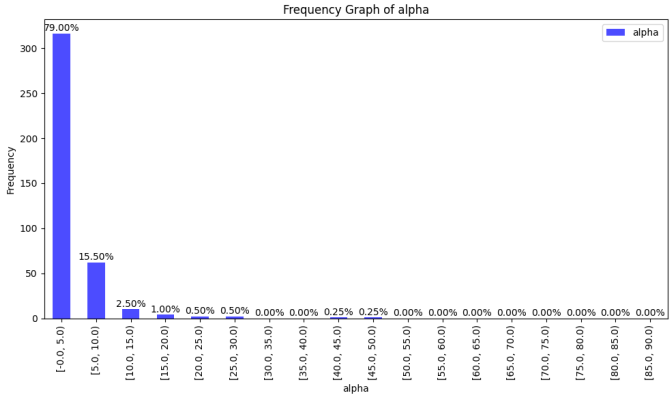

Figure 7.7: Detailed graph of alpha frequency for optimized phase (6,true) of the second training session

The success rate for shots meeting the conditions of $\alpha < 15°$ and a velocity exceeding 2 m/s was approximately 88.029%.

7.4.3 Analysis on the second training session

Following the training and the analysis of the reward graph (Figure 7.4), it became evident that the results were highly promising, especially when compared to reward graphs from previous training sessions using the same function (5.1).

In terms of velocity, the most frequently recorded speeds of 3 m/s and 4 m/s highlight the robot's exceptional ability during this training to execute precise shots and effectively strike the ball in the shooting phase. Additionally, the accuracy graph for α values revealed outstanding performance, with the robot correctly aligning with the ball's direction in an impressive 97% of the shots taken, resulting in a remarkable 88% success rate.

These collective findings unequivocally establish this training endeavor as an impressive success. It has demonstrated the capacity to seamlessly transition from dribbling to a kick in motion, achieving a remarkable success rate and remarkably swift preparation and execution speeds. This outcome substantiates the feasibility of swiftly transitioning from a dribble to a shot using this particular technique.

It is clear that the change in direction during the dribbling phase, turning left instead of right, has produced the desired result of reducing the distance between the robot and the ball when entering the shooting phase, thus promoting better results. Despite the initial visual impression that using the opposite foot to the side of the turn might be more logical, the results have defied this expectation. In fact, the opposite has proven true, as the ball's positioning facilitates precise alignment for a left-footed shot when turning left, resulting in an exceedingly low occurrence of missed shots.

Due to these excellent results, this model developed for this dribbling phase can be employed in competitive environments, given its high success rate. It can be used in both defensive and offensive situations, with a high degree of efficiency almost guaranteed.

The positive results have also motivated us to further optimize this shooting technique. In the next training session, we plan to introduce some direction into the shots to make them more versatile and usable in a wider range of situations.

7.5 Third training

7.5.1 Changing State Space

To further enhance the optimization of our previous shot, we are planning to incorporate directional control into our shots. This strategic adjustment aims to increase the versatility and adaptability of our shots across a wider range of scenarios. To achieve this, several modifications will need to be made to enable the system to perceive the intended direction of the ball's trajectory. Based on this direction, the system will calculate the accuracy of the shot and subsequently assign the appropriate reward.

In the context of the state space, we have introduced a new variable, as mentioned earlier, to determine the intended direction of the shot. At the onset of the shooting phase, this variable will be initially randomized, falling within a range of -20 to +20 degrees relative to the robot's initial orientation. To clarify, this means that the robot's targeted shot will have the potential to span a range of roughly 40 degrees when the shooting phase commences.

All other parameters remain constant. The robot undergoes the same pre-training process as in the previous training session and then transitions to the shooting phase. However, in this phase, it is imperative that the direction applied to the ball matches the randomly chosen value to maintain a high reward. The α value in the function 5.1 will now represent the difference between the direction of the ball and the randomly selected intended direction. If this training

proves successful, the randomly chosen direction may, in the future, be employed by the team as a deliberate choice for executing shots.

7.5.2 Results of the third attempt

After 2 million steps, we decided to stop the training as the average reward per training session had stabilized at its peak value. This trend can be observed in Figure 7.8, which illustrates the evolution of the reward over time steps.

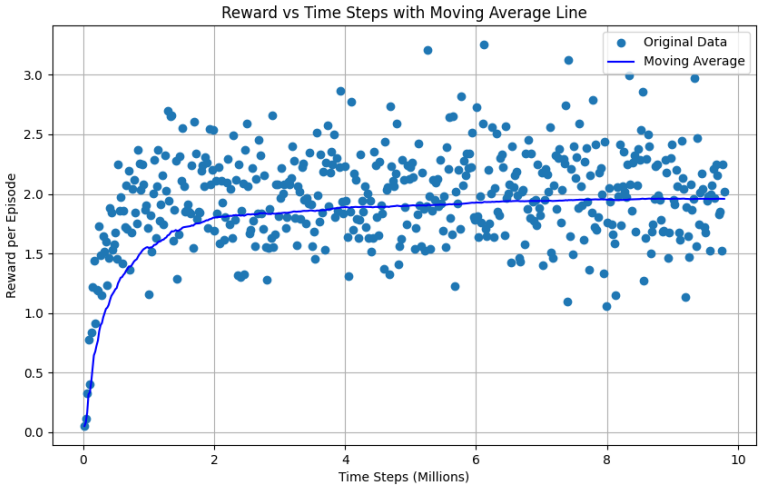

Figure 7.8: Reward Evolution Over Time Steps from third training session of phase (6,true) of LFDL

The outcomes are visually represented in the speed and alpha frequency graphs, following approximately 400 test shots.

In Figure 7.9, the speed graph showcases promising results, with the most frequently recorded speed hovering around 3 m/s.

The average shot velocity recorded was 3.378 m/s, while the average velocity for successful shots ($0° < \alpha < 15°$) amounted to 3.675 m/s.

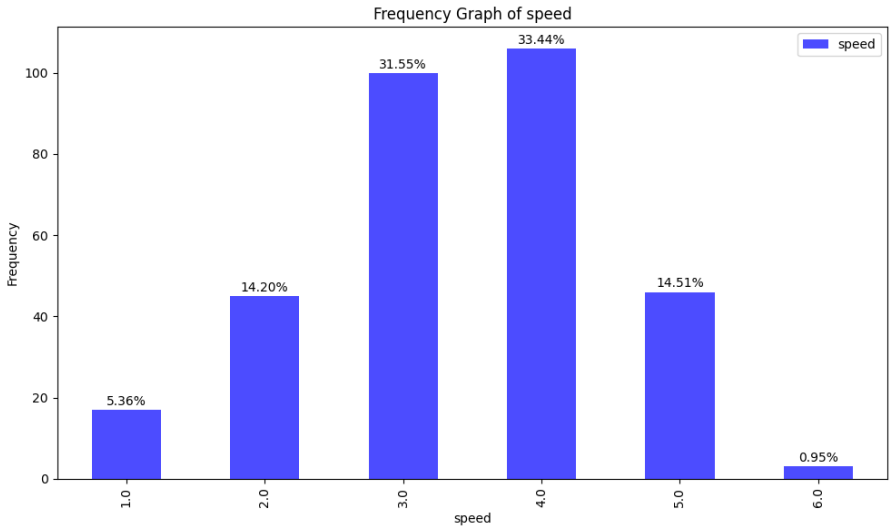

Figure 7.9: Graph of alpha frequency for optimized phase (6,true) of the third training session

Due to the requirement for high precision in this type of targeted shot, we present the graph in figure 7.10, illustrating the angle α that represents the difference between the desired final direction of the shot and the actual direction of the shot, in increments of 5 degrees instead of the previously presented 15 degrees. In this case, approximately 40% of the shots exhibited very precise direction, with $\alpha < 5º$.

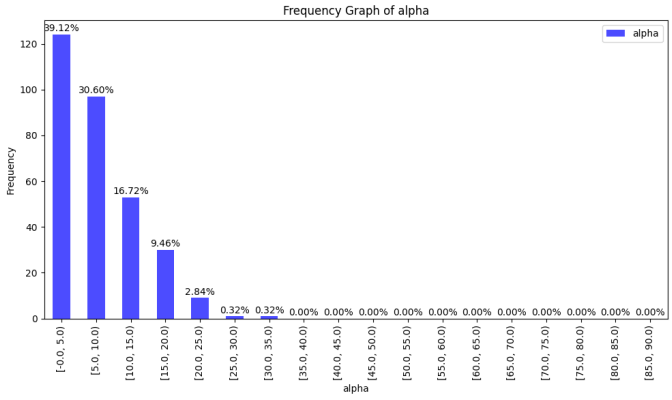

Figure 7.10: Graph of alpha frequency for optimized phase (6,true) of the third training session

The hit rate, maintaining the previous criteria of $\alpha < 15°$ and a velocity exceeding 2 m/s, was determined to be 77.673%.

7.5.3 Analysis of the third training session

In this third training session, a change was made to the state space to enable targeted shots. However, the results reveal that the agent learns to impart some direction to the ball during the shot.

Nevertheless, the speed at which the robot enters the shooting phase, coming from the dribbling phase, is too high for the robot to achieve high precision in its shots. As a result, approximately only 40% of the shots exhibit very precise direction, with $\alpha < 5º$.

Regarding shot velocity, compared to the previous model, the average shot velocities were similar.

This model proves viable for introducing some direction into the shot when needed, such as in passes, clearances, or quick shots where the agent has to impart some direction to the shot. However, precision is limited, indicating that additional adjustments may be necessary to improve accuracy in targeted shots.

7.6 Conclusion

A summary of the optimizations performed can be observed in Table 7.1, which presents, for each of the previous training sessions, the hit rate, average speed, and average speed of successful shots.

Training Trial	**Hit Rate**	**Average Spee0**d	**Average Speed of Successful Shots**	**Targeted shot**
1	23.5%	1.783 m/s	3.885 m/s	NO
2	88.029%	3.352 m/s	4.646 m/s	NO
3	77.673%	3.378 m/s	3.675 m/s	YES

Table 7.1: Training Optimization Summary of LFDL

After conducting a comprehensive analysis of various training variable variations, we have successfully developed a kick in motion after dribbling with an accuracy of approximately 90% when shooting straight ahead. Additionally, we have devised a shooting technique where the agent attempts to add some direction to the shot despite the inherent difficulty in the process. This is due to the robot's high speed when entering the shooting phase, leaving little time for precise alignment. However, the primary goal of these shots has always been to execute them as quickly as possible.

The high level of accuracy achieved in the model developed during the second training session makes this shot suitable for use in competitions. With a success rate of 90%, it establishes this shot as a valuable defensive option for clearing the ball from dangerous areas and also in offensive situations like passes or shots, where the accuracy rate is high. Furthermore, by reversing the state space and action space, it will be possible to achieve these high success rates when dribbling, both to the left and to the right.

This preference for turning to the left for a shot with the left foot, as opposed to turning to the right, in the model from the second training session, stems from the fact that the agent keeps the ball at a much closer distance to the left foot when turning left than when turning right. This closer proximity allows for better alignment and contributes to the higher success rate when executing shots in that direction. It may not be as visually intuitive, but it proves to be a more effective approach in practice.

Chapter 8

Conclusion

This thesis delved into the utilization of reinforcement learning methodologies to enhance the kicking capabilities of NAO robots within the RoboCup framework. The primary goals encompassed an exploration of the Proximal Policy Optimization (PPO) deep reinforcement learning algorithm, a comprehensive grasp of the FCPortugal work environment, and the proposal of efficient strategies for augmenting agent performance.

The document initiates by presenting an introduction and rationale for this endeavor in chapter 1, within the context of the FCPortugal team and the broader RoboCup competition.

Subsequently, we embarked on an exhaustive survey of machine learning, deep learning, and reinforcement learning in Chapter 2. This exploration encompassed an array of algorithms, including SARSA, Q-Learning, DQN, DDPG, Policy Gradient, TRPO, and PPO. These algorithms laid the foundation for devising effective learning strategies for the NAO robots.

Chapter 3 brought us to the heart of the FCPortugal work environment, with a particular focus on the RoboCup simulator, SimsPark. Here, we dissected the components of Roboviz, delved into the intricacies of the NAO robots, and familiarized ourselves with the regulations governing the 2023 competition. These insights established the framework for our subsequent experimental endeavors.

In Chapter 4, we took on the challenge of achieving a kick in motion following a Double Footed Dribble Long. This involved a meticulous examination of the problem, the definition of action and state spaces, and a careful tuning of network shapes and hyperparameters. Our dedicated experimentation in this chapter provided valuable insights into the optimal phases for transitioning from dribbling to a shot.

Chapter 5 was dedicated to optimizing the kicking technique following a dribbling phase of (2,True). While our experiments led to significant improvements in kicking techniques during this phase, we have developed a model that achieves approximately 70% accuracy.

In Chapter 6, our focus shifted to enhancing the kick in motion following a Left Footed Dribble Long. Here, the experimental findings enriched our understanding of the most promising phases for executing a successful shot in this specific dribbling context.

Chapter 7 reached its climax with the culmination of our journey to optimize the kicking phase during the dribbling process (6,True). Extensive testing of various episode layouts resulted in achieving a remarkable shot success rate of approximately 90%. We attained these results because, during this phase of the LFDL, the robot keeps its feet close together. Furthermore, the robot manages to stay close to the ball while dribbling to the left, something that didn't happen when dribbling to the right. All of these factors allow the robot, in the limited time available during the shooting phase, to close the gap between the ball and the robot and successfully take a shot.

In conclusion, this thesis represents a comprehensive exploration of the Proximal Policy Optimization (PPO) algorithm in the realm of RoboCup, specifically within the FCPortugal team. Through our collective efforts, we have successfully developed a shot technique that aligns with the objectives set forth in this dissertation. Our journey not only advances the capabilities of NAO robots but also contributes to the broader field of robotics and reinforcement learning.

8.1 Future Work

Despite the significant advancements made in this dissertation, numerous avenues for future research and development in the realm of reinforcement learning for robot soccer remain open, offering promising opportunities for further exploration and improvement.

Algorithmic Exploration: Future work could consider alternative algorithms beyond PPO (Proximal Policy Optimization). Exploring a variety of algorithms may yield valuable insights and improvements in skill development.

Reward Function Experimentation: Experimenting with diverse reward functions, state space configurations, and episode structures can potentially unlock novel and previously unattainable results.

Transfer Learning to Real Robots: Adapting the learned policies from simulation to real-world NAO robots is a critical aspect for practical applicability. Research in this area should tackle challenges such as the reality gap, calibration issues, and domain adaptation techniques to seamlessly deploy learned behaviors on physical robots.

In conclusion, the future of reinforcement learning for robot soccer holds immense potential to revolutionize the fields of robotics and autonomous systems. By pursuing these research directions, we can pave the way for more sophisticated and capable robot soccer players. This not only contributes to advancements in sports but also has broader implications in real-world applications involving complex decision-making in dynamic environments.

References

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

[2] Ahmad Taher Azar, Anis Koubaa, Nada Ali Mohamed, Habiba A Ibrahim, Zahra Fathy Ibrahim, Muhammad Kazim, Adel Ammar, Bilel Benjdira, Alaa M Khamis, Ibrahim A Hameed, et al. Drone deep reinforcement learning: A review. *Electronics*, 10(9):999, 2021.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[4] Soccer simulation · wiki · robocup simulation / simspark · gitlab. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Soccer-Simulation.

[5] Softbank robotics documentation. URL: http://doc.aldebaran.com/2-8/home_nao.html.

[6] Models · wiki · robocup simulation / simspark · gitlab. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Models.

[7] Robocup federation official website, Dec 2022. URL: https://www.robocup.org/.

[8] Robocup simulation 3d league. URL: https://www.robocup.org/leagues/25.

[9] Objective. URL: https://www.robocup.org/objective.

[10] Miguel Abreu, Mohammadreza Kasaei, Luís Paulo Reis, and Nuno Lau. Fc portugal: Robocup 2022 3d simulation league and technical challenge champions. In *Robot World Cup*, pages 313–324. Springer, 2022.

[11] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[13] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.

[14] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire, Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt Hanna, et al. A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nature genetics*, 43(5):491–498, 2011.

[15] B Sujatha and K VSSR Murthy. Advanced machine learning technique to handle filtering unwanted messages in online social networks.

[16] Kandan Chitra and Balakrishnan Subashini. Data mining techniques and its applications in banking sector. *International Journal of Emerging Technology and Advanced Engineering*, 3(8):219–226, 2013.

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

[18] What are neural networks? URL: https://www.ibm.com/topics/neural-networks.

[19] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[20] Ekin Keserer. Deep learning vs reinforcement learning: Key differences and use cases, Nov 2020. URL: https://www.akkio.com/post/deep-learning-vs-reinforcement-learning-key-differences-and-use-cases.

[21] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[22] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[23] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[24] Robert N Boute, Joren Gijsbrechts, Willem Van Jaarsveld, and Nathalie Vanvuchelen. Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, 298(2):401–412, 2022.

[25] Lan Zou. Meta-learning: theory, algorithms and applications. *(No Title)*, 2022.

[26] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[27] Hao Dong, Hao Dong, Zihan Ding, Shanghang Zhang, and Chang. *Deep Reinforcement Learning*. Springer, 2020.

[28] Tuyen P. Le, Ngo Anh Vien, and TaeChoong Chung. A deep hierarchical reinforcement learning algorithm in partially observable markov decision processes. *IEEE Access*, 6:49089–49102, 2018. doi:10.1109/ACCESS.2018.2854283.

[29] Daniel Bennett, Yael Niv, and Angela J Langdon. Value-free reinforcement learning: policy optimization as a minimal model of operant behavior. *Current Opinion in Behavioral Sciences*, 41:114–121, 2021.

[30] Mengran Yu and Shiliang Sun. Policy-based reinforcement learning for time series anomaly detection. *Engineering Applications of Artificial Intelligence*, 95:103919, 2020.

[31] Part 2: Kinds of rl algorithms. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[32] Huang (Steeve) Kung-Hsiang. Introduction to various reinforcement learning algorithms. part i (q-learning, sarsa, dqn, ddpg), Sep 2018. URL: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa

[33] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[34] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[35] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[36] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[37] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[38] Arthur Juliani. Simple reinforcement learning with tensorflow part 8: Asynchronous actor-critic agents (a3c), Jun 2017. URL: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-crit

[39] URL: https://openai.com/research/openai-baselines-acktr-a2c.

[40] URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html.

[41] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[42] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[45] Russell Smith et al. Open dynamics engine. 2005.

[46] Yuan Xu and Hedayat Vatankhah. Simspark: An open source robot simulator developed by the robocup community. In *RoboCup 2013: Robot World Cup XVII 17*, pages 632–639. Springer, 2014.

[47] About simspark · wiki · robocup simulation / simspark · gitlab. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/About-SimSpark.

[48] Agent proxy · wiki · robocup simulation / simspark · gitlab. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Agent-Proxy.

[49] magmaOffenburg. Magmaoffenburg/roboviz: Monitor and visualization tool for the robocup 3d soccer simulation league. URL: https://github.com/magmaOffenburg/RoboViz.

[50] Justin Stoecker and Ubbo Visser. Roboviz: Programmable visualization for simulated soccer. In *RoboCup 2011: Robot Soccer World Cup XV 15*, pages 282–293. Springer, 2012.

[51] Nao the humanoid and programmable robot. URL: https://www.aldebaran.com/en/nao.

[52] Home. URL: https://ssim.robocup.org/3d-simulation/3d-rules/.

[53] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[54] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, et al. Stable baselines, github repository. *URL https://github. com/hill-a/stable-baselines*, 2018.

[55] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines (2017). *URL https://github. com/openai/baselines*, 2016.

[56] Welcome to stable baselines docs! - rl baselines made easy¶. URL: https://stable-baselines.readthedocs.io/en/master/.

[57] URL: https://openai.com/research/openai-baselines-ppo.

[58] URL: https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html.

[59] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[60] Openai. Openai/gym: A toolkit for developing and comparing reinforcement learning algorithms. URL: https://github.com/openai/gym.

[61] Miguel Abreu, Tiago Silva, Henrique Teixeira, Luís Paulo Reis, and Nuno Lau. 6d localization and kicking for humanoid robotic soccer. *Journal of Intelligent & Robotic Systems*, 102(2):30, 2021.

[62] Henrique Teixeira, Tiago Silva, Miguel Abreu, and Luís Paulo 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. IEEE, 2020.

[63] Abbas Abdolmaleki, David Simões, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. Learning a humanoid kick with controlled distance. In *RoboCup 2016: Robot World Cup XX 20*, pages 45–57. Springer, 2017.

[64] Miguel Abreu, David Simes, Nuno Lau, and Luis Paulo Reis. Fast, human-like running and sprinting.

[65] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. Learning to run faster in a humanoid robot soccer environment through reinforcement learning. In *Robot World Cup*, pages 3–15. Springer, 2019.

[66] Margarida Inês de Almeida Santiago. Fc portugal-high-level skills within a multi-agent environment. 2021.

[67] Pedro Lavarinhas Amaro. Multi-robot learning of high-level skills in robocup. 2019.

[68] Tiago Rafael Ferreira da Silva. Humanoid low-level skills using machine learning for robocup. 2019.

[69] Mohammadreza Kasaei, Miguel Abreu, Nuno Lau, Artur Pereira, and Luis Paulo Reis. Robust biped locomotion using deep reinforcement learning on top of an analytical control approach. *Robotics and Autonomous Systems*, 146:103900, 2021.

[70] Mohammadreza Kasaei, Miguel Abreu, Nuno Lau, Artur Pereira, Luis Paulo Reis, and Zhibin Li. Learning hybrid locomotion skills–learn to exploit residual dynamics and modulate model-based gait control. *arXiv preprint arXiv:2011.13798*, 2020.

[71] Mohammadreza Kasaei, Miguel Abreu, Nuno Lau, Artur Pereira, and Luis Paulo Reis. A cpg-based agile and versatile locomotion framework using proximal symmetry loss. *arXiv preprint arXiv:2103.00928*, 2021.

[72] S Mohammadreza Kasaei, David Simões, Nuno Lau, and Artur Pereira. A hybrid zmp-cpg based walk engine for biped robots. In *ROBOT 2017: Third Iberian Robotics Conference: Volume 2*, pages 743–755. Springer, 2018.

[73] David Simões, Pedro Amaro, Tiago Silva, Nuno Lau, and Luís Paulo Reis. Learning low-level behaviors and high-level strategies in humanoid soccer. In *Robot 2019: Fourth Iberian Robotics Conference: Advances in Robotics, Volume 2*, pages 537–548. Springer, 2020.