Exploring Reinforcement Learning in Path Planning for Omnidirectional Robot Soccer

José Victor Silva Cruz (jvsc@cin.ufpe.br)

Federal University of Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~graduacao

Recife
2023

José Victor Silva Cruz (jvsc@cin.ufpe.br)

Exploring Reinforcement Learning in Path Planning for Omnidirectional Robot Soccer

A B.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Concentration Area: *Computational Intelligence*
Advisor: *Edna Natividade da Silva Barros*
(ensb@cin.ufpe.br)

Recife
2023

FICHA

BANCA

ACKNOWLEDGEMENTS

First, I would like to thank God and my family, especially Mrs. Inês and Mr. Paulo Cruz, for their unconditional support: you are my greatest mirrors in everything I do, and being your son is the most incredible pride I have in life. I want to thank my brothers Eudes, Pedro, Lucas, and my cousin Bruno, for all the shared moments; we will always be together, and no distance will be able to change that. I dedicate all my efforts during the graduation to my family on behalf of my paternal grandparents, whom I did not have the opportunity to meet, but who guided my steps until here.

I would also like to thank all the professors who taught me everything I know and supported my academic choices, especially my math teacher, Edmar, for helping me with the course decision; Prof. Nivan, for the partnership during the Maratona period; Prof. Hansenclever Bassani, for the orientation and contribution in writing; and Prof. Edna, for the words said during the LARC 2019 and for all the support given during graduation, fundamental in the most critical moments of this trajectory.

I would also like to thank the city of Recife, where nowadays I consider myself to have citizenship highlighted by my hybrid accent, for the opportunity to meet great friends such as Raul, João, Luan, Clodes, Leon, Gap, and Victor Hugo. I want to thank Andresa for being by my side, supporting my choices, and inspiring me with her dedication.

I want to thank CIn, the place for which I chose to leave my hometown and concentrate my studies in Computer Engineering, for the infrastructure, research support, and excellence in computing, which allowed me to have a unique formation to make dreams come true and to conquer things I never imagined. I want to thank my school, Diocesano, for all my primary and life education.

I also want to thank Maratona for being the project that allowed me to meet highly talented people and directly contributed to my skills as a programmer. Special thanks to Bezaliel for all the moments we shared and everything he taught me, Lucas Santana for the support, and my team, *experts com sono*, with whom I had the honor of being part alongside Serra and Mendes.

It is impossible not to thank RobôCIn, where I dedicated most of my efforts during this period. But yet, I especially want to thank Lucas Cavalcanti for allowing me to join the team and the Reinforcement Learning student members Breno, Felipe, and Gonça for the orientations that made this work possible.

Finally, I would like to thank my friends who have become brothers in life: Francisco, Heitor, and Miguel. To my favorite quartet: André, Vitória, Luísa, and Myrna. To my team Peladeiros F.C., and to my kitten, Mia, who spent early mornings programming with me. I really miss you all.

"I might end up somewhere in Mexico"

— Soul to Squeeze – Red Hot Chili Peppers ✱

ABSTRACT

Path Planning consists of a widely studied computational problem of great applicability in autonomous robotics and virtual reality environments that aims to solve the following problem: given the origin of an entity in space, obtain a feasible collision-free route to the destination. From the characteristics of a given environment, in this case, a soccer field in conventional game conditions that imply a greater complexity given the dynamics of the obstacles, it is intended to use Reinforcement Learning – technique that has gained expression over time due to the ability of its applications to perform better than humans in different scenarios –, to optimize the trajectories performed by an agent as it maximizes the reward accumulated within the executed iterations.

Keywords: Path Planning. Omnidirectional Robot. Reinforcement Learning.

RESUMO

Planejamento de rotas consiste em um problema computacional amplamente estudado, de notável aplicabilidade na robótica autônoma e em ambientes de realidade virtual, que propõe-se resolver o seguinte problema: dado a origem de uma entidade em um espaço, obtenha uma rota factível e sem colisão até o destino. A partir das características de um dado ambiente, neste caso, um campo de futebol em condições de jogo convencionais que implicam em uma complexidade maior dado a dinamicidade dos obstáculos, pretende-se utilizar Aprendizagem por Reforço – técnica que tem ganhado expressão ao longo do tempo devido a capacidade de suas aplicações performarem melhor que humanos em diferentes cenários –, visando otimizar as trajetórias realizadas por um agente na medida em que ele maximiza a recompensa acumulada dentro das iterações executadas.

Palavras-chave: Planejamento de Rotas. Robô Omnidirecional. Aprendizagem por Reforço.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1

INTRODUCTION

One of the fundamental processes of autonomous robotics is the *path planning*. It can be seen as a part of robot's trajectory steps, where after locating the robot in space, the following problem seeks to be solved: given the origin position of an entity, obtain a collision-free feasible route. The study area has broad applicability in the movement of any autonomous entity in a real or simulated environment.

In this work, the entity considered is an omnidirectional robot soccer, where the main characteristic is that it can move in any direction inside a 2D plane. There are many competitions related to robot soccer, with an emphasis on RoboCup [1] [12]. Inside it, lies the *Small Size League (SSL)* (Figure 1), one of the oldest categories that use four-wheeled intelligent multi-agent collaborative omnidirectional robots controlled in a highly dynamic environment with a hybrid centralized/distributed system.

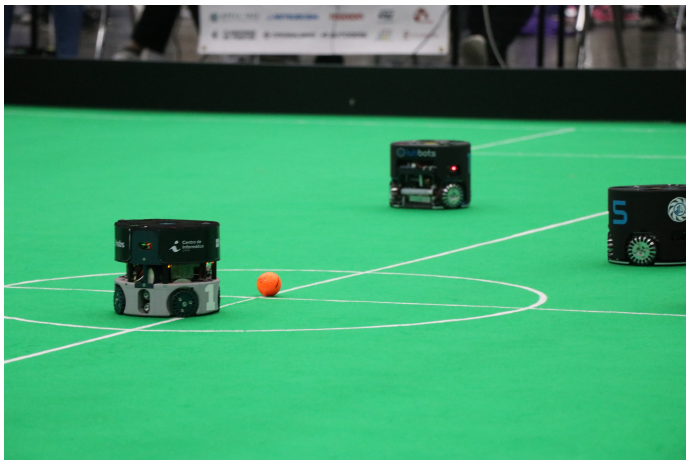


Figure 1: SSL robots in RoboCup field during real game. Source: The author.

In path planning, choosing the best model to solve the problem may depend on several factors, including: number and format of obstacles, processing time, distance to the objective, and environment. Several approaches have obtained great notoriety in general, but more specifically

[1] RoboCup is a world competition of several modalities of autonomous robots existing since 1997, which has the ambitious intention of developing a team of humanoid robots capable of defeating the most recent FIFA World Cup champion in the middle of 21st century [9].

in robot soccer: either by heuristics [17], potential fields assisted by evolutionary algorithms [15], or sampling-based planning [10, 8]. However, the focus of this work is on Reinforcement Learning use as a possible alternative to obtain an optimal path for omnidirectional robot soccer.

Reinforcement Learning (RL) is a subarea of Machine Learning where the agent's learning occurs from its interaction with the environment within a reward system, without direct interference from the developer [26]. Applications using this technique have gained expression over time due to their ability to perform better than humans in different scenarios, such as driving autonomous cars [11], playing Atari [20], natural language processing (NLP) [28] and more. Inside RoboCup and SSL scope, although some studies already include RL technique, as demonstrated by Volodymyr, M. *et al.* [30], B. R. Kiran, *et al.* [32] and Uc-Cetina, V. *et al.* [34], none of those relates to path planning.

Due to the nature of the problem when focusing more generally on soccer, some characteristics are preestablished. This is, as soccer occurs in an open field, the dynamic obstacles are entities of the same aspect, whether these are allies or opponents, and the static obstacles occur in specific foul situations or are predetermined by rules of the modality under study. Therefore, this work focuses on this and other peculiarities involving robot soccer when associated with the SSL category, aiming to establish an optimal solution for path planning using Reinforcement Learning.

1.1 OBJECTIVES

The general objective of this project is to study and propose a path-planning model for omnidirectional robots using Reinforcement Learning. Thus, the focus does not rely on the analysis of different techniques or theoretical foundations of RL, but rather on the choice of determining heuristics for an agent already considered as optimal. Besides, it also does not involve the analysis of static and dynamic obstacles.

The specific objectives encompassed in the general objectives are:

i. Proposition of the a learning method, *i.e* definition of the agent's inputs and rewards.

ii. Implementation of the method in a simulated environment.

iii. Comparative analysis of the results in different preestablished situations, measuring the output of the model in terms of performance and quality.

This work is divided in 6 chapters. Chapter 2 introduces some key concepts about omnidirectional robot navigation and path planning applied to the RoboCup SSL category. Chapter 3 details RL and the training algorithm used in this study. Chapter 4 describes the step-by-step implementation of this study. Chapter 5 presents the results of the experiments. Finally, Chapter 6 discusses each result and provides some ideas for future results improvements.

2

NAVIGATION

This chapter explains what robot navigation is, where path planning is inserted, what challenges are present in this area within the SSL modality, and how the teams are exploring it.

2.1 OVERVIEW

Robot navigation refers to the ability of a robot to move from one location to another in an environment, it is a crucial component of robotics, as it enables robots to perform a wide range of tasks in various environments. It involves the integration of perception, decision-making, and control to enable a robot to autonomously navigate through an environment to achieve a given goal.

The main components [24] of robot navigation (Figure 2) are:

- *Sensors*: the physical components that provide the necessary input for the agent's perception, which allows the robot to make decisions and move safely. These sensors can be cameras, ultrasound or infrared sensors, and others.
- *Perception*: processing of sensor data to extract relevant information about the robot's environment, such as the location of obstacles, the position of the robot, and the location of the goal. This component includes algorithms for object detection, segmentation and recognition, for example.
- *Localization*: the process of determining the robot's location in the environment, including methods such as odometry, GPS, and SLAM (Simultaneous Localization and Mapping).
- *Mapping*: operation responsible for creating a representation of the environment that the robot can use to plan its path. This component can use techniques such as occupancy grid, topological or geometric mapping.
- *Path Planning*: analysis aiming to find an optimal path from the robot's current location to its destination, while avoiding obstacles and other hazards in the environment. This component tipically uses techniques such as A* search or Dijkstra's algorithm.

- *Control*: execution of the planned path by sending commands to the robot's actuators that, in this study, can be divided into off-field and in-place control – post-processing of path planning generated route and motors and/or servos acting in field, respectively.

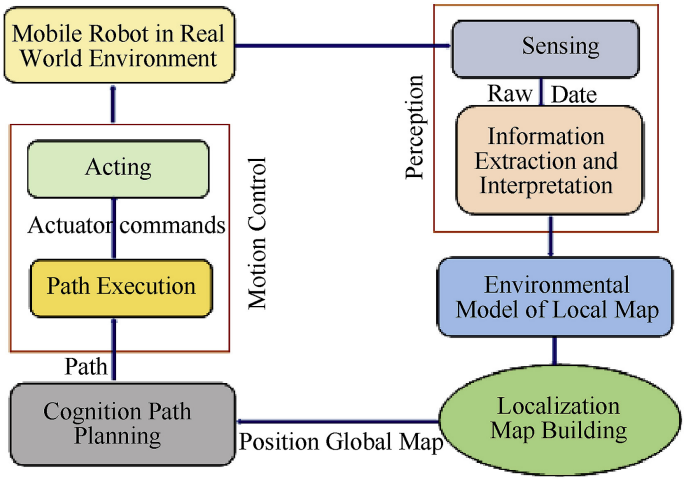


Figure 2: Flow diagram for mobile robot navigation. Source: Patle *et al.* [24].

All these components work together to enable a robot to navigate through its environment autonomously. The choice of sensors, perception algorithms, localization techniques, mapping, and path planning methods depend on the specific robot platform and environment in which it will operate.

Navigation is also broadly classified as global and local. *Global* robot navigation refers to the process of planning a robot's path from a starting point to a destination, taking into account the robot's surroundings and any obstacles in its path. This typically involves using a map of the environment and algorithms to plan a safe and efficient route. *Local* robot navigation, on the other hand, is concerned with the robot's ability to navigate in real-time as it moves through its environment. This involves using sensors to detect obstacles and other objects in the robot's immediate vicinity, and using algorithms to adjust the robot's path to avoid collisions.

In essence, global navigation is concerned with planning the best possible route for the robot to take, dealing with a completely known environment, while local navigation is concerned with making real-time adjustments to that route to ensure the robot safely reaches its destination. Both approaches have their advantages and disadvantages. Global navigation is typically more efficient and can lead to faster travel times, but it requires accurate maps of the environment and can be computationally intensive. Conversely, local navigation is more reactive and can adapt to changes in the environment, but it may not always find the most efficient route.

Global navigation is typically employed on Small Size Soccer. In this modality, a standardized vision system developed and maintained by the SSL's community as an open-source project [33] tracks all the objects on the field and works by processing data from one or more cameras that are mounted approximately 4 *meters* above the playing surface (Figure 3).

The coordination and control of robots relie on off-field computers, which are responsible for sending referee commands and position information to the robots. In addition to this communication function, these computers also typically handle the majority of the processing required for coordination and control. To enable wireless communication with the robots, dedicated commercial transmitter/receiver units are commonly used. By leveraging these powerful computing and communication technologies, teams are able to remotely control and manage the movements of their robots on the field.

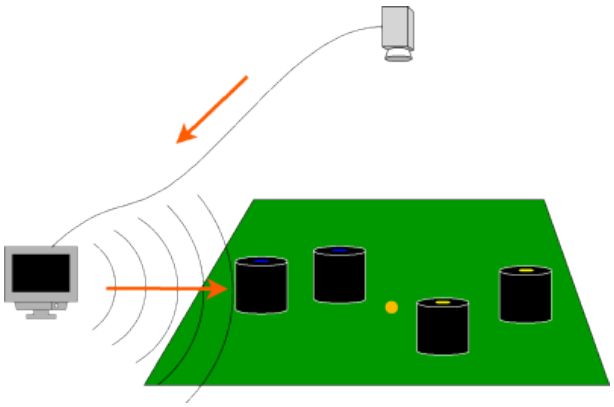


Figure 3: General structure of the Small Size League operation. Source: RoboCup Federation [1].

However, there are ongoing studies in the category, such as the one proposed by J. Melo *et al.* [19], which aims to decentralize the agents' vision and gradually give more power to the local navigation strategy by embedding cameras and sensors into the robot, consisting in a greater challenge due to the limitations of their sizes, that must fit within a 180 *millimeters* diameter circle and cannot be higher than 15 *centimeters*.

2.2 PATH PLANNING

Path planning algorithms are an essential component for the success of a team in a robot soccer competition, as it allows the robots to move quickly and effectively around the field, making quick and accurate decisions to outmaneuver their opponents and score goals. These algorithms are used to generate efficient and safe trajectories for the robots to follow, taking into account the position of other robots and the ball, as well as the boundaries of the field and any obstacles that may be presented. They must be fast and efficient to keep up with the fast-paced nature of the game, while also being robust enough to handle unexpected situations that may arise during gameplay.

One of the most widely used algorithms in this context due to its efficiency and versatility is the *Rapidly-exploring Random Tree (RRT)* [13]. RRT is a probabilistic approach based on the generation of a tree-type graph rooted from the origin that, using heuristics of sampling nodes in the environment linked to a random factor, performs connections until one of its branches

reaches the target position. For a while, RRT presented satisfactory results only for applications with a fixed destination point, becoming even better years later, from an asymptotically optimal variant proposed by Karaman and Frazzoli [10] called RRT* [1]. This algorithm uses a cost-to-go heuristic to guide the sampling toward the goal region and rewires the tree to minimize the cost of the path to each node, being able to converge to an optimal solution and producing smoother paths than RRT. However, the sampling function improvements and the connection with suitable control functions are still fields widely explored by SSL teams [27, 31].

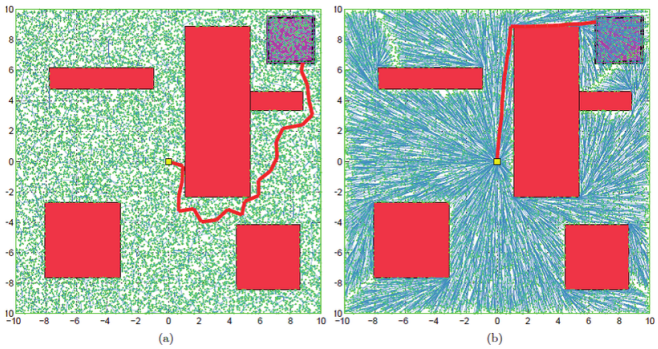


Figure 4: Comparison of the RRT (a) and RRT* (b) algorithms on a simulation example with obstacles. Source: Karaman and Frazzoli [10].

While RRT can incorporate some physical constraints – such as obstacles or kinematic constraints, target angle or target velocity – resulting in paths that are feasible in terms of collision avoidance, it may not always be possible or profitable to fully account for all physical constraints. Since it is a probabilistic algorithm, the generated paths may not always be the same, even for the same input parameters, making it challenging to validate and reproduce the generated paths in a physical sense.

As path planning is a constantly evolving field, where teams are always exploring new approaches and techniques to improve their robots' performance. Traditional teams as Tigers Mannheim and ER-Force used RRT but more recently proposed specific solutions for the creation of new paths taking advantage of game characteristics such as speed and dynamism. Instead of just applying changes to the RRT, the new approach focus on the generation of physically possible paths with improvements in the efficiency of route generation by avoiding the necessary post-processing.

In 2019, Tigers introduced a new planning algorithm [22] that prioritizes the quick reaction of the robots within a time limit over the generation of an optimal path. In this way, the chosen path can count on the presence of collisions within a predetermined time if another path cannot be obtained. In the following year, ER-Force [29] presented its new path planning algorithm capable of reaching positions with the desired velocity and acceleration.

[1] For the sake of simplicity, in this study, RRT* and others optimized variants will be refereed only as RRT.

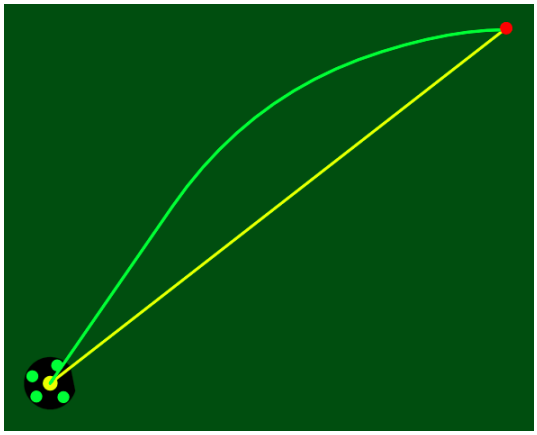


Figure 5: Trajectories with (in green) and without (in yellow) a maximum target velocity generated by ER-Force's path planning. Source: Andreas *et al.* [29].

For both teams, the observed limitations were overcome with a standard trajectory using bang-bang trajectories. A *bang-bang trajectory* is a type of motion characterized by abrupt and rapid changes in direction or velocity [16], and the leading term *bang-bang* refers to the sudden and sharp changes in the control signal used to manipulate the system. In this type of trajectory, the control signal switches between two extreme values corresponding to the maximum acceleration or deceleration of the system, allowing the trajectory generation as a function of the robot's physical motion.

2.3 CONTROL

There is a traditional and widely used procedure based on global navigation adopted by most of the SSL teams, which is available as a standard way for simulations in the category [21, 3]. The main information sent by off-field computers as the in-place control input in this procedure consists of the tuple $\langle v_x, v_y, \omega \rangle$ (explained below), usually a result from off-field control, directly related to control error techniques with emphasis on *Proportional-Integral-Derivative (PID)* controllers. A PID controller is a type of feedback controller commonly used in engineering and control systems to regulate a process variable. It consists in three main components that name this technique: the proportional term (P) – which generates an output signal proportional to the difference between the setpoint and the actual process variable – the integral term (I) – that generates an output signal that is proportional to the integral of the error signal over time – and the derivative term (D) – that generates an output signal that is proportional to the rate of change of the error signal.

By combining these three components, a PID controller can effectively regulate a process or a system by adjusting the output signal based on the difference between the setpoint and the process variable. The gains for each component can be adjusted to optimize the controller's performance for a particular application.

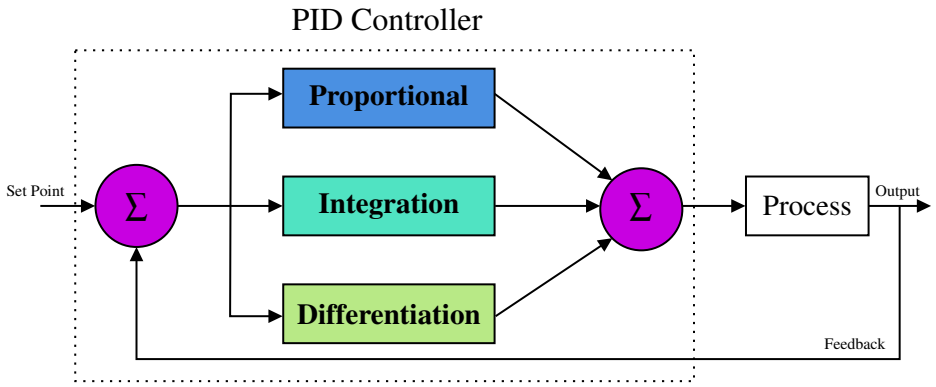


Figure 6: A block diagram of a PID controller in feedback loop. Source: The author.

The mentioned triplet is composed of two components related to the linear velocity of the robot – v_x and v_y –, usually measured in *meters per second*, and a third component indicating its angular velocity – ω –, usually measured in *radians per second*. Together, these three values specify the robot's motion, allowing it to move in any direction and turn.

To effectively apply the in-place control, the robot must perform three steps. First, it must convert the 3-tuple into individual wheel velocities – σ_N, where each wheel $N \in 1,4$ – to achieve the desired motion using the equations below [25] – where M_{N_θ}, R_r, W_r represents the angles, robot and wheel radius, respectively (Figure 7).

$$\sigma_1 = (sin(M_{1_\theta})/W_r) \times v_x + (cos(M_{1_\theta})/W_r) \times v_y + (R_r/W_r) \times \omega$$
$$\sigma_2 = (sin(M_{2_\theta})/W_r) \times v_x + (-cos(M_{2_\theta})/W_r) \times v_y + (R_r/W_r) \times \omega$$
$$\sigma_3 = (-sin(M_{3_\theta})/W_r) \times v_x + (-cos(M_{3_\theta})/W_r) \times v_y + (R_r/W_r) \times \omega$$
$$\sigma_4 = (-sin(M_{4_\theta})/W_r) \times v_x + (cos(M_{4_\theta})/W_r) \times v_y + (R_r/W_r) \times \omega$$

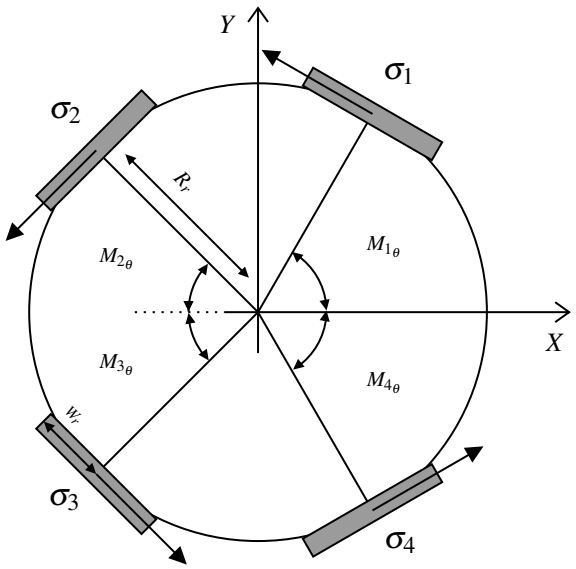


Figure 7: Force diagram of SSL's robot with wheels angular distribution. Source: Adapted from RobôCIn [2].

Second, the robot must control the wheel velocities using a feedback loop, where it adjusts the wheel velocities to maintain the desired motion and ensure the robot moves as intended. The robot's sensors and actuators provide feedback to the control system, which adjusts the wheel velocities as necessary.

And finally, the robot must continuously adjust the wheel velocities to maintain the desired motion as it moves around the playing field. The control system receives feedback from the robot's sensors and adjusts the wheel velocities to keep the robot on track. This ensures that the robot moves accurately, precisely, and that it can navigate the playing field effectively.

In summary, a SSL robot uses a combination of translational and rotational motion, along with a feedback loop, to achieve the desired motion.

3

REINFORCEMENT LEARNING

This chapter provides an overview of Reinforcement Learning, along with the specific technique employed in this study.

3.1 OVERVIEW

Reinforcement Learning is a subarea of Machine Learning that focuses on how an agent can learn to make decisions and take actions in an environment to maximize its cumulative reward. It is inspired by the hypotheses that the natural learning process of animals consists in making mistakes, receiving feedback, and adjusting actions accordingly. RL algorithms seek to replicate this process by enabling agents to interact with its environment, receive feedback in the form of reward signals, and learn from their experiences to make better decisions in the future. This trial-and-error learning process allows the agents to adapt to changing environments and optimize their behavior over time, much like animals do through experience.

To get into the subject, some main concepts (Figure 8) must be understood, which are:

- *Agent*: is the learner or decision maker that interacts with the environment.
- *Environment*: is the context in which the agent operates, and it can be anything from a simple game to a complex real-world scenario. An environment is formally defined as a *Markov Decision Process (MDP)* [6], described as a 4-tuple $\langle S, A, P_a, R_a \rangle$, where $P_a(S, S')$ is the probability of transitioning from state S to state $S' = S_{t+1}$ under action $a = A_t$.
- *State or Observation* (S): is the current situation or configuration of the environment used to determine the agent's next action. A state $s = S_t$ describes the state in a certain time t.
- *Action* (A): is the decision made by the agent based on the current state, and it affects the subsequent state of the environment.
- *Reward* (R): is a scalar feedback signal that the agent receives from the environment used to evaluate how good the agent's actions were so far. A reward $R_a(S, S')$ is the

immediate reward received upon transitioning from state S to state S' under action a. A reward $R_t = r$ describes the reward in a certain time t.

- *Policy* (π): is the agent's strategy for selecting actions in each state where the optimal policy is called π^*.

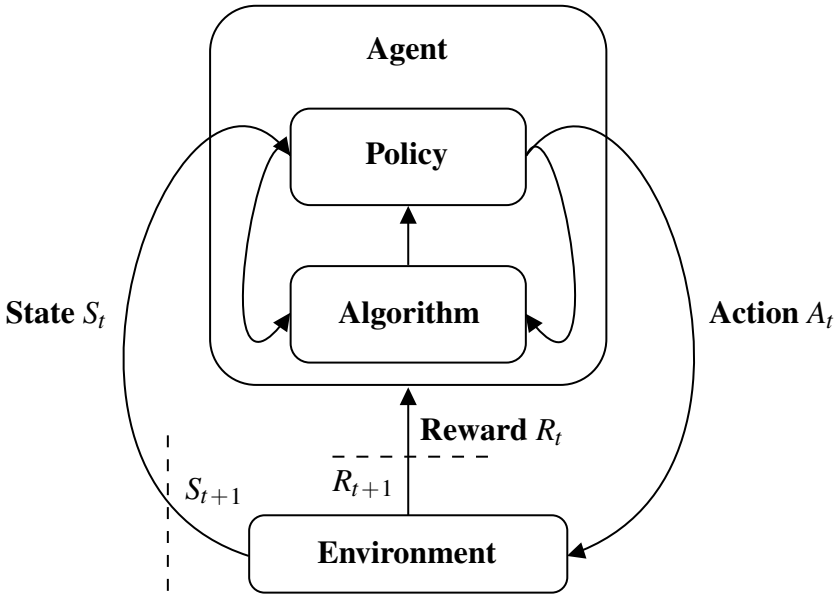


Figure 8: Execution cycle of an RL model with main concepts connection. Source: Adapted from Sutton [26].

Some important definitions based on previously introduced concepts are:

- *Episode or Epoch* (τ): is a sequence of steps taken by the agent in the environment, starting from an initial state and ending in a terminal state. The duration of an episode can be fixed or variable, depending on the environment.

- *Step*: is a single interaction between the agent and the environment, in which the agent observes the current state and takes an action based on its policy, receiving a reward, and the next state from the environment.

- *Experience Replay or Memory*: is a technique used to store and reuse experiences that an agent obtained during training. In experience replay, a subset of past experiences is randomly sampled from a memory buffer and used to update the agent's policy or value function.

- *Value* (V): is a measure of how good it is for the agent to be in a state or take an action. It is defined as the expected cumulative reward that the agent can receive starting from that state and following its policy. A value $V(s,a)$ describes the particular measure of how good it is to take action a in state s.

- *Q-Value* (Q): is defined as the expected cumulative reward that the agent can receive starting from a state, taking an action and following its policy. A Q-Value $Q(s,a)$ describes the particular cumulative reward from a state s taking an action a and the optimal Q-Value is called Q^*.

All these concepts fit perfectly into a robot soccer scenario. In this case, the *agent* can be a single robot or the entire team. The *environment* is the soccer field and the other team. The *state* is the positions of the robots, the ball, and the opposing team, but it could also include information about the score, time remaining and any penalties or other events that occur during the game. The *action* can be any movement or manipulation of the robot or ball – for example, a robot could move forward, backward, sideways, kick the ball or pass the ball to another robot. The *reward* can be a positive value for scoring a goal, a negative value for allowing the opposing team to score or a small positive or negative value for actions that contribute to or detract from the team's overall performance. Finally, the *policy* can be the strategy that the robots use to make decisions about their actions based on the current state of the game – for example, the policy might be to pass the ball to a teammate who is in a better position to score or to defend the goal when the opposing team has possession of the ball.

3.2 RSOCCER GYM

Reinforcement Learning algorithms are currently being trained using a variety of techniques and tools, including *Gyms*, which are platforms that provide a set of environments and APIs for developing, testing, and comparing RL algorithms. These gyms are often used to train and evaluate agents in different scenarios, such as playing video games, controlling robots, or optimizing financial portfolios. Some of the key features of an RL gym may include support for multiple programming languages, visualization tools, and integration with popular deep learning frameworks like *TensorFlow* [5] or *PyTorch* [23].

For this study, the rSoccer Gym was chosen. *rSoccer* was introduced by Martins, F.B. *et al.* [18] as an open-source framework optimized for Reinforcement Learning applications in the context of SSL and IEEE Very Small Size Soccer (VSS) [4], another category in robot soccer competition, environments.

This framework offers a flexible and customizable platform for researchers and practitioners to experiment with various RL algorithms and techniques, and to develop advanced strategies and tactics for autonomous robotic soccer players. It is comprises three core modules: simulator, environment, and render. Each of these modules (Figure 9) plays an essential role in the overall functionality of the framework.

The *simulator* module is based on grSim – the official SSL's simulator -, and it is responsible for the physics of the soccer game environment – *i.e.* simulates the movement of the robots, ball, and other objects within the game world. The *environment* module, which complies with the OpenAI Gym framework [7], serves as the bridge between the agent and the simulator,

receiving the agent's actions, communicating with the other modules using the entities structure (explained later) and returning new observations and rewards. Finally, the *render* module is responsible for visualizing the environment and rendering it for the user – this module creates a 2D visual representation of the game world, allowing users to see the robots, ball, and other objects in action.

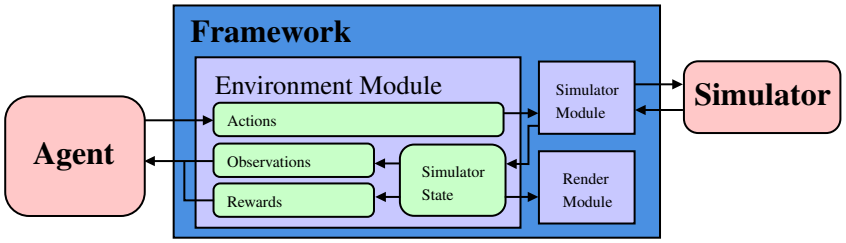


Figure 9: rSoccer Gym Framework modules architecture. Source: Adapted from Martins *et al.* [18].

In order to facilitate communication between modules within every environment, a specific set of data structures known as *entities* has been defined:

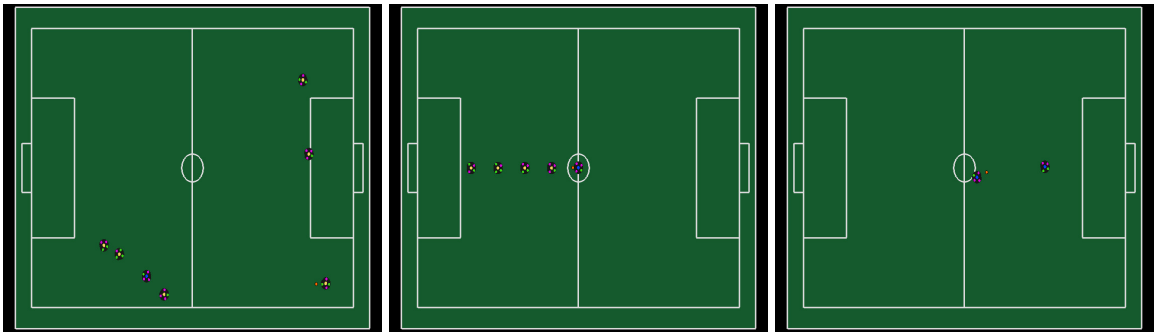
- *Ball*: contains information about the position and velocity of the ball.
- *Robot*: contains identification information, as well as position, velocity and wheel speed values. This entity can be used to read the current state of the robot or to send control commands that affect its movement and actions.
- *Frame*: contains both Ball and Robot entities for each robot in the game, which are structured in a way that makes it easy to index each robot by team color and ID. This entity is used to store the complete state of the simulation and to enable modules to access all relevant information about the game environment.
- *Field*: contains important specifications about the simulation values, such as the geometry and parameters of the field and robots. This entity serves as a reference point for the entire simulation and provides a standardized set of parameters that can be used to ensure consistent performance and behavior across different modules and environments.

For a complete definition of the RL environment, it is necessary to implement the following four methods:

- **`frame_to_observations`**: returns an array of observations that will be sent to the agent as specified by the environment.
- **`get_commands`**: returns a list of Robot entities containing the commands sent to the simulator from a list of actions.

- **`get_initial_positions_frame`**: returns a Frame entity used to define the initial positions of both ball and robots.
- **`calculate_reward_and_done`**: returns the computed step reward and a boolean value that denotes whether the present state is terminal.

In order to demonstrate the environment's workability, in rSoccer Gym, several tasks such as **`Go To Ball`**, **`Dribbling`** and **`Pass Endurance`** (Figure 10) using different RL algorithms were successfully implemented and tested. As result of the validation, the Deep Deterministic Policy Gradient stood out for its performance in solving different proposed problems, configuring one of the best and more commonly RL methods and being, therefore, chosen for use in the object of this study.



(a) Go To Ball (b) Dribbling (c) Pass Endurance

Figure 10: Example of SSL's environments on rSoccer's render view, where it is possible to visualize differences in the environment's distribution within each agent's learning needs. Source: Martins *et al.* [18].

3.3 DEEP DETERMINISTIC POLICY GRADIENT

Deep Deterministic Policy Gradient (DDPG) is a type of Reinforcement Learning algorithm developed by Lillicrap *et al.* [14] at Google DeepMind in 2015 that can learn to make decisions in environments with continuous action spaces, such as robotics or game playing.

The *policy* in DDPG refers to the agent's decision-making strategy, as explained before; the *gradient* part of the name refers to the use of gradient descent to update the neural network parameters; it is *deterministic* because it learns a deterministic policy, which means that given a particular state, it will always output the same action – this is different from stochastic policies, which output a distribution over the action space.

This algorithm uses a combination of two neural networks: a *critic* and an *actor* network (Figure 11). The critic network – which represents the value function –, evaluates how good the agent's actions are in a given state, providing feedback on how good or bad the actions were in achieving the overall goal. The actor network – which represents the policy –, decides which actions to take in a given state, being responsible for selecting the actions that maximize the

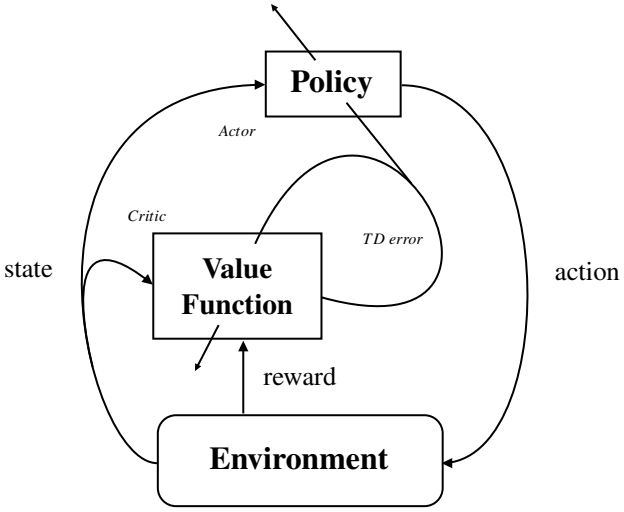


Figure 11: Execution cycle of an Actor-Critic System. Source: Adapted from Sutton [26].

expected cumulative reward in the long run and mapping the state to actions at the end. These networks are connected by the *TD error*, which is the difference between the predicted value of the current state and the observed reward received by the agent, being used to update the weights of the critic network to reduce the prediction error.

During training, the critic network is used to calculate a Q-Value and the actor network is then updated using the gradient of the calculated value with respect to the actor's parameters. Thus, by iteratively updating both the actor and critic networks, DDPG can learn to make better decisions in the environment over time.

4

PROPOSED APPROACH

In this chapter, the proposed approach is presented. It aims to use RL to create path planning for a single SSL robot, moving it from a position given by the 5-tuple $\langle s_x, s_y, v_x, v_y, \theta \rangle$ to a target $\langle s'_x, s'_y, v'_x, v'_y, \theta' \rangle$ where the target position corresponds to the vector (s_x, s_y), the target velocity to the vector (v_x, v_y) and the target angle to θ (Figure 12).

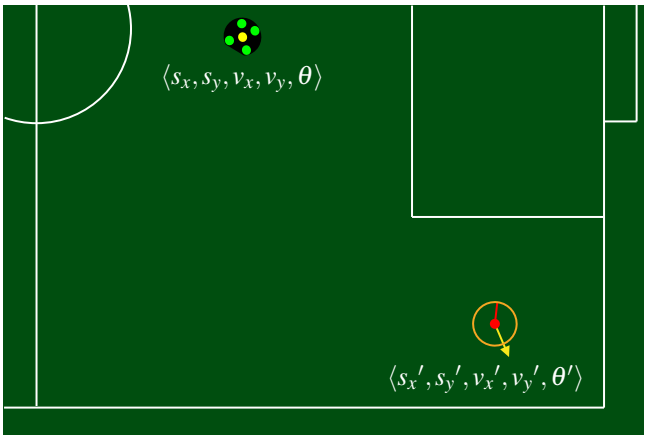
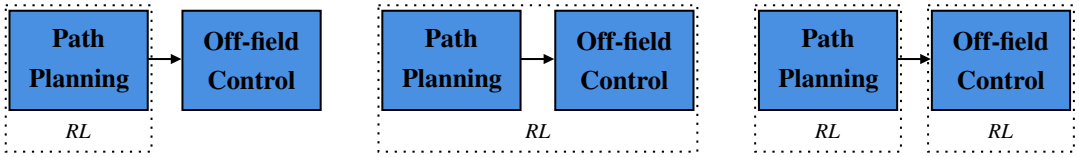


Figure 12: Overview of the proposed application, showing the robot's target position (in orange), angle (in red), and velocity with magnitude and direction (in yellow). Source: The author.

This approach focuses on two of the three possible approaches to apply Reinforcement Learning in the path-planning context shown in Figure 13: *Path Planning For A Given Control Approach* and *Single Component Approach*. The first is path planning with the aid of a high-level control, initially applied to create a base comparison model for the final approach. In other words, the supply of a off-field control that already worked in a real-world environment makes the analysis and study of path planning more deterministic, eliminating the probability of occurrence of problems that are not necessarily related to path planning. Thus, it generates a base behavior for future comparison of more complex approaches, such as the Single Component Approach, where the network has the ability to learn to move without aids, directly providing the necessary input to the low-level control.

The third possible alternative known as *Hierarchical Approach* was disregarded as it is outside the scope of the study since it consists of a two-step hierarchical learning, where the

out-of-field control stage would be part of an isolated network, receiving as input the output of the path planning created by another one.



(a) Path Planning for a Given Control Approach

(b) Single Component Approach

(c) Hierarchical Approach

Figure 13: Three possible approaches for path planning with Reinforcement Learning. Source: The author.

4.1 ENVIRONMENT SETUP

The entire development used an SSL environment and a Reinforcement Learning algorithm: the rSoccer as the Gym responsible for providing the desired environment and rendering animations capable of displaying the preliminary results, and the DDPG algorithm, responsible for guiding the entity's learning, respectively.

Besides, in order to implement the RL environment in rSoccer, the **`SSLBaseEnv`** class provided by the framework was inherited and the necessary observations were defined as an array of real numbers parameterized as shown below:

- $T = \langle s_x, s_y, v_x, v_y, \cos(\theta), \sin(\theta) \rangle$ be the tuple of observations related to the target.
- $B = \langle s_x, s_y, v_x, v_y \rangle$ be the tuple of observations related to the ball.
- $R_n = \langle s_x, s_y, v_x, v_y, v_\theta, \cos(\theta), \sin(\theta) \rangle$ be the tuple of observations related to the n^{th} robot [1].

Then, the observations size can be expressed as:

$$|O| = |T| + |B| + \sum_{n=1}^{N} |R_n|$$

Where $N \geq 1$ is the total number of robots. For all applicable entities, (s_x, s_y) represents the position in the field, (v_x, v_y) the velocity vector, v_θ the angular velocity, and θ, the orientation [2]. The values represented in the observations and the reward's accumulated value were normalized in the range [-1, 1] by dividing each value by the constant representing its maximum value.

[1] The robot observations for $n = 1$ are relative to the robot that must be controlled.

[2] While a range of [-1, 1] may help represent normalized values in some contexts, this range cannot represent angles. Therefore, the angle was represented in polar format to ensure functionality in the desired four quadrants.

However, for the complete definition of the environment, the necessary methods defined in Section 3.2 were overridden with the implementations described below, except the **`calculate_reward_and_done`**, since it represents the most significant and mutable function, defining the best possible result for both approaches.

The **`frame_to_observations`** performs a direct conversion of rSoccer frame to observations in the described format. The **`get_initial_positions_frame`** defines the random positioning of the entities within the field, ensuring a minimum distance during the generation and the target considering the limitations of each magnitude already mentioned. The **`get_commands`** receives the network actions, denormalizes them to the appropriate position, velocity, and angle values, performs the off-field control call, when it exists, and sends the displacement triplet to the robot as required by rSoccer.

4.2 PATH PLANNING FOR A GIVEN CONTROL APPROACH

For the implementation of the path planning for a given control approach, a minimum off-field control capable of helping the agent to learn was initially made, giving the agent the ability to identify the best route for a given control based on different reward functions. Additionally, the environment actions were defined as the 5-tuple target of displacement described earlier, which will serve as input to off-field control.

4.2.1 Off-field Control Algorithm

The off-field control implementation (Algorithm 1) was developed based on a simplification of the algorithm initially used by RobôCIn team – available in the team's open source [2]. It consists of a PID control of the angular error, without integrative-derivative corrections, and essentially establishes two distances to the target point to guide the optimal movement: one for the proportional velocity adjustment and the other for the final angle adjustment (Figure 14).

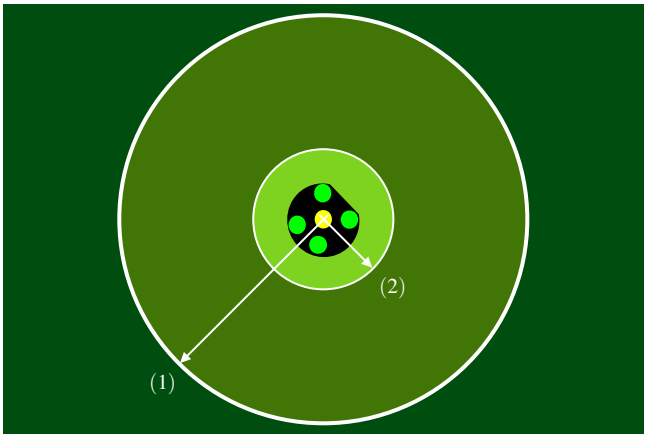


Figure 14: The proportional velocity adjustment distance (1) and the final angle adjustment distance (2). Source: The author.

The proportional velocity adjustment distance (v_{p_r}) is an offset centered on the target point where the robot must gradually approach the final velocity. Within this range, a mapping of the distance from the robot to the target is performed between a minimum value ($v_{p_{lb}}$) within the range [0, 1] to 1, where the minimum value corresponds to the mapped distance when the applied proportional velocity gives enough power to the motors move the robot, which can remain at maximum velocity (v_{max}) for longer distances to reach the desired destination as fast as possible. The final angle adjustment distance (a_{a_r}) is an offset in which the robot is considerably closer to the target point. Inside this radius, the robot only tries to adjust its face to the desired direction until it is within the desired tolerance (a_ε). Otherwise, it moves in the direction of the goal, limiting the linear velocity on the angular error, first adjusting its face to the target, and only after that prioritizing the displacement.

Algorithm 1: Off-field Control Algorithm

function OFFFIELDCONTROL($robot, target$)

$max\_velocity \leftarrow v_{max}$

$target\_velocity\_norm \leftarrow$ VECTORNORM($target_v$)

$distance\_to\_goal \leftarrow$ DISTANCEBETWEEN($robot_s, target_s$)

$angle\_error \leftarrow$ SMALLESTANGLEDIFF($robot_\theta, target_\theta$)

if $distance\_to\_goal \leq v_{p_r}$ **then**

$max\_velocity \leftarrow$
$max\_velocity \times$ RANGEMAPPER($distance\_to\_goal, 0.0, v_{p_r}, v_{p_{lb}}, 1.0$)

$max\_velocity \leftarrow$ MAX($max\_velocity, target\_velocity\_norm$)

if $distance\_to\_goal > a_{a_r}$ **then**

$theta \leftarrow$ VECTORANGLE($target_s - robot_s$)

$v\_prop \leftarrow$
ABSSMALLESTANGLEDIFF($\pi - a_\varepsilon, angle\_error$) $\times$ ($max\_velocity/(\pi - a_\varepsilon)$)

return $\langle$FROMPOLAR($theta, v\_prop$), $ANGLE\_K\_P \times angle\_error\rangle$

```
/* ANGLE_K_P is the PID control's proportional
constant.                                          */
```

return $\langle target_v, ANGLE\_K\_P \times angle\_error\rangle$

4.2.2 Reward Function

With the environment defined, obtaining the best reward was the main challenge involved in this study. It was decided to add velocity and angle, the other requirements incrementally, and testing, each receiving the target, the current frame, and the next frame, prioritizing reaching the terminal state before time runs out to maximize its reward. The reward composition was performed using means with empirical weightings and obtained through a binary search for all cases.

4.2.2.1 Initial Reward

The initial reward (Algorithm 2) consists in a minimal non-sparse monotonic function, that prioritizes arrival at the destination position, characterizing the main requirement of the solution and representing the simplest solution for achieving the desired goal.

Algorithm 2: Initial Reward Function

function REWARDFUNCTION($frame$, $frame'$, $target$)

$distance\_robot\_target \leftarrow$ DISTANCEBETWEEN($frame'_{robot_s}, target_s$)

if $distance\_robot\_target \leq DISTANCE\_TOLERANCE$ **then**

return $\langle 0, TRUE \rangle$

$last\_distance\_robot\_target \leftarrow$ DISTANCEBETWEEN($frame_{robot_s}, target_s$)

$step\_reward \leftarrow \frac{last\_distance\_robot\_target - distance\_robot\_target}{2 \times MAX\_POSITION}$

return $\langle step\_reward, FALSE \rangle$

This function subtracts the last distance from the current distance to the target, incentivizing the robot to progress toward the target location while also considering its past performance. When the robot starts moving, the current distance to the target is typically large, so the reward for moving toward the target will be high and still be positive if the robot progresses to the same destination. If the robot stops progressing or moves away, the current distance to the target will increase. Consequently, the reward for moving toward the target will decrease accordingly, incentivizing it to continue progressing. It can help the robot avoid getting stuck in undesired states or wandering in the environment.

Since the path performed is control-dependent and the environment already took into account the physical possibilities for the robot to move, there was no need for post-processing, and the improvement of the reward function was the determining factor for changing the generated path.

4.2.2.2 Angle addition

The addition of the target angle is the first increment proposed to the original solution. Besides being close to the target for training completion, it was also necessary to be within an angle tolerance. However, for some approaches, the angular reward studied also subtracts the previous angular error from the current one, consisting in the angular correction, forcing a gradual convergence to the acceptable tolerance, similar to what happened with distance.

Three different approaches to this requirement were applied: rewarding angular correction only at the control's angular adjustment radius – mentioned previously in Section 4.2.1 –, not rewarding angular correction at all – only account with angle tolerance –, and rewarding angular correction for the entire path.

4.2.2.3 *Velocity addition*

The addition of the velocity constant is the biggest challenge for determining the final reward function, since it is a physical quantity that can imply a local worsening of distance for a global improvement of the complete state in the shortest possible time. For instance, only the ER-Force among SSL teams, as mentioned in Section 2.2, can create a route considering non-null velocities.

Initially, this reward function was designed independently of the angle, with the aim of isolating the problem and reducing the randomness of the results. For this case, the reward function was analyzed for null and non-null velocities, focusing on three types of heuristics: managing to finish the training without rewarding, rewarding or penalizing the agent for the difference in its velocity to the desired velocity, the last one being accounted only at the end of the training.

In the first, the training only ended when the robot's velocity was within the target margin, given as a tolerance for the target velocity. For the second, the agent was rewarded when, within distance tolerance, the velocity approached the target velocity. Moreover, for the last one, the assigned value was given by the difference to the goal multiplied by a constant.

Thereafter, considering the position-angle-velocity composition, the same approaches were used, and a final reward function was finally established (Algorithm 3).

Algorithm 3: Final Reward Function

function REWARDFUNCTION($frame$, $frame'$, $target$)

$last\_distance\_robot\_target \leftarrow$ DISTANCEBETWEEN($frame_{robot_s}, target_s$)

$distance\_robot\_target \leftarrow$ DISTANCEBETWEEN($frame'_{robot_s}, target_s$)

$last\_angle\_error \leftarrow$ SMALLESTANGLEDIFF($frame_{robot_\theta}, target_\theta$)

$angle\_error \leftarrow$ SMALLESTANGLEDIFF($frame'_{robot_\theta}, target_\theta$)

$last\_velocity\_error \leftarrow$ DISTANCEBETWEEN($robot_v, target_v$)

$velocity\_error \leftarrow$ DISTANCEBETWEEN($robot'_v, target_v$)

$distance\_reward \leftarrow K\_DISTANCE \times \frac{last\_distance\_robot\_target - distance\_robot\_target}{2 \times MAX\_POSITION}$

$angle\_reward \leftarrow K\_ANGLE \times \frac{last\_angle\_error - angle\_error}{\pi}$

$velocity\_reward \leftarrow K\_VELOCITY \times \frac{(last\_velocity\_error - velocity\_error)}{MAX\_VELOCITY}$

```
// K_DISTANCE, K_VELOCITY and K_ANGLE are constants
 that make up the normalization of the final reward.
```

if $distance\_robot\_target \leq DISTANCE\_TOLERANCE$ **then**

  if $velocity\_error \leq VELOCITY\_TOLERANCE$ **then**

    return $\langle angle\_reward, angle\_error \leq ANGLE\_TOLERANCE \rangle$

  return $\langle angle\_reward + velocity\_reward, FALSE \rangle$

return $\langle distance\_reward + angle\_reward, FALSE \rangle$

This function rewards position and angle as described previously, with the addition of a reward for velocity while subtracting the error of the previous velocity from the current one. The angle reward, like the distance reward, occurs throughout the trip, while the velocity reward is only given when within position tolerance. The terminal state is reached when the position-speed-angle triple is satisfied, respectively, within its tolerances.

4.3 SINGLE COMPONENT APPROACH

The proven versatility of RL algorithms enables the idea of encompassing steps necessary for an environment's development. Therefore, the implementation of the Single Component Approach can be seen as a natural evolution of this study, where after the validation of different reward functions based on a given pre-determined control, it was supposed that the network could learn without the aid of it, obtaining outcomes in the same direction.

Due to the previous analysis, it was possible to obtain a comparative basis to determine which results are considered satisfactory and to imagine the creation of paths capable of circumventing the limitations of the previously presented control. The final and most promising reward functions based on the position-angle and position-angle-velocity composition were applied to this new approach. However, the specific scenario of not rewarding angle correction at all was also applied, since it is directly impacted by the control.

The structure of the environment remained as described in the previous section, with the exception of the actions, which now directly return the control triplet $\langle v_x, v_y, \omega \rangle$ mentioned in Section 2.3.

5

RESULTS

This chapter presents the results obtained from the reward functions described previously, as well as comparisons between the approaches studied and the limitations found during development.

5.1 INITIAL REWARD

For the evaluation of the robot's movement, all other entities that would characterize an obstacle were removed. As a result of running the environment in this scenario, a rapid convergence in agent learning was observed, achieving the expected result. In addition, the robot moved quickly but in a non-linear trajectory, unlike the linear route taken by the direct call to the off-field control (Figure 15).

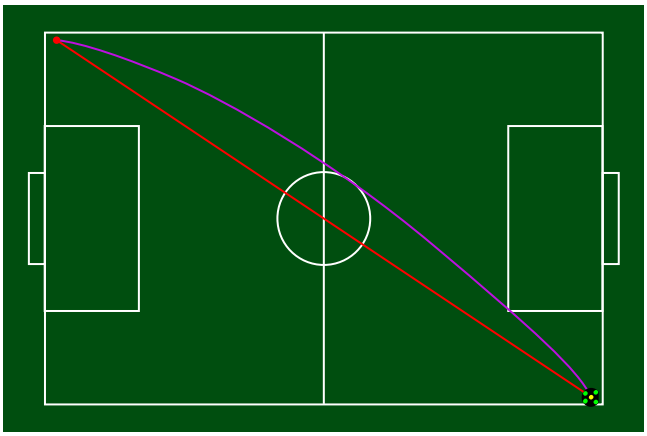
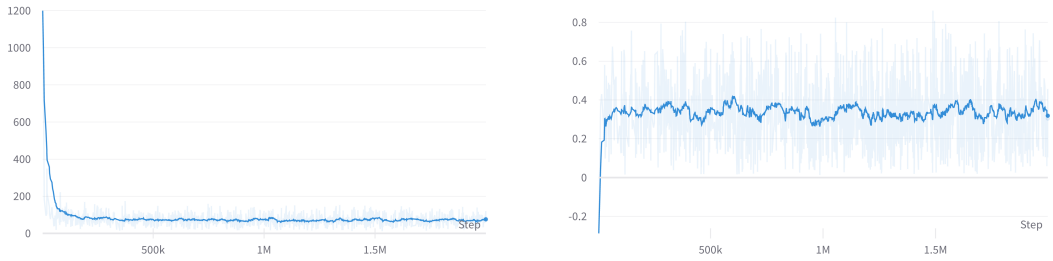


Figure 15: Route comparison between direct control (in red) and path planning for a given off-field control (in magenta) using the initial reward. Source: The author.

Since there are no velocity or angle limitations, going in a straight path is probably the best option. However, since the path obtained by applying RL to the off-field control relies in an angular limitation for the control implemented as a function of the received angle, and taking into account the environment and all the surrounding factors, the network possibly tried to circumvent the existing angular limitation, while trying to get as close as possible to the goal, generating a non-linear path.

As the initial reward aimed mainly the validation of the environment and learning method, the results were considered as good enough to proceed to more complex functions, where for less than two hundred thousand iterations out of a total of two million – performed in all analyses onward –, the algorithm was able to achieve the total cumulative reward mean and get the lowest average number performed reward over time, as can be seen in Figure 16.



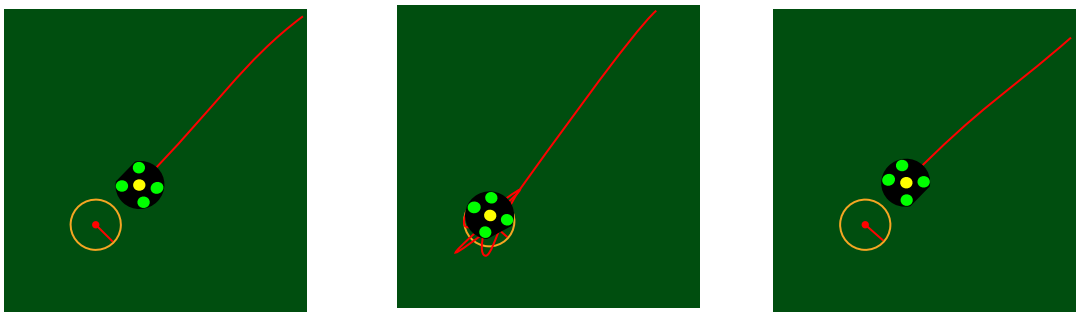
(a) Number of steps

(b) Total reward accumulated

Figure 16: Mean (lines) and standard deviation (shades) of the number of steps and total reward metrics accumulated (Y-axis) both over time (X-axis) to the first reward. Source: The author.

5.2 POSITION-ANGLE REWARD

For this composed reward, three main results were obtained using the off-field control (Figure 17), each related to one of the approaches discussed in Section 4.2.2.2. For all of them, the ease of reaching the target position was observed, as visualized in the initial reward. However, none of the three approaches could avoid angular instabilities altogether when close to the target position, which was reduced and became infrequent with an increase in the chosen angular tolerance value, except when there was no angular reward, where the robot could not converge as in the other cases.



(a) Rewarding angular correction only at the control's angular adjustment radius

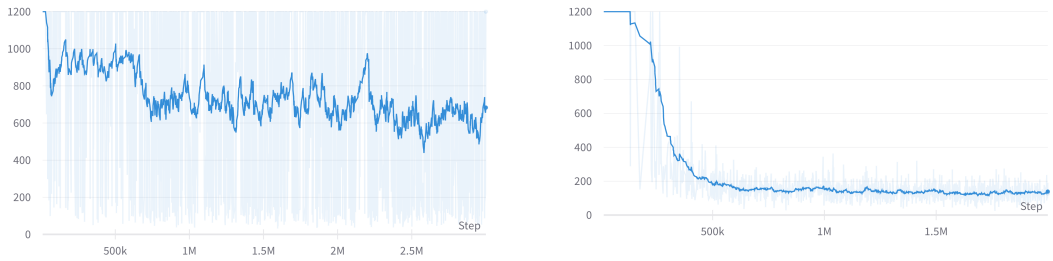
(b) Not rewarding angular correction

(c) Rewarding angular correction for the entire path

Figure 17: Visualization of the approach's behavior close to the target when using the off-field control, where the target angle consists of the vector (in red) starting from the center of the objective (in orange). Source: The author.

Rewarding angular correction only at the control's angular adjustment radius induced the agent to arrive with the most different angle in order to accumulate the largest amount of available reward.

Even though the network with not rewarding angular correction heuristic has reached quickly close to the target position, a difficulty in the convergence over time was noticed when it depends on the implemented control, not achieving such good results. It can be explained due to angular instabilities when close to the target position, which prioritizes an angular correction as a function of displacement, as mentioned in Section 4.2.1, impacting the speed peaks and the agent performance. Thus, the agent positioning was more stable for the Single Component Approach, which has its control.



(a) With aid of off-field control

(b) Without aid of off-field control

Figure 18: Mean (lines) and standard deviation (shades) of the number of steps (Y-axis) over time (X-axis) for both approaches. Source: The author.

Finally, rewarding angular correction for the entire path presented itself as the correct solution within those presented, since it approximates what the control expects for linear movement. The application of this last heuristic for the Single Component Approach resulted in a similar generated route, but with improvements in velocity as shown in Table 1.

Table 1: Comparison of completion time of rewarding angular correction for the entire path heuristic between Path Planning for a Given Control and Single Component approaches, running across the field diagonal with the target angle right since the beginning of the movement.

Path Planning for a Given Control Approach (ms)	Single Component Approach (ms)
4221	3957
4138	3874
4193	3896
4215	3901
4239	3875
4207	3941
4095	3899
4108	3910
4213	3813
4183	3980
4181.2	3904.6

5.3 POSITION-ANGLE-VELOCITY REWARD

From start, the most evident result for the Position-Angle-Velocity reward was that the model converged when the target velocity was null, but the same was not true when it assumed a not null value. As mentioned before, this reward consists of the hardest composition due to physical variables: the robot may be close to the final goal in terms of distance, but far away in terms of velocity. In such a scenario (Figure 19), there is not enough space for the robot to get fast enough, and thus a distance from the final state position presents itself as necessary, an intrinsic limitation of the environment in question.

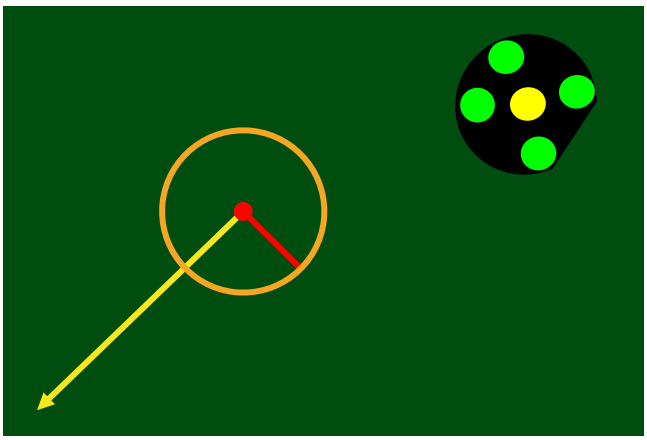


Figure 19: Robot with initial position near the target position (in orange), angle near the target angle (in red) and without enough space to reach the desired speed (in yellow). Source: The author.

Based on the discussed approaches in Section 5.2, for both Position-Velocity and Position-Angle-Velocity, instabilities in angular correction near the goal position continued to occur, potentially due to a lack of verification in the angular velocity values obtained upon arrival at the goal. However, for the scenario where the target velocity is null, it is possible to get around this problem, since the approximation of an already small distance can be performed by heuristics, avoiding singularities performed by the model.

Thus, the path obtained over time using the two approaches studied and calling the control directly, *i.e.*, without using the planning done using RL, was compared in two ways by traversing the field diagonal: with the angle already correct from the beginning of the path, and with it totally opposed to the target angle (Tables 2, 3).

For the way the final reward was constructed, the Single Component Approach was able to outperform, in terms of time, the control used as a reference for the study, which in turn performed worse than for an approach without the use of RL. This can perhaps be explained by the fact that the agent did not learn to extract the full potential of the existing control in cases without an obstacle, where it consists of a direct movement to the goal, which is possible to visualize in Figures 20 and 21. Although the routes of the Single Component Approach traveled the longest distance, it achieved the best times, on average, for the two tests performed, probably

because without the limitations of the control, the robot was able to combine the steps in a more adjusted, and therefore, faster way.

Table 2: Comparison of completion time of final reward between no RL applied, Path Planning for a Given Off-fiel Control and Single Component approaches, running across the field diagonal with the target angle right since the beginning of the movement.

Without RL application (ms)	With aid of control (ms)	Without aid of control (ms)
4351	4401	4099
4434	4510	4237
4454	4438	4143
4694	4561	4164
4436	4509	4080
4454	4432	4047
4419	4542	4180
4455	4479	4058
4445	4593	4132
4404	4582	4079
4454.6	4504.7	4121.9

Table 3: Comparison of completion time of final reward between no RL applied, Path Planning for a Given Off-fiel Control and Single Component approaches, running across the field diagonal starting with a totally opposite angle.

Without RL application (ms)	With aid of control (ms)	Without aid of control (ms)
4555	4554	4232
4556	4672	4360
4511	4621	4505
4583	4534	4606
4523	4659	4504
4538	4633	4253
4557	4652	4676
4514	4663	4367
4561	4591	4581
4639	4650	4656
4553.7	4622.9	4474

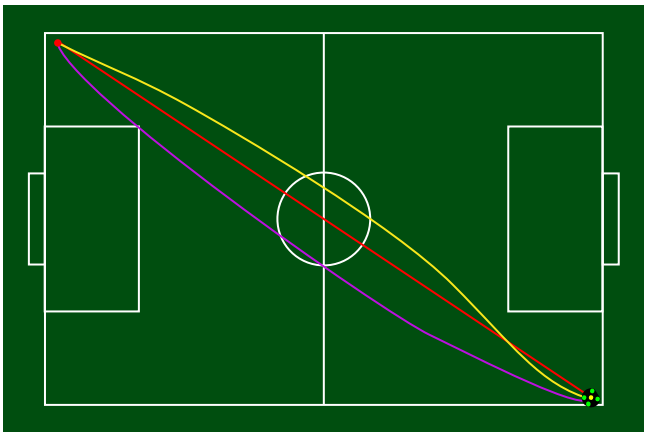


Figure 20: Route comparison between direct control (in red), path planning for a given off-field control (in magenta) and single component (in yellow) using the final reward with the target angle right since the beginning of the movement. Source: The author.

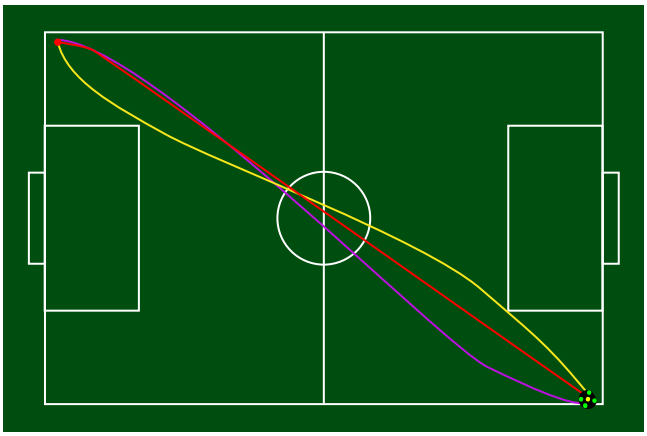


Figure 21: Route comparison between direct control (in red), path planning for a given off-field control (in magenta) and single component (in yellow) starting with a total opposite angle. Source: The author.

However, even without a control dependency, the agent was unable to dissociate the linear movement from the angular movement, which in theory, can be solved independently.

It is important to note that, although the results were obtained with only one robot in the field, the average inference time of the already trained network was below 1 *millisecond* using CPU, without the need for post-processing, which makes it capable of performing within the time required for real-time game situations, since for an application like SSL, this is far below the generally used camera refresh rate of 60 fps, which implies a maximum limit of approximately 16 *milliseconds*.

6

CONCLUSION

In the context of moving autonomous agents, navigating efficiently is essential. This is important in different applications, whether for cleaning robots, autonomous cars, or delivery drones. The challenge becomes even more significant in dynamic environments with requirements other than position, such as in real-time or simulated robot soccer game. Hence, adding physical quantities to the desired state, as angle and velocity, is a well-studied open problem in the SSL category. The non-convergence of the proposed solutions for not null velocities will potentially imply a change in the perception of the main reward initially considered, given that increasing the distance from the objective position may be necessary to obtain the objective velocity.

As intended at the beginning, proposals for learning methods and the definition of the specific agent for the problem were made, the necessary inputs and outputs were given, and the different methods were tested, with comparisons performed simultaneously, satisfying the specific objectives initially proposed. Thus, although the tests happened with only one robot in the field, the inference time below 1 *millisecond* makes the solution extremely efficient even not using GPU. Also, the tolerance values chosen have proven to be fundamental in obtaining good results, completely changing the convergence of a model.

Furthermore, it can be concluded that off-field control functions can be encompassed in the study of path planning since they can obtain better or equally satisfactory results, eliminating the limitations presented by control algorithms, like the one used in the study, which limited linear velocity in favor of angular correction. Although the results comprised the validation of the control to use path planning, they made it possible to understand that a change in the perception of how the reward was constructed is necessary to achieve the requirements of modern path planning.

In future works, reward functions that can disassociate local maxima to obtain a global maximum will be studied, also with scenarios where there is a change in the target, something that happens quite often in a soccer match, as well as the addition of static and dynamic obstacles.

REFERENCES

[1] (2015). About robocup small size league (ssl). https://ssl.robocup.org/about/. Accessed: January 21, 2023.

[2] (2016). RobôCIn's Open Source Code. https://github.com/robocin. Accessed: March 03, 2023.

[3] (2018). Robotics Erlangen Framework for RoboCup SSL Simulation. https://www.robotics-erlangen.de/framework. Accessed: March 19, 2023.

[4] (2020). Very Small Size Soccer League Website. https://ieeevss.github.io/vss/index.html. Accessed: March 22, 2023.

[5] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

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

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. cite arxiv:1606.01540.

[8] Elbanhawi, M. & Simic, M. (2014). Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77.

[9] Geiling, N. (2014). RoboCup: Building a Team of Robots That Will Beat The World Cup Champions. https://www.smithsonianmag.com/innovation/robocup-building-team-robots-will-beat-world-cup-champions-180951713. Accessed: March 10, 2023.

[10] Karaman, S. & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning.

[11] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., & Pérez, P. (2020). Deep reinforcement learning for autonomous driving: A survey.

[12] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: The robot world cup initiative. In *International Conference on Autonomous Agents*.

[13] LaValle, S. M. (1998). Rapidly-exploring random trees : a new tool for path planning. *The annual research report*.

[14] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2019). Continuous control with deep reinforcement learning.

[15] Lim, Y., Choi, S.-H., Kim, J.-H., & Kim, D.-H. (2008). Evolutionary univector field-based navigation with collision avoidance for mobile robot. *IFAC Proceedings Volumes*, 41(2):12787–12792. 17th IFAC World Congress.

[16] Lynch, K. M. & Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, USA, 1st edition.

[17] Mac, T. T., Copot, C., Tran, D. T., & De Keyser, R. (2016). Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28.

[18] Martins, F. B., Machado, M. G., Bassani, H. F., Braga, P. H. M., & Barros, E. S. (2022). rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer. In Alami, R., Biswas, J., Cakmak, M., & Obst, O., editors, *RoboCup 2021: Robot World Cup XXIV*, 165–176.

[19] Melo, J. G. & Barros, E. (2022). An embedded monocular vision approach for ground-aware objects detection and position estimation.

[20] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[21] Monajjemi, V., Koochakzadeh, A., & Ghidary, S. S. (2011). grsim - robocup small size robot soccer simulator. In *RoboCup*.

[22] Ommer, N., Ryll, A., & Geiger, M. (2019). Tigers mannheim (team interacting and game evolving robots) extended team description for robocup 2019. RoboCup Small Size League, Mannheim, Germany, 2019.

[23] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

[24] Patle, B., Babu L, G., Pandey, A., Parhi, D., & Jagadeesh, A. (2019). A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606.

[25] Purwin, O. & D'Andrea, R. (2005). Trajectory generation for four wheeled omnidirectional vehicles. In *Proceedings of the 2005, American Control Conference, 2005.*, 4979–4984 vol. 7.

[26] Sutton, R. S. & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[27] Takamichi Yoshimoto, Takato Horii, S. M. Y. I. & Zenji, S. (2019). Op-amp 2019 extended team discription paper. RoboCup Small Size League, Asagami Works, Osaka, Japan 2019.

[28] Uc-Cetina, V., Navarro-Guerrero, N., Martin-Gonzalez, A., Weber, C., & Wermter, S. (2022). Survey on reinforcement learning for language processing. *Artificial Intelligence Review*, 56(2):1543–1575.

[29] Wendler, A. & Heineken, T. (2020). Er-force 2020 extended team description paper. RoboCup Small Size League, Erlangen, Germany, 2020.

[30] Yoon, M. (2015). Developing basic soccer skills using reinforcement learning for the robocup small size league.

[31] Zheyuan Huang, Lingyun Chen, J. L. Y. W. Z. C. L. W. J. G. P. H. & Xiong, R. (2019). Zjunlict extended team description paper for robocup 2019. Zhejiang University, Zheda Road No.38, Hangzhou, Zhejiang Province, P.R.China 2019.

[32] Zhu, Y., Schwab, D., & Veloso, M. (2019). Learning primitive skills for mobile robots. In *2019 International Conference on Robotics and Automation (ICRA)*, 7597–7603.

[33] Zickler, S., Laue, T., Birbach, O., Wongphati, M., & Veloso, M. (2010). Ssl-vision: The shared vision system for the robocup small size league. In Baltes, J., Lagoudakis, M. G., Naruse, T., & Ghidary, S. S., editors, *RoboCup 2009: Robot Soccer World Cup XIII*, 425–436.

[34] Zolanvari, A., Shirazi, M. M., & Menhaj, M. B. (2019). A q-learning approach for controlling a robotic goalkeeper during penalty procedure.