Universidade do Minho
Escola de Engenharia

Bruno António Rodrigues de Sousa

Parallel, Angular and Perpendicular Parking for Autonomously Driven Vehicles

December 2021

Parallel, Angular and Perpendicular Parking for Autonomously Driven Vehicles

Bruno Sousa

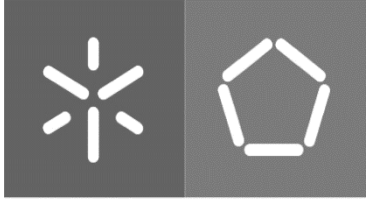
UMinho | 2021

Universidade do Minho
Escola de Engenharia

Bruno Sousa

Parallel, Angular and Perpendicular Parking for Autonomously Driven Vehicles

Guimarães, December 2021



Universidade do Minho
Escola de Engenharia

Bruno Sousa

Parallel, Angular and Perpendicular Parking for Autonomously Driven Vehicles

Dissertação de Mestrado
Mestrado Integrado em Engenharia
Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Fernando Ribeiro

Guimarães, December 2021

Acknowledgements

Throughout my academic journey, several people contributed with essential support and incentive that contributed to the completion of this dissertation, to whom I cannot forget to thank.

In first place, a very special thanks to my parents Antonio Sousa and Inês Rodrigues for always supporting me and providing unconditional love and belief in my capabilities. It would be impossible to complete this journey without their constant support. I would also like to thank my sister for all the love and support.

In second place, a big thanks to my supervisor, Professor Fernando Ribeirofor for the opportunity, the valuable guidance and the availability demonstrated throughout this dissertation. To the *Laboratório de Automação e Robótica* members for the great work environment, in particular to Tiago Ribeiro for always guided me in the right direction when nedded.

In third place, I would like to thank all my friends and classmates with whom I had the pleasure to share great adventures, especially to my closest friends André Gonçalves, Rafael Meleiro and Rafael Araújo, that always were there for me and in their own way always supported, encouraged me and made this journey filled with unforgettable memories.

Lastly, to all my family members who helped raise me and have always supported me throughout my life.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

O progresso para criar um veículo completamente autónomo tem aumentado constantemente nas últimas décadas e por consequência, o estacionamento autónomo tem sido uma área bastante investigada, uma vez que todas as viagem de carro têm de terminar com uma manobra de estacionamento. Nos últimos anos, com o recente sucesso da Aprendizagem por Reforço, a ideia de aplicar esta tecnologia para resolver o problema do estacionamento autónomo tem sido cada vez mais explorada.

Um veículo equipado com sistema de estacionamento autónomo tem de estacionar em três tipos de lugares de estacionamento, perpendicular, angular e paralelo. Qualquer sistema de estacionamento autónomo visa controlar o ângulo de direção e a velocidade do veículo, tendo em consideração o estado do ambiente para garantir uma manobra sem colisões dentro do espaço disponível. Assim, nesta dissertação, são apresentados dois métodos que visam resolver o problema do estacionamento autónomo para os três tipos de lugares de estacionamento, perpendicular, angular e paralelo, utilizando a Aprendizagem por Reforço.

Nesta dissertação, para cada método implementado é apresentada uma extensa explicação do método com a respectiva função de recompensa. A construção do ambiente e do agente no CopppeliaSim são apresentados juntamente com as configurações da implementação de ROS que é responsável por estabelecer a comunicação entre o CopppeliaSim e o Python script onde o algoritmo de Aprendizagem por Reforço foi implementado. O algoritmo de Aprendizagem por Reforço implementado foi o Deep Deterministic Policy Gradient (DDPG). Para os dois métodos, muitos treinos foram realizados para encontrar os hiperparâmetros ideais. O treino final e todas as etapas intermédias tambem são apresentadas. Por fim, foi realizada uma análise do comportamento do agente em todos os testes.

palavras-chave: Inteligência artificial, Aprendizagem Máquina, Aprendizagem por Reforço, Robô móvel autónomo, Robótica, Estacionamento autónomo, DDPG

Abstract

The progress to create a fully autonomous vehicle has been steadily increasing in the recent decades and by consequence, autonomous parking has been a well research field, since every driving trip has to end with a parking maneuver. In recent years, with the recent success of Reinforcement Learning, the idea of applying it to solve autonomous parking problem has been more and more explored.

A vehicle equipped with a complete autonomous parking system must be able to park in three types of parking spots, perpendicular, angular and parallel parking spots. Any autonomous parking system aims to control the steering angle and speed of the vehicle by taking into account the actual situation of the environment to ensure collision-free motion within the available space. Thus, in this dissertation, two approaches that aim to solve the autonomous parking problem for the three mentioned types of parking spots using Reinforcement Learning are presented.

In this dissertation, for each implemented approach an extensive explication of the method with the respective reward function is presented. The construction of the environment and the agent in CopppeliaSim are presented together with the configurations of the ROS implementation which is responsible for establishing the communication between CopppeliaSim and the Python script where the Reinforcement Learning algorithm was implemented. The Reinforcement Learning algorithm implemented was the Deep Deterministic Policy Gradient (DDPG). For the two approaches, an extensive search for the optimal hyper-parameters was realized and the final training and all the intermediate stages that lead to it are presented. Lastly, the agent's behaviour for all the tests was analyzed.

keywords: Artificial Intelligence, Machine Learning, Reinforcement Learning, Autonomous Mobile Robot, Robotics, Autonomous parking, DDPG

Table of Contents

List of Figures

List of Tables

Acronyms

AI - Artificial Intelligence

DDPG - Deep Deterministic Policy Gradient

FNR - The National Robotics Festival

GPU - Graphics Processing Unit

Lidar - Light Detection and Ranging

MC - Monte Carlo

ML - Macjine Learning

NN - Neural Network

PG - Policy Gradient

ROS - Robot Operating System

SAE - Society of Automotive Engineers

SPR - The Portuguese Robotic Society

TD - Temporal Different Learning

Chapter 1

Introduction

On every driving trip, in the final stage of the itinerary, the vehicle will have to be parked in a proper place. There are three different types of parking maneuvers that self-driving vehicles must be able to perform, parallel parking, angular parking and perpendicular parking. For autonomous cars to be able to park, they must be aware of the type of parking spot in which they have to park. For each type of parking, the vehicle has to know the parking spot position, it must prioritize safety by dynamically avoiding any obstacles that may appear, it must carry out all the necessary maneuvers in order to place the car in the correct place and be properly parked between the lines. In recent years Reinforcement Learning has been explored as a solution to the autonomous parking problem.

Reinforcement Learning uses a trial and error strategy for learning. For human beings, this idea of learning by experience in a trial and error strategy, that is, by interacting with the world is something natural because, since childhood, humans learn that every action has a reaction. Regardless of what human beings are doing, whether walking, running, having a conversation or even driving a car, they are fully aware of how the environment responds to each action taken and seek to influence what happens through their behaviour. This idea of goal-oriented learning through interactions with the environment is the basic idea behind Reinforcement Learning and is what separates it from other types of Machine Learning [9]. Reinforcement Learning can be defined as learning what to do in specif situations. In Reinforcement Learning the agent is the entity to be trained in order to learn how to perform the task in question. Each action it performs in the environment will have a reward, positive or negative, depending on the action performed. The goal is to learn how to map situations to actions in order to maximize a numeric reward sign. In figure 1.1 it can be seen for better visualization the agent's interaction with the environment.

Figure 1.1: Agent-Environment interaction

The implementation of Reinforcement Learning to autonomous vehicle parking processes makes it possible to generalize parking in various situations, abstracting from any external factors that may surprise more traditional implementations, creating a more complete and versatile system.

1.1 Motivation

Reinforcement Learning refers to goal-oriented algorithms that learn through interactions with the environment following a trial-and-error strategy. This offers a different approach from any other method because, in this approach, the agent alone, without human intervention, must discover the best strategy to complete the pretended task. Over the years, a few Reinforcement Learning achievements already proved that Reinforcement Learning algorithms can develop innovative strategies that lead to achieving superhuman performance. One of these achievements is AlphaGo Zero [2]. This algorithm was designed to play the strategy board game Go. This algorithm is one of Reinforcement Learning greatest achievements because in 2017 it became the first program to defeat a world champion in the game of Go, which is one of the most complex broad games.

Reinforcement Learning has been explored in many other areas outside strategy board games. One of them is autonomous driving, where it provides a more complete and adaptable system capable of generalizing different scenarios from the ones it learns. In 2018 Wayve demonstrates the first example of Deep Reinforcement Learning on a self-driving car, learning to lane follow from scratch with just 10 minutes of feedback [10] [11]. Another example of Reinforcement Learning being applied to autonomous driving, in this case, to solve the autonomous parking problem, was

Audi Q2 deep learning concept. This project was presented by Audi in 2016 where Reinforcement learning was used to teach a 1:8 scale model car of Audi Q2 to park. Within an area measuring 3 x 3 meters, it detects a suitable parking space in the form of a metal frame and then autonomously parks itself there [12].

1.2 Objectives

This dissertation proposes to solve autonomous parking tasks by implementing a self-learning system using Reinforcement Learning solutions.

The first step was to deeply study the current Reinforcement Learning algorithms to do a more knowledgeable and insight choice of the Reinforcement Learning algorithm implemented.

To learn how to autonomous park, the system has to perform the following goals:

- To perform all the necessary maneuvers to park the autonomously driving vehicle in three different types of parking spots, perpendicular parking spot, parallel parking spot and angular parking spot:

  After being aware of the parking spot position and understanding what type of parking it will have to perform, the vehicle will proceed to carry out the correct maneuvers for the desired type of parking.

- To avoid any obstacles that may appear:

  Since safety is a priority, while performing all the necessary maneuvers to park itself in the pretended place, the vehicle will have to avoid obstacles that may appear.

- To properly park the vehicle between the lines:

  The parking maneuvers will only be completed when the vehicle is properly parked inside the parking spot lines.

Since the participation in the Autonomous Driving test at the RoboCup Portuguese Open is one of the goals of this project, the Reinforcement Learning system will be trained and tested in the simulated track of this competition. In figure 1.2 examples of the simulated and real track of the competition are presented. This event is characterized by creating a reduced and more controlled version of a two-lane road, crosswalks, obstacles, traffic lights, traffic signals, tunnels,

parking spaces and construction sites with temporary signage. The robot has to be completely autonomous and respect all traffic rules in a more controlled version of the public road. In this event, there are three types of parking, already mention above, with and without obstacles, creating a wide variety of possible parking types and conditions external to the vehicle.

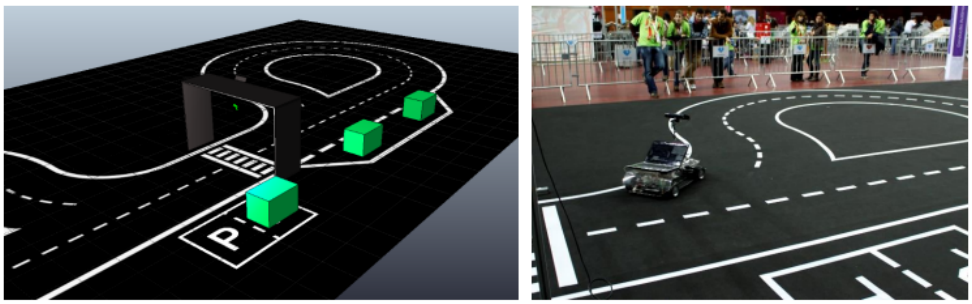


Figure 1.2: Autonomous Driving Test simulated Track (left) and real Track (right)

1.3 Dissertation Structure

This document is divided into five chapters. Chapter 1 refers to the introduction of this document, which describes the motivation and objectives of this dissertation.

In chapter 2, all the necessary theoretical concepts of Reinforcement Learning are explained, the RoboCup Portuguese Open competition on which this project is applied is introduced. Lastly, it is presented the state of the art of Reinforcement Learning and Reinforcement Learning in autonomous parking.

In chapter 3, all the implementations carried out in this dissertation are explained. The agent and environment, which is the simulated track of the RoboCup Portuguese Open competition, are presented. Lastly, a brief explanation of the theoretical concepts and implementation of Robot Operating System (ROS) developed in this project is presented.

In chapter 4, all the phases of the training and tests the agent was subject are presented. The chapter ends with an analysis of the results obtained in the test phase.

Lastly, in chapter 5, all conclusions drawn from this project are presented as well as further work.

Chapter 2

Literature Review

This chapter is divided into three sections, Theoretical concepts 2.1, Portuguese Robotics Open 2.2 and State of the Art 2.3. Section 2.1 presents a brief review of theoretical concepts of Machine Learning and Artificial Intelligence and a more comprehensive introduction of the theoretical concepts of both Reinforcement Learning and Deep Reinforcement Learning. In section 2.2, the RoboCup Portuguese Open competition, which is the competition that this project is inserted on, is introduced. This section is more focused on the parking challenges since that is the objective of this dissertation. Lastly, in section 2.3, the more noticeable accomplishments in Reinforcement Learning, as well as the most recent advances of Reinforcement Learning in the fields of Autonomous Driving and Autonomous Parking, are presented.

2.1 Theoretical Concepts

In this section, machine learning is introduced and the theoretical foundations about Reinforcement Learning needed for a better understanding of all concepts used in this dissertation are presented. In this dissertation, only the mandatory concepts to introduce the Deep Deterministic Policy Gradient (DDPG) algorithm, were mentioned. These concepts consist of the explore-exploit dilemma, Markov Decision Processes (MDP), Temporal Difference Learning vs Monte Carlo where a comparison between online learning and offline learning is presented, Policy Gradient Methods, Actor-Critic Methods and lastly, the DDPG algorithm.

2.1.1 Machine Learning

Artificial Intelligence (AI) is the area of computer science that emphasizes the creation of intelligent machines that function and react like human beings. It involves developing computer programs to complete tasks that would otherwise require human intelligence. AI algorithms can tackle learning, perception, problem-solving, language-understanding and/or logical reasoning [13].

Machine learning (ML) is a branch of AI and although it is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve, it is based on defining each step that the program must perform to obtain a result. ML algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, ML facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs [14].

In ML there exists three different categories, which differ on how data is received or how feedback on the learning is given to the system developed. The three categories are Supervised Learning, Unsupervised Learning and Reinforcement Learning and in figure 2.1 it is possible to visualize the AI hierarchy.

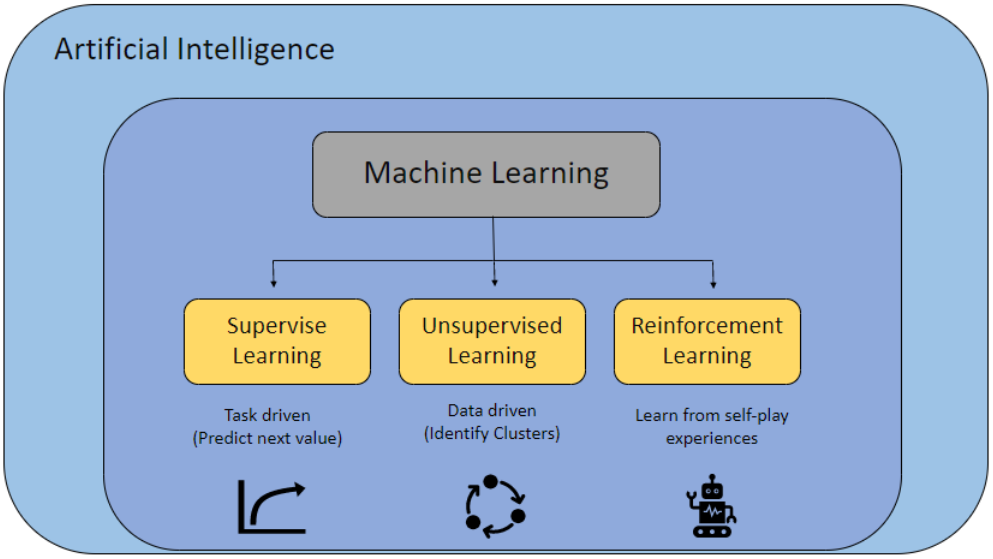


Figure 2.1: IA hierarchy

2.1.2 Reinforcement Learning

Reinforcement learning is used in robotics to teach a robot to master a specific task. This is accomplished in Reinforcement Learning by mapping situations to actions to maximize a numerical signal of reward. In Reinforcement Learning there exists two entities, the agent and the environment. In this kind of learning, the agent is placed in the environment and carries out an action, in response, the environment returns a reward based on the carried out action, this line of events can be visualized in figure 2.2. The objective of the agent is to maximize the reward that receives. This way the agent explores every state in the environment to map the best action to each state. From this, a strategy is deduced. Reinforcement Learning enables a robot to autonomously discover an optimal behaviour through trial-and-error interactions with its environment [15]. To better understand all of these concepts, in 2.1.2.1 are presented and explained some of the most important keywords in Reinforcement Learning.

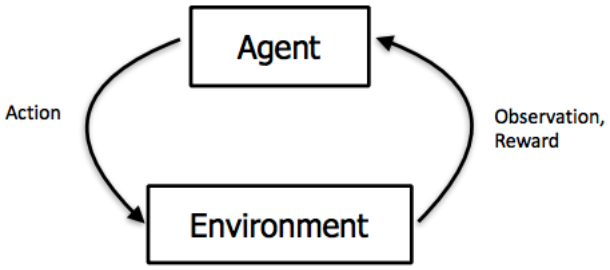


Figure 2.2: Reinforcement Learning

2.1.2.1 Reinforcement Learning Vocabulary

In this section, a dictionary is presented to explain essential elements of Reinforcement Learning, to a better understanding of more complex subjects referred ahead in this dissertation.

- Agent: Simulated or physical component that must perform the task.
- Environment: The world through which the Reinforcement Learning agent acts.
- Actions (a): The Agent's methods to interact and change the environment and thus transfer between states. Every action performed by the Agent yields a reward from the environment.

- State (s): The immediate situation in which the agent finds itself in the environment.
- Terminal State: Last state. When the agent reaches this state the episode terminates.
- Reward: Feedback from the environment that indicates to the agent how good or bad his action was.
- Return: Sum of all rewards in an episode.
- Transition Probabilities ($p(s^{'}, r|s, a)$): Represents the one-step dynamics of the Environment, that is, the probability of next state and rewards, given the current state and current action.
- Policy (π): Defines the learning agent's way of behaving at a given time. In other words, is the agent's strategy.
- Episode: an episode is a sequence of interactions from the initial state to some terminal state.
- Value function (V(s)): Qualitative value of how good a state is. Measures potential future rewards that may get from being in this state s.
- Action Value (Q(s)): Qualitative value of how good the taken action was. Measures potential future rewards that may get from taking the action (a).

2.1.2.2 The explore-exploit dilemma

As it was previously presented, in Reinforcement Learning, two entities exist, the agent and the environment. At time step t, the agent takes an action and in response, the environment returns the new state and a reward signal. The goal of the agent is to maximize the cumulative reward it receives from the environment, so the agent will always take the best possible action that will lead to this goal. The problem with this approach is that it could lead to some of the states in the environment never being visited and in an early stage of the training, it is recommended the agent visits all the states, because only then it can be sure that the final policy is the best possible. On the other hand, too much exploration is not necessarily a good approach, because it will take too long to master the task at hand. Simply, the exploration versus exploitation dilemma can be

described as the search for a balance between exploring the environment to find profitable actions while taking the best action as often as possible.

2.1.2.3 Markov Decision Processes

In this section, some of the key elements/concepts in Reinforcement Learning and Markov Decision Processes (MDP) such as Markov property, returns, values, Bellman equation and Optimal policy are introduced.

MDP is one of the most essential concepts in Reinforcement Learning because it is the framework from which every Reinforcement Learning algorithm is derived. MDP are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent states and future rewards.

1. **Markov Property**

   Markov property states that the next state (s_{t+1}) and the reward (R_t) depends only on the current state (s_t) and the action taken (a_t), even if conditioned other previous states, the transition probabilities and the reward would remain the same. Succinctly, the probability of the next state knowing the current state would be the same as the probability of the next state knowing all the previous states, as presented in equation 2.1.

   $$P(S_{t+1}|S_t) = P(S_{t+1}|S_t, S_{t-1}, S_{t-2}, ..., S_1) \tag{2.1}$$

   When a Reinforcement Learning problem satisfies this property, this is, the next state and the reward depend only on the current state and in the action taken, it is formulated as an MDP.

2. **Interaction between the agent and the Environment**

   In an MDP framework, an agent interacts with the environment on a discrete time scale. At each time step (t), the agent receives the state of the environment ($s_t \in S$), and depending on the state, performs one of the possible actions ($a_t \in A_{st}$). In response to the chosen action (a_t) the environment returns to the next state (s_{t+1}) and the reward (r_t) for the action taken, as presented in figure 2.3. This cycle (s, a, r) is repeated until the agent reaches a terminal state [9].

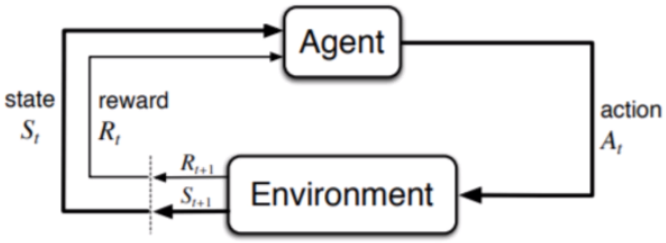


Figure 2.3: Agent-Environment Interaction

In an MDP, the probabilities characterise the environment dynamics entirely. Thus, as referred in section 2.1.2.1, the function p, also known as Transition Probabilities, represents the one-step dynamics of the Environment. So, from the function p it is possible to compute any information about the environment, such as the state-transition probabilities:

$$p(s^{'}|s,a) = p(S_{t+1} = s^{'}|S_t = s, A_t = a) = \sum_{r \in R} p(s^{'}, r|s, a) \tag{2.2}$$

3. **Return and Discounted Return**

At each time step, a reward signal $r_T \in R$ is sent from the environment to the agent. In order to maximize the efficiency of the agent's behaviour, the agent should not only focus on maximizing the immediate reward but also on maximizing the long-term reward. In order words, the agent should focus on maximizing the sum of all future rewards for a given episode [16]. This is called the return (G_t).

$$G_t = R_{t+1} + R_{t+2} + ... + R_T = \sum_{\tau=0}^{\infty} R(t + \tau + 1) \tag{2.3}$$

Since the return is the sum of all future rewards, it is possible to rewrite the equation for calculating the return as follows:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.4}$$

$$G_{t+1} = R_{t+2} + R_{t+3} + ... + R_T \tag{2.5}$$

$$G_t = R_{t+1} + G_{t+1} \tag{2.6}$$

R_T represents the reward that the agent receives to reach the goal (the terminal state).

Although this method works in small episodic tasks (tasks that have a terminal state), it will fail when applied to continuous tasks (tasks that do not have a terminal state), because in this case there is no terminal state and the return would be infinite ($G_t = \infty$).

To solve this problem it is introduced the discount factor (γ) in the return. This way, the more in the future the reward is, the more insignificant it is for the current time step.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{\tau=0}^{\infty} \gamma^\tau R(t + \tau + 1) \tag{2.7}$$

$$G_t = R_{t+1} + \gamma G_{t+1} = \sum_{\tau=0}^{\infty} \gamma^\tau R(t + \tau + 1) \tag{2.8}$$

4. **Action-Value and Value-function**

In Reinforcement Learning it is not always convenient to wait until the end of the episode to get the return. So, instead of calculating the return, most of the algorithms in Reinforcement Learning calculate the value-function or the action-value function. As it was mentioned in 2.1.2.1, the value-function, also known as the expected return, essentially is the function that estimates the best state for the agent to proceed considering the current state. With this approach, the primary goal of the agent is to maximise the expected return. So, as the episode progresses, the agent maps the best actions for each state, creating the policy. Mathematically, the value-function can be expressed as the expected return given a state (s) under the policy distribution π:

$$V_\pi(s) = E_\pi[G_t | S_t = s] \tag{2.9}$$

In order hand, the action-value, also known as the expected return, is the function that estimates the best action to take considering the current state. In this case, the agent maps the best action for each state (policy). Similarly, the action-value function can be

expressed mathematically as the expected return given a state (s) and an action (a) under the policy distribution π.

$$Q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] \tag{2.10}$$

5. **Bellman Equation**

In Markov decision processes, a Bellman equation is a recursion for expected rewards. This equation describes the expected reward for taking the action (a) defined by the policy π. This way to calculate the value-function it only needs to look one step ahead. This means that the agent can plan without actually having to look very far in the future.

The Bellman equation for the value-function has a recursive property. For any policy and state, it is possible to iteratively calculate the value-function for the current state from the value-function of the next state. It is presented in equation 2.11 the Bellman equation for the value-function:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s^{'}, r|s, a)[r + \gamma V_\pi(s^{'})] \tag{2.11}$$

In the same way as the value-function, this equation recursively calculates the action-value following a policy π

$$Q_\pi(s,a) = \sum_{s',r} p(s^{'}, r|s, a)[r + \gamma V_\pi(s^{'})] \tag{2.12}$$

This equation is extremely important because it simplifies the computation of the value-function or action-value, such that rather than summing over multiple time steps, we can find the optimal solution of a complex problem by breaking it down into simpler, recursive sub-problems and finding their optimal solutions [17].

6. **Optimal Policy**

As mentioned before, the goal of the Agent is to maximize the total cumulative reward in the long run. The policy that contains the best possible strategy for the given environment,

this is, the policy that will lead to the highest return possible is called the optimal policy (π_*).

By definition, the optimal policy (π_*) has to be better or equal to any other policy (π). For this to happen the optimal value-function (V_*) has to be better or equal than any other value-function (V_π) for all possible states. An optimal policy is guaranteed to exist but may not be unique. That means that there could be different optimal policies, but they all have the same value-functions, the optimal value-functions (V_*). The optimal value-function can be defined by the following equations:

$$V_*(s) = max_\pi V_\pi(s) \tag{2.13}$$

In the formula above, the optimal value-function (V_*) allow the agent to recognize what is the best possible state to follow.

Similarly, the optimal action-value (Q_*) inform the agent what is the best action to take in the current state. The optimal action-value (Q_*) can be defined by the following equations:

$$Q_*(s,a) = max_\pi Q_\pi(s,a) \tag{2.14}$$

Knowing this it is possible to rewrite the optimal value-function (V_*) via the optimal action-value (Q_*) so the value of some state equals the value of the maximum action the agent can execute for the current state:

$$V_*(s) = max_a Q_*(s,a) \tag{2.15}$$

$$V_*(s) = max_a \sum_{s',r} p(s', r|s,a)[r + \gamma V_*(s')] \tag{2.16}$$

Similarly, it is also possible to express the optimal action-value using the Bellman equation:

$$Q_*(s,a) = \sum_{s',r} p(s', r|s,a)[r + \gamma max_{a'} Q_*(s', a')] \tag{2.17}$$

2.1.2.4 Temporal Difference Learning vs Monte Carlo

In Reinforcement Learning there are two types of learning, Online Learning and Offline Learning:

- Online Learning: Type of learning where the value is updated for each step of the episode. Shortly, the update occurs on a step-by-step basis.
- Offline Learning: Type of learning where the value is updated in the end of the episode. Shortly, the update occurs on an episode-by-episode basis.

Monte Carlo (MC) and all Temporal Difference Learning (TD) algorithms learn from raw experience. The main difference between these two algorithms is that MC uses Offline Learning and TD uses Online Learning.

MC is an Offline Reinforcement Learning algorithm, therefore it must wait until the end of the episode to receive the Return (G). Then it uses the Return as a target to estimate the value-function, as presented in the equation 2.18.

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{2.18}$$

On the other hand, all TD algorithms, such as SARSA and Q-Learning, are Online Reinforcement Learning algorithms. Therefore, the value is updated on a step-by-step basis, to accomplish this, is used the process of Bootstrapping. Bootstrapping is the name of the process that uses the difference between temporally successive predictions to estimate a prediction/value, thus the name Temporal Difference [18]. In other words, TD updates estimates based on the other learned estimates, this way it is not necessary to wait to the end of the episode. This can be expressed mathematically using the equation 2.20

$$V(S_t) \leftarrow V(S_t) + \alpha(Target - prediction) \tag{2.19}$$

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.20}$$

This equation can also be written in order to estimate the action-value (equation 2.22) and it is important because the most acknowledged algorithms in Reinforcement and Deep Reinforcement Learning, such as Q-Learning, Deep Q-Learning, Deep Deterministic Policy Gradient, among others, use the action-value.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(Target - prediction) \tag{2.21}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{2.22}$$

It is important to note that part the equation calculate an error, measuring the difference between the estimated value of $Q(S_t, A_t)$ and the target estimate ($R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$). This quantity is called the TD error and can be expressed by:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \tag{2.23}$$

All TD learning algorithms are an efficient solution when dealing with small problems, but in a scenario where the state space and the action space are too large, it would become too much computational expensive to store all the necessary data. To solve this problem it is used a Neural Network as an approximation for the Q function. In this case, the Neural Network receives the Environment state as input, and it returns as output the value of each possible action, as shown in figure 2.4. An example of this approach is the Deep Q-Learning algorithm [19].

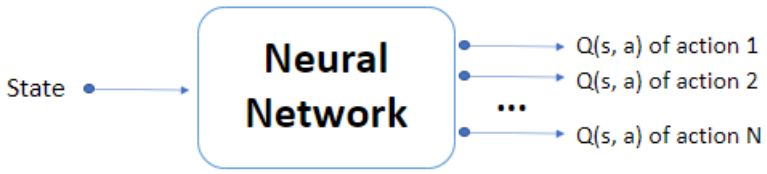


Figure 2.4: Deep Reinforcement Learning Value-Based architecture

2.1.2.5 Policy Gradient Methods

In this section, the theoretical concepts of Policy Gradient methods are introduced. Reinforcement Learning can be divided into two different methods: Value-Based methods and Policy Gradient methods.

So far it was only introduced Value-Based methods. These are the methods that estimate the value (V(s) or Q(s,a)) to deduce which is the best action to each state, in these methods the policy is created through the value, Q-Learning and Deep Q-Learning are examples of this method. On the other hand, Policy Gradient (PG) methods do not use the value to learn the best actions, instead, these methods learn a parameterised policy ($\pi(a|s,\theta)$) that selects an action without checking the value. The goal of this method is to find the best parameters/weights (θ) of the Neural Network that generates the best policy [1].

In PG methods, the Neural Network takes as input the state of the environment and returns the policy. In this method, the policy is a probability distribution of all the possible actions with respect for the parameters θ, as it is described by equation 2.24. This means that if the agent ends up in the same state twice, it may not end up taking the same action every time.

$$\pi(a|s,\theta) = p(A_t = a|S_t = s, \theta_t = \theta) \tag{2.24}$$

The key idea underlying PG methods is to increase the probabilities of actions that lead to a higher return and decrease the probabilities of actions that lead to a lower return. For this purpose, PG methods, learn the policy parameter θ based on a gradient of a scalar performance measure $J(\theta)$ concerning the policy parameter. Since the objective of PD methods is to maximise the performance, the parameters θ are updated using gradient ascent in J, as shown in the equation 2.25

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{2.25}$$

Where the α is the learning rate and the $\nabla J(\theta_t)$ denotes a stochastic estimate, whose expectation approximates the performance measure gradient concerning θ_t. Using the Policy

Gradient Theorem, the gradient of the performance can be described as:

$$\nabla J(\theta_t) = \sum_{t=1}^{T} G_t * log\pi(a_t|S_t, \theta) \tag{2.26}$$

With this, the return indicates how to change the weights of the policy θ. For example, in a scenario in which the agent completes the episode with success, at the end of the episode it adds up all the gradient directions that should be steeped in to increase the log probability of selecting each state-action pair. The bigger the return, the bigger the step. That is equivalent to just taking T simultaneous steps where each step corresponds to a state-action pair in the episode. On the opposite side, if the agent fails the episode, the gradient will evolve in the direction that will decrease the log probability of selecting each state-action pair [20].

In summary, PG methods behaviour can be described into five steps:

1. Play the episode and collect the State-Action-Reward sequence.
2. Calculate the return, G.
3. Calculate the performance's gradient, $\nabla J(\theta_t)$.
4. Update the weights θ of the policy.
5. Repeat until it converges.

The methods that follow this process logic are named policy gradient methods, independently of also learning an approximate value function. A method, known as the Actor-Critic method, learns approximations to both policy and value function. The actor learns the policy, and the critic learns the value function. This method will be covered in the next section.

2.1.2.6 Actor-Critic Methods

An Actor-Critic method is a PG learning algorithm that learns both a policy and the value function. This means that the Actor-Critic methods are a combination of both Value-Based methods and Policy-Gradient methods, as shown in figure 2.5.

The standard architecture of the actor-critic methods is constituted by two Neural Networks:

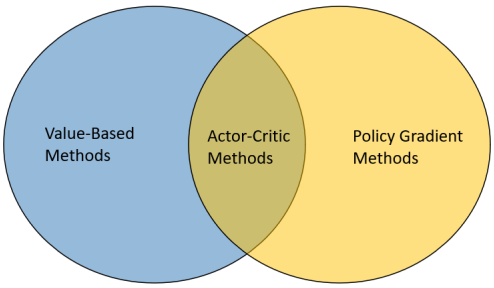


Figure 2.5: Deep Reinforcement Learning Value-Based architecture

- Actor/Policy Neural Network: This Neural Network receives as input the environment state and returns as output a probability distribution of all possible actions. The goal of the Actor-Critic methods is to find the Actor-Network weights that will lead to the optimal policy. Shortly, this is called the Actor-Network because defines the action, it acts. This represents the PG component of the Actor-Critic methods.

- Critic/Value Neural Network: This Neural Network receives as input the environment state and returns as output the state-value function, as shown in figure 2.6. The function of this Neural Network is to "judge" the actions performed by the Actor-Network. Shortly, this is called the Critic Network because it defines how good was the action taken by the Actor through the TD error. This represents the Value-Based component of the Actor-Critic methods.

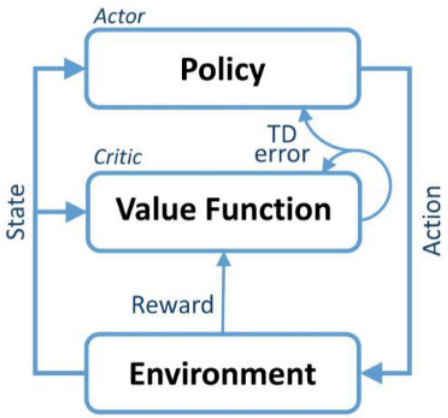


Figure 2.6: Diagram of the Actor-Critic method architecture, where the actor represents the policy and the critic the value function. Figure from [1], displayed with permission from Tiago Ribeiro

Actor-Critic methods can be seen as a Temporal-Difference version of the PG methods. This is, the PG methods are offline, which means that it relies on the Monte-Carlo estimate, in other words, it has to wait until it receives the return to update the parameters. The problem with this approach is that typically it suffers from high variance and large sampling cost [21]. On the other hand, Actor-Critic methods use bootstrapping, by updating the value estimate for a state from the estimated values of subsequent states. Then the TD-error is calculated and it replaces the return in the parameter update equation of the Actor parameters, as is shown in equations 2.27, 2.28.

$$\theta_{t+1} = \theta_t + \alpha * [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] * log\pi(a_t|S_t, \theta) \tag{2.27}$$

$$\theta_{t+1} = \theta_t + \alpha * \delta_t * log\pi(a_t|S_t, \theta) \tag{2.28}$$

In summary, Actor-Critic methods behaviour can be described into five steps [22]:

1. Take an action chosen by the Actor and receive the next state and reward.
2. Estimate the value-function and calculate the TD-error.

   $\delta_t \leftarrow R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
3. Update the Actor parameters, θ.

   $\theta_{t+1} = \theta_t + \alpha * \delta_t * log\pi(a_t|S_t, \theta)$
4. Update the Critic weights, w.

   $w = w + \alpha * \delta_t \nabla V(s)$
5. Repeat until finding the optimal policy.

2.1.2.7 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [23] is an actor-critic, model-free algorithm based on the deterministic policy gradient (DPG) that can operate over continuous action spaces. This algorithm was presented by Google's Deepmind. It is a combination of the DPG principles with the ideas underlying the Deep Q-Learning success.

Deep Q-Learning was able to learn value functions using non-linear function approximators stably and robustly, prior to what was initially believed, due to two innovations: Experience Replay Buffer and Target Neural Networks [23]:

- Experience Replay Buffer: One challenge when using neural networks for reinforcement learning is that most optimization algorithms assume that the samples are independently and identically distributed. Since the samples are generated from exploring sequentially in an environment this assumption is not valid. Additionally, it is more efficient and stable to learn in minibatches, rather than online. Such as in Deep Q-Learning, DDPG uses the experience replay buffer to address this issue. The experience replay buffer is a finite sized cache where it is stored the tuple (s_t, a_t, r_t, s_{t+1}) from the interaction of the agent with the environment. When the replay buffer is full the oldest samples are discarded and replaced by the more recent ones. At each time step, the actor and critic are updated by randomly sampling a minibatch from the replay buffer, this behaviour is presented in figure 2.7. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions [23].

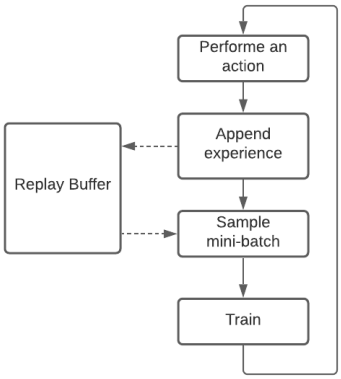


Figure 2.7: Replay buffer line of events

- Target Neural Networks: As it was presented in section 2.1.2.6, the Actor-Critic methods use bootstrapping to estimate the value. The problem with this approach is that the target is not the real target but an estimation of the real target. This could easily lead to

instability and lack of convergence. So, to solve this problem in DDPG, two extra Neural Networks (Target Neural Networks) are created to estimate the target ($TD_{target} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$). The target Neural Networks are initially a copy of the Actor-Critic Neural Networks and are updated, in DDPG, using the soft-update. So, with soft-update, the target networks weights are updated by slowly tracking the weights of the Actor-Critic networks. As shown in equation 2.29.

$$\theta^{'} = \tau\theta + (1 - \tau)\theta^{'} \tag{2.29}$$

Where $\theta^{'}$ are the weights of the target networks, θ are the weights of the Actor-Critic networks and τ defines how much influence the Actor-Critic networks has over the target networks weights. Normally $\tau << 1$ because this will improve the targets network stability what is necessary to consistently train the critic network without divergence. Since the target networks delay the value estimation propagation, it causes the learning process to be slower. However, this problem is outweighed by learning stability.

Combining the Actor-Critic principles with the target networks idea from Deep Q-Learning results in a four Neural Networks architecture, two pairs of the Actor-Critic architecture presented in figure 2.8. Where the Target Actor-Critic estimate the target and the Actor-Critic estimates the action-value.

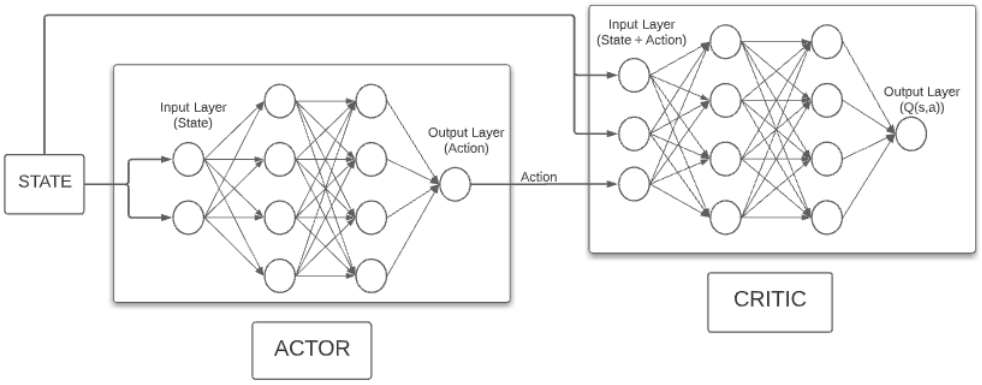


Figure 2.8: Deep Deterministic Policy Gradient Architecture

Since the Critic represents the Value-Based component of this Actor-Critic algorithm it is optimized by minimizing the loss:

$$L(\theta^Q) = E_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t|\theta^Q) - y_t)^2] \tag{2.30}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(S_{t+1}, \mu(s_{t+1})|\theta^Q) \tag{2.31}$$

Where $Q(S_{t+1}, \mu(s_{t+1})|\theta^Q)$ is the action-value estimated by the target Critic for the state (s_{t+1}) and action ($\mu(s_{t+1})$), the $\mu(s_{t+1})$ is the deterministic policy (action) define by the target Actor for the state (s_{t+1}) and $Q(s_t, a_t|\theta^Q)$ is the action-value estimated by the Critic for the state (s) and action (a) performed by the Actor. Note that the s_t, a, $r(s_t, a_t)$ and s_{t+1} are the transitions sampled form that replay buffer. It is also important to note that, in summery, the Critic Loss is the square of the TD-error.

To update the Actor is used the same method implemented in the DPG. The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The actor is updated by following the chain rule to the action-value from the start distribution J with respect to the actor parameters:

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}] \tag{2.32}$$

$$\nabla_{\theta^\mu} J = E_{s_t \sim \rho^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s = s_t] \tag{2.33}$$

A major challenge of learning in continuous action spaces is exploration. To address this issue, in DDPG, is constructed an exploration policy $\mu^{'}$ by adding noise sampled from a noise process N to the Actor policy.

$$\mu^{'} = \mu(s_t|\theta_t^\mu) + N \tag{2.34}$$

The process used to determine N can be chosen to suit the environment. However, in DDPG, was used an Ornstein-Uhlenbeck process [25].

To better understand the sequence of events of this algorithm is presented a pseudo-code of the DDPG algorithm.

Algorithm 1: Deep Deterministic Policy Gradient Pseudo-code

Randomly initialize The Critic $Q(s, a|\theta^Q)$ and the Actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ;
Initialize Target Networks with the weights of the Actor and Critic. $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$;
Initialize replay buffer R;
while *Episode* = *1, M* **do**
  Initialize a random process N for action exploration;
  Receive initial observation state s_0
  while $t = 0, T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the Actor and the exploration noise;
    Execute action a_t and observe reward r_t and new state s_{t+1};
    Store Transition (s_t, a_t, r_t, s_{t+1}) in R;
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R;
    Set $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}})|\theta^{Q'})$;
    Update the Critic by minimizing the loss:

$$L(\theta^Q) - \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

    ;
    Update the Actor using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|s = s_i]$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end
end

2.2 RoboCup Portuguese Open

The *Festival National de Robotica* (FNR) is currently the largest scientific meeting with robotics competitions taking place in Portugal, having been held since the first edition that took place in Guimarães in 2001. The FNR is an initiative of the Portuguese Robotics Society, also known as *Sociedade Portuguesa de Robotica* (SPR). Throughout its various editions, the FNR has been held throughout the country, providing the dissemination of Robotics and related areas in a geographically balanced manner. The main goals of the FNR are [26]:

- Make a positive contribution to the development of research in Robotics and Automation;
- Motivate students from primary, secondary schools and universities to a technologically advanced and highly multidisciplinary area;

- Contribute to the dissemination of Science and Technology developed in Portugal;
- Increase interest on robotics, technology and science in the general public;
- Promote meetings and synergies between national and international working groups in the field of robotics;
- Qualify the Portuguese teams for the RoboCup world competition.

2.2.1 Autonomous Driving Competition

The Autonomous Driving Competition is a competition for fully autonomous robots that takes place in a track with the shape of a traffic road, as presented in figure 2.9. The robot that enters this competition has to be a completely autonomous vehicle in which all decisions must be taken by the systems included in it. Thus, it is not allowed to include any kind of communication between the robot and other electronic devices external to it [27].

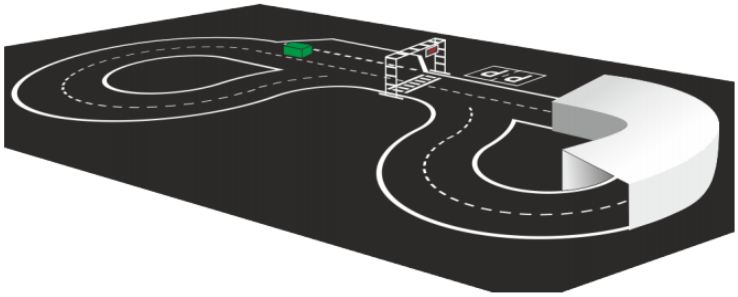


Figure 2.9: Track overview

Two TFT panels are mounted right above the zebra crossing, in the inverted position, and vertically aligned with each other of the driving bands of the track. These panels are used to show indicating signals to the competing vehicles. When these panels display the parking signal it indicates the vehicle to start the parking maneuvers [27], as shown in figure 2.10.

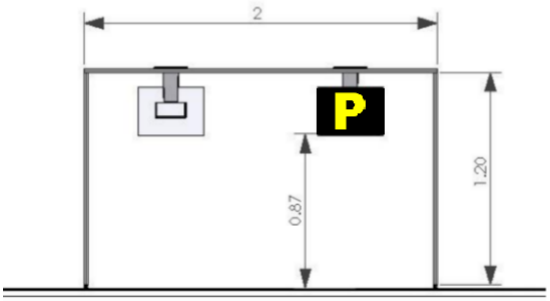


Figure 2.10: Panels overview

2.2.2 Parking areas

On the track there exist two parking areas, one for each type of parking (perpendicular or parallel). One is located after the zebra crossing, on the right side of the lane. It consists of a band, parallel to the right driving lane and with the same width, as presented in figure 2.11. This is the parallel parking area. The other parking area, is a perpendicular parking area, with two places, located before the zebra crossing. It consists of two rectangles with the letter "P" inscribed [27], as presented in figure 2.11.

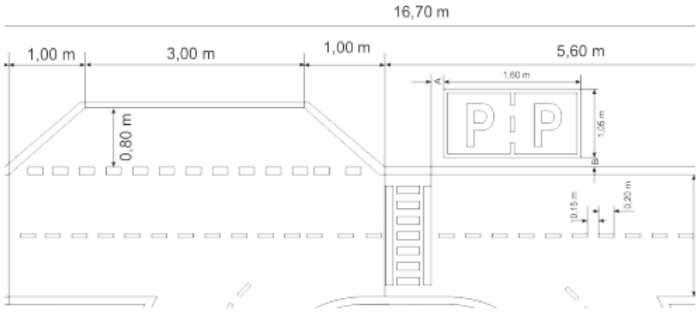


Figure 2.11: Parking areas schematic

2.2.3 Parking Challenges

The parking challenges start when displayed the parking signal in the panel. There are four different types of challenges in this competition, two for each parking area. In the perpendicular parking area two challenges are carried out, perpendicular parking without obstacles and perpendicular parking with obstacles. In the latter case, the parking place with the obstacles is not known in advance [27] and these two challenges are presented in figures 2.12 and 2.13.

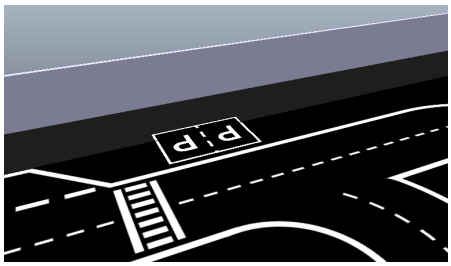


Figure 2.12: Perpendicular parking without obstacles

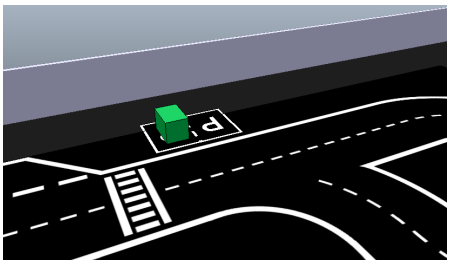


Figure 2.13: Perpendicular parking with obstacles

In the parallel area are also carried out two challenges, parallel parking without obstacles and parallel parking with obstacles, as presented in figures 2.14 and 2.15. Similarly to the perpendicular parking challenge, the location of the obstacles is not known in advance. In these challenges, the robots must stop parallel to the driving lane [27].

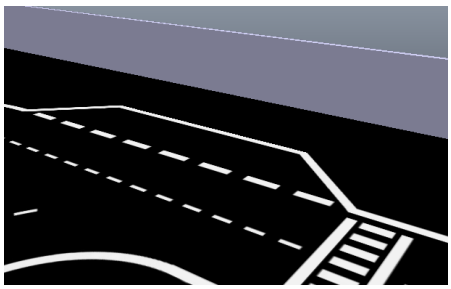


Figure 2.14: Parallel parking without obstacles

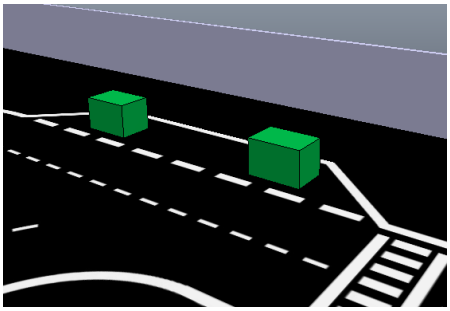


Figure 2.15: Parallel parking with obstacles

2.3 State of the Art

2.3.1 Reinforcement Learning

Over the years Reinforcement Learning has gained more and more interest. The idea of potting an agent in an unknown environment to master a defined task and with nothing more than a reward signal, he alone learns how to do the task has awakened the interest of many people. With this growing interest, the investment in this area has increased and the results have been appearing in recent years.

One of the most recent significant achievements of reinforcement learning is the AlphaGo Zero [2]. The AlphaGo Zero is a Reinforcement Learning algorithm created to learn to play the game of go. Go is an abstract strategy board game for two players in which the aim is to surround more territory than the opponent.

AlphaGo Zero is the most recent version of the Alpha Go family, which has learned entirely from scratch without any human data. This Reinforcement Learning algorithm is one of the most significant achievements in this era because in 2017 it became the first program to defeat a world champion in the game of Go. The algorithm behind the AlphaGo Zero is a Monte Carlo Tree Search with a deep convolutional neural network.

AlphaGo Zero was his own teacher, this is, at the beginning of his training, AlphaGo Zero didn't know anything about the game of go and it learns the game only by self-playing the game without any human interventions. This way, AlphaGo Zero came out with new strategies never seen before.

The training stage of this algorithm lasted 40 days. Throughout the training, 29 million games of self-play were generated. In the figure 2.16 the learning progress of the AlphaGo Zero is presented. As it is presented in the figure 2.16 this algorithm with only three days of training reaches the abilities of the version AlphaGo Lee. This was the version that defeated world champion Lee Sedol. With 21 days of training reaches the level of AlphaGo Master. This was the version that defeated the strongest human professional players 60–0 in online games in January 2017. By the end of the training, AlphaGo Zero surpasses all other versions of Alpha Go.

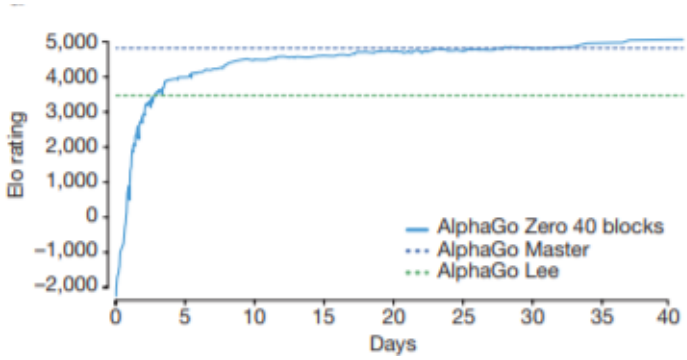


Figure 2.16: Training progress of AlphaGo Zero (figure from [2] displayed with permission from David Silver)

Another great achievement of a Reinforcement Learning algorithm against a human or in this case a group of humans was the OpenAI Five [28]. This algorithm became a great success when, on April 13th 2019, became the first AI system to defeat the world champions at an Esports game. The game in question is DOTA 2. Dota 2 is a multiplayer real-time strategy game produced by Valve Corporation in 2013 it is a 5v5 game with many heroes to choose from. Each team's base contains a structure called an ancient and wins the game the time that destroys the ancient of the enemy time first.

OpenAI Five leveraged existing Reinforcement Learning techniques, scaled to learn from batches of approximately 2 million frames every 2 seconds. It was developed a distributed training system and tools for continual training which allowed to train the OpenAI Five for 10 months. This algorithm, like AlphaGo Zero, started without knowing anything from the game and learned only from millions of self-played games and just like AlphaGo Zero, developed new strategies never seen before.

To create this algorithm, it was define a policy (π) as a function from the history of observations to a probability distribution over actions, which was parameterized as a recurrent neural network with approximately 159 million parameters. The policy was trained using Proximal Policy Optimization (PPO) algorithm. The neural network consists primarily of a single layer 4096-unit. This algorithm only takes between 164ms and 264ms to decide what his next action will be.

The OpenAI Five, such as AlphaGo Zero, are the proof that Reinforcement Learning algorithms have enormous potential.

2.3.2 Autonomous Driving

Due to the technological advances that have taken place in recent years, especially in the area of artificial intelligence, there has been a rapid increase in the progress of self-driving vehicle technology. Autonomous vehicles promise to improve traffic safety and reduce congestion. They represent the main trend in future intelligent transportation systems. So, because of that and many other reasons, autonomous driving is a topic that has been researched more and more in recent years. There are six levels of driving automation, ranging from 0 (fully manual) to 5 (fully autonomous) defined by The Society of Automotive Engineers (SAE) [29] [30] [31].

- Level 0 (No Driving Automation): Most vehicles on the road today are Level 0. This means that there is no autonomous system implemented in the car, it is the human that performs all driven tasks, but can have help from some implemented systems. An example would be the emergency braking system since it technically does not drive the vehicle, it does not qualify as automation;

- Level 1 (Driver Assistance): Level 1 represents the lowest level of automation. The vehicles classified with this level have a single automated system for driver assistance, such as steering or accelerating. An example would be the adaptive cruise control, where the vehicle can match the speed of the car ahead keeping, this way, the same distance. Once the human driver monitors the other aspects of driving such as steering and braking this system only qualifies as a level 1;

- Level 2 (Partial Driving Automation): Level 2 represents the advanced driver assistance systems (ADAS) and the vehicles in this level can control both steering and acceleration but it is still necessary for the driver full attention over all driving tasks because he may have to take control of the car at any time. A few examples of advanced driver assistance systems are the emergency braking system, the collision avoidance system, lane departure warning/correction and traffic sign recognition. However, these are not fully autonomous vehicles, as a driver continues to remain in control and must always pay attention to traffic;

- Level 3 (Conditional Driving Automation):

It is from this level that the complexity of systems takes a huge leap, from the technological perspective, because the vehicles with this ADS level or higher have environmental detection capabilities and can make informed decisions for themselves. Any car that qualifies as ADS level three can drive itself, but only under certain conditions. At this level, the car can drive long distances by itself, but monitoring of drivers is still necessary because these are not fully autonomous vehicles. In addition, ADS level three operate only in limited operational design domains such as highways;

- Level 4 (High Driving Automation):

  Any ADS level 4 car, in most circumstances, can operate in self-driving mode without any human interaction. This is under complex circumstances, the car is capable of handling the majority of driving situations without any input from the driver. However, a human still has the option to manually override. Despite ADS level 4 vehicles are able to drive themselves in most scenarios because legislation and infrastructure have yet to catch up with the technology, it is only allowed to do so within a limited area and limited speed;

- Level 5 (Full Driving Automation)

  ADS level 5 vehicles are fully autonomous, which means that they are capable of handling all driving responsibilities in all road conditions. At this level, there is no need for a driver. ADS level 5 vehicles won't even have steering wheels or acceleration and braking pedals.

2.3.3 Autonomous Parking

In the process of creating fully autonomous vehicles, autonomous parking is one of the most important technologies, since any journey ends with a parking maneuver. Therefore, many companies like Tesla, Audi, Mercedes, General Motors and many others have already developed an autonomous parking system or at the very least an assistant parking system. Current parking products can be divided into two categories [32]:

- The assistant parking system is a manual control parking system which provides drivers with broader perspectives and prompts relevant operations by sensors and video cameras, but the vehicle is controlled by the driver;

- The automatic parking system is an intelligently control steering and acceleration to complete parking operations through the environmental information collected by sensors and cameras.

The problem of parking can be described as a see-and-act task. This means that before an automated parking maneuver can begin, the parking system must search, identify and accurately localise an available parking spot. Knowing this, the autonomous parking problem can be divided into two sub-problems [33]:

- Visual Perception
- Trajectory Planner

2.3.3.1 Visual Perception

As previously mentioned, for a car to be able to autonomously park itself, it has to be aware of its surroundings. In the autonomous parking problem, two different types of detection systems must be implemented, parking spot detection and Static and dynamic obstacles detection.

Parking spot detection is critical for any automated parking system because before the vehicle can park itself, it has in the first place to know where to park and how to park, more specifically, the parking spot position and the type of spot. There are three types of parking spots, perpendicular parking spot, parallel parking spot and angular parking spot, each one is shown in figure 2.17 [3].

Figure 2.17: Types of parking spots. Figure adapted from [3]

With the continuous growth of demand for parking assistance systems, over the years researchers have proposed a variety of parking space detection methods with different technologies. The following are the most used approach's [33]:

- Ultrasonic-sensing based approach

  In [34] and [35] were proposed an implementation of a parallel parking assist system and semi-automated parallel parking system, respectively, based on ultrasonic sensors. The main idea of these two approaches is when the vehicle is driving parallel to the parking space, the system continuously detects the distance to the surrounding vehicles through the ultrasonic sensor to check if it has the necessary space to park 2.18.

Figure 2.18: Schematic view of a parallel parking maneuver, where the ultrasonic sensors capture the parking space dimensions and relative position to the car. Figure from [4]. Author unreachable

- Visual-sensing based approach

  An alternative to ultrasonic sensors is the Light Detection and Ranging system (Lidar) and the Around View Monitor system (AVM), which consists of four fish-eye cameras covering a 360° close-range area around the vehicle. In [36] and [5] were implemented a vision-based automatic parking slot detection method to detect various parking spots, using an Around View Monitor system (AVM). In [5] a convolution neural network was also used to detect the type, position, length, and direction of the parking spot from the AVM images, as is shown in figure 2.19.

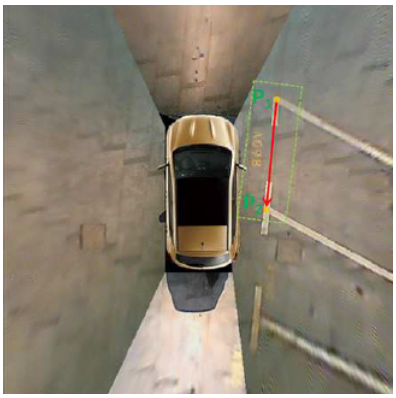


Figure 2.19: Parking spot detection using a AVM system. Figure from [5]. Author unreachable

2.3.3.2 Trajectory Planner

Traditional controllers make use of a model composed of fixed parameters. When robots or other autonomous systems are used to solve complex tasks, such as autonomous driving, traditional controllers cannot foresee every possible situation. In comparison, learning controllers make use of training information to learn their models over time in a trial and error strategy, creating a more versatile system [37]. Although most car companies do not disclose what system is used to plan the trajectory of the car, in a presentation from Tesla Motors it is stated that it is used Imitation Learning as a trajectory planner [38]. In 2016 in the Conference and Workshop on Neural Information Processing Systems (NIPS), Audi showcased a scale model of the Audi Q2 model, see figure 2.20, that was able to autonomous park itself using Reinforcement Learning as a Trajectory Planner [12].

Figure 2.20: Scaled model of the Audi Q2. Figure adapt from [6]

In recent years, in an academic research environment, a few papers were published where Deep Reinforcement Learning was used as a trajectory planner for an autonomous parking system. Every analysed paper can be divided into three groups. In [39] and [7] were proposed a Deep Reinforcement Learning method as a motion planner for parallel autonomous parking, in [8], [40], [41] and [42] were proposed a Deep Reinforcement Learning method as a motion planner for perpendicular autonomous parking. In [33] was proposed a Deep Reinforcement Learning method as a motion planner for both perpendicular and parallel parking.

Concerning parallel autonomous parking, two papers were published, in 2020, regarding the implementation of a model-based Deep Reinforcement Learning algorithm as a motion planner for parallel autonomous parking [39] and [7]. Both papers use a truncated Monte Carlo tree search to evaluate parking states and action selection. Monte Carlo Tree Search (MCTS) is a general-purpose planning Reinforcement Learning algorithm that has found great success in several unrelated applications, the greatest one being the Alpha Go. Both implementations were capable of successfully detecting the parking spot and learning to park the car both in simulation and real life, as it is shown in figure 2.21.

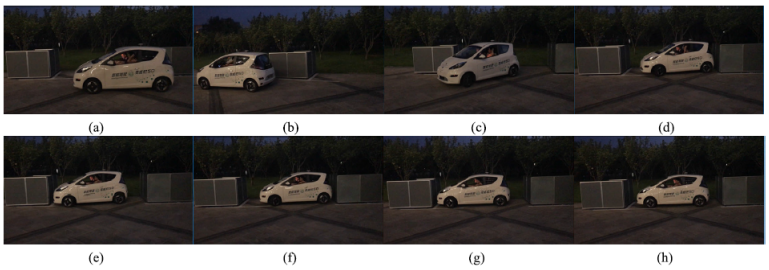


Figure 2.21: Sequential figures of the parallel parking maneuvers. Figure from [7]. Author unreachable

Concerning the perpendicular parking maneuvers, in [8] a Deep Reinforcement Learning algorithm was implemented to solve this problem. The algorithm implemented was the PPO (Proximity Policy Optimization) with LSTM (Long Short-Term Memory) network, and it was able to find a generalized solution to a variety of different initial positions, as is shown in the figure 2.22.

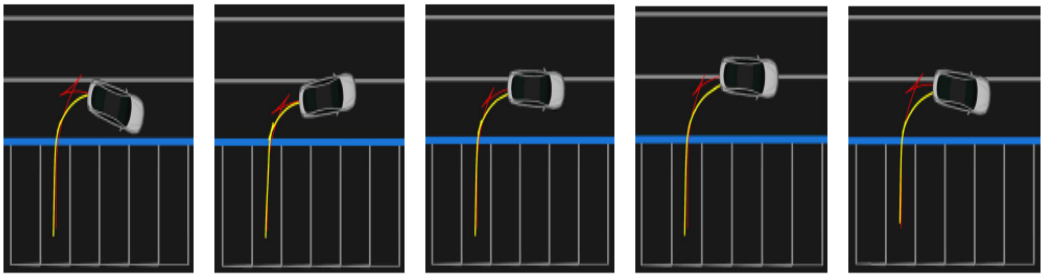


Figure 2.22: Trails with different initial positions. Figure from [8], the author unreachable)

In the referred paper, a comparison of the benchmark results for robustness to the initial state variation between their approach and the baseline approaches was also presented, more specifically, the Geometric Planning approach and the DQFD (Deep Q-Learning for Demonstration) approach. Deep Q-learning from Demonstrations is a Deep Reinforcement Learning algorithm that starts the training by leveraging small sets of demonstration data to accelerate the learning process and only after it begins to learn by interacting with the environment [43].

The authors of this paper concluded that their approach is similar and even better in some scenarios than the baseline approaches when regarding the adaptability of the system at initial position variations. It was also concluded that the trajectory learned by the proposed algorithm was more efficient than the compared approaches.

In [40] a control technique for reverse perpendicular parking car-like vehicles based on Deep Reinforcement Learning was presented. The algorithm implemented was the DDPG. The effectiveness of the proposed technique was tested on a car-like vehicle in a simulation environment implemented in MATLAB. The system was able to park the car-like vehicle in a variety of different initial and final positions proving this way the adaptability of the system.

In similarity with the previous referred paper, in [41] and [42] a DDPG algorithm was also implemented as a motion planner for perpendicular parking. In [41] to prove the efficiency of the algorithm, a comparison between the proposed approach, a PID controller (Proportional–Integral–Derivative controller) approach and SMC (Statistical Model Checking) approach was performed. When analysing the behaviour of the three mentioned approaches for the variation of the initial orientation of the vehicle, the authors concluded that although all approaches were able to park the vehicle, "The parking performance of reinforcement learning is obviously superior over those of the other two methods". In [42], the algorithm was also able to adapt the movements to

a variety of initial positions and orientations that have not been experienced before and a study of the behaviour of the vehicle in the different stages of the maneuver was also presented.

Regarding the papers that address the two maneuvers (parallel and perpendicular), in [33], two different Deep Reinforcement Learning algorithms were implemented as a motion planner for perpendicular and parallel parking. The two algorithms implemented were the Deep Q-Learning (DQN) and the Deep Recurrent Q-Learning (DRQN), both approaches were able to successfully learn to park the vehicle in both parking spots from a variety of different initial positions and orientations. Although DRQN presented a high success rate with 96.5% for parallel parking and 96.22% for perpendicular parking, the DQN presented a slightly higher success rate with 98.81% for parallel parking and 97.58% for perpendicular parking.

Chapter 3

System Specification

In this chapter, every implementation developed and used in this dissertation is explained. Each algorithm developed to control a robot has to be tested in a simulation environment, thus, section 3.1 presents the environment simulator used and the reasons for its choice, as well as the implemented track where the agent was trained. In section 3.1.1, the agent is presented, more specifically, all the components present in the agent and their limitation, for example, limit in velocity and steering angle. In section 3.1.2, it is exhibited a closer view of the types of parking spots in the simulated track and the challenges presented to the agent in the training phase, for each of the parking maneuvers. Since the reinforcement learning algorithm was implemented in a Python script, it was necessary to create a ROS framework to stabilise a connection between the Python IDE Pycharm and CoppeliaSim. Thus, in section 3.2, the theoretical foundations of ROS and all the implementation process in this project, are explained. Lastly, in section 3.3, it is presented a brief summary of the chosen Reinforcement Learning algorithm, an overview of the action and state space of the environment, a detailed explanation of the reward function used in each of the two approaches implemented and the structure of the Neural Networks of the Reinforcement Learning algorithm.

The software was implemented on Ubuntu 20.04 operating system on a Toshiba Satellite p50-a with an Intel Core i7 7th Gen 4700MQ CPU an Nvidia GeForce GT 745M and 8Gb of RAM.

3.1 Simulation Environment: CoppeliaSim

When creating any kind of algorithm to control a robot it is highly beneficial to test it first in a simulated environment. The main advantage of robot simulation is that it provides proof of

concept so that flaws are not built into the robotic systems that would most likely damage the real robot. This would make the entire project extremely time consuming and cost-benefit inefficient.

The simulation environment chosen to simulate the entire competition environment was CoppeliaSim Edu, Ubuntu 20.04 version. CoppeliaSim is a robot simulator based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or BlueZero node, a remote API client or a custom solution making it a very versatile simulator for multi-robot applications [44]. There are many reasons for the choice of CoppeliaSim as a simulator for this project, the main one being that it is the one used by the entire autonomous driving team of the *Laboratório de Automação e Robótica* for the development of the project. Thus, the simulated track for the competition, that is necessary to accomplish the objectives of this dissertation, was already developed by Inês Ribeiro which is a member of the *Laboratório de Automação e Robótica*. In figure 3.1, a simulated version, in CoppeliaSim, of the track for the Autonomous Driving Competition of the RoboCup Portuguese Open is presented.

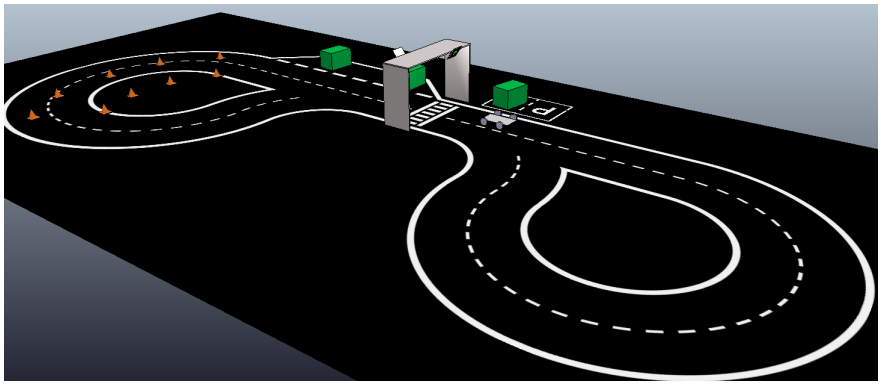


Figure 3.1: Simulated version of the track in CoppeliaSim

3.1.1 The Agent

In order to create a realistic simulation, it was necessary to develop a simulated version of the real robot in the CoopeliSim simulated environment. Based on the real robot, Inês Ribeiro had already created the simulated version of the robot. This simulated robot consisted of a rectangle shape with 60 centimetres of length and 30 centimetres width. In figure 3.2, the simulated robot in CoppeliaSim is presented. The robot is a four-wheel vehicle with rear-wheel traction.

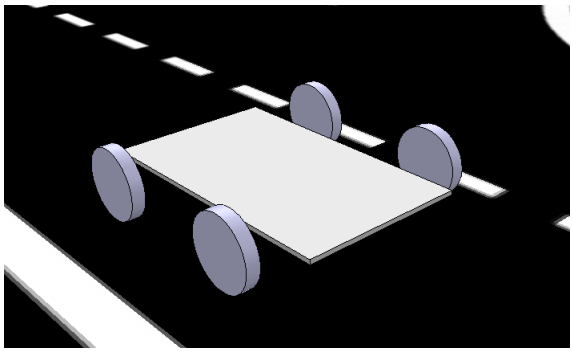


Figure 3.2: Simulated robot in CoppeliaSim

For the traction system, revolute joints from CoppeliaSim were used to simulate the motors. Revolute joints are a CoppeliaSim object that allows the user to emulate a motor by rotating the object that is attached to. The front wheels have joints in which the main purpose is to control the car direction.

Parking maneuvers are usually carried out at low speed because a certain level of precision is necessary to avoid colliding with nearby entities. Thus, the maximum motors speed is limited via software to 1 m/s.

For the robot to be able to avoid obstacles, it has to detect them first. So, six proximity sensors were placed in the robot, three in the front and three in the back, so that the robot is able to park both front and rear, as presented in figure 3.3. These are ultrasonic proximity sensors with a minimum range of detection of 1 cm and a maximum range of detection of 30 cm.

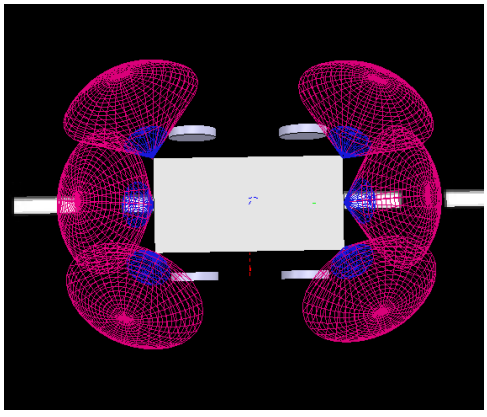


Figure 3.3: Final version of the robot with the proximity sensors in CoppeliaSim

3.1.2 Environment: Types of parking

The main objective of this dissertation is the development of a Reinforcement Learning algorithm able to park a robot in three different types of parking spots. Perpendicular parking spot, parallel parking spot and angular parking spot.

Throughout this dissertation, the same method was used to train the agent in each of the following scenarios. First, the agent was trained in the simplest version of each one of the parking spots and then the difficulty and complexity of the task were progressively increased. To increase the difficulty of the task it was placed an obstacle, the position and orientation of the agent were randomly changed in each episode. Lastly, the parking spot position would also vary from one episode to another. For each of these steps, the training and testing phases would be repeated until the agent successfully master the task in question.

3.1.2.1 Perpendicular parking

In this scenario, the agent must be able to park itself in a perpendicular parking spot with and without obstacles. Since the track, referred in section 3.1, already has a perpendicular parking spot incorporated, this was used in the training and testing of the agent in this scenario. In figure 3.4, the part of the track where the perpendicular parking spots are placed is presented.

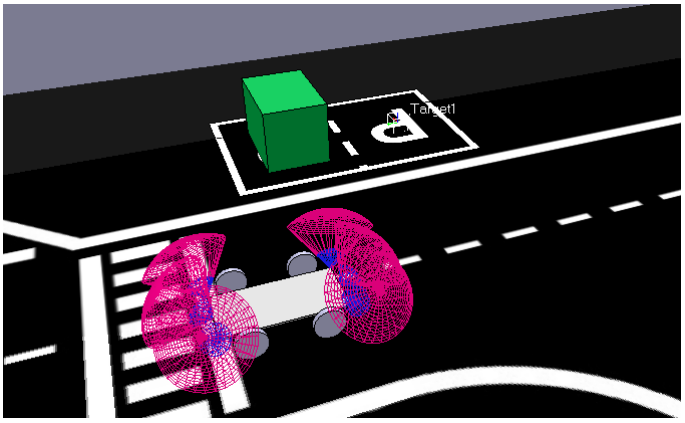


Figure 3.4: Perpendicular parking spot in CoppeliaSim

The agent started the training knowing only the data from the sensors, its orientation in the current time step, its desired final orientation and his position (X and Y coordinates) relative to

the target placed in the desired parking spot. In the first instance, the agent was presented with a scenario where he had to learn how to park in a perpendicular parking spot from a fixed initial position. After successfully learning to complete the previous scenario, the agent was placed in a more complex and challenging scenario. The difficulty and complexity of the perpendicular parking scenario were progressively increased throughout all phases of the training. All phases of this training are explained in more detail in section 4.

One of the goals of this dissertation was the participation in the Autonomous Driving Competition of the RoboCup Portuguese Open which would be the ultimate test of the agent but due to the coronavirus pandemic, the competition did not take place.

3.1.2.2 Parallel parking

Similarly to the perpendicular parking challenge, in this scenario, the agent must be able to park itself in a parallel parking spot with and without obstacles. Since the track referred in section 3.1 already has a parallel parking spot incorporated, this was used in the training and testing of the agent in this scenario. In figure 3.5, the part of the track where the parallel parking spot is placed is presented.

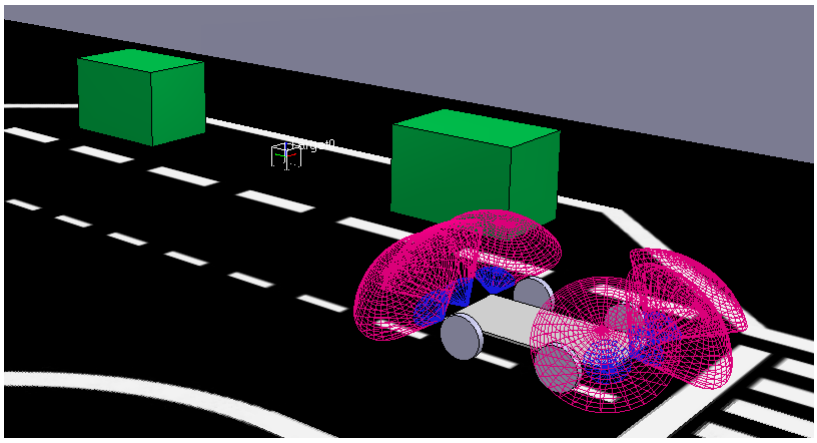


Figure 3.5: Parallel parking spot in CoppeliaSim

Similarly to the perpendicular parking, the only information that the agent knows about the environment is the data from the sensors, its orientation in the current time step, its desired final orientation and its position relative to the target, which is placed in the desired parking spot. In this

scenario, the agent had to learn to parallel parking in an obstacle's free spot from a fixed position. Upon learning to park in the previous scenario, the difficulty and complexity were increased and the trained restarted. The complexity of the parallel parking scenario was progressively increased throughout all phases of the training. All phases of this training are explained in more detail in section 4. These phases in the training of the agent are important because it increases the adaptability of the system.

Since one of the challenges of the Autonomous Driving Competition of the RoboCup Portuguese Open was the parallel parking, this would be the ultimate test of the agent but due to the coronavirus pandemic, the competition did not take place.

3.1.2.3 Angular parking

Similarly to the previously referred challenges, in this scenario, the agent must be able to park itself in an angular parking spot with and without obstacles. Since does not exist an angular parking challenge in the autonomous driving competition of the RoboCup Portuguese Open event, the track referred in section 3.1 does not have an angular parking spot incorporated. Thus, it was placed in the parallel parking spot as presented in figure 3.6.

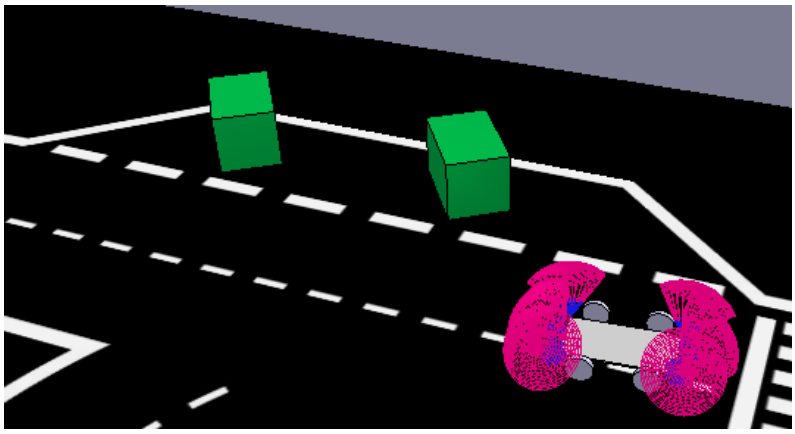


Figure 3.6: Angular parking spot in CoppeliaSim

The setup for this scenario is similar to the previously presented scenarios. The only information that the agent knows about the environment is the data from the sensors, its orientation in the current time step, its desired final orientation and its position relative to the target, which

is placed in the desired parking spot. The agent starts the training in the simplest version of the angular parking scenario. The agent starts the training with an obstacle's free spot from a fixed position. Upon completing this phase of the training, the agent was placed in a more complex version of the same scenario. The complexity of the angular parking scenario was progressively increased throughout all phases of the training. All individual phases of this training are explained in more detail, in section 4. These phases in the training of the agent are important because it increases the adaptability of the system.

3.2 ROS Framework

Since the Reinforcement Learning algorithm software was developed on a Python script external to CoppeliaSim, it was necessary to stabilise a connection between the Python IDE Pycharm and the simulator CoppeliaSim. One of the ways provided by CoppeliaSim to address this issue is via ROS. Robot Operating System (ROS) is a set of software libraries and tools for creating robotic applications. From drivers to cutting-edge algorithms and with powerful development tools that simplify the task of creating complex and robust robot behaviour across multiple robotic platforms. ROS is a complex open-source platform, a "Middleware" based on an anonymous publication/subscription mechanism that allows the transfer of messages between different ROS processes. ROS can have multiple processes running in parallel, each one is called a node, every node is responsible for one task and communicates with eachother by passing messages via logical channels called topics. Each node has to have a unique name. A message is a strictly typed data structure. Standard primitive types (integer, floating-point, boolean, etc.) are supported.

As previously mentioned, ROS works in a publication/subscription mechanism. This means that a node can communicate with other node by publishing the pretended message (data) into a given topic and then the other node to receive it subscribes to the same topic. Shortly, one node publishes the data in the topic to send it and the other subscribes to the same topic to receive it. It is possible to have multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of eachother's existence [45]. After the communication is established the nodes

communicate with eachother in a peer-to-peer topology, but before nodes start to transmit data over topics they must first advertise the topic name and the type of message.

A system built using ROS consists of a number of nodes connected at runtime in a peer-to-peer topology. The peer-to-peer topology requires some sort of lookup mechanism to allow nodes to find each other. This is called the Master. The ROS Master is responsible for managing names and registration services to the nodes within a ROS system. Publishers and subscribers are monitored by the ROS Master to ensure associated topics. The ROS Master also enables location and communication between nodes within the robotics system. Lastly, the ROS Master commonly initiates the node communication. In figure 3.7 it is presented the Publisher/Subscriber architecture of a ROS system.

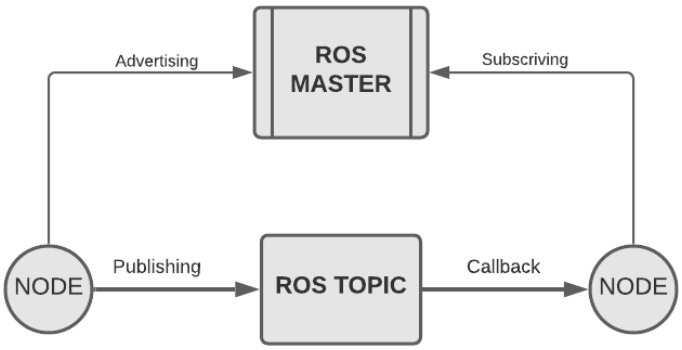


Figure 3.7: Publisher/Subscriber architecture of a ROS system

Although CoppeliSim offers a wide range of possible robot control systems, the choice of ROS rested in the practical applicability of ROS software.

All Reinforcement Learning algorithms learn in a trial and error strategy, so the agent must experience an enormous amount of episodes to master a certain task. For this reason, it was necessary to optimize the process of restart the simulation after every episode. The CoppeliSim already provided a customization script named "RosInterfaceHelper" that solved this problem. This script is extremely helpful because it is being executed while CoppeliSim is running and not only when the simulation is running. It also contains a variety of pre-created subscribers and publishers that helped to create an optimized simulation. From this script the following subscriber were used:

- startSub (Boolean) - Subscriber that indicates to CoppeliSim to start the simulation;
- stopSub (Boolean) - Subscriber that indicates to CoppeliSim to end the simulation.

To establish a connection between the python script and the simulator two nodes were created. The first node was for the simulator and it was named /sim_ros_interface. The second node was for the Python script and it was named /talker. In order to establishe an organised and efficient communication between the two ROS nodes it was necessary to use six different ROS topics either for data transition or simulation control:

- /startSimulation (Boolean) - This ROS topic sends information from Pycharm to CoppeliaSim to indicate the start of a new episode;
- /stopSimulation (Boolean) - This ROS topic sends information from Pycharm to CoppeliaSim to indicate the end of the current episode for reaching a terminal state.
- /chatter (Float32MultiArray) - This ROS topic sends the action the agent will perform from Pycharm to CoppeliaSim
- /sensores (Float32MultiArray) - This ROS topic sends the state of the environment from CoppeliaSim to Pycharm. The state contains the information of the sensors and the robot coordinates relative to the target.
- /collision (Boolean) - This ROS topic sends information from CoppeliaSim to Pycharm to indicate if a collision occurred.
- /privateMsgAux - This ROS topic is automatically created by CoppeliaSim, it does not perform anything relevant to this specific ROS framework;

In Figure 3.8, the implemented ROS rqt_graph is presented describing how the two ROS nodes interact with the six ROS topics.

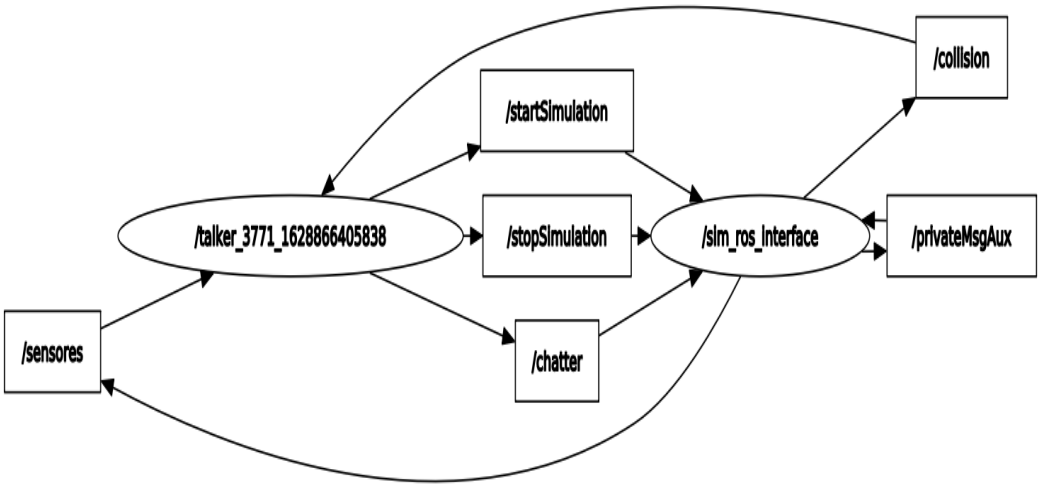


Figure 3.8: Graphical representation of how both of the ROS nodes communicate

In CoppeliaSim, the control and customization of a robot and the simulation are implemented via Lua scripts. Lua scripts are embedded scripts provided by CoppeliaSim to configure all the simulation parts, it is a flexible and straightforward implementation, with guaranteed compatibility with every other default CoppeliaSim installation. This method allows users to control every element of the simulation. Lua is powerful, fast and designed to be a lightweight embeddable scripting language.

All the publishers and subscribers previously mentioned, necessary to establish an efficient communication between Pycharm and CoppeliaSim, were developed in a non threaded associated child script in CoppeliaSim. Regarding the ROS communication two scripts were used in this dissertation, one was the ROSinterfacehelper already mentioned and the second was developed to create the remaining publishers and subscribers.

3.3 Reinforcement Learning

After a study of different Reinforcement Learning algorithms, the chosen one is the Deep Deterministic Policy Gradient (DDPG), already explained in section 2.1. The DDPG is an Actor-Critic

algorithm that estimates a deterministic policy, which is considered easier to learn when compared with other approaches and presents a more consistent behaviour. This algorithm works with continuous action-space and state-space which is another reason for being the chosen algorithm.

DDPG is a model-free, off-policy algorithm that learns by mini-batches stored in the replay buffer, the replay buffer behaviour is presented in figure 3.9. To increase the stability, this algorithm is composed of four networks, Actor, Critic, Target Actor and Target Critic, more specifically two sets of the Actor-Critic architecture, Actor-Critic and target Actor-Critic, where both Actors and both Critics are a copy of each other. Each of the Neural Networks has different functions. The Actor function defines the action the agent performs and the Critic function judges how good it was the action, through the TD error, as presented in figure 3.9. The target Neural Networks define the target value (Q_{target}) used in the TD error equation. The deterministic policy gradient theorem provides the weight update rules for the Actor. The Critic network is updated regarding the gradients obtained from the TD error. The target Neural Networks are updated via soft-update.

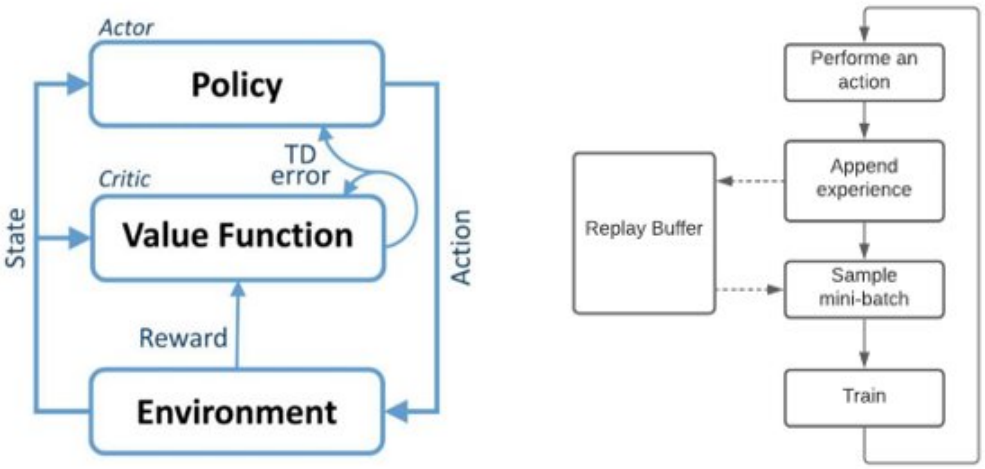


Figure 3.9: Actor-Critic architecture/left and Replay Buffer behavior/right

In this section, the action space and state space of the environment where the variables that constitute the state and the action and its limitations are explained. An extensive review and explication of the reward functions implemented and lastly, the Neural Networks architecture used, are presented.

3.3.1 Action-Space and State-Space

As previously mentioned, the DDPG takes as inputs the state of the environment and outputs the action. The state is constituted by the data of the six proximity sensors, the robot X and Y coordinates relative to the target, placed in the desired parking spot, the current orientation of the robot and the desired final orientation, this is the final posture that the robot should have at the end of the parking maneuver. The sensors have a 30 cm range, if nothing is in this range a "nil" value is returned. The coordinates have no limit so it sends a real value for every possible position of the robot on the track. The final orientation is a fixed value set at the beginning of the episode and it depends on the type of parking spots. The orientation is limited to the range of -180° to 180°.

Regarding the action, the DDPG algorithm deploys as an output, the linear velocity and the steering angle. Because parking maneuvers are a low-speed maneuver, the speed was limited to 1 m/s. Relatively to the steering angle, to create a more realistic version of the real robot, it was limited to the same range of the real robot is physically limited. Thus, the steering angle is limited to the range -40° to 40°, where -40° represents the maximum turn to the left and 40° represents the maximum turn to the right, as presented in figure 3.10.

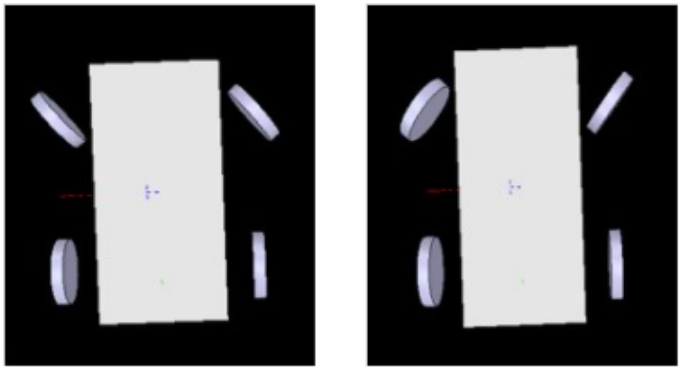


Figure 3.10: Orientation limits of simulated robot (-40°/left and 40°/right)

3.3.2 Reward Function

Throughout this dissertation, two different approaches were tested for the three parking maneuvers previously described. For each approach different reward functions were tested and iteratively improved until derived the final reward function. In this section, the two approaches and their respective reward functions are presented.

3.3.2.1 One Target Approach

For the first approach, one target was placed in the centre of the parking spot, as shown in figure 3.11. Throughout this dissertation, this approach is referred to as one target approach.

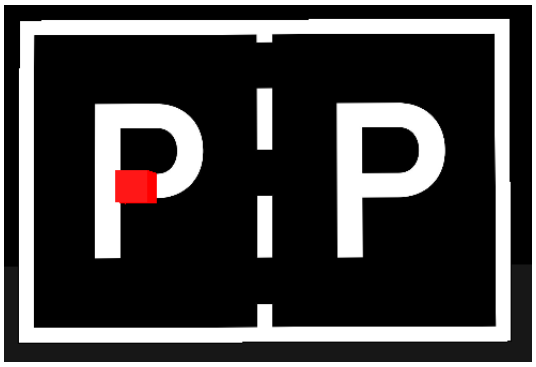


Figure 3.11: One target approach

Since the goal is for the robot to park itself in the middle of the parking spot while avoiding collisions, the reward function was created considering these factors. The final reward function is divided into 5 parts, distance reward, collision reward, orientation reward, incentive reward and success reward.

- **Distance Reward:**

  To encourage the agent to drive itself in the target direction, at each time step a reward is given, relative to the agent's current distance from the target. The closer the agent is to the target the highest is the reward. The distance reward is obtained by the following equation:

$$R_{Distance} = e^{-(k_x.X^2+k_y.Y^2)} \tag{3.1}$$

Where X and Y are the X and Y coordinates of the agent relative to the target and k_x and k_y are constants that prioritize the movement in one of the axis, for example, if k_y is greater than k_x the movement in the Y axis is prioritized. The exponential is important to normalize the reward since the exponential of a negative number is set to the range [0, 1].

- **Collision Reward:**

  To make sure that the agent avoids the obstacles, a penalty was set for a collision. Since the collision is a terminal state, every time the agent collides with an obstacle, the episode stops and the agent receivs a penalty.

$$R_{Collision} = \begin{cases} 0 & if \quad collision = False \\ -100 & if \quad collision = True \end{cases} \tag{3.2}$$

  This reward is extremely important because showing the agent what not to do is as important as showing it what to do.

- **Orientation Reward:**

  It was necessary to create a reward to encourage the agent to orient itself in the desired orientation. Thus, the orientation reward was created to reward the agent for its orientation being closer to the pretended orientation. For example, in a scenario where the pretended orientation is 90°, the closer the agent's orientation is to 90° the higher is the reward. The equation used in the orientation function reward is presented below:

$$R_{Orientation} = e^{-k_\theta(\theta_{target}-\theta)^2} \tag{3.3}$$

  Where θ represents the robot's orientation in the current time step, θ_{target} represents the final orientation and k_θ is a constant that was experimentally deducted. As in the distance reward, the exponential in this equation has the function of normalizing the rewards, so that the reward obtained from this function is in the range [0, 1].

- **Incentive Reward:**

  In order to encourage the agent to complete the maneuver as quickly as possible, a penalty is given at each time step, as shown in the next equation:

$$R_{Incentive} = -1 \tag{3.4}$$

  This reward is important because the longer the agent takes to complete the parking maneuver the smaller the total sum of rewards is, which forces the agent to find better and quicker strategies.

- **Success Reward:**

  To encourage the agent to complete the maneuver, a reward is given every time the agent is able to park. Although, certain criteria has to be respected to accept the final position of the robot as a success attempt. Thus, to be accepted, the robot has to be in a range of 5 centimetres of the target and the error between its orientation and the desired orientation has to be less than 10°, this rejects the scenarios where the robot is parked too far off the centre of the parking spot and the scenarios where the robot is not aligned with the parking spot:

$$\begin{cases} Distance <= 5cm \\ \quad e_\theta <= 10^o \end{cases} \tag{3.5}$$

  Where e_θ is the error between its orientation and the desired final orientation.

  When this is respected the episode stops, since this is a terminal state, and it is given the highest reward the agent receives in the episode, as shown in the next equation:

$$R_{Success} = \begin{cases} 0 & if \quad success = False \\ 200 & if \quad success = True \end{cases} \tag{3.6}$$

  This is the highest reward given to the agent in the episode because a greater reward should be given to the successful runs with lower time consumption and no collision, in order to encourage the agent to repeat that attempt.

- **Final Reward Function:**

  The final reward function is a combination of all the previously mentioned rewards and it is given by the following equation:

$$Reward = (1-w)*R_{Distance}+w*R_{Orientation}+R_{Incentive}+R_{Collision}+R_{Success} \tag{3.7}$$

Where w is the prioritize coefficient between the orientation reward and the distance reward. If $w = 0$ the orientation reward is ignored, if $w = 1$ the distance reward is ignored. This coefficient can change, depending on each phase of the maneuver the agent is currently on. For each parking maneuver, two phases were considered. The first phase is the target approximation phase, in figure 3.12 an example of this phase is presented. In this phase, w is zero. Thus, the orientation reward is ignored because if the agent is far from the parking spot, it is not relevant that the agent's orientation is equal to the desired final orientation. In this phase, it is only relevant that the agent is moving towards the parking spot.

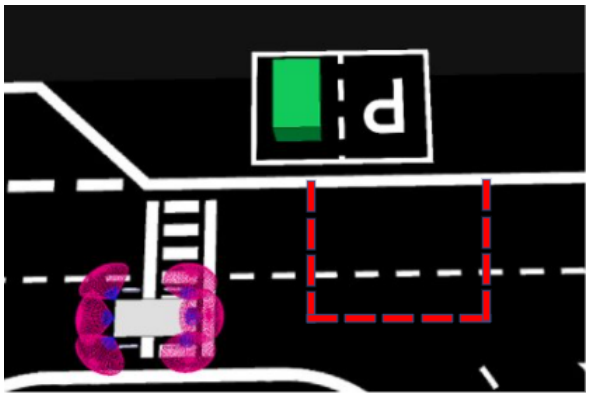


Figure 3.12: Phase 1

When the agent enters the area defined in figure 3.13, the second phase starts. The area's size is different for each parking maneuver and it was deducted by trial and error. In this phase the orientation reward is taken into consideration, rewarding the agent when tending to the pretended orientation.

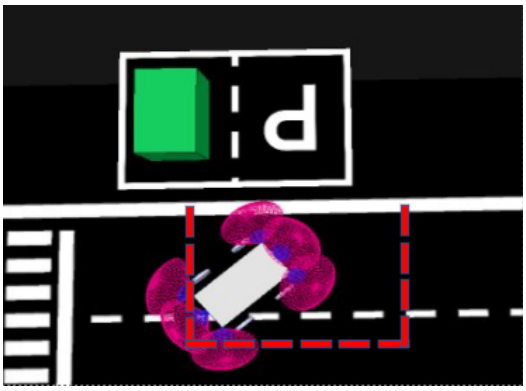


Figure 3.13: Phase 2

Since w is in the range [0, 1], the final reward function is normalized and contained in an interval of [-1, 0]. This way, all the rewards given on a time step basis are negative, which forces the agent to complete the task as fast as possible and since the only positive reward is the success reward, this encourages the agent to tend to this state.

All the coefficients referred above were deducted experimentally and are presented in the next table:

Table 3.1: Value of the coefficients used in the one target approach

Coefficient	***Value***
k_x	0.04
k_y	0.05
k_θ	40
w	0 or 0.1

3.3.2.2 Four Targets Approach

For the second approach, four targets were placed in the corners of the parking spot, as shown in figure 3.14. Throughout this dissertation, this approach is referred to as four targets approach.

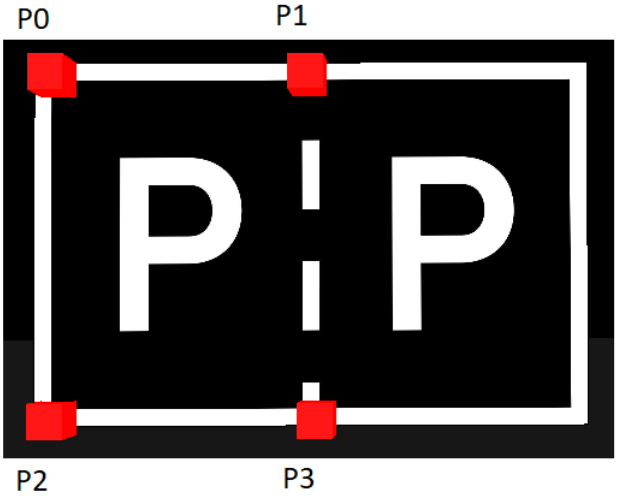


Figure 3.14: Four target approach

As in the one target approach, the function reward is divided into smaller rewards to teach the agent different goals. This approach is divide into 4 parts, distance reward, orientation reward, collision reward and success reward.

- **Distance Reward:**

  In this approach, the distance to the centre of the parking spot is deducted by the four points shown in figure 3.14. As previously mentioned, the distance reward increases when the distance to the centre of the parking spot decreases. The equation for the distance reward is next presented:

$$R_{Distance} = R_{DistanceX} + R_{DistanceY} \tag{3.8}$$

$$R_{DistanceY} = -(\frac{1}{2}.abs(Y_{P0} + Y_{P1}) + \frac{1}{2}.abs(Y_{P2} + Y_{P3})) \tag{3.9}$$

$$R_{DistanceX} = -(\frac{1}{2}.abs(X_{P1} + X_{P3}) + \frac{1}{2}.abs(X_{P0} + X_{P2})) \tag{3.10}$$

  Where $Reward_{DistanceX}$ and $Reward_{DistanceY}$ is the distance reward in the X axis and Y axis, respectively, to the centre of the parking spot and X_{P_x} and Y_{P_x} is the normalized X and Y coordinates of the agent relative to the point P_x.

- **Orientation Reward:**

  To encourage the agent to orientate itself to the pretended final orientation a reward is given to the agent at each time step, which penalizes the further away its orientation is from the pretended orientation. The orientation reward is given by the following equation:

$$R_{orientation} = abs(\theta_{target} - \theta) \tag{3.11}$$

  where θ_{target} is the target final orientation that the agent should have in the parking spot and θ is the orientation of the agent in the current time step.

- **Collision Reward:**

  As explained in the one target approach, to teach the agent to avoid obstacles, a negative reward is given to the agent and the episode terminates every time the agent collides with an obstacle. The collision reward is presented below:

$$R_{collision} = \begin{cases} 0 & if \quad collision = False \\ -100 & if \quad collision = True \end{cases} \tag{3.12}$$

- **Success Reward:**

  To encourage the agent to complete the maneuver more often, every time the agent reaches the parking spot and the following criteria is verified, the episode terminates and the agent receives a positive reward.

$$\begin{cases} Distance <= 5cm \\ e_\theta <= 10^o \end{cases} \tag{3.13}$$

  Where e_θ is the error between its orientation and the desired final orientation.

  The success reward is given by the following equation:

$$R_{Success} = \begin{cases} 0 & if \quad success = False \\ 200 & if \quad success = True \end{cases} \tag{3.14}$$

- **Final Function Reward:**

  The final reward function on this approach is similar to the final reward function of the one target approach. Thus, it is a combination of all previously mentioned rewards and it is given by the following equation:

$$Reward = (1 - w) * R_{Distance} + w * R_{Orientation} + R_{Collision} + R_{Success} \quad (3.15)$$

  Where w, is the prioritize coefficient between the orientation reward and the distance reward.

  In this approach, similarly, to the one target approach, two phases exist. In phase one, w is zero, therefore, the orientation reward is ignored and it is only considered the distance reward. The agent is in phase one every time that it is outside the area defined by the targets $P0$ and $P1$ as shown in figure 3.15.

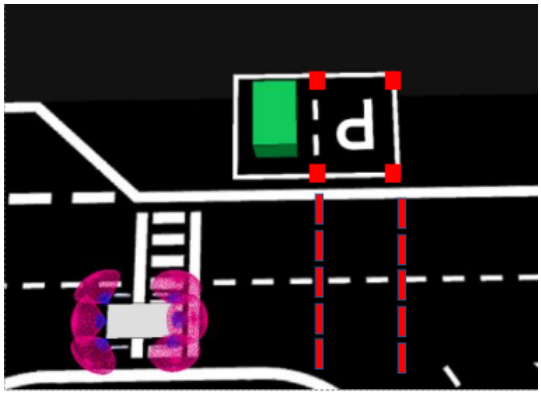


Figure 3.15: Phase 1

  When the agent enters the area defined by the targets $P0$ and $P1$ as shown in figure 3.16, phase two starts. In this phase, w is 0.1. Thus, the orientation reward is taken into consideration.

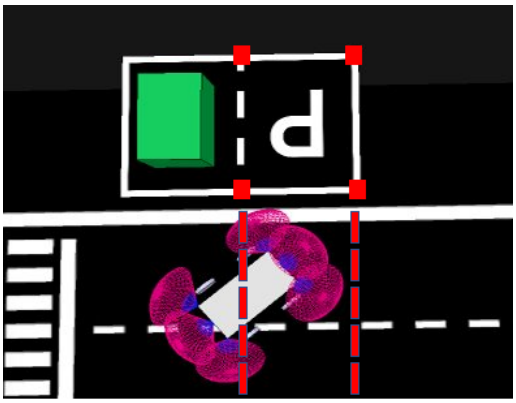


Figure 3.16: Phase 2

Since w is contained in the range [0, 1], the final reward function is normalized and contained in the interval [-1, 0].

3.3.3 Neural Networks

Regarding the Neural Networks, the DDPG algorithm is constituted by four Neural Networks. Since the target Actor and target Critic have the same structure as the Actor and the Critic, respectively, only the Actor and Critic structure are explained.

The Actor is a Neural Network constituted by two fully connected layers. The first layer is constituted by 400 neurons and the second layer by 300 neurons. The activation function of the two hidden layers is "ReLU". The Actor has one input layer with 12 neurons, which has to be the same number as the state space. The Actors output layer has two neurons with the activation function "tahn", to bind the actions. The Actor structure is presented in figure 3.17.

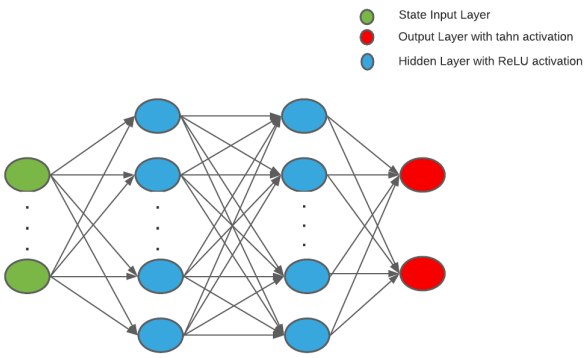


Figure 3.17: Actor structure

The Critic is a Neural Network constituted by two fully connected layers. The first layer is constituted by 400 neurons and the second layer by 300 neurons. The Critic has two input layers one for the actions and other for the state, but the action is only included in the second hidden layer. The output layer has one neuron which outputs the action value. The Critic structure is presented in figure 3.18.

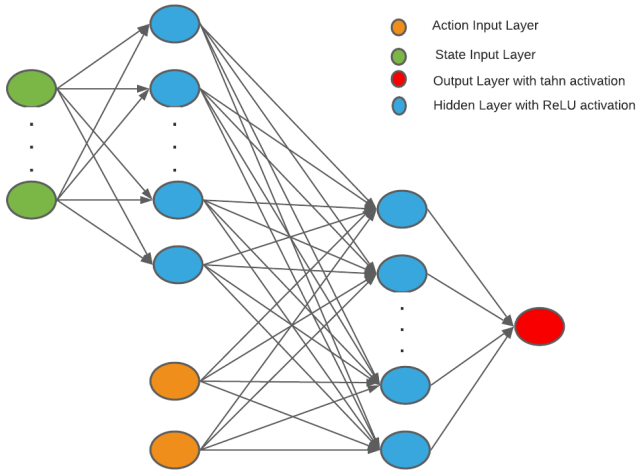


Figure 3.18: Critic structure

Chapter 4

Tests and Results

In this chapter, the final training and all the intermediate phases that lead to it are presented for the three mentioned parking maneuvers. Next, all the tests carried out and an analysis of the results obtained are also presented. For all three parking maneuvers, multiple trainings were carried out, in a trial and error strategy, in the search of the optimal hyper-parameters. Although multiple hyper-parameters were tested, only one set was found that was able to lead to a policy capable of completing the training. To accomplish this, it was necessary to decrease the frequency of transition from 20Hz to 5Hz. For the frequency of 20Hz, the variation in the state was too small for the agent to find a successful policy. In table 4.1 the hyper-parameters and the training specifications used in all the phases of the training are presented.

Table 4.1: Hyper-parameters and the training specifications

Parameters	***Value***	***Parameters***	***Value***
Actor Learning Rate	0.0001	Critic Learning Rate	0.0002
Experience Replay Buffer size	30000	Training batch size	512
Reward Decay Factor	0.995	Soft update parameters	0.001
Number of training rounds	2500-5000	Maximum step per round	300

4.1 Perpendicular Parking

4.1.1 Training

To create a versatile and robust system for this maneuver the training was divided into six phases, each one more difficult and complex than the one before. This idea of progressively increasing

the training difficulty had the purpose of facilitating the searche for hyper-parameters and to better adjust the reward function previously mentioned. The agent must be able to complete the current phase of the training before it can pass to the next phase. The initial idea was to transfer the learning of the first phase and use it to improve the training of the second phase and to obtain a faster conversion. When tested, it was discovered that transferring the learning of one phase to another, would decrease the performance of the agent and most times lead to the lack of conversion in that training. Thus, in all phases of this training described below the agent starts learning from zero. Two sets of weights were withdrawn for all phases of this training. The latest weights, withdrawn at the end of the episode and the best performance weights, withdrawn in the episode that was registered the higher average episodic reward of the training.

4.1.1.1 First Training

For the first phase of the training the agent was trained in the simplest possible version of this maneuver. The agent was placed in a fixed position at the right of the parking spot, as shown in figure 3.4. For this scenario the obstacles were not added. The agent was trained using the two approaches previously mentioned, the one target approach and the four target approach. The one target approach training lasted 4000 episodes and the progress is presented in figure 4.1(a). The four target approach training lasted 3300 episodes and the progress is presented in figure 4.1(b).

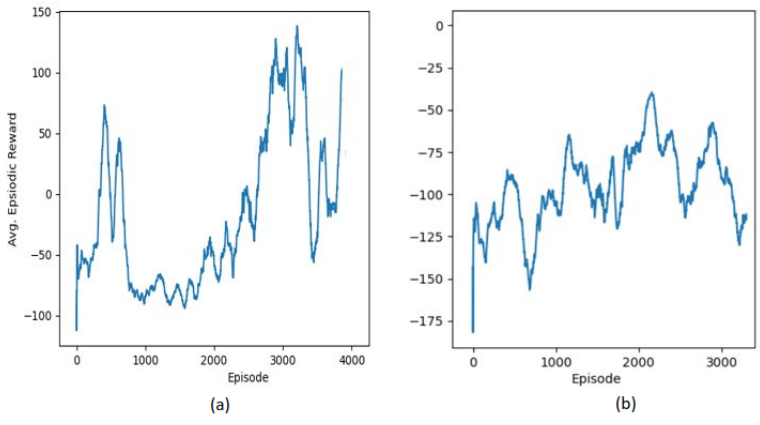


Figure 4.1: Agent's training progress for the first training of the perpendicular parking challenge for the one target approach (a) and for the four target approach (b)

Regarding the four target approach, throughout the whole training, the agent was unable to learn a policy capable of parking. The best performance, in this training, was registered at episode 2100 with an average episodic reward of -40. Since the agent was unable to learn a policy capable of parking in the simplest version of this maneuver, in the remaining phases of the training, for this maneuver only the one target approach was used.

Regarding the one target approach, the average episodic reward for all 4000 episodes is presented in figure 4.1(a). This represents the reward evolution along with this phase of training, which allows a better understanding of all the agent's attempts in this phase. As shown in figure 4.1, in the first 500 episodes the agent is able to complete a few episodes, obtaining an average reward of 75, but since it is exploring all possible alternatives in the search for the optimal policy, the performance decreases. The agent obtains the best performance in this phase with an average reward of 145, in episode 3200, but it was unable to maintain this performance. The agent ends the training with an average episodic reward of 110.

As previously mentioned, two sets of weights were withdrawn, the best performance weights and the latest weights. For the latest weights, the agent learned a policy that was not able to complete all episodes. For the best performance weights the agent learned a policy that was able to complete all episodes but an average reward of 145 is too low to be considered the optimal policy.

4.1.1.2 Second Training

For the second training, the agent was placed in the same scenarios used for the first training. To increase training difficulty, it was added randomly a small variation to the initial position of the agent every episode and one obstacle was placed in the parking spot adjacent, as shown in figure 3.4. The training lasted 5000 episodes and the progress is presented in figure 4.2.

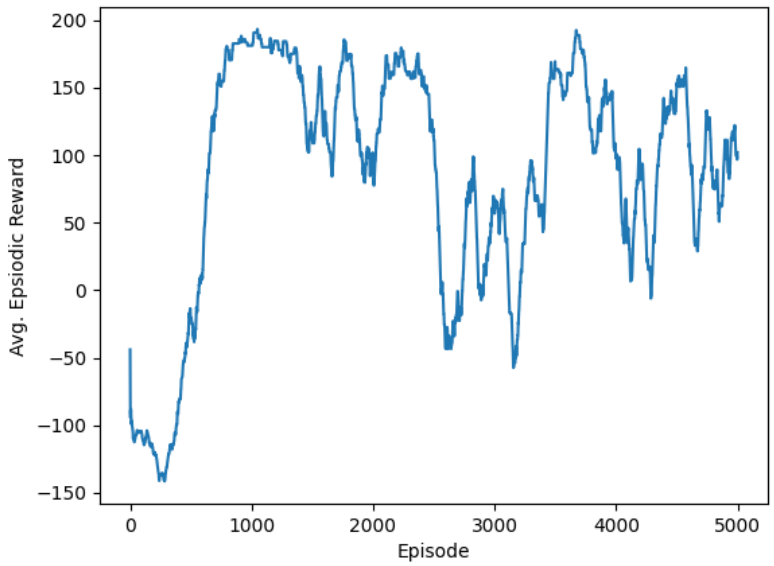


Figure 4.2: Agent's training progress for the second training of the perpendicular parking challenge

Upon analysing the progress of the agent in this training, it is possible to observe that the agent reaches the best performance in episode 1100 with an average episodic reward of 195. Since the agent was exploring, the performance kept decreasing and increasing throughout the episode. The agent ends the episode with an average reward of 100.

Once again two sets of weights were withdrawn. For the latest weights, the agent learned a policy that was not able to complete the maneuver for all trained initial positions. For the best performance weights, the agent was able to complete all scenarios of this training.

4.1.1.3 Third Training

For the third training, the agent was placed at the left of the parking spot. A small variation was randomly added to the initial position of the agent every episode and one obstacle was placed in the adjacent parking spot. The training lasted 2500 episodes and the progress is presented in figure 4.3.

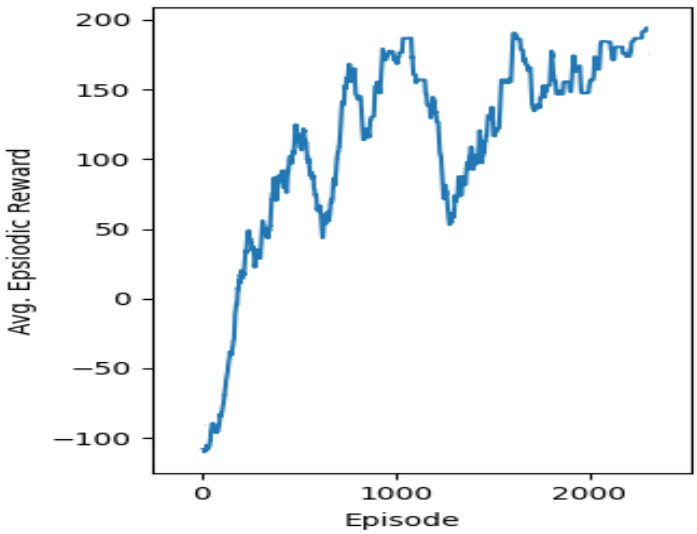


Figure 4.3: Agent's training progress for the third training of the perpendicular parking challenge

The progress of the agent in this training is straightforward, consistently improving performance throughout the training with only one considerable fall around episode 1100. Since, the best performance was archived at the end of the episode, only one set of weights was withdrawn. The agent obtained an average episodic reward of 195 at the end of the episode, proving that it can complete all scenarios of this test. This was already expected because this scenario is similar to the scenario used in the previous test, where the agent was also able to complete all trained situations.

4.1.1.4 Fourth Training

In the fourth training, the difficulty highly increased because the agent was randomly placed on both sides of the parking spot where its initial position would still suffer a small variation. This scenario was more challenging than the ones described before, because in addition to having to adapt to the small variations in its initial position, it also has to learn a policy that enables to perform two different movements. The training lasted 2500 episodes and the progress is presented in figure 4.4.

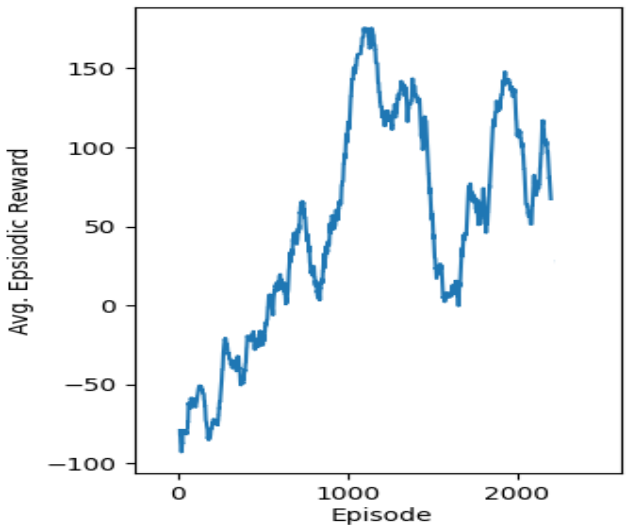


Figure 4.4: Agent's training progress for the fourth training of the perpendicular parking challenge

Upon analysing the progress of the agent throughout this training, the agent consistently improved its performance, until obtaining its best with an average episodic reward of 170 in episode 1100. For the remainder of the training, the agent kept on exploring but was not able to learn a better policy, ending the training with an average episodic reward of 70.

The agent was unable to learn the optimal policy for this scenario but the sub-policy learned was still able to complete all trained positions. The decrease in performance in comparison with the previous training was already expected due to the difficulty of learning a policy able to adapt its movement to opposite positions of the parking spot and it is extremely higher than adapting its movement for positions separated by just a few centimetres.

4.1.1.5 Fifth Training

For the fifth training, the setup used in the previous training was reused. In every episode, the agent was randomly placed on one of the sides of the parking spot and a small randomly variation was added to its initial position. The obstacle was also kept. The goal of this training was to teach the agent to park and reverse park. The training lasted 5000 episodes and the progress is presented in figure 4.5.

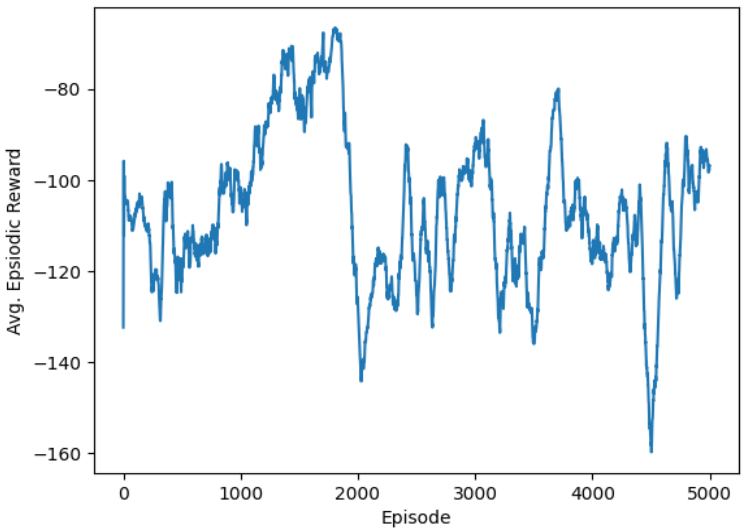


Figure 4.5: Agent's training progress for the fifth training of the perpendicular parking challenge

Throughout all 5000 episodes, the agent did not learn a policy that was able to successfully park. Although in previous training, the agent has successfully learned a policy able to adapt the steering movements from both sides of the parking spot but it was unable to learn a policy that could adapt between negative and positive velocities based on the position of the agent.

4.1.1.6 Sixth Test

For the final training of the agent in the perpendicular parking maneuver, the agent was training with any kind of variation on position and orientation, so it would learn a policy capable of adapting to the majority of situations. Thus, in every episode the agent was randomly placed on one of the sides of the parking spot. On both sides of the parking spot, the initial position of the agent would also surfer a small variation. Thus, the distance of the agent to the parking spot could vary between 1.2 meters to 2 meters. The initial orientation of the agent was set to 0 degrees for the scenario in which was placed at the left of the parking spot and 180 degrees for the scenario in which was placed at the right side of the parking spot because the agent was unable to adapt between park and reverse park, it was only trained to reverse park. A variation between -10 degrees and 10 degrees was randomly added to the initial orientation of the agent. The training lasted 3000 episodes and the progress is presented in figure 4.6.

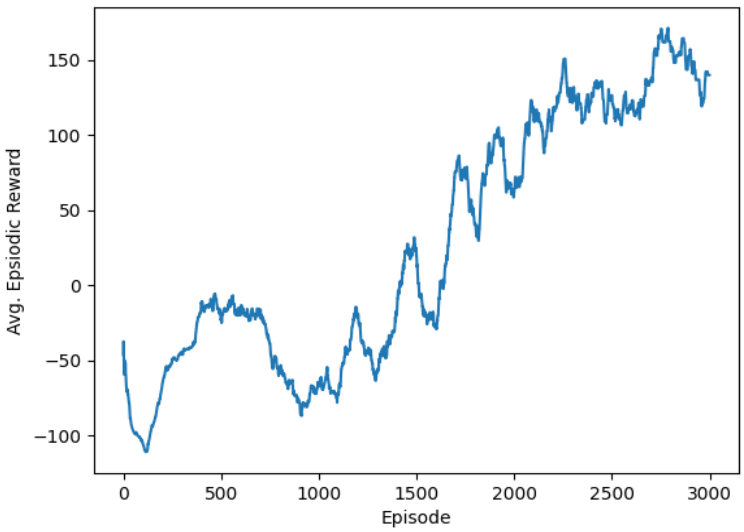


Figure 4.6: Agent's training progress for the final training of the perpendicular parking challenge

The agent learned a policy that enabled it to adapt to all the variations added to the initial position and orientation. Upon analysing the progress of the agent, it improved its performance throughout this training consistently. In episode 2800 it reached its best performance with an average episodic reward of 165. Until the end of the episode, the performance decreased, ending the training with an average episodic reward of 150.

Although the average episodic reward obtained by the agent in this training is lower than in other training previously described, this scenario is more complex than any other described training. Due to the complexity and all the variations in this scenario, the agent was able to learn a policy better at adapting to new situations. For all tests described below, the best performance weights were used.

4.1.2 Tests

Although the agent already learned a policy capable of adapting to the various situations in which was trained, it also should be able to adapt to new situations. In addition, it is important to register the limitations of the agent. Thus, in this section, a variety of tests is presented, to show the limits of the agent relative to variations in the agent's position, agent's orientation and parking spot position.

4.1.2.1 Test A

For test A, the final training scenario was utilized. The agent's initial position would change between both sides of the parking spot. On both sides, the distance of the agent to the parking spot could vary between 1.2 meters to 2 metres. The agent was tested in two scenarios, the scenario without obstacles and the scenario with obstacles. The goal was to determine the impact of the obstacle on the performance of the agent in the perpendicular parking maneuver. This test lasted 50 episodes for each scenario and the progress is presented in figure 4.7.

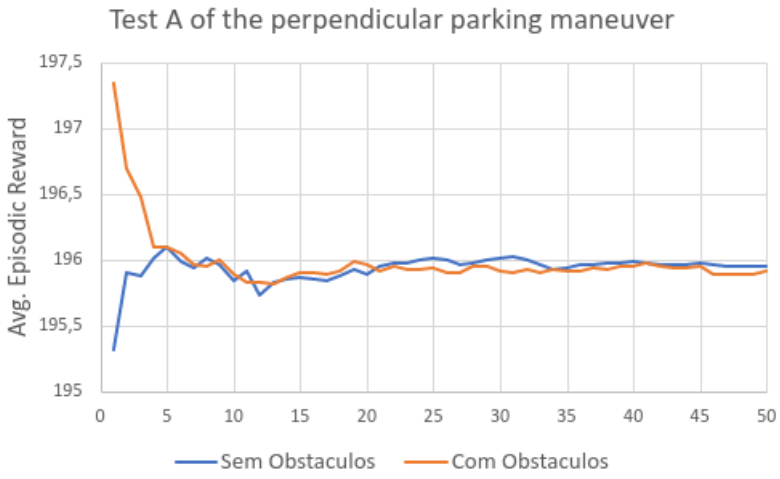


Figure 4.7: Agent's performance of the test A for perpendicular parking maneuver

Figure 4.7 shows that for all 50 episodes the agent obtained an average episodic reward of 195.95 for the scenario without obstacles and 195.92 for the scenario with obstacles. The initial episodes present a higher average reward for the scenario with obstacles than for the scenario without obstacles. This happened because the initial position of the agent was randomly set between 1.2 meters and 2 meters of the parking spot and in the initial episodes, the agent was further away from the parking spot in the obstacle-free scenario than in the scenario with obstacles, which lead to a faster conclusion of the maneuver in the obstacle scenario and consequently a better reward.

4.1.2.2 Test B

The goal of test B was to evaluate the agent's performance for different distances and to verify the maximum distance between the agent and the parking spot for which the agent was still able to park. To accomplish this, the initial distance between the agent and the parking spot was set to 1 meter. Every time that the agent completed all episodes of the test, the distance was increased by 0.5 meters. For every distance tested, the initial position of the agent would randomly change between a fixed position on both sides of the parking spot. The test lasted 50 episodes for each distance and the progress is presented in figure 4.8.

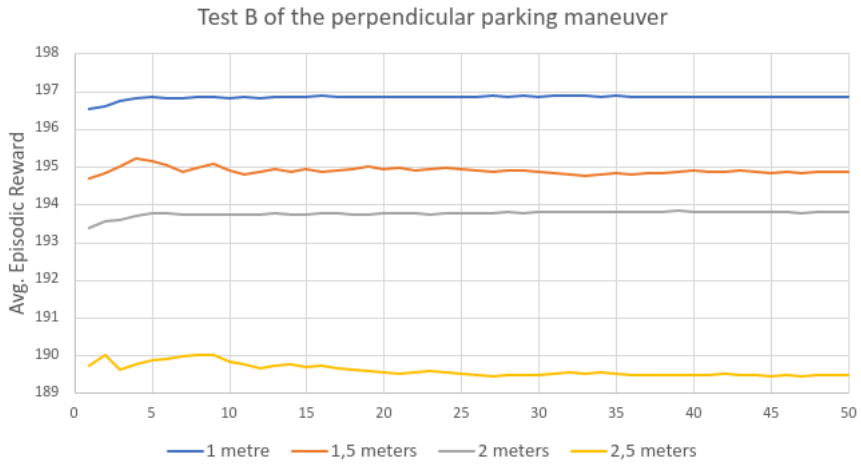


Figure 4.8: Agent's performance of the test B of the perpendicular parking maneuver

In figure 4.8, the average episodic reward for the distances: 1 meter, 1.5 meters 2 meters and 2.5 meters are presented. The agent was able to park for all referred distances. It obtained an average episodic reward of 197 for the 1 meter distance, 194.9 for the 1.5 meters distance, 193.8 for the 2 meters distance and 189.5 for the 2.5 meters distance. The average episodic reward decrease with the increase of distance. This was expected since the longer the agent takes to complete the task, the lower the reward is. With higher distance, more time and more steps are necessary to complete the task. The agent was also tested for a distance of 3 meters but it was unable to park in this scenario. The agent performance during all distance test is very stable, showing a small variation in the average reward. The small variation in performance demonstrates that the policy learned by the agent presents a similar movement from both sides of the parking spot.

4.1.2.3 Test C

The goal of test C was to evaluate the difference in performance for different initial orientations and to determine until what angle of the agent's initial orientation would the agent still be able to park. To accomplish this, the agent was placed in a fixed position 2 meters from the parking spot on both sides. The initial orientation of the agent was set to 0 degrees or 180 degrees depending on the side of the parking spot in which it was placed. Every time the agent completed the test a variation of ± 10 degrees was added to the agent's initial orientation. All orientations used in this test are presented in table 4.2.

Table 4.2: Orientations used in test C

Orientations	***Left Side***	***Right Side***	***Variation***
Orientation A	0°	180°	0°
Orientation B	-10° or 10°	170° or 190°	± 10°
Orientation C	-20° or 20°	160° or 200°	± 20°
Orientation D	-30° or 30°	150° or 210°	± 30°
Orientation E	-40° or 40°	140° or 220°	± 40°

For every orientation tested the test lasted 50 episodes and the progress is presented in figure 4.9.

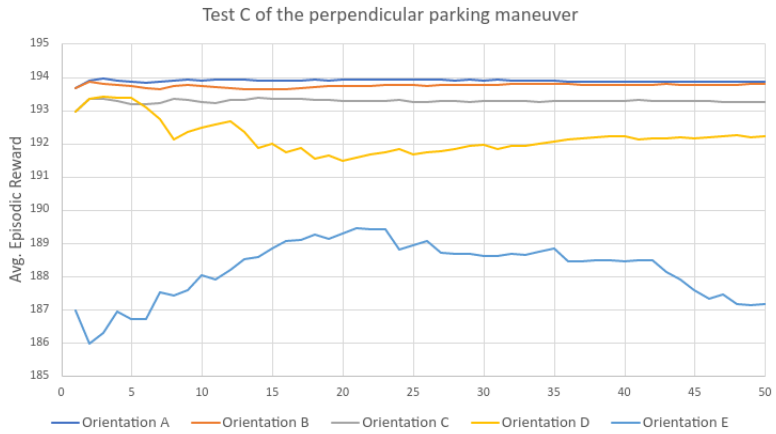


Figure 4.9: Agent's performance of the test C of the perpendicular parking maneuver

For the first test, orientation A was used and the agent was able to park with an average episodic reward of 193.9. For the second test, orientation B was used. The agent completed

all episodes with an average episodic reward of 193.8. As presented in figure 4.9, the average reward obtained in the first two tests was similar. This was expected since the agent already experienced those scenarios in the training phase. In the third test, orientation C was used. The agent completed all the episodes of this test with an average episodic reward of 193.2. Although this was a scenario that the agent never experienced before, still obtain a similar average reward from the scenarios in which it was trained, proving that the agent is not memorizing a movement and it can adapt. In the fourth test, orientation D was used. Although the average episodic reward is not as stable as the previously referred tests the agent was still able to complete all episodes of this test with an average reward of 192.2. In the last test, orientation E was used. In this test, the agent completed all episodes with an average episodic reward of 187.2, but as shown in figure 4.9, the average reward in this test was unstable presenting a great variation along with the 50 episodes. Orientations with a higher variation of orientation E were tested, but the agent was unable to park in those scenarios.

4.1.2.4 Test D

The goal of test D was to verify if the agent was able to park in different parking spots. Thus, the parking spot position and the obstacle position would randomly alternate between two possible positions. For comparison terms, two more tests were carried out, one for the left parking spot and other for the right parking spot. For these tests all variations in the agent's initial position used in the previous tests, were kept but no variations were added to the initial orientation of the agent. All three tests lasted 50 episodes and the progress is presented in figure 4.10.

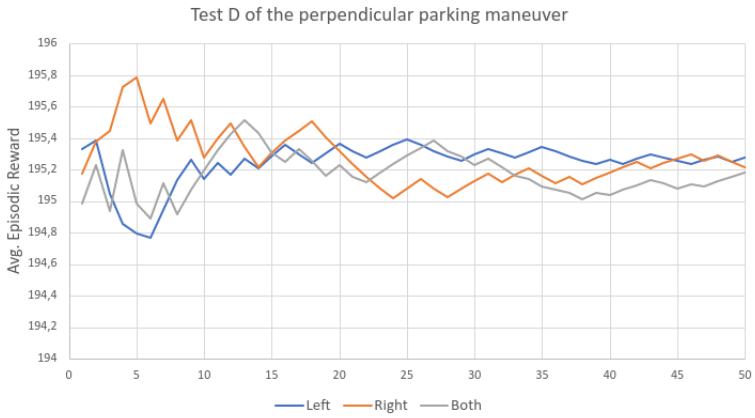


Figure 4.10: Agent's performance of the test D of the perpendicular parking maneuver

The first test was for the left parking spot. This was the scenario in which the agent was trained, so it was already expected that the agent completed all 50 episodes with success. As shown in figure 4.10, the average reward for the 50 episodes obtained by the agent in this test was 195.3.

The second test was for the right parking spot. Since the agent was not trained in this parking spot, this was a new scenario for the agent. The agent was able to adapt and completed all 50 episodes with an average episodic reward of 195.2, as shown in figure 4.10.

For the last test, the parking spot in which the agent had to park, would randomly change between the right and the left parking spots. Once again the agent was able to complete all 50 episodes with an average reward of 195.15, as shown in figure 4.10.

As presented in figure 4.10, the average episodic reward obtained by the agent in all three tests were similar, presenting only less than 0.15 of average reward in the new scenarios than the scenario in which it was trained. This demonstrates the agent did not memorize one movement and it is able to adapt to new situations.

4.1.3 Results Discussion

The four target approach turned out to be an inefficient method leading to unstable training and lack of convergence. As presented before, the agent was unable to learn a policy that enables it to perpendicular park in the simplest scenario of this maneuver.

Regarding the one target approach, the method revealed to be fluctuated, constantly learning and unlearning the maneuver throughout the majority of the phases of this training. The search for the most optimal hyper-parameters turned out to be extremely difficult. A great number of training for different sets of hyper-parameters was carried out, but only the presented set lead to a successful policy. Despite this, the agent was able to learn a policy that enable it to adapt to all the applied variations.

Concerning test A of the perpendicular parking maneuver, in figure 4.11, the agent's behaviour for this test is presented.

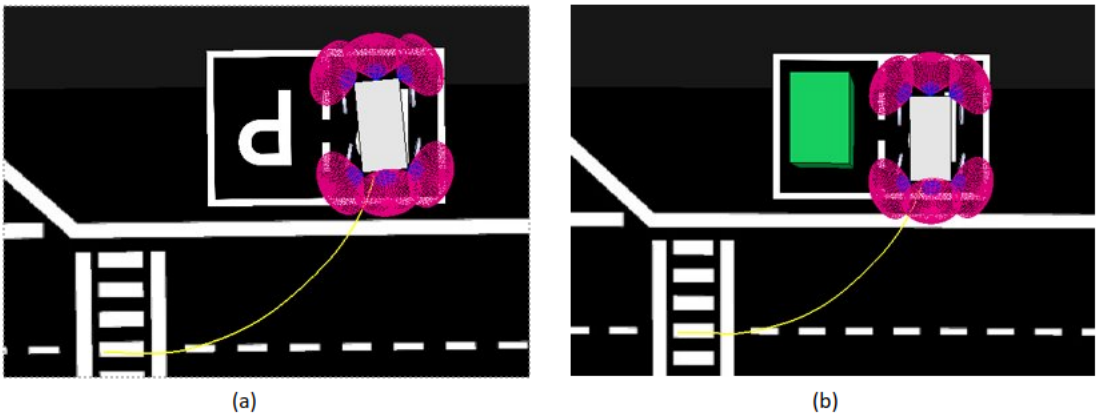


Figure 4.11: Agent's behaviour in the test A of the perpendicular parking maneuver. Scenario with obstacles (a) and scenario without obstacles (b)

Upon analysing figure 4.11, it is possible to observe the similarity in both scenarios. This was already predictable since the average episodic reward obtained by the agent, in both scenarios of test A, was extremely close. This proves that, for the perpendicular parking maneuver, the obstacles have almost no impact on the performance of the agent.

Concerning test B, the agent's behaviour for the 1 meter scenario of this test is presented in figure 4.12.

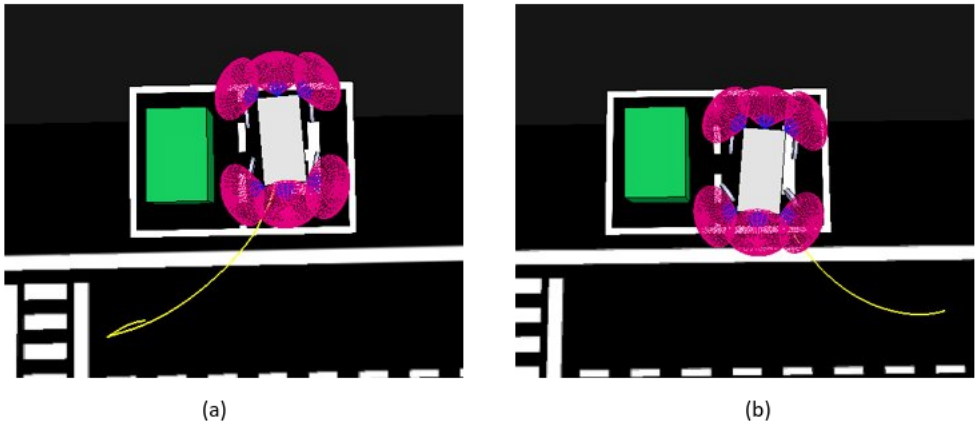


Figure 4.12: Agent's behaviour in the test B of the perpendicular parking maneuver. 1 meter distance at the left (a) and 1 meter distance at the right (b)

Upon analysing the agent behaviour presented in figure 4.12, it is noticeable that the agent learned different strategies for each side of the parking spot. In the scenario presented in figure 4.12a, the agent drives forward to gain some space and then reverse parks. For the scenario presented in figure 4.12b, the agent follows a different strategy, it swerves to the left to gain the necessary space to park.

The agent's behaviour in test A for the 2.5 meters distance is presented in figure 4.13

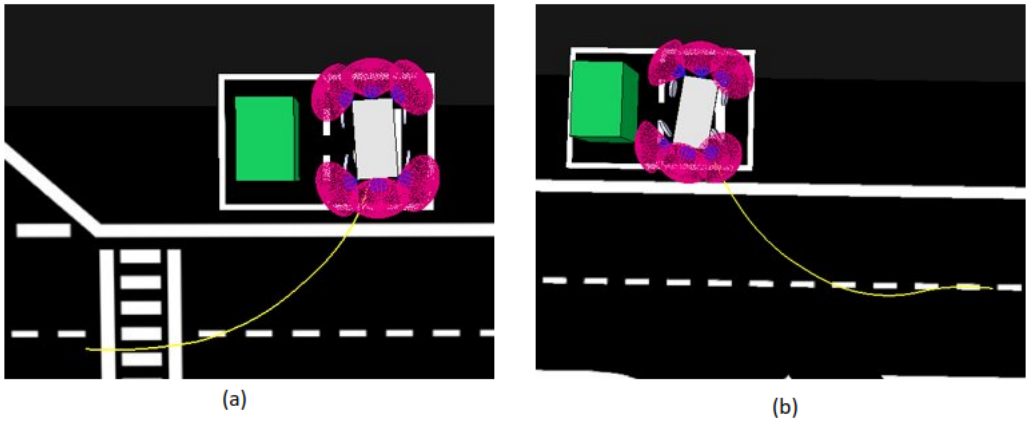


Figure 4.13: Agent's behaviour in the test B of the perpendicular parking maneuver. 2.5 meters distance at the left (a) and 2.5 meters distance at the right (b)

The agent's behaviour for the distances presented in figure 4.13, is similar to the behaviour presented in the 1 meter distance and all the other tested distances up to 2.5 meters. In the scenario presented in figure 4.13a, since it has space it does not need to do a forward drive correction but still performs a continuous left turn until is correctly positioned inside of the parking spot. In the scenario presented in figure 4.13b the agent continues to perform a left swerve to enter the parking spot with better orientation. It was also observed that the agent was able to end the maneuver in the centre of the parking spot of all distances at the left of the parking spot. On the contrary, for the scenarios tested at the right of the parking spot, the greater the distance, the more off-centre the agent was from the parking spot.

In figure 4.14, the agent behaviour for test C with orientations E are presented. Orientation E is the orientation with higher variation from which the agent is still able to park.

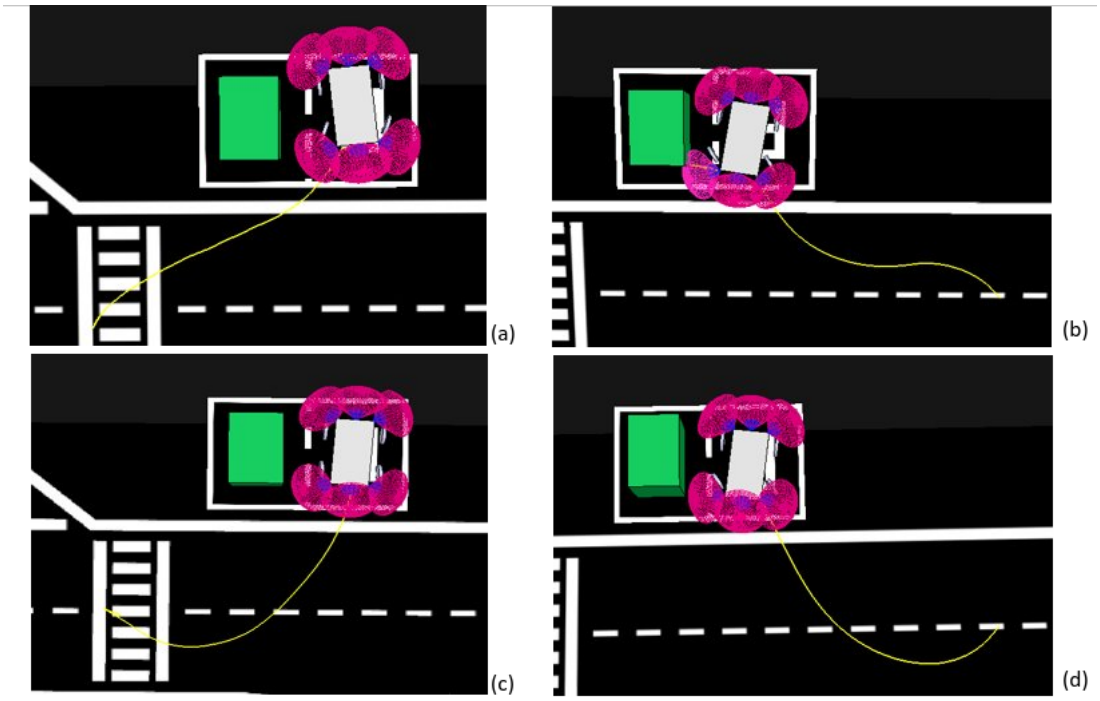


Figure 4.14: Agent's behaviour in the test C of the perpendicular parking maneuver. Orientation E for the left side (a) (c) and the right side (b) (d)

Upon analysing the agent's behaviour in the tests presented in figure 4.14a and 4.14b, it is perceptible that the agent had a more realistic movement and enter the parking spot with a better orientation in the scenario where it was placed at the right of the parking spot. In this scenario, for all tested orientations, the agent presents a final position more consistently centered in the parking spot. In the scenarios where the agent is placed at the left of the parking spot, the more negative it was the variation added to its initial orientation, the more off-centre the agent was from the parking spot.

The agent performs a similar movement in the scenarios where orientation E was added to the start orientation (orientation A). These scenarios are presented in figure 4.14c and 4.14d. For these scenarios, the agent continuously turns in the parking spot direction until it is correctly parked. Similar to the test B, the agent is able to end the maneuver correctly centred in the parking spot with the proper orientation for all orientations tested from the left side of the parking spot, but for the orientations tested from the right of the parking spot, with the increased of the variation added to the starting orientation (orientation A), the more off-centre the agent was from the parking spot.

In test D, the versatility and the adaptability of the agent are tested in a different parking spot. In figure 4.15, the behaviour of the agent in this test is presented.

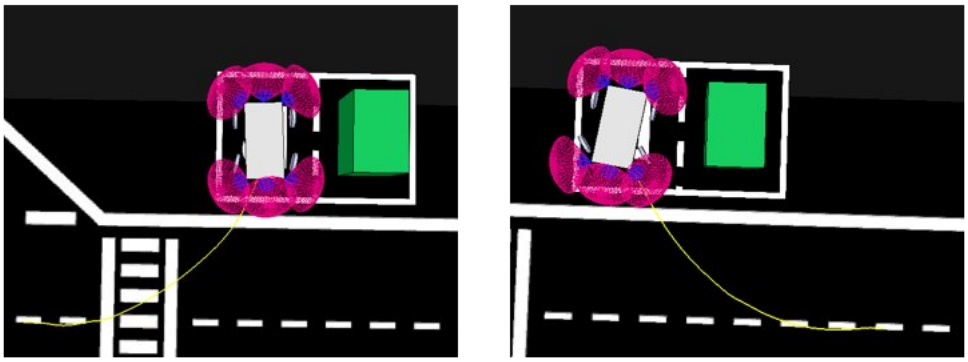


Figure 4.15: Agent's behaviour in the test D of the perpendicular parking maneuver for the left side (a) and for the right side (b)

In this test, the same behaviour presented in the previous tests is verified. Once again, in the scenario presented in figure 4.15a, the agent ends the maneuver correctly centred in the parking spot with an orientation extremely close to the ideal orientation. In the scenario presented in figure 4.15b, the agent is able to park but it presents a final position off-centre from the parking spot with a non-ideal orientation.

In conclusion, the agent is able to park for all situations presented in table 4.3. For the majority of the scenarios tested with a starting position at the left of the parking spot, the agent presents an almost ideal final position. For the scenarios tested with a starting position at the right of the parking spot, the agent demonstrates that it has more difficulties in the tests with higher variations, presenting a final position further off-centre to scenarios with higher variations added to the initial position. Due to the reward function implemented, the agent presents a lack of concern for the final orientation and position. This happens because, in the reward function, the maneuver was only accepted for an orientation error less than 10 degrees and a distance inferior to 5 cm, but there were no discounts implemented in the reward considering these deviations. As a consequence, the difference in the reward obtained by the agent for all the scenarios that respect the distance and orientation requirements was extremely small.

Table 4.3: Agent limitations for the perpendicular parking maneuver

Limitations	***Value***
Side of the parking spot	left and right
Range of the initial orientations	-40 degrees to 40 degrees
Maximum initial distance to the parking spot	2.5 meters

4.2 Angular Parking

4.2.1 Training

The angular parking training followed the same idea of the perpendicular parking training. To create a more versatile and robust system it was necessary to introduce variations in the agent's initial position and orientation so that the agent could learn how to adapt. Thus, the training of this maneuver was divided into four phases. The method of progressively increasing the training difficulty, used in the perpendicular parking training, was also used in this training. Thus, the agent must be able to complete the current phase of the training before it can pass to the next phase since the next phase is more challenging.

Like it was already discovered in the perpendicular parking training, transferring the learning from one phase to another is more harmful than helpful, because the agent starts all phases of the training from zero. From all phases of the training two sets of weights were withdrawn, the latest weights and the best performance weights. Depending on the situation both weights could be a good choice because, on one hand, the best performance weights registered the best performance in the training, but on the other hand, the latest weights experience more episodes, so it is more likely to be better at adapting to new situations.

4.2.1.1 First Training

In the first training of the angular parking, the agent was placed in the simplest version of this maneuver, where no variation was added to the agent's initial position and orientation. Thus, the agent was set in a fixed position at the left of the parking spot, as shown in figure 3.6. In this scenario, the obstacles were not placed. This training was implemented for the two approaches previously mentioned, the one target approach and the four target approach. The angular parking training for the one target approach lasted 2200 episodes and the progress is presented in figure

4.16(a). The angular parking training for the four target approach lasted 2500 episodes and the progress is presented in figure 4.16(b).

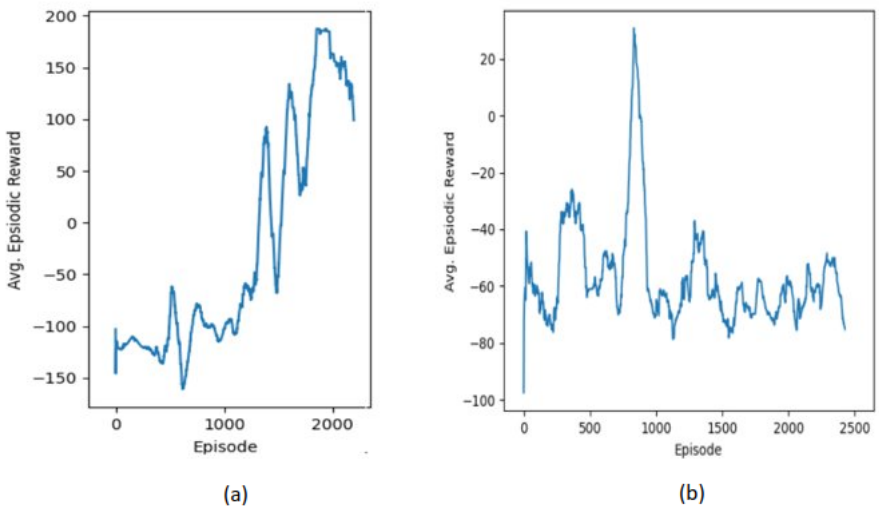


Figure 4.16: Agent's training progress for the first training of the angular parking challenge for the one target approach (a) and for the four target approach (b)

Regarding the four target approach, the agent did not learn a policy capable of successfully performing the angular parking maneuver. Throughout the training, the best performance was registered around episode 900 with an average episodic reward of 25. Since the agent was unable to learn a policy capable of parking in the simplest version of this maneuver, the remaining phases of the training were performed only using the one target approach.

Regarding the one target approach, as presented in figure 4.16(a), the progress of the average episodic reward is unstable. In this training, the agent learned and unlearned the movement two times before mastering it. This happened because the agent is still exploring the environment. Around episodes 1500 and 1800, this process of learning and unlearning the movement is noticeable. Despite this, around episode 2000, the agent reaches its best performance of this training, with an average episodic reward of 190. Until the end of the training the agent's performance decrease, ending the training with an average episodic reward of 100.

4.2.1.2 Second Training

In the second training the agent was placed at the left of the parking spot. A variation was introduced in the initial position of the agent. This could randomly vary the distance from the agent to the parking spot, from 1.2 meters to 2 meters. Lastly, two obstacles were placed on both sides of the parking spot as presented in figure 3.6. This training lasted 3000 episodes and the progress is presented in figure 4.17

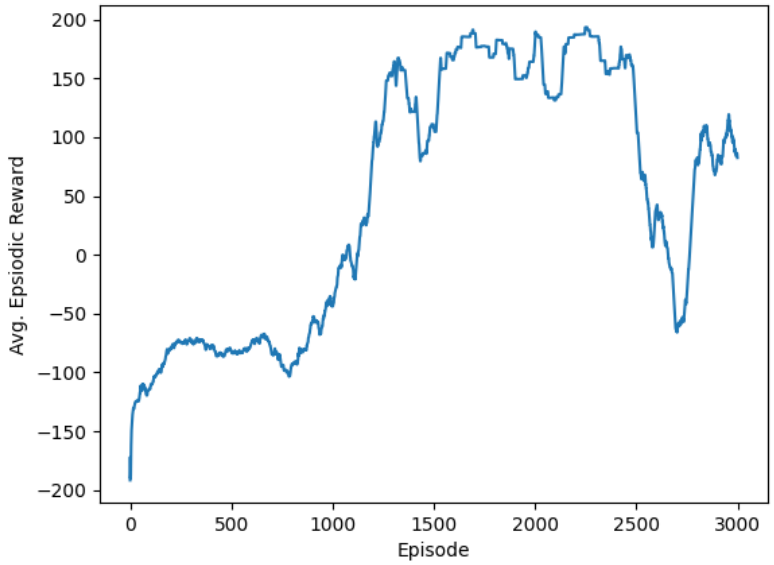


Figure 4.17: Agent's training progress for the second training of the angular parking challenge

In this training, the agent had already learned a policy capable of performing the angular parking maneuver before reaching episode 1500. The agent kept a similar performance until episode 2500 where it starts to unlearn the maneuver. The agent was able to learn it again, ending the training with an average episodic reward of 95. The best performance was registered in episode 2300, with an average episodic reward of 197.

4.2.1.3 Third Training

The goal of the third training was to teach the agent to park from both sides of the parking spot. Thus, the agent's initial position would randomly change between both sides of the parking spot

where small variations were also added. This would variate, the initial position of the agent, from 1.2 meters to 2 meters on both sides of the parking spot. Since the perpendicular parking training verified that the agent cannot adapt between parking and reverse parking depending on the agent's initial position, in this test the agent was rotated 180 degrees on the right side of the parking spot. Thus, the agent's initial orientation was set to 180 degrees in the left position and 0 degrees in the right position. Considering the parallel parking movement, the agent's final orientation also had to change, the left position was set to 135 degrees and the right position was set to 45 degrees. This training lasted 3200 episodes and the progress is presented in figure 4.18.

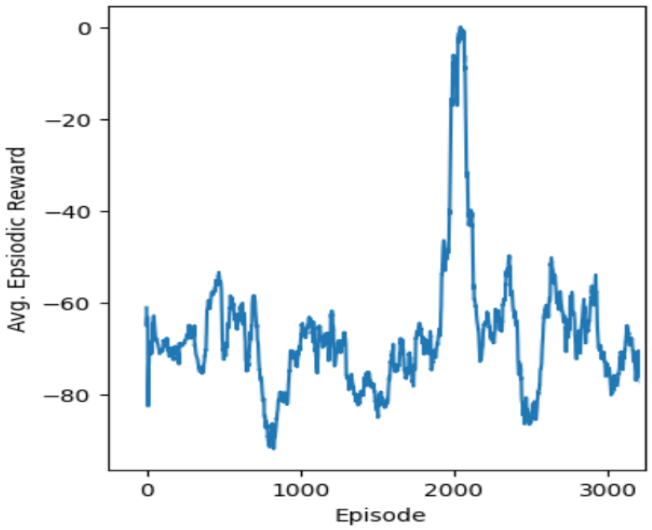


Figure 4.18: Agent's training progress for the third training of the angular parking challenge

As shown in figure 4.18, the agent reaches the best performance of this training in episode 2100 with an average episodic reward of 0.

Throughout all 3200 episodes, the agent did not learn a policy that could park from both sides of the parking spot. The agent was not able to adapt to different final orientations.

4.2.1.4 Fourth Training

Since the agent was not able to park from both sides of the parking spot, for the final training of the angular parking maneuver, the agent's initial positions would vary between 1.2 meters to 2

meters at the left of the parking spot. The initial orientation of the agent was set to 180 degrees and a variation of ± 10 degrees was added. This training had the goal of improving the adaptability of the agent to new situations. This training lasted 3000 episodes and the progress is presented in figure 4.19.

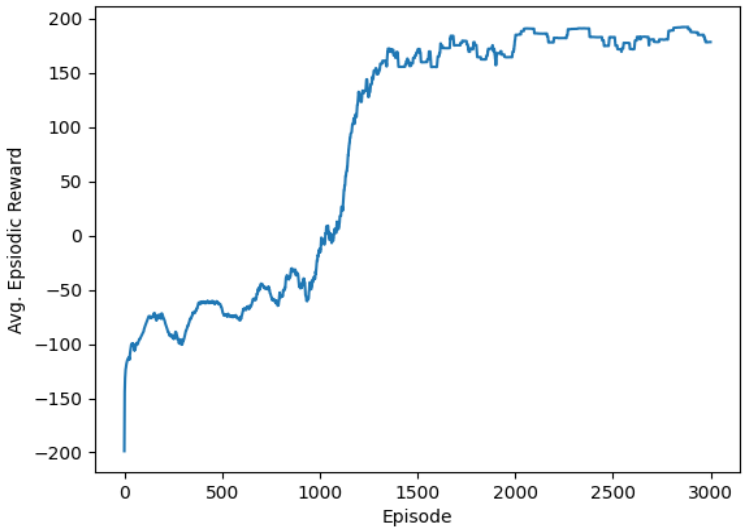


Figure 4.19: Agent's training progress for the final training of the angular parking challenge

The average episodic reward of all 3000 episodes is presented in figure 4.19. Upon analysing the progress, the agent consistently improved its performance throughout the training. The policy learned by the agent enabled it to adapt to all the variations added to the initial position and orientation. In episode 1500, the agent was capable of parking for all the different situations presented in this training and it kept a similar performance until the end of the training. In episode 2800 the agent reached the best performance of the training with an average episodic reward of 196. The agent ends the training with an average episodic reward of 188.

Although the average episodic reward obtained by the agent in episode 2800 is higher than the average episodic reward at the end of the training, the weights used in the tests presented below were withdrawn from the end of the training. For those weights, the agent experienced more episodes and still kept a similar performance, therefore, it is more likely to adapt better to new situations.

4.2.2 Tests

To demonstrate the adaptability of the agent to new situations and register the agent limitations, it was tested with unknown situations. In this section, all the tests carried out are presented, to test the limits of the agent relatively to variations in the agent's position and orientation.

4.2.2.1 Test A

In test A, the agent was placed at the left of the parking spot. The initial position of the agent could vary between 1.2 meters to 2 meters of the parking spot. The agent was tested in two scenarios, the scenario without obstacles and the scenario with obstacles. The goal was to determine the impact of the obstacle on the performance of the agent in the angular parking maneuver. This test lasted 50 episodes for each scenario and the progress is presented in figure 4.20.

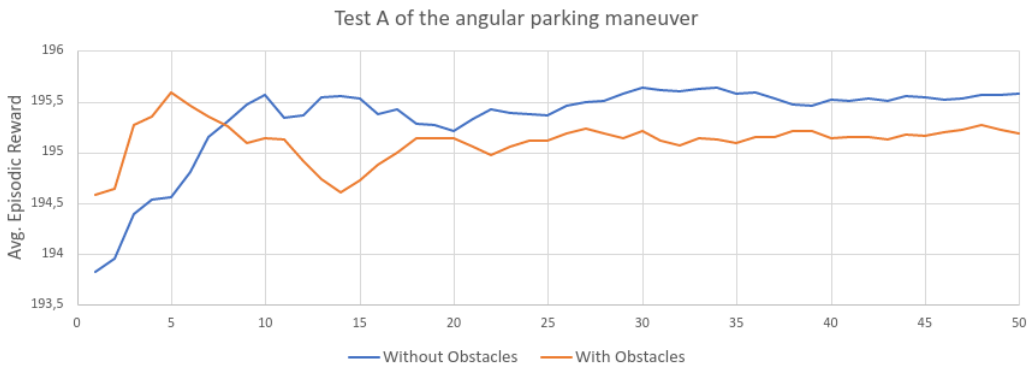


Figure 4.20: Agent's performance of the test A of the angular parking maneuver

As presented in figure 4.20, for the scenario without obstacles the agent obtained an average episodic reward of 195.6. In the scenario with obstacles the result was 195.2, showing a small difference in performance between both scenarios, but as expected, the agent performed better without the obstacles.

4.2.2.2 Test B

In test B, the agent was placed in a fixed position 1 meter at the left of the parking spot. Every time that the agent completed the test for the current distance, the distance was increased by

0.5 meters. The goal of this test was to evaluate a difference in performance between all tested distances and to determine the maximum distance between the agent and the parking spot for which the agent was still able to park. For every distance tested the test lasted 50 episodes and the progress is presented in figure 4.21.

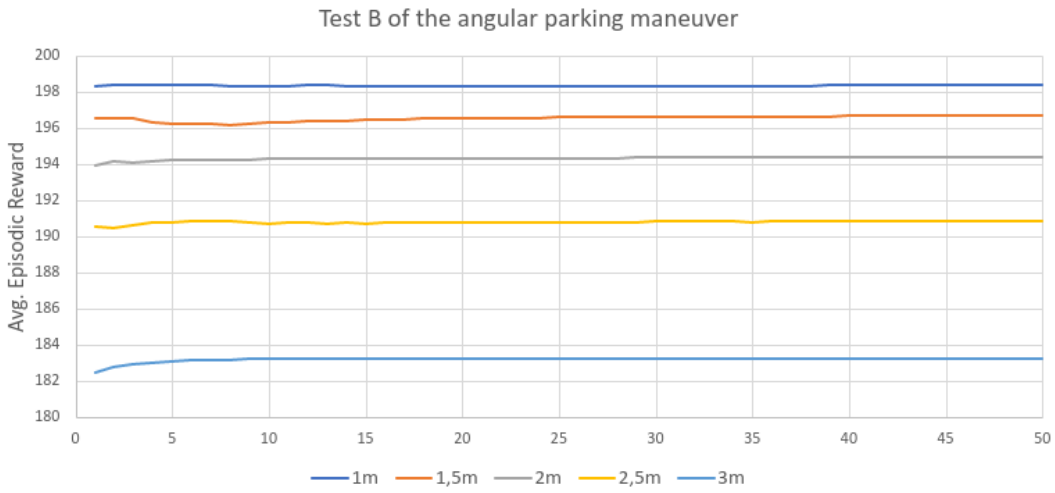


Figure 4.21: Agent's performance of the test B of the angular parking maneuver

The agent has completed all tests up to a distance of 3 meters. As presented in figure 4.21, the agent obtained an average episodic reward of 198.4 for the 1 meter distance, 196.45 for the 1.5 meters distance, 194.2 for the 2 meters distance 190.9 for the 2.5 meters distance and 183.3 for the 3 meters distance. The agent was also tested for distances higher than 3 meters but it was not able to successfully park in those scenarios.

4.2.2.3 Test C

In test C the agent was placed in a fixed position 1.5 meters at the left of the parking spot and no variations were added. The agent orientation was set at 180 degrees. Every time the agent completed the test the orientation was increased and decreased 10 degrees. All orientations used in this test are presented in table 4.4.

Table 4.4: Orientations used in test C

Orientations	***Value***	***Variation***
Orientation A	180°	0°
Orientation B	170° or 190°	± 10°
Orientation C	160° or 200°	± 20°
Orientation D	150° or 210°	± 30°
Orientation E	140° or 220°	± 40°
Orientation F	130° or 230°	± 50°
Orientation G	120° or 240°	± 60°
Orientation H	110° or 250°	± 70°
Orientation I	100° or 260°	± 80°
Orientation J	90° or 270°	± 90°

This test had two purposes. The first was to evaluate the difference in performance for a variety of initial orientations. The second was to determine until what angle of the agent's initial orientation would the agent still be able to park. For every orientation tested, the test lasted 50 episodes and the progress is presented in figure 4.22.

The agent has completed all tests up to an initial orientation of ± 50 degrees. As shown in figure 4.22, the agent got an average episodic reward of 196,35 for the initial orientation A. For the second test, orientation B was used. The agent completed all 50 episodes with an average reward of 196.3. In the third test, orientation C was used. The agent completed the test with an average reward of 196.28. For the fourth test, orientation D was used. The agent completed all episodes of the training and obtained an average episodic reward of 195.87. For the fifth test orientation E was used. The agent completed all 50 episodes with an average reward of 195.5. In the sixth test, orientation F was used. Although the agent completed all 50 episodes with an average episodic reward of 170.4, the performance was too unstable to be accepted. Lastly, orientation G was used but the agent was only able to complete the episodes where it was subtracted from the orientation. The same happened with orientations H, I and J. When tested with orientations with a higher variation than the orientation J, the agent was not able to complete any of the episodes.

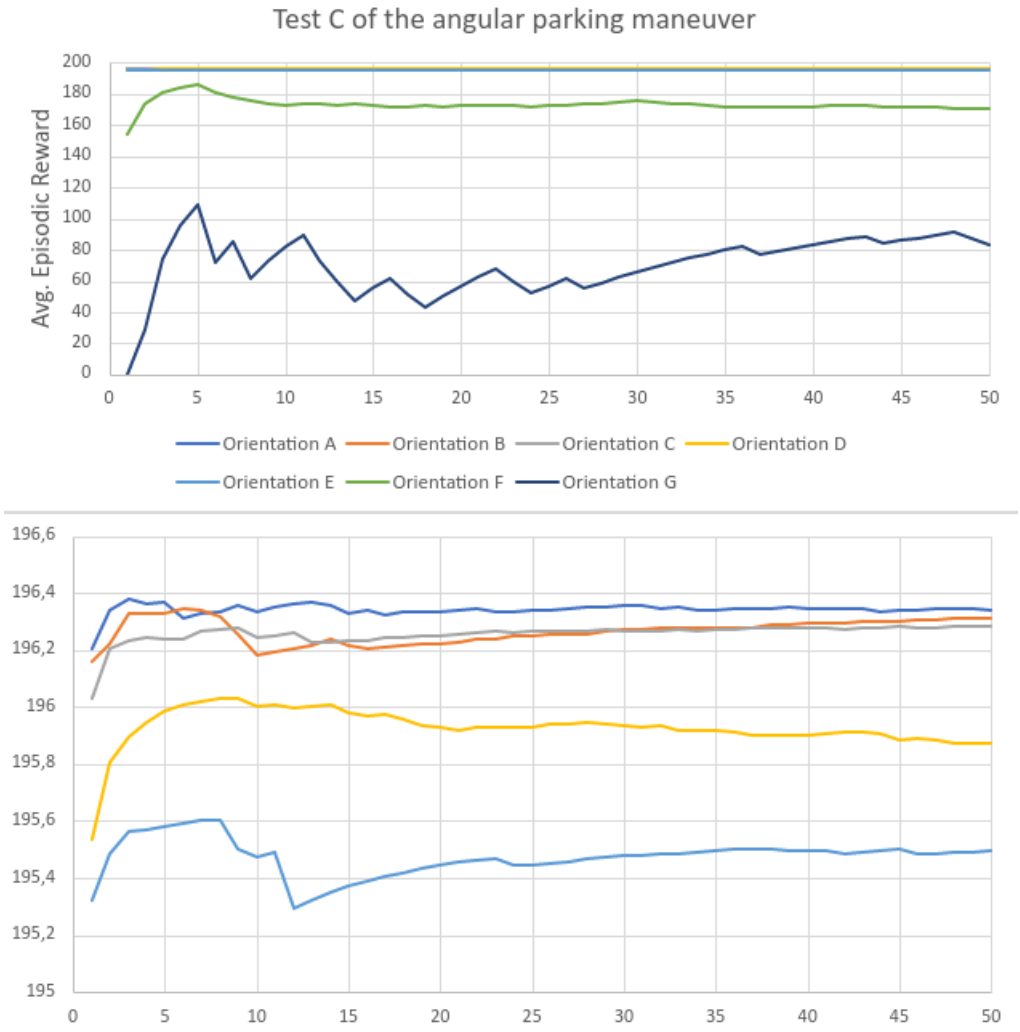


Figure 4.22: Agent's performance of the test C of the angular parking maneuver

4.2.3 Results Discussion

The four target approach revealed to be an inefficient method leading to unstable training and lack of convergence. As presented before, the agent was unable to learn a policy that enables it to angular park in the simplest scenario of this maneuver.

Regarding the one target approach, the search for the most optimal hyper-parameters turned out to be extremely difficult. A great number of training for different sets of hyper-parameters was carried out but only the presented set of hyper-parameters lead to a successful policy. Despite this, the agent was able to learn a policy that enabled it to park for a variety of different scenarios. The learning progress for all the training phases turned out to be more stable with the increase of the variations added.

Concerning test A, in figure 4.24, the agent's behaviour performed in this test is presented.

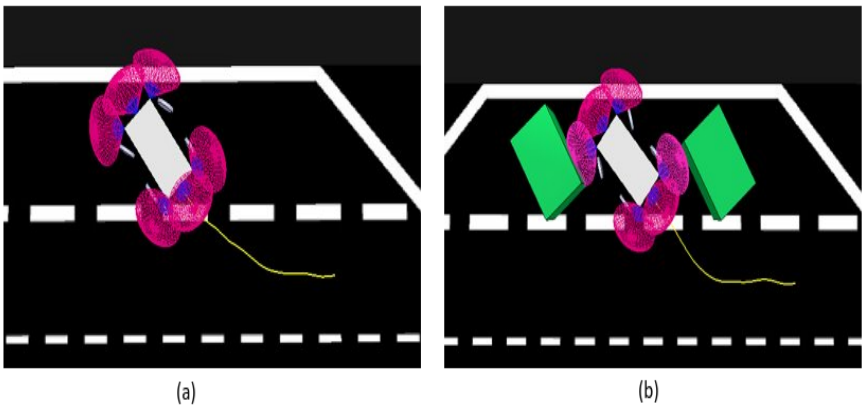


Figure 4.23: Agent's behaviour in the test A of the angular parking maneuver for the scenario without obstacles (a) and for the scenario with obstacles (b)

Since the average episodic reward obtained by the agent in test A was relatively close, it was already expected that the agent presented a similar behaviour for both scenarios. Although the behaviour is similar, upon analysing the agent's behaviour, it was verified that the obstacles have a small impact on the agent movement. For the scenario without obstacles presented in figure 4.23a, the agent turned right sooner and drives straight to the parking spot. On the other hand, for the scenario with obstacles presented in figure 4.23b, the agent went ahead longer to avoid the obstacle, made a sharper right turn and went straight to the parking spot. The obstacles did not appear to have a great impact on the agent's performance because in this scenario the obstacles were 90 cm apart. When tested in scenarios with closer obstacles, the agent was unable to park. This happened because the agent was not trained in scenarios with obstacles that close and the penalty for the collision was too high. Thus, the agent learned a policy that would make it move away when an obstacle was detected. This policy worked for scenarios in which the distance between obstacles was equal to or greater than 90 cm because the agent had the necessary space to complete the maneuver regardless.

Regarding the test B, in figure 4.24, the agent's behaviour performed in this test is presented.

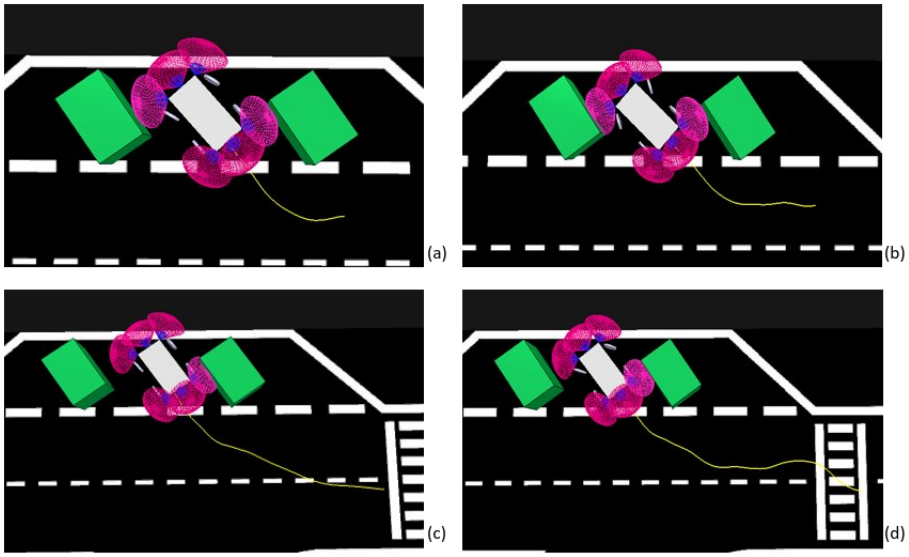


Figure 4.24: Agent's behaviour in the test B of the angular parking maneuver. 1 meter distance (a), 1.5 meter distance (b), 2.5 meter distance (c) and 3 meter distance (d)

In figures 4.24a and 4.24b it can be observed that the agent presents a similar strategy to the one used in the perpendicular parking maneuver. The agent performs a left swerve to enter the parking spot with a more ideal orientation. The agent's performance decreases for higher distances, the movement becomes unstable with a lot of fluctuations in the direction. Despite this, the agent can park up to a distance of 3 meters.

Regarding test C, in figure 4.25, the agent's behaviour performed in this test is presented.

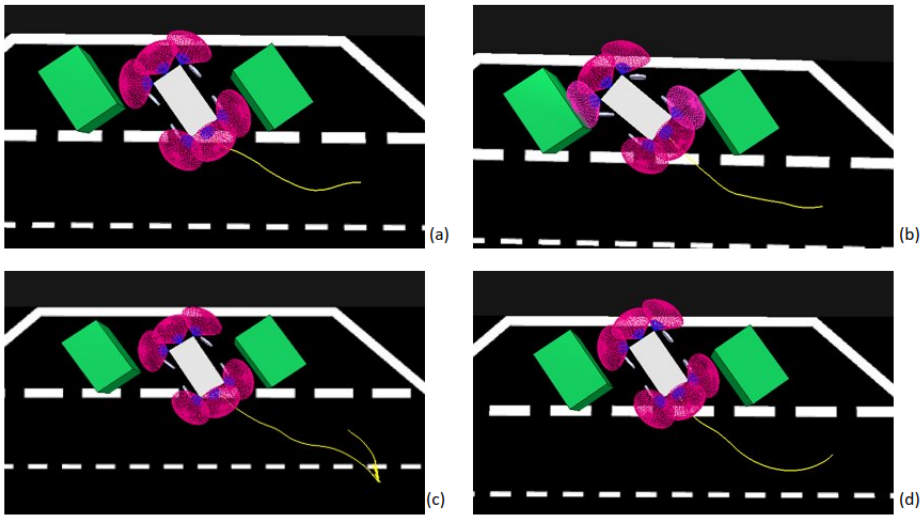


Figure 4.25: Agent's behaviour in the test C of the angular parking maneuver. -10 degrees orientation (a), 10 degrees orientation (b), -50 degrees orientation (c) and 50 degrees orientation (d)

For all tested orientations in the range of 140 degrees to 230 degrees, the agent's behaviour is similar. The agent drives towards the parking spot while keeping a safe distance from obstacles. The more off-centre the agent is from the 235 ($45^{\circ} + \pi$) degrees orientation the greater is the correction but the strategy applied is similar for those scenarios. In the scenario tested for the orientation 230 degrees presented in figure 4.25c, the agent does not have the necessary space to complete the maneuver but the agent was able to adapt and reversed drove until it had the necessary space to perform the maneuver safely.

In conclusion, the agent learned to park for all scenarios presented in table 4.5, but with the increased of variations, the agent's behaviour becomes increasingly unstable with greater fluctuations in the steering. The agent was not able to learn a policy that enable it to park from both sides of the parking spot. Similarly to the perpendicular parking maneuver, in this maneuver, with higher variations, an increase in the difference between the ideal final orientation and the agent's orientation at the end of the maneuver was verified. The reason for this was already explained in the perpendicular parking maneuver. Despite this, the agent was able to learn a policy that enabled it to park for the trained scenarios and to adapt to new scenarios.

Table 4.5: Agent limitations for the angular parking maneuver

Limitations	***Value***
Side of the parking spot	right
Range of the initial orientations	130 degrees to 230 degrees
Maximum initial distance to the parking spot	3 meters
Minimum distance between obstacles	90 centimeters

4.3 Parallel Parking

4.3.1 Training

In the parallel parking, the difficulty of the training was progressively increased. Similarly to the training of the perpendicular and angular parking maneuvers.

To create a versatile and robust system, throughout the training, random variations were added to the agent's initial position and orientation. Thus, this maneuver training was divided into

four phases. Like, it was already discovered in the perpendicular parking training, transferring the learning from one phase to another would harm the performance of the agent, because the agent start all phases of this training from zero. For this training two sets of weights were withdrawn for all the training phases described ahead. The first set of weights were withdrawn in the best performance of the episode. The second set of weights were withdrawn at the end of the episode.

4.3.1.1 First Training

For the first training of the parallel parking, the simplest version of the maneuver was used. The agent was placed in a fixed position at the left of the parking spot. There were no variations added to the agent's initial position and orientation. In this scenario the obstacles were not placed. For this training it was registered the progress for the two implemented approaches. For the one target approach, the training lasted 3200 episodes and the progress is presented in figure 4.26(a). For the four target approach, the training lasted 3000 episodes and the progress is presented in figure 4.26(b).

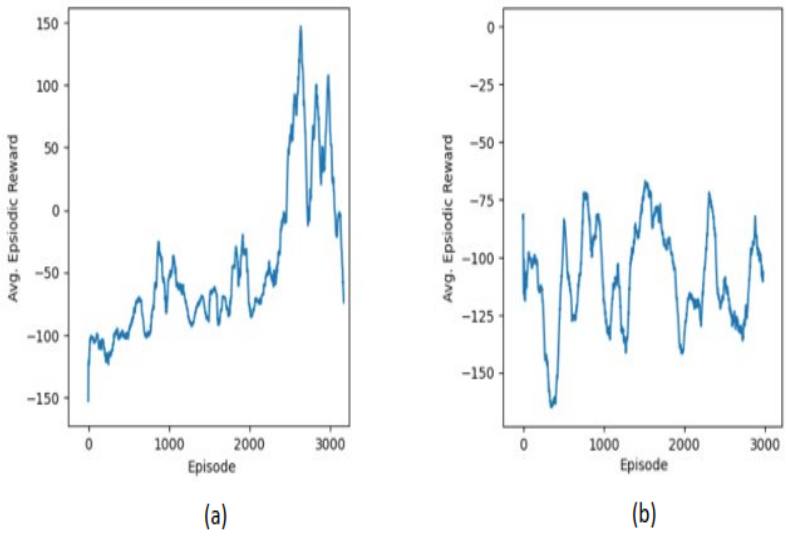


Figure 4.26: Agent's training progress for the first training of the parallel parking challenge for the one target approach (a) and for the four target approach (b)

Regarding the four target approach, similarly to the perpendicular and angular parking, the agent did not learn a policy that enabled it to parallel park. Since the agent did not learn how

to park in the simplest version of this scenario, the remaining phases of the training were only performed for the one target approach.

Regarding the one target approach, as presented in figure 4.26 the progress of the average episodic reward is unstable. In this training, the agent learned the movement, obtaining an average episodic reward of 150 at episode 2600. Until the end of the episode the agent unlearns the maneuver ending the training with an average episodic reward of -75. This happened because the agent is still exploring the environment.

4.3.1.2 Second Training

In the second training, the agent was placed at the left of the parking spot. A variation was introduced to the initial position of the agent, which varied its initial position from 1.2 meters to 2 meters. Lastly, two obstacles were placed on both sides of the parking spot as presented in figure 3.5. This training lasted 3000 episodes and the progress is presented in figure 4.27.

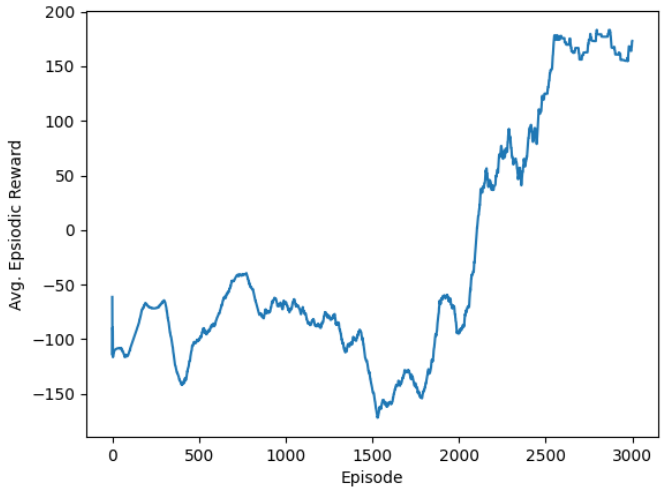


Figure 4.27: Agent's training progress for the second training of the parallel parking challenge

As presented in figure 4.27, in episode 2600 the agent learned a policy capable of performing the angular parking maneuver for the situations presented and it kept a similar performance until the end of the episode. After 2800 episodes, the agent reaches its best performance with an average episodic reward of 185 and ends the training with an average reward of 175.

4.3.1.3 Third Training

The goal of the third training was to teach the agent to park from both sides of the parking spot. Thus, the agent's initial position would randomly change between both sides of the parking spot where small variations were also added. Since the perpendicular parking training verified that the agent cannot adapt between parking and reverse parking depending on the agent's initial position, in this test the agent was rotated 180 degrees on the right side of the parking spot. Thus, the agent's initial orientation was set to 180 degrees in the left position and 0 degrees in the right position. Considering the parallel parking movement, the agent's final orientation also had to change, the left position was set to 180 degrees and the right position was set to 0 degrees. The training lasted 3000 episodes and the progress is presented in figure 4.28.

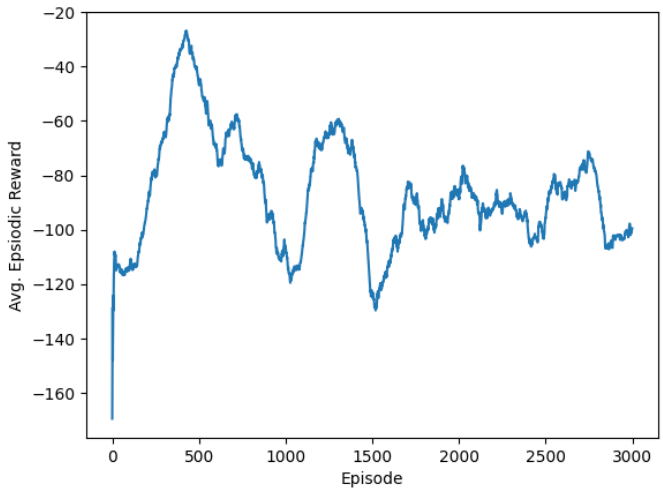


Figure 4.28: Agent's training progress for the third training of the parallel parking challenge

Throughout the training, the agent was unable to learn a policy capable of performing the parallel parking maneuver for both sides of the parking spot. Similarly to the angular parking training, the agent could not adapt to different final orientations. In episode 500, the best performance of the agent was registered with an average episodic reward of -25. The agent ends the training with an average episodic reward of -97.

4.3.1.4 Fourth Training

Since the agent was unable to park from both sides of the parking spot, in the final training of the parallel parking maneuver, the agent was placed at the left of the parking spot. A variation was added to the initial position of the agent which made it vary between 1.2 meters to 2 meters from the parking spot. The agent initial orientation was set to 180 degrees and a variation of -10 degrees to 10 degrees was randomly added every episode. The goal of this training was to enable the agent to parallel park in diverse situations and to improve the adaptability of the agent to new situations. This training lasted 3000 episodes and the progress is presented in figure 4.29.

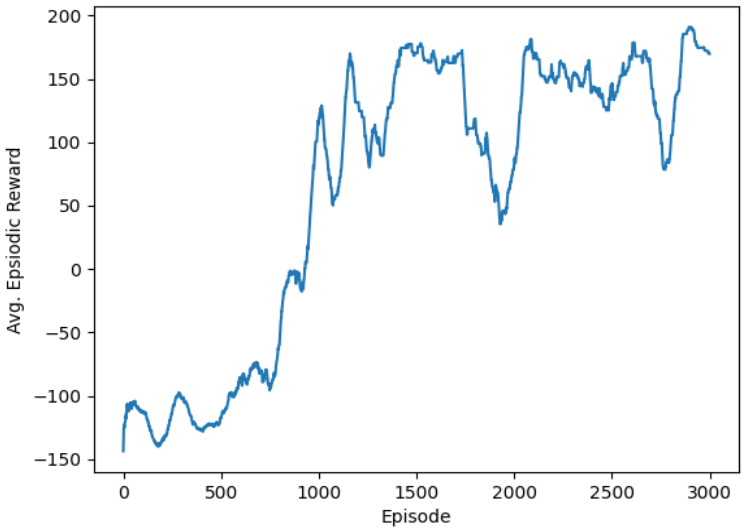


Figure 4.29: Agent's training progress for the final training of the parallel parking challenge

In figure 4.29, the average episodic reward of all 3000 episodes is presented. The agent learned a policy that enabled it to adapt to all the variations added to the initial position and orientation. Upon analysing the progress of the agent in this training, it is verified that the agent learned to park early in the training, archiving an average episodic reward higher than 100 at episode 1100. Due to the exploration, around episodes 1250, 1800 and 2700 a few declines in performance are verified. Despite this, the agent was able to improve its performance again ending the training with an average episodic reward of 175. In episode 2900, the best performance of the training was registered with an average episodic reward of 190.

In this training, two sets of weights were withdrawn. The first set was withdrawn at episode 2900 where it was registered the best performance of the training. The second was withdrawn at the end of the training. Since the performance from the two sets of weights is similar, in the tests presented below were used the set of weights withdrawn at the end of the episode. For those weights, the agent experienced more episodes and still kept a similar performance, therefore it is more likely to be better at adapting to new situations.

4.3.2 Test

The agent was tested with unknown scenarios, to demonstrate the adaptability of the agent and register the limitations. In this section, all the tests performed are presented, to test the limits of the agent relative to variations in the agent's position and orientation.

4.3.2.1 Test A

For test A, the agent was randomly placed between 1.2 meters and 2 meters at the left of the parking spot. The agent was tested in two scenarios. In the first scenario, no obstacles were added. In the second scenario two obstacles were placed, one on each side of the parking spot. The goal was to determine the impact of the obstacle on the performance of the agent in the angular parking maneuver. This test lasted 50 episodes for each scenario and the progress is presented in figure 4.30.

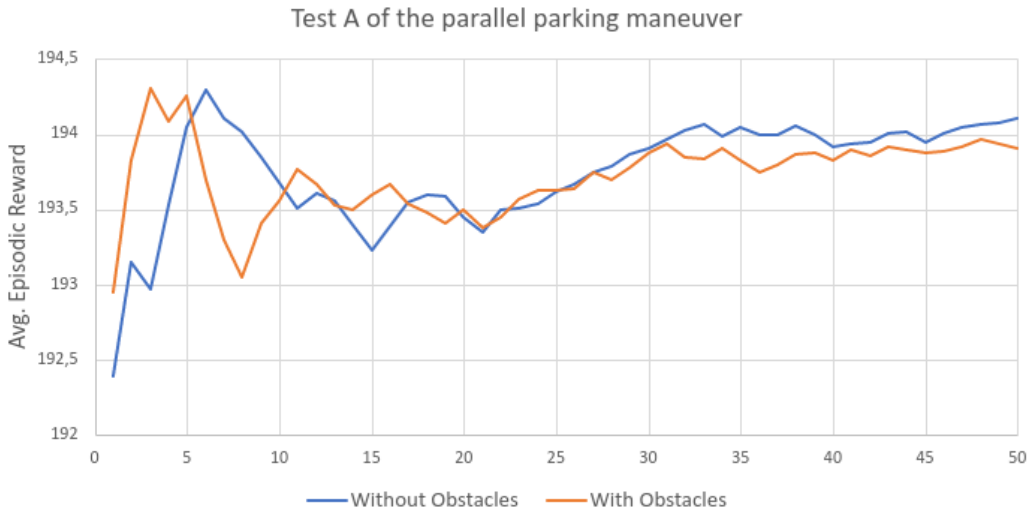


Figure 4.30: Agent's performance of the test A of the parallel parking maneuver

For the scenario without obstacles, the agent archived an average episodic reward of 194.1. For the scenario with obstacles, the agent obtained an average episodic reward of 193.9. Although in the scenario without obstacles the performance is higher, the difference in performance between both scenarios is small. Showing that, in this scenario, the obstacles have a small impact on the agent's performance for this maneuver.

4.3.2.2 Test B

In test B, the agent was placed in a fixed position 1 meter at the left of the parking spot. Every time the agent completed the test for the current distance, the distance of the agent to the parking spot was increased by 0.5 meters. The goal of this test was to evaluate a difference in performance between all tested distances and to determine the maximum distance between the agent and the parking spot for which the agent was still able to park. For every distance tested, the test lasted 50 episodes and in figure 4.31 the progress is presented.

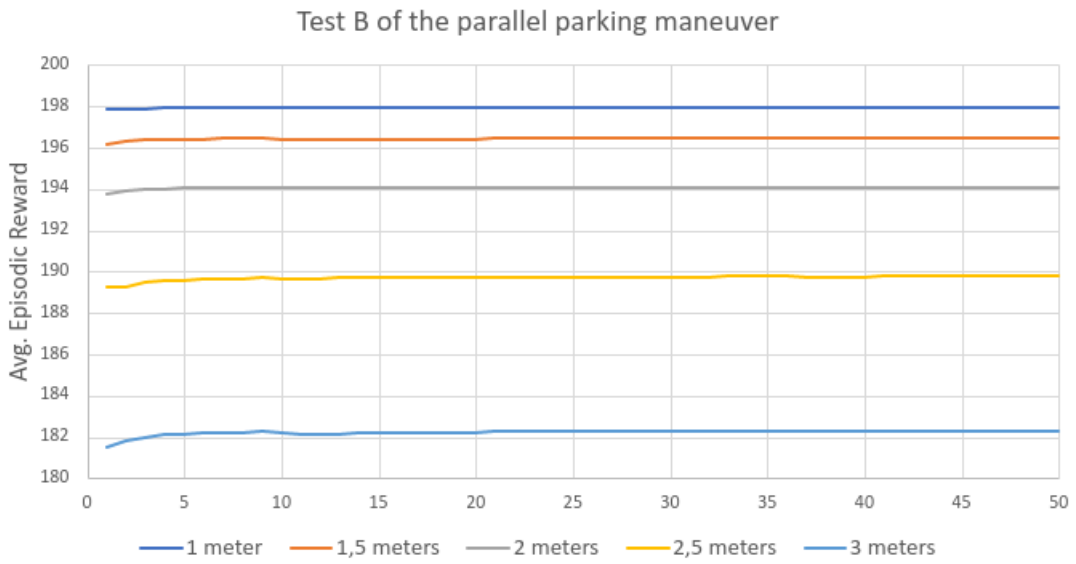


Figure 4.31: Agent's performance of the test B of the parallel parking maneuver

The agent has completed all tests up to a distance of 3 meters. As shown in figure 4.31, the agent archived an average episodic reward of 197.95 for the 1 meter distance, 196.5 for the 1.5

meters distance, 194.1 for the 2 meters distance, 189.8 for the 2.5 meters distance and 182.3 for the 3 meters distance. Lastly, the agent was also tested for the 3.5 meters distance but despite completing all 50 episodes, the movement was too unstable to be accepted. Upon analysing the performance of the agent in this test, it is noticeable that the performance decreases with the distance. This happens because the average episodic reward is influenced by how fast the agent can perform the maneuver. Since, as expected, higher distances require more steps and time to complete the maneuver the performance decreases.

4.3.2.3 Test C

In test C the agent was placed in a fixed position 1.5 meters left of the parking spot. To the agent's initial position, no variations were added. The orientation of the agent was set at 180 degrees. Every time the agent completed the test the orientation was increased and decreased by 10 degrees. All orientations used in this test are presented in table 4.6.

Table 4.6: Orientations used in test C

Orientations	***Value***	***Variation***
Orientation A	180°	0°
Orientation B	170° or 190°	± 10°
Orientation C	160° or 200°	± 20°
Orientation D	150° or 210°	± 30°
Orientation E	140° or 220°	± 40°
Orientation F	130° or 230°	± 50°
Orientation G	120° or 240°	± 60°
Orientation H	110° or 250°	± 70°

The goal of this test was to evaluate the difference in performance for different initial orientations and to determine until what angle of the agent's initial orientation would the agent still be able to park. For every orientation tested, the test lasted 50 episodes and in figure 4.32, the progress is presented.

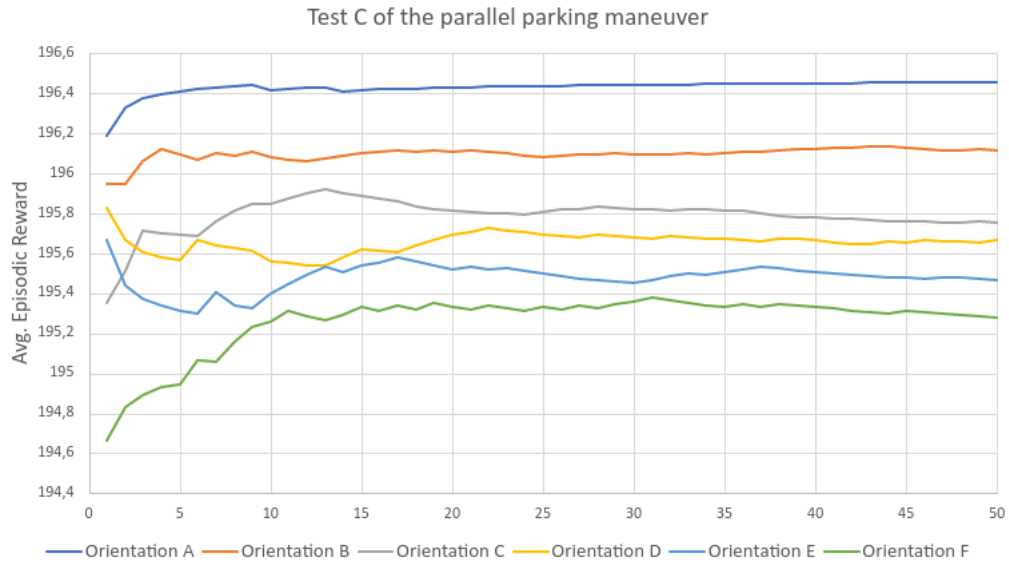


Figure 4.32: Agent's performance of the test C of the parallel parking maneuver

The agent completed all tests up to an initial orientation of $\pm$ 50 degrees. As shown in figure 4.32, for the initial orientation A, the agent got an average episodic reward of 196.5. For the second test, orientation B was used and the agent got an average episodic reward of 196.1. For the third, orientation C was used and the agent obtained an average episodic reward of 195.75. For the fourth test orientation D was used and the agent obtained an average episodic reward of 195.66. For the fifth test, orientation E was used and the agent got an average episodic reward of 195.46. Lastly, for the fifth test, orientation F and the agent got an average episodic reward of 195.27. The agent was also tested for orientations G and H but the agent was unable to complete these scenarios. As can be seen in the figure 4.32, the agent's performance decrease for orientations with higher variations.

4.3.3 Results Discussion

The four target approach turned out to be an inefficient and unreliable learning method. Throughout the whole training of the agent in the simplest version of the parallel parking training, for all tested hyper-parameters, it was unable to learn a policy capable to park.

Regarding the one target approach, the agent was able to parallel park in a great variety of different situations from the left side of the parking spot, only failing in learning a policy capable of adapting its movement from both sides of the parking spot.

Concerning test A of this maneuver, in figure 4.33 the agent's behaviour for this test is presented.

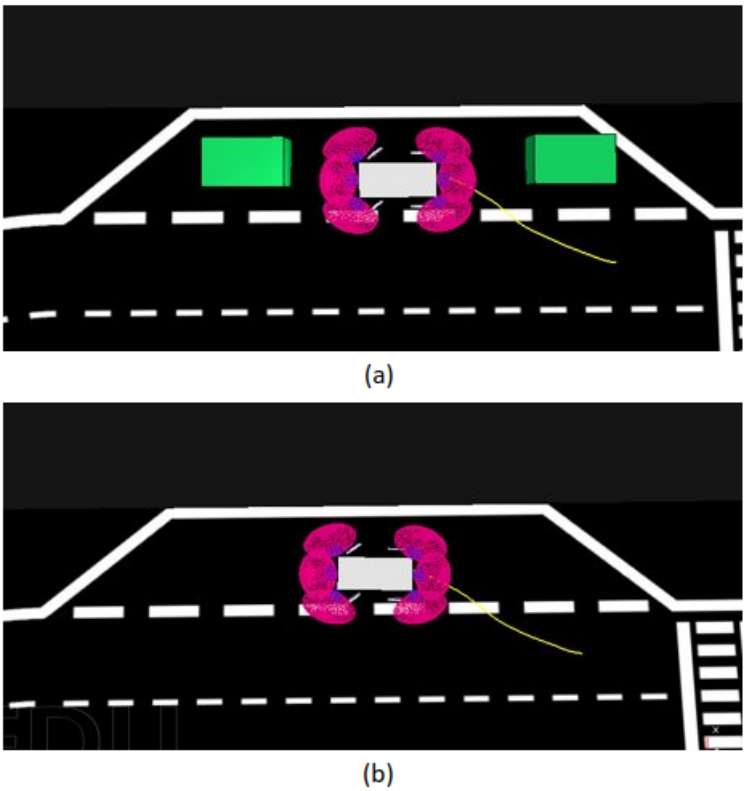


Figure 4.33: Agent's behaviour in the test A of the parallel parking maneuver. Scenario with obstacles (a) and scenario without obstacles (b)

As seen in figure 4.33, the agent presented a similar movement for both scenarios. As expected, since the agent archived a similar average reward in both scenarios. Although, the performance is similar, parallel parking is easier without obstacles. This was verified in the analysis of tests B and C because in some scenarios the agent did not have the necessary space to perform the maneuver, due to the obstacles and it could not adapt.

Concerning test B of this maneuver, in figure 4.34 the agent's behaviour for this test is presented.

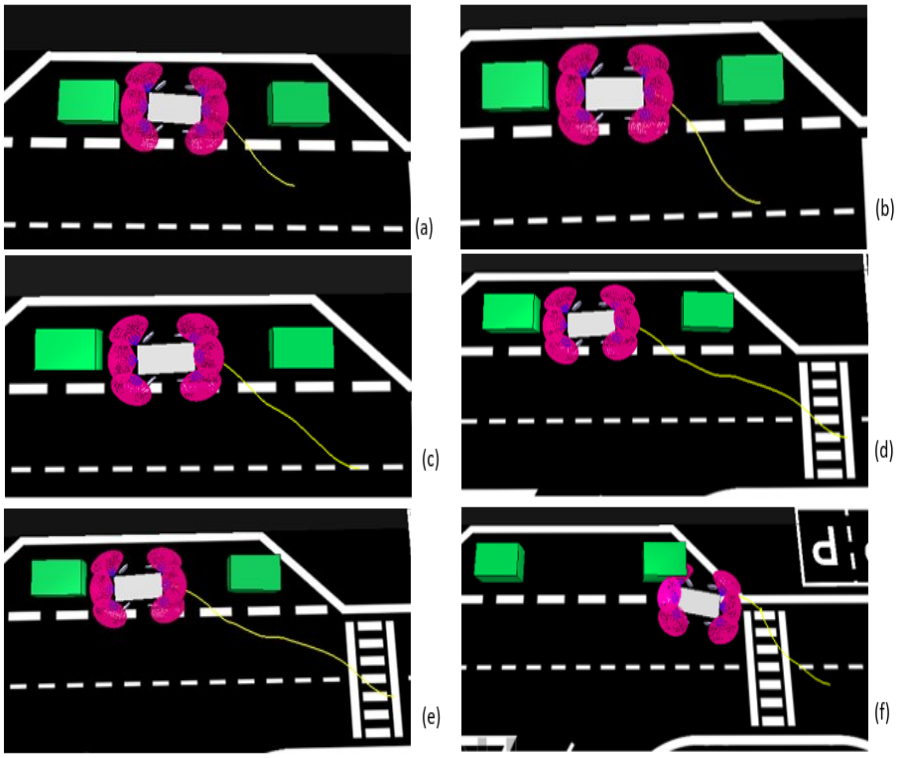


Figure 4.34: Agent's behaviour in the test B of the parallel parking maneuver. 1 meter distance (a), 1.5 meters distance (b), 2 meters distance (c), 2.5 distance meters (d), 3 meters distance (e) and 3.5 meters distance (f)

Upon analysing all the tests presented in figure 4.34, it is noticeable that the policy learned by the agent consists of driving in the direction of the parking spot. When the right front sensor detects the obstacle, the agent turns right to position itself inside of the parking spot. After the right back sensor detects the obstacle, the agent turns left until reaching a horizontal position. For distance between 2 and 3 meters the agent detects the obstacle earlier and has to do a correction to avoid the obstacle. This policy enables the agent to park up to a distance of 3 meters, but when tested for higher distances, the agent collides with the first obstacle when driving in the direction of the parking spot.

Concerning test C, figure 4.35 show's the agent's behaviour for the orientations 170 degrees, 190 degrees, 140 degrees and 220 degrees.

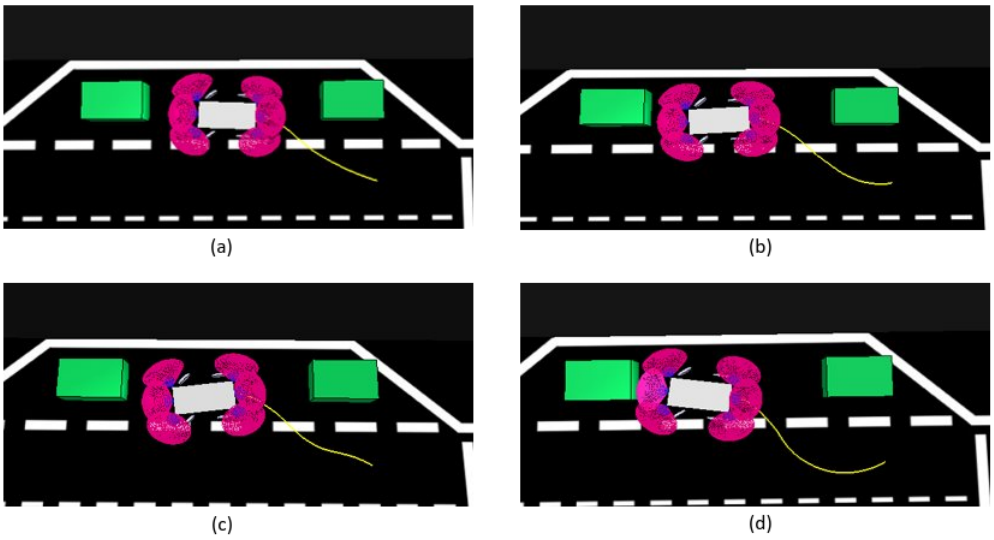


Figure 4.35: Agent's behaviour in the test C of the parallel parking maneuver. (a) -10 degrees orientation,(b) 10 degrees orientation, (c) -40 degrees orientation and (d) 40 degrees orientation

Regarding the 170 degrees orientations, the agent presents a similar behaviour to the previous tests, where it uses the first obstacle as reference. For the 140 degrees orientation, the agent detects the obstacle earlier and is able to adapt by doing a left turn correction to avoid the obstacle. For the orientation of 40 degrees the agent, due to physical limitations, does not turn enough to detect the first obstacle but is still able to park. Although the agent can park up to orientations of 180 $\pm$ 50 degrees, it was verified that the higher the initial orientation, the closer to the second obstacle the agent parks, not having the necessary space to park for orientations with a greater variation than $\pm$ 50 degrees. Thus, it was concluded that the higher the variation in the initial orientation of the agent the more impact the position of the obstacles have on the agent's performance.

In conclusion, for the four target approach, despite the number of trainings carried out, the agent did not learn a policy capable to park. For the one target approach, the agent learned a policy that enabled it to parallel park for a great variety of different scenarios but it was not able to park in a scenario where the distance between the obstacles is lower than 1.5 meters. In table 4.7, all the different situations in which the agent is able to park are presented.

Table 4.7: Agent limitations for the parallel parking maneuver

Limitations	***Value***
Side of the parking spot	left
Range of the initial orientations	130 degrees to 230 degrees
Maximum initial distance to the parking spot	3 meters
Minimum distance between obstacles	1.5 meters

Chapter 5

Conclusions and Future Work

The work presented in this dissertation describes the development of a Reinforcement Learning system capable to park an autonomously driven vehicle in the three most common parking types, perpendicular, angular and parallel.

In the first part of this dissertation, a review of the state of the art in the area of Reinforcement Learning and Deep Reinforcement Learning applied to the autonomous parking problem is presented. The theoretical concepts of Artificial Intelligence more focused on Reinforcement Learning are also presented with the goal of better explaining the Deep Deterministic Policy Gradient algorithm.

The second part of this dissertation focus on practical implementations. Initially, it describes all the implementations carried out in CoppeliaSim, where it explains the construction of the agent and its limitations regarding speed and steering. Then, a brief explanation of ROS theoretical concepts and the implementation of the ROS architecture carried out in this project to ensure the correct Communication between Pycharm and CoppeliaSim is presented. Lastly, all the implementations regarding DDPG are explained. It presents the action-space and the state-space of the environment. A deeper look at the architecture of DDPG is also presented. This part of the dissertation ends with an extensive explanation of the two approaches implemented in this project and their respective reward function.

The third part of this dissertation is divided into three sections of each parking maneuver. The first section consists of the final training of the agent and all the intermediate phases that lead to it. From this training, two sets of weights were withdrawn and one was chosen to be used in the tests. In this section, an explanation for this choice is presented. The second section is composed of all the tests to which the agent was subjected to study the agent's limits to diverse situations.

Lastly, the third section presents the discussion of the results, where the agent's behaviour for all the tests is commented.

For all parking maneuvers the four target approach revealed to be an inefficient method from which the agent was unable to learn a policy capable of performing any of the trained parking maneuvers even for the simplest scenarios.

The one target approach, although in some scenarios lead to an oscillatory training constantly learning and unlearning the maneuver for the intermediate phases of the training. In the final version of the training, for each one of the parking maneuver, the method turned out to be stable, leading to a policy that enabled the agent to park in all three parking spots and adapt its movement for a variety of different situations. This was verified throughout the whole training, the more variations were added to the agent's initial position and orientation the better the policy learned by the agent was to adapt, leading to a more stable training and better performance in the tests.

Regarding the perpendicular parking maneuver, the agent learned a policy that enable it to park from both sides of the parking spot with a great variety of different positions and orientations. The agent has some limitations, already referred, but for the situations in which it is able to park it performs a stable movement with no unnecessary steering. In most cases, the agent ends the episode with a close to the optimal final position.

Regarding the angular parking maneuver, the agent was able to learn a policy that enable it to park in a great variety of different scenarios, but it was not able to adapt in order to park from both sides of the parking spot. For the trained scenarios, the agent performs a stable movement with close to no unnecessary steering, but the greater the distance to the parking spot the more unstable the movement became. Due to the collision penalisation used in this training being too high, the agent just avoids the obstacle when detecting it. This limits the agent's ability to park in a parking spot with a distance between obstacles below 90 cm.

Regarding the parallel parking maneuver, the agent learned to park from a variety of initial positions and orientations but was unable to adapt its movement to park from both sides of the parking spot. The agent performed a stable and direct maneuver for the trained scenarios. Although this strategy completes the maneuver efficiently and quickly, it limits the parking spot space to which the agent can park. It was verified that the greater the initial distance and the variation added to the orientation the closest the final position of the agent was from the second obstacle. This limits the agent's ability to park in a parking spot with a distance between obstacles

less than 1.5 meters.

A video of all the carried out tests was created and in appendix A.1 the link of the video is presented.

The work developed in this Dissertation allowed the creation and submission of a scientific paper.

During the test phase for all the parking maneuvers, in scenarios where higher variations were added to the agent's initial position and orientation, the agent showed a lack of concern for the orientation of the wheels at the end of the maneuver. This is due to the non-inclusion of a penalty for the final orientation of the wheels in the reward function.

For the three parking maneuvers, an extensive amount of trainings were performed in search of the optimal hyper-parameters, but only one set of hyper-parameters was found that lead to a policy capable of completing the parking maneuvers. This happens because DDPG suffers from instability in the form of sensitivity to hyper-parameters and propensity to converge to poor solutions or even diverge. Since PPO is considerable more data-efficient and less sensitive to hyper-parameters than DDPG, it could be a more adequate approach for the autonomous parking problem.

5.1 Future Work

Regarding the autonomous parking problem, the introduction of a new algorithm is necessary to improve both the overall efficiency as well as the capability to transfer the learning from simulation to the physical robot. Since, DDPG revealed to be very poor sample efficiency and extreme sensitivity to hyper-parameters and prone to converge to poor solutions or even diverge, the future of this project aims:

- To implement the Reinforcement Learning algorithm, Proximal Policy Gradient (PPO);
- To train the PPO in this simulation:
- To transfer the learning system to a physical robot.

Appendix A

A.1 All tests for the three parking maneuvers

https://www.youtube.com/watch?v=4o7DuFK93_c

A.2 Code Snippets

```
def get_actor():

    last_init = tf.random_uniform_initializer(minval=-0.0003, maxval=0.0003)

    inputs = layers.Input(shape=(num_states,))
    out = layers.BatchNormalization()(inputs)
    out = layers.Dense(400, activation="relu")(inputs)
    out = layers.BatchNormalization()(out)
    out = layers.Dense(300, activation="relu")(out)
    out = layers.BatchNormalization()(out)
    outputs = layers.Dense(2, activation="tanh")(out)
    model = tf.keras.Model(inputs, outputs)
    return model
```

Figure A.1: Function responsible to create the two Actors

```
def get_critic():
    # State as input
    state_input = layers.Input(shape=(num_states))
    state_out = layers.BatchNormalization()(state_input)
    state_output = layers.Dense(400, activation="relu")(state_out)
    state_output = layers.BatchNormalization()(state_output)

    # Action as input
    action_input = layers.Input(shape=(num_actions))

    concat = layers.Concatenate()([state_output, action_input])

    out = layers.Dense(300, activation="relu")(concat)
    outputs = layers.Dense(1)(out)

    model = tf.keras.Model([state_input, action_input], outputs)

    return model
```

Figure A.2: Function responsible to create the two Critics

```python
class Buffer:
    def __init__(self, buffer_capacity=100000, batch_size=64):

        self.buffer_capacity = buffer_capacity
        self.batch_size = batch_size
        self.buffer_counter = 0

        self.state_buffer = np.zeros((self.buffer_capacity, num_states))
        self.action_buffer = np.zeros((self.buffer_capacity, num_actions))
        self.reward_buffer = np.zeros((self.buffer_capacity, 1))
        self.next_state_buffer = np.zeros((self.buffer_capacity, num_states))
        self.terminal_state_buffer = np.zeros(self.buffer_capacity, dtype=np.bool)


    def record(self, obs_tuple):
        index = self.buffer_counter % self.buffer_capacity

        self.state_buffer[index] = obs_tuple[0]
        self.action_buffer[index] = obs_tuple[1]
        self.reward_buffer[index] = obs_tuple[2]
        self.next_state_buffer[index] = obs_tuple[3]
        self.terminal_state_buffer[index] = obs_tuple[4]

        self.buffer_counter += 1
```

Figure A.3: Class of the Replay Buffer

```python
def learn(self):
    if self.buffer_counter < self.batch_size:
        return

    # Get sampling range
    record_range = min(self.buffer_counter, self.buffer_capacity)
    # Randomly sample indices
    batch_indices = np.random.choice(record_range, self.batch_size, replace=False)

    # Convert to tensors
    state_batch = tf.convert_to_tensor(self.state_buffer[batch_indices])
    action_batch = tf.convert_to_tensor(self.action_buffer[batch_indices])
    reward_batch = tf.convert_to_tensor(self.reward_buffer[batch_indices])
    reward_batch = tf.cast(reward_batch, dtype=tf.float32)
    next_state_batch = tf.convert_to_tensor(self.next_state_buffer[batch_indices])
    done_batch = self.terminal_state_buffer[batch_indices]

    self.update(state_batch, action_batch, reward_batch, next_state_batch)
```

Figure A.4: Function responsible to select a random bach from the Replay Buffer and sen it to the update function

```python
@tf.function
def update(self, state_batch, action_batch, reward_batch, next_state_batch, ):
    # Training and updating Actor & Critic networks.

    with tf.GradientTape() as tape:
        target_actions = target_actor(next_state_batch, training=True)
        target = reward_batch + gamma * target_critic(
            [next_state_batch, target_actions], training=True)

        critic_value = critic([state_batch, action_batch], training=True)
        critic_loss = tf.math.reduce_mean(tf.math.square(target - critic_value))

    critic_grad = tape.gradient(critic_loss, critic.trainable_variables)
    critic_optimizer.apply_gradients(
        zip(critic_grad, critic.trainable_variables)
    )

    with tf.GradientTape() as tape:
        actions = actor(state_batch, training=True)
        critic_value = critic([state_batch, actions], training=True)
        actor_loss = -tf.math.reduce_mean(critic_value)

    actor_grad = tape.gradient(actor_loss, actor.trainable_variables)
    actor_optimizer.apply_gradients(
        zip(actor_grad, actor.trainable_variables)
    )
```

Figure A.5: Update function

```python
@tf.function
def update_target(target_weights, weights, tau):
    for (a, b) in zip(target_weights, weights):
        a.assign(b * tau + a * (1 - tau))
```

Figure A.6: Soft-update function

```python
def get_reward():
    global state, prev_state, done, collision, sucess, steps

    distance = math.sqrt(pow(state[0] * 15, 2) + pow(state[1] * 15, 2))
    orientation = abs(state[5] * 180 - (state[4] * 180))
    reward_distancia = 0
    reward_collision = 0
    reward_sucess = 0
    reward_orientation = 0

    reward_distancia = math.exp(-(0 * pow(state[0] * 15, 2) + 0.06 * pow(state[1] * 15, 2)))
    orientation_rad = orientation * math.pi / 180
    reward_orientation = math.exp(-40 * pow(orientation_rad, 2))

    if distance <= 0.4 and orientation <= 10:
        print("DONE")
        done = True
        reward_sucess = 200
        sucess += 1

    if collision and steps > 60:
        print("Colidiu")
        done = True
        collision = False
        reward_collision = -100

    if -0.6 <= state[1] * 15 <= 0.6 and state[0] * 15 >= -3:
         w = 0.1
         reward_distancia = math.exp(-(0.04 * pow(state[0] * 15, 2) + 0.06 * pow(state[1] * 15, 2)))
    else:
         w = 0.0

    reward = (1 - w) * reward_distancia + w * reward_orientation + reward_sucess + reward_collision - 1
    return reward
```

Figure A.7: Reward function

```python
def talker():
    global done, steps

    while not rospy.is_shutdown():
        if steps > steps_per_episode:
            done = True

        if done:
            return

        start_pub.publish(True)
        rate.sleep()
```

Figure A.8: Function responsible to wait for the ROS connection to be established

```python
def callback(data):
    global steps, score, state, prev_state, done, total_steps, score, prev_action
    velocidade = 0
    angulo = 0  # -40 a 40
    if steps > steps_per_episode:
        done = True
    if done:
        return
    steps += 1
    total_steps += 1
    if steps == 1:
        prev_state = data.data
    state = data.data
    if steps >= 50:

        tf_prev_state = tf.expand_dims(tf.convert_to_tensor(prev_state), 0)
        tf_state = tf.expand_dims(tf.convert_to_tensor(state), 0)
        action_nn = choose_action(tf_state, num_actions, ou_noise)  # state
        action = action_nn[0]
        # print(steps)
        if steps == 50:
            prev_action = action
        velocidade = action[0] * 5
        angulo = action[1] * 40
        velocidade = np.clip(velocidade, -5, 5)
        angulo = np.clip(angulo, -40, 40)
        action_send = [velocidade, angulo, steps]
        action_send = Float32MultiArray(data=action_send)
        pub.publish(action_send)
        reward = get_reward()
        score += reward
        if training:
            buffer.record((prev_state, prev_action, reward, state, done))
            buffer.learn()
            update_target(target_actor.variables, actor.variables, tau)
            update_target(target_critic.variables, critic.variables, tau)
        prev_state = state
        prev_action = action
```

Figure A.9: Function responsible to control the program. It exchange variables between DDPG and CoppeliaSim via ROS

References

[1] T. A. Ribeiro, "Deep reinforcement learning for robot navigation systems," 2019. [Online]. Available: http://repositorium.sdum.uminho.pt/handle/1822/64866http://repositorium.sdum.uminho.pt/

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

[3] M. Heimberger, J. Horgan, C. Hughes, J. McDonald, and S. Yogamani, "Computer vision in automated parking systems: Design, implementation and challenges," *Image and Vision Computing*, vol. 68, pp. 88–101, 2017.

[4] Y. Ma, Y. Liu, L. Zhang, Y. Cao, S. Guo, and H. Li, "Research Review on Parking Space Detection Method," *Symmetry 2021, Vol. 13, Page 128*, vol. 13, no. 1, p. 128, jan 2021. [Online]. Available: https://www.mdpi.com/2073-8994/13/1/128/htmhttps://www.mdpi.com/2073-8994/13/1/128

[5] W. Li, H. Cao, J. Liao, J. Xia, L. Cao, and A. Knoll, "Parking Slot Detection on Around-View Images Using DCNN," *Frontiers in Neurorobotics*, vol. 14, jul 2020.

[6] "Audi uses machine learning to refine self-parking technology." [Online]. Available: https://www.motorauthority.com/news/1107621{_}audi-uses-machine-learning-to-refine-self-parking-technology

[7] J. Zhang, H. Chen, S. Song, and F. Hu, "Reinforcement Learning-Based Motion Planning for Automatic Parking System," *IEEE Access*, vol. 8, pp. 154 485–154 501, 2020.

[8] Y. Zhuang, Q. Gu, B. Wang, J. Luo, H. Zhang, and W. Liu, "Robust Auto-parking: Reinforcement Learning based Real-time Planning Approach with Domain Template," *NIPS 2018 Workshop MLITS*, no. Nips, oct 2018.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction Second edition*, 2018.

[10] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to Drive in a Day," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 8248–8254, jul 2018. [Online]. Available: https://arxiv.org/abs/1807.00412v2

[11] "About | Wayve." [Online]. Available: https://wayve.ai/about/

[12] Audi AG, "Automatic intelligent parking: Audi at NIPS in Barcelona," pp. 2–3, 2016. [Online]. Available: https://www.audi-mediacenter.com/en/press-releases/automatic-intelligent-parking-audi-at-nips-in-barcelona-7139

[13] M. Ziyad, "Artificial Intelligence Definition, Ethics and Standards," *Artificial Intelligence Definition, Ethics and Standards*, pp. 1–11, 2019. [Online]. Available: https://www.researchgate.net/publication/332548325_Artificial_Intelligence_Definition_Ethics_and_Standards

[14] Lisa Tagliaferri, "An Introduction to Machine Learning | DigitalOcean," 2017. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning

[15] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, sep 2013.

[16] L. Pack Kaelbling, M. L. Littman, A. W. Moore, and S. Hall, "Reinforcement Learning: A Survey," *Journal of Artiicial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[17] "The Bellman Equation. V-function and Q-function Explained | by Jordi TORRES.AI | Towards Data Science." [Online]. Available: https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7

[18] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, aug 1988.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," dec 2013. [Online]. Available: https://arxiv.org/abs/1312.5602v1

[20] "Policy-Gradient Methods. REINFORCE algorithm | by Jordi TORRES.AI | Towards Data Science," 2020. [Online]. Available: https://towardsdatascience.com/policy-gradient-methods-104c783251e0

[21] H. Shen, K. Zhang, M. Hong, and T. Chen, "Asynchronous Advantage Actor Critic: Non-asymptotic Analysis and Linear Speedup," 2020. [Online]. Available: http://arxiv.org/abs/2012.15511

[22] "The Actor-Critic Reinforcement Learning algorithm | by Dhanoop Karunakaran | Intro to Artificial Intelligence | Medium." [Online]. Available: https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016. [Online]. Available: https://goo.gl/J4PIAz

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *31st International Conference on Machine Learning, ICML 2014*, vol. 1, 2014, pp. 605–619.

[25] G. E. Uhlenbeck, L. S. Ornstein, I. Of Xll, . Higan, A. Arbor, Physisc', H. Laboratorium, and D. R. U. Utrecht, "PH Y5ICA L RE VIE H" ON THE THEORY OF THE BROKNIAN MOTION," Tech. Rep., 1930.

[26] SPR, "Festival Nacional de Robótica," 2019. [Online]. Available: http://www.sprobotica.pt/index.php?option=com_content&view=article&id=108&Itemid=

62https://www.festivalnacionalrobotica.pt/?lang=en%0Ahttp://www.sprobotica.pt/index.php?option=com_content&view=article&id=108&Itemid=62

[27] ——, "Robótica 2019 - Rules for Autonomous Driving," Tech. Rep., 2019.

[28] C. Berner, G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. De Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *arXiv*, 2019.

[29] M. V. Rajasekhar and A. K. Jaswal, "Autonomous vehicles: The future of automobiles," *2015 IEEE International Transportation Electrification Conference, ITEC-India 2015*, pp. 1–6, 2016.

[30] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[31] "The 6 Levels of Vehicle Autonomy ExplainThe 6 Levels of Vehicle Autonomy Explained | Synopsys Automotive. (n.d.). Retrieved July 2, 2021, from https://www.synopsys.com/automotive/autonomous-driving-levels.htmled | Synopsys Automotive." [Online]. Available: https://www.synopsys.com/automotive/autonomous-driving-levels.html

[32] W. Wang, Y. Song, J. Zhang, and H. Deng, "Automatic parking of vehicles: A review of literatures," *International Journal of Automotive Technology*, vol. 15, no. 6, pp. 967–978, oct 2014. [Online]. Available: https://link.springer.com/article/10.1007/s12239-014-0102-y

[33] Z. Du, Q. Miao, and C. Zong, "Trajectory Planning for Automated Parking Systems Using Deep Reinforcement Learning," *International Journal of Automotive Technology*, vol. 21, no. 4, pp. 881–887, aug 2020. [Online]. Available: https://link.springer.com/article/10.1007/s12239-020-0085-9

[34] S. H. Jeong, C. G. Choi, J. N. Oh, P. J. Yoon, B. S. Kim, M. Kim, and K. H. Lee, "Low cost design of parallel parking assist system based on an ultrasonic sensor," *International*

Journal of Automotive Technology, vol. 11, no. 3, pp. 409–416, jun 2010. [Online]. Available: https://link.springer.com/article/10.1007/s12239-010-0050-0

[35] J. Pohl1, M. Sethsson2, P. Degerman2, and J. Larsson2, "A semi-automated parallel parking system for passenger cars," 2006.

[36] L. Li, C. Li, Q. Zhang, T. Guo, and Z. Miao, "Automatic parking slot detection based on around view monitor (AVM) systems," in *2017 9th International Conference on Wireless Communications and Signal Processing, WCSP 2017 - Proceedings*, vol. 2017-Janua. Institute of Electrical and Electronics Engineers Inc., dec 2017, pp. 1–6.

[37] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A Survey of Deep Learning Techniques for Autonomous Driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, oct 2019. [Online]. Available: http://arxiv.org/abs/1910.07738http://dx.doi.org/10.1002/rob.21918

[38] "Tesla's Deep Learning at Scale: Using Billions of Miles to Train Neural Networks | by Yarrow Bouchard | Towards Data Science." [Online]. Available: https://towardsdatascience.com/teslas-deep-learning-at-scale-7eed85b235d3

[39] S. Song, H. Chen, H. Sun, and M. Liu, "Data efficient reinforcement learning for integrated lateral planning and control in automated parking system," *Sensors (Switzerland)*, vol. 20, no. 24, pp. 1–24, dec 2020. [Online]. Available: www.mdpi.com/journal/sensors

[40] E. Bejar and A. Moran, "Reverse parking a car-like mobile robot with deep reinforcement learning and preview control," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019*. Institute of Electrical and Electronics Engineers Inc., mar 2019, pp. 377–383.

[41] P. Zhang, L. Xiong, Z. Yu, P. Fang, S. Yan, J. Yao, and Y. Zhou, "Reinforcement learning-based end-to-end parking for automatic parking system," *Sensors (Switzerland)*, vol. 19, no. 18, sep 2019.

[42] L. Junzuo and L. Qiang, "An Automatic Parking Model Based on Deep Reinforcement Learning," in *Journal of Physics: Conference Series*, vol.

1883, no. 1. IOP Publishing Ltd, apr 2021, p. 12111. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1883/1/012111https://iopscience.iop.org/article/10.1088/1742-6596/1883/1/012111/meta

[43] T. Hester, T. Schaul, A. Sendonaris, M. Vecerik, B. Piot, I. Osband, O. Pietquin, D. Horgan, G. Dulac-Arnold, M. Lanctot, J. Quan, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3223–3230, 2018.

[44] V-REP, "Robot simulator CoppeliaSim: create, compose, simulate, any robot. - Coppelia Robotics," 2020. [Online]. Available: https://www.coppeliarobotics.com/

[45] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System." [Online]. Available: http://stair.stanford.edu