FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Deep Reinforcement Learning for Automated Parking

Dinis Moreira

Mestrado Integrado em Engenharia Informática e Computação

Supervisors: Pedro d'Orey, Rosaldo Rossetti

August 4, 2021

Deep Reinforcement Learning for Automated Parking

Dinis Moreira

Mestrado Integrado em Engenharia Informática e Computação

August 4, 2021

Abstract

Private transportation has been steadily increasing for decades, as each year a bigger percentage of the worlds population owns their private car, with some people even acquiring multiple vehicles, increasing the vehicle/person ratio to values that are now higher than ever. One of the consequences of this growth in vehicle population is the high demand for parking, which is especially concerning in densely populated areas. Car parking has become one of the biggest drawbacks of living in, or visiting, these urban areas when using a private vehicle, which for some people might be the only transportation method available. Developing a methodology for vehicles to follow predefined paths inside confined environments like traditional parking lots or parking lots designed exclusively for autonomous vehicles is a challenging task, as in this type of environments cusps and overlapping sections might be an essential part of the path in order to create feasible trajectories through narrow passages towards the vehicle's goal position. This type of paths also possesses an increased degree of complexity and diversity. Moreover, the path following agent must be able to ensure a consistent low distance from the path in order to avoid collisions with surrounding obstacles.

To this end, a methodology for teaching an agent through deep reinforcement learning how to guide a wheeled vehicle inside a confined space through a predefined path with these characteristics is proposed. It focuses on achieving a stable path tracking method to determine which part of the path the vehicle should track at any given point, and rewarding the agent for closely following the path in the correct direction. Additionally, a comparison between the results obtained by the three reinforcement learning algorithms tested in this environment during training and testing phases is presented, namely Soft Actor Critic, Deep Deterministic Policy Gradient and Twin Delayed Deep Deterministic Policy Gradient. The potential benefits to be extracted from automated parking include significantly reducing the time and resources spent on looking for a place to park, and through novel parking lot models designed exclusively for autonomous vehicles with the potential to be twice as space efficient, reducing the footprint of parking facilities in cities, allowing city planners to reorganise and rebuild green spaces and other infrastructure in their place, while giving the people who live and visit these densely populated areas added and improved space for their activities.

Keywords: Deep Reinforcement Learning, Artificial Intelligence, Autonomous Driving, Automation, Path Following

Resumo

O transporte privado tem vindo a crescer durante as últimas décadas, e a cada ano uma maior percentagem da população do planeta possui o seu próprio carro, com algumas pessoas inclusive a chegarem a adquirir vários veículos, aumentando o rácio de veículos por pessoa até um máximo histórico. Uma das consequências deste crescimento na população de veículos é o aumento da procura por lugares de estacionamento, o que é especialmente preocupante em áreas densamente populadas. O estacionamento de veículos tornou-se nos últimos anos numa das maiores desvantagens de viver, ou visitar, estas áreas urbanas com um veículo privado, o que para algumas pessoas é o único meio de transporte disponível. O desenvolvimento de uma metodologia para veículos seguirem um caminho pré definido dentro de ambientes confinados como parques de estacionamento tradicionais ou parques desenhados exclusivamente para veículos autónomos é uma tarefa desafiante, pois neste tipo de ambientes mudanças de direção e secções do caminho que se sobrepõem podem ser uma parte essencial da trajetória para que seja possível atravessar passagens estreitas até alcançar o objetivo final. Este tipo de caminhos também possui um elevado grau de complexidade e diversidade, para além disso, o agente que segue o caminho tem de ser capaz de garantir constantemente uma curta distância ao caminho para evitar colisões com obstáculos circundantes.

Com este objetivo, é proposta uma metodologia para treinar um agente através de *deep reinforcement learning* como aprender a guiar um veículo dentro de um espaço confinado através de um caminho predefinido com estas características. Este foca-se em alcançar um ponto de referência consistente para determinar por qual parte do caminho o veículo se deve guiar a cada momento, assim como recompensar o agente por seguir uma trajetória próxima do caminho no sentido correto. Adicionalmente é feita uma comparação entre os resultados obtidos pelos três algoritmos de *reinforcement learning* testados neste ambiente durante a fase de treino e de teste, nomeadamente: *Soft Actor Critic*, *Deep Deterministic Policy Gradient* e *Twin Delayed Deep Deterministic Policy Gradient*. Os potenciais benefícios que podem ser extraídos de estacionamento autónomo incluem uma redução significativa no tempo e recursos gastos à procura de um local para estacionar, e através de novos modelos de parques de estacionamento feitos exclusivamente para veículos autónomos com o potencial para serem duas vezes mais eficientes em termos de espaço, reduzindo a área de terreno utilizada por parques de estacionamento nas cidades, proporciona também aos organizadores das cidades a oportunidade de construir novos espaços verdes e outros tipos de infraestrutura no seu lugar, proporcionando aos residentes e visitantes destas áreas densamente populadas novos e melhores espaços para as suas atividades

Palavras-chave: Deep Reinforcement Learning, Inteligência Artificial, Condução Autónoma, Automatização, Seguimento de Caminho

"Não tenhamos pressa,
mas não percamos tempo"

José Saramago

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 3
- 1.3 Problem Definition . . . 3
- 1.4 Objectives & Approach . . . 5
  - 1.4.1 Objectives . . . 5
  - 1.4.2 Methodology . . . 5
- 1.5 Document Structure . . . 6

2 State of the art **7**
- 2.1 Parking systems for Autonomous Vehicles . . . 7
- 2.2 Novel AV exclusive parking lots . . . 8
- 2.3 Path Planning . . . 9
  - 2.3.1 Path Planning for Wheeled Vehicles . . . 10
  - 2.3.2 Reinforcement Learning for Path Planning . . . 12
  - 2.3.3 Multi Agent Path Planning . . . 12
- 2.4 Path Following . . . 15
- 2.5 Deep Reinforcement Learning . . . 16
  - 2.5.1 Introduction . . . 16
  - 2.5.2 Algorithms . . . 19
  - 2.5.3 Hindsight Experience Replay . . . 22
- 2.6 Research Gap . . . 23

3 Problem Approach **25**
- 3.1 Path and Vehicle Characteristics . . . 25
- 3.2 State Space . . . 26
  - 3.2.1 How to Determine What Point of the Path the Vehicle Should Track? . . . 27
- 3.3 Action Space . . . 28
- 3.4 Reward Function . . . 28
- 3.5 Deep Reinforcement Learning Architecture and Algorithm Parameters . . . 31

4 Used tools and Results **33**
- 4.1 Used Tools . . . 33
  - 4.1.1 Environment Simulation . . . 33
  - 4.1.2 RL Algorithms Library . . . 33
  - 4.1.3 Path Planning Library . . . 34
  - 4.1.4 Data Collection . . . 35
- 4.2 Evaluation metrics . . . 35

4.3 Training 36
4.3.1 Training Phase 1 36
4.3.2 Training Phase 2 38
4.3.3 Training phase 3 38
4.4 Training Phases Results Analysis 39
4.4.1 Training phase 1 results analysis 39
4.4.2 Training Phase 2 results analysis 42
4.4.3 Training phase 3 results analysis 47
4.5 Testing Phase Results 52
4.5.1 Testing environment 52
4.5.2 SAC's agent performance review 52
4.5.3 DDPG's agent performance review 53
4.5.4 TD3's agent performance review 56

5 Conclusions **59**
5.1 Future work 60

References **63**

List of Figures

1.1 Path with overlapping sections 4
1.2 Path with a cusp 4

2.1 Researched topic during the development of this thesis 8
2.2 Modules that constitute an AV parking system 9
2.3 High-density parking lot 10
2.4 Reeds-Shepp curves 11
2.5 Continuous Curvature turn 11
2.6 Example of Hybrid Curvature Steer path 12
2.7 Multi agent path planning problem environment 13
2.8 Problem not solvable through Cooperative A* 14
2.9 Reinforcement Learning agent interaction 17
2.10 Taxonomy of RL algorithms 19
2.11 Psuedo code for SAC 21
2.12 Psuedo code for DDPG 22
2.13 Psuedo code for TD3 23

3.1 Vehicle and path's angle of heading 26
3.2 vehicle's front wheels angle of direction 27
3.3 Angle difference between the vehicle and the path's heading 28
3.4 Tangential velocity 29
3.5 Lateral distance and longitudinal distance 30

4.1 Libraries used and the data shared between them 34
4.2 Four examples of paths generated for training phases one and two 37
4.3 Four examples of paths generated for training phase three and the testing phase . 38
4.4 Reward achieved in training phase one by each DRL algorithm 39
4.5 Average lateral distance achieved in training phase one by each DRL algorithm . 41
4.6 Success rate achieved in training phase one by each DRL algorithm 42
4.7 Reward achieved in training phase two by each DRL algorithm 43
4.8 Entropy and Policy Loss parameters of SAC algorithm throughout training phase two 43
4.9 Policy Loss parameter of TD3 algorithm throughout training phase two 45
4.10 Average lateral distance achieved in training phase two by each DRL algorithm . 46
4.11 Success rate achieved in training phase two by each DRL algorithm 47
4.12 Reward achieved in training phase three by each DRL algorithm 48
4.13 Average lateral distance achieved in training phase three by each DRL algorithm 50
4.14 Success rate achieved in training phase three by each DRL algorithm 51
4.15 Challenging type of path 51

4.16 Average lateral distance recorded in each episode of testing with SAC's agent . . 53
4.17 SAC's agent metrics values during a test episode 54
4.18 Average lateral distance recorded in each episode of testing with DDPG's agent . 54
4.19 DDPG's agent metrics values during a test episode 55
4.20 Average lateral distance recorded in each episode of testing with TD3's agent . . 56
4.21 TD3's agent metrics values during a test episode 57

Abbreviations

AV	Autonomous Vehicle
CA*	Cooperative A*
COA	Conditional Order on Arrival
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
FAR	Flow Annotation Replanning
HCA*	Hierarchical Cooperative A*
HER	Hindsight Experience Replay
MDP	Markov Decision Process
OoA	Order on Arrival
RL	Reinforcement Learning
RRA*	Reverse Resumable A*
RRT	Rapidly-exploring random tree
SAC	Soft Actor Critic
SLS	Smallest Length Stack
SLSP	Smallest Length Stack Platoon
TD3	Twin Delayed Deep Deterministic Policy Gradient
WHCA*	Windowed Hierarchical Cooperative A*

Chapter 1

Introduction

1.1 Context

Private Vehicle Overpopulation. Private transportation has been steadily increasing for decades, as more people are able to buy their own private cars, sometimes even multiple vehicles, and this vehicle/person ratio has never been higher [15]. In some of the most vehicle dense regions, like the USA, the automobile/person ratio is equal to 0.48 [53], which is a concern regarding ecological issues but also the physical space occupied by so many vehicles, both on the road and when parked. Recently there has been a surging force countering this movement towards car ownership, transportation as a service. Although digitally managed taxi services might motivate their users to not buy their own private vehicle, the ones being used for these services can also benefit greatly from efficient parking solutions. The elevated number of vehicles on the road and the high demand for parking is especially concerning in densely populated areas. The inefficient use of private vehicles, when most of them carry only one person, the driver, transporting no passengers or materials inside the vehicle on a daily basis, intensifies the concern regarding the space occupied by each vehicle on the road, which is also partially responsible for the urban sprawl phenomena.

Parking Scarcity and its Consequences. For the increasing number of car owners worldwide, car parking in densely populated areas is one of the biggest drawbacks of living in, or visiting, these urban areas. In some countries, drivers spend an average of more that 40 hours every year looking for an available parking spot, some examples being the United Kingdom (44 h), and Germany (41 h) [10]. During these periods the combined energy and resources spent towards the goal of finding a place to park are valued in billions of dollars. With today's vast majority of manually driven vehicles the two most commonly available options for parking are either on parking spaces located on the side of the road or in big parking facilities. Both occupy a significant area of the precious space in our cities and deteriorate the visual appeal of the city landscape, in some cases, land used exclusively for parking purposes can even be the single biggest parcel of land used in some cities, reaching values higher than 20% (e.g. Atlanta, USA) [14]. In large parking facilities for today's vehicles it is imperative that there exist big aisles between parking spaces so

that drivers can get in and out of each parking space, while having a permanently unobstructed connection path between them and the main road, so that they are not dependent on any other vehicle's movement in order to reach the exit. This implies that a sizable portion of these parking lots is only being used for a small fraction of time. If we consider a parking lot made exclusively for autonomous vehicles, other more efficient models for parking lot layouts are possible, with no need for a permanent connection from every parking spot to the nearest exit, as vehicles that block the path of others can be moved out of the way at any moment if necessary [28].

High-density Parking Industry Status. High-density parking refers to parking solutions that achieve a higher vehicle density than traditional parking lots. In terms of innovative parking solutions, other approaches have been explored for decades. The land scarcity issue in densely populated areas and the lack of space for parking vehicles, although now being more impactful than ever, has been a problem for many years [37]. Automatic robotic parking systems have been developed for decades, but the common approach has never been to resort on the vehicle's own source of energy for movement, but instead rely on electric elevators and rotating or sliding platforms. Such is the case of Parkmatic[1] that has explored this approach since 1984. This type of system requires high capital and operational costs in order to build such infrastructure, and having qualified personnel to maintain it and operate it consequently resulting in high operation prices, as well as for the end user. The advantages of parking lot model for AVs proposed in [7, 16, 28] when compared to traditional high-density parking systems come from (1) resorting to the vehicles' own source of energy for the necessary movements, (2) the absence of other moving parts, and (3) the lack of necessity for human operation, which drastically reduce the cost of maintaining the infrastructure, and consequently allowing for a larger profit margin.

Autonomous Parking in Confined Spaces. Several studies have been published regarding autonomous parking, from end-to-end RL parking methodologies [56, 6], to path planning for automated vehicles in tight spaces [31, 8] . In addition to this, most of the automobile industry's private companies have also heavily invested in research and development in these areas, as automation is nowadays seen as the future for private vehicles. In order to navigate and achieve the desired parking location and orientation in confined spaces, each vehicle needs to be able to autonomously perform well calculated high precision maneuvers, which is a more challenging task that planning maneuvers on wide and well demarcated roads due to visibility constraints and the necessity for multiple step maneuvers in these tight spaces, where a few centimeters can mean the difference between a well adjusted maneuver, and a disastrous collision. Achieving a navigation method based on a predefined path is the primary focus of this thesis, and the first step towards achieving a fully autonomous parking lot.

[1] https://www.parkmatic.com/

1.2 Motivation

The development of a methodology for closely following detailed predefined paths with cusps within confined spaces can severely improve the reliability and predictability of autonomous vehicle's maneuvers when traveling through narrow aisles between other vehicles. This predictability can also improve path planning and vehicle synchronization results for systems with multiple agents moving through overlapping paths at the same time and open the doors for innovative parking solutions.

The end goal of autonomous parking can improve the lives of not only drivers who spend less time and resources looking for a place to park, but also all the people living in densely populated areas. Previous works [7, 16] have shown that, when it comes to total land area used per vehicle, novel parking models made exclusively for AVs have the potential to be twice as space efficient when compared to regular parking lots. As such, taking advantage of this reduction in space occupied by stationary vehicles, city planners would have the opportunity to reorganise and rebuild these sections of cities, designing more visually appealing roads, with less parked cars, creating more green spaces, bicycle lanes, or even reusing these parking spaces to accommodate for other smaller vehicles, namely bicycles or motorcycles, giving the people who live and visit these densely populated areas more welcoming spaces for their activities.

Although the focus of this project lies on realistically sized vehicles and parking environments, the potential of the methodology presented in this thesis is also scalable to four wheeled robots or even nautical vehicles of any size, including automated wheeled robots working inside warehouses or boats moving through confined environments like tight canals or harbours, although to this end the proper adjustments regarding the vehicle's specific movement characteristics would need to be considered.

1.3 Problem Definition

The challenge consists of guiding a vehicle with Ackermann steering geometry from point A to point B through a confined environment, where even a small change of direction can mean the difference between a successfully cleared obstacle and a devastating collision between the controlled vehicle and the obstacles that compose the surrounding environment, meaning walls or other vehicles. This is a particularly difficult problem because, in addition to the traditional path following challenges of keeping the vehicle's trajectory as close to the predefined path as possible while moving through the path, and respecting the vehicle's movement constraints, in this case reverse maneuvers and direction changes need to be considered as well, in order to achieve the desired destination point and then forward park, otherwise there might not be a feasible trajectory connecting the start and finish positions. This brings some additional challenges to the task of following a predetermined path, for example knowing which part of the path the vehicle should track when the path overlaps itself at different points, this is a common phenomenon when the path involves back and forwards maneuvers and is represented in Fig 1.1.

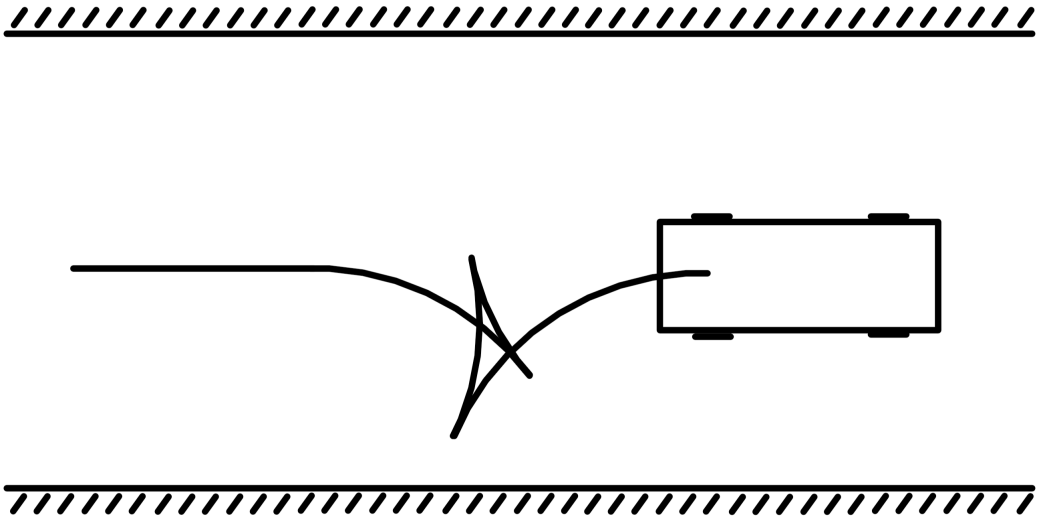

Figure 1.1: A representation of a path with overlapping sections derived from the need to perform a change in the vehicle's direction within a confined space

Another additional challenge is observed when there exist cusps in the predefined path, which require the vehicle to stop and change direction, in these situations the vehicle needs to know how to distinguish the previous section of the path that led it to that point, and the following path section that the vehicle should follow next. This type of situation is illustrated in Fig 1.2

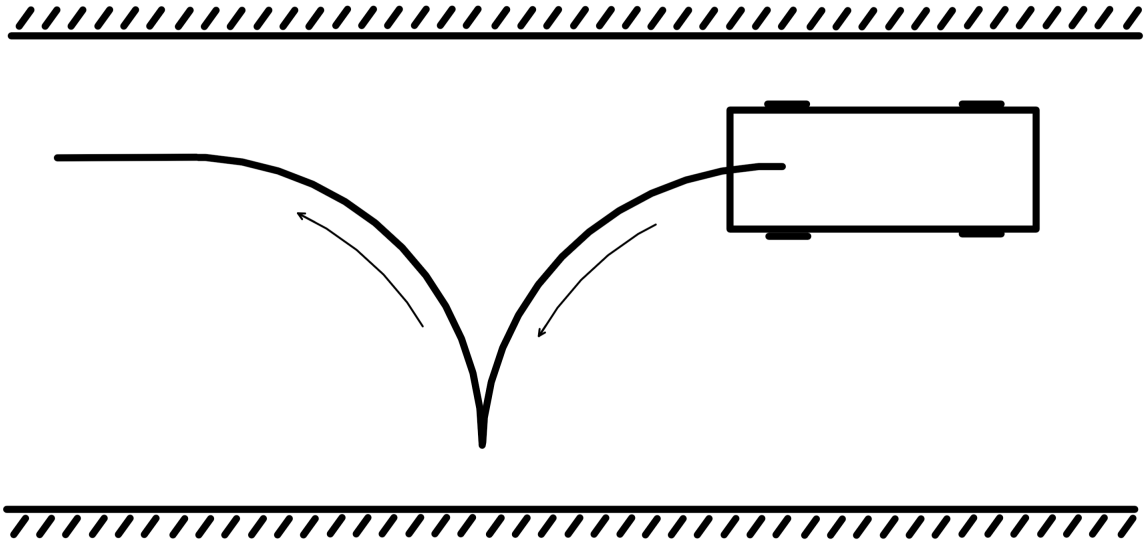

Figure 1.2: A path with a cusp, requiring a change in direction on the vehicle's part in order to be followed

The path consist of a sequence of points progressively leading from the vehicle's position towards the goal position. The vehicle has the average dimensions of a big SUV vehicle, as well as reasonable acceleration and steering angle limits. A success condition is activated when

the vehicle reaches its goal position with the correct orientation, fitting completely inside the goal parking spot, also created with standardized dimensions. Further details about the problem environment are clarified in section 3.1

1.4 Objectives & Approach

1.4.1 Objectives

The main purpose of this dissertation project is the development of an artificial intelligent agent capable of performing the said path following task, in paths that may include cusps, letting the agent learn from experience how to perform the adequate maneuvers and reach its desired parking destination. In order to achieve this, the two following objectives are established:

- The development of a methodology for learning how to efficiently guide a vehicle towards its goal parking spot, closely following a predetermined path, with cusps and overlapping sections, while traversing a tight confined space.

- Achieving a great understanding of how different Deep Reinforcement Learning algorithms perform during the learning process and the results they are able to obtain.

1.4.2 Methodology

Although some types of path following problems are possible to be resolved through rule-based algorithms, the process of coming up with meticulously designed heuristics and manually testing and optimizing a solution until every possible scenario achieves a satisfactory result can be very time consuming and require a deep understanding of the vehicle's kinematic behaviour, turning this process into a long and difficult one. Moreover these types of geometric algorithms typically have very few parameters to tune accordingly to the vehicle's or the trajectory's characteristics. This type of path following problem can be more easily tackled using more advanced AI techniques, which are capable of optimizing a policy through trial and error, learning from the environment, each time getting closer to reaching an optimal policy. For these reasons Deep Reinforcement Learning is employed in order to achieve the desired objectives.

Through Reinforcement Learning we are able to teach an agent whether or not the actions performed during training were good or bad by rewarding or punishing the agent for the chosen action in that state, making it more or less likely to repeat that behavior in the future [35], the main challenge when employing this type of approach is precisely tailoring the way in which the agent is rewarded. By combining this approach with deep neural networks we enable automatic feature engineering and end-to-end learning, meaning that domain knowledge can be very reduced. This approach has been applied to a vast domain of problems in the past, including other path following problems, with an overwhelming degree of success [32, 38, 30, 39, 49, 51, 26].

1.5 Document Structure

The remainder of this document is divided into four more chapters. In chapter 2 an overview of the state of the art of parking systems, path planning, path following and deep reinforcement learning is presented. In Chapter 3 the approach employed in order to tackle the problem is described, detailing the environment and vehicle characteristics, state space, action space, reward function, the neural network structure and the parameters used for the configuration of the DRL algorithms. Chapter 4 describes the third party tools utilized during the development of this thesis, the metrics considered for the evaluation of the performance of the agents, the different training phases conducted and an analysis of the results obtained during both the training phases and the testing phase. And finally, the main conclusions and prospects for future work are summarized in chapter 5.

Chapter 2

State of the art

In Fig 2.1 a diagram representing the topics researched for this thesis is represented. Parking systems that explore the possibilities and challenges that AVs face when parking in traditional environments, as well as novel AV exclusive parking lot models are investigated. Path planning and existing path following methodologies are surveyed, and a brief explanation of deep reinforcement learning and its latest advancements is presented. In order to look for the right literature regarding these topics, Scopus, Google Scholar and IEEE Xplore Digital Library where utilized.

2.1 Parking systems for Autonomous Vehicles

A typical parking system is constituted of four main components, as described in Fig. 2.2. Its modules are (1) environment recognition through sensors, (2) path planning generation, (3) path tracking control, and (4) actuators, namely throttle, breaks and steering.

Apart from this, in some scenarios other factors need to be taken into consideration, for example when considering human interactions in the environment. Motion and intent prediction of human driven vehicles is not a simple task, especially in confined spaces where complex maneuvers need to be performed, as is the case of parking lots. In [45] the integration of AV with manually driven vehicles is investigated. This study shows very promising results towards achieving functional mixed AV and human driven vehicle parking lots. It is concluded that, through a Long Short-Term Memory prediction model and Convolution Neural Networks, the intent of the human driver can be estimated with roughly 85% accuracy, and its real intent is kept on the top-3 predictions of the model almost 100% of the time. Knowledge regarding which is the parking spot that the human driver intends to park on has a major impact on predicting the parking trajectory. And through semantic image inputs, from where information regarding the environment and the location of possible obstacles can be extracted, long term predictions are improved.

In [31] a methodology for providing AVs useful information about the parking lot or garage they are entering with the goal of finding a parking spot to park on, instead of relying exclusively on the vehicles own sensors as a source of information for the environment, has been studied. This methodology is based on two layers of information. A semantic layer where lanes and parking

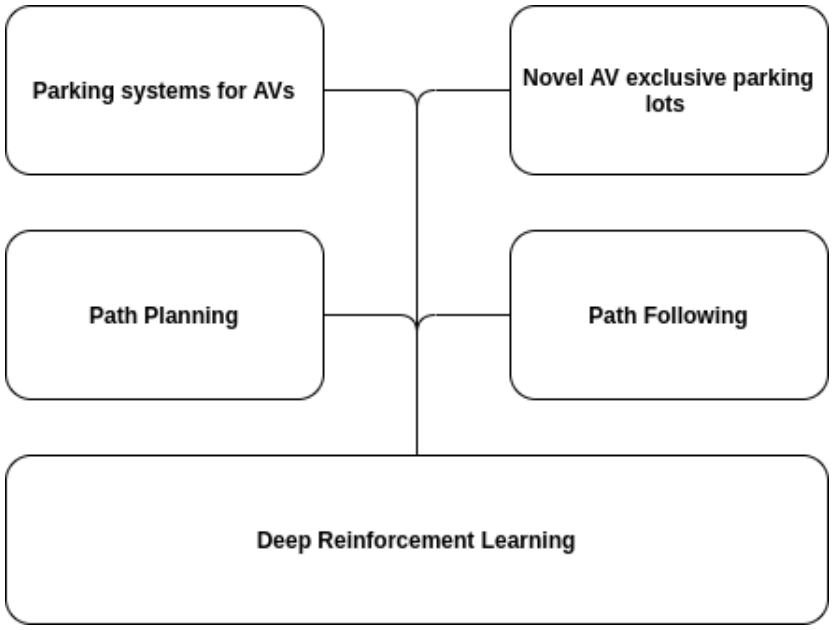

Figure 2.1: Researched topic during the development of this thesis

spots can be imported to create a directed graph that is later used for path planning. A metric layer that consists of a location-based grid map, which is used to store metric information about obstacles and non-drivable areas, enabling obstacle avoidance. A-priori information about the environment makes the AV's task of finding a parking spot more efficient and structured.

Autonomous Parking in traditional parking lots faces similar allocation and path planning with collision avoidance challenges. It is proven that for traditional parking lot layouts different allocation strategies for AVs achieve significantly different results regarding the fleet parking time and the queue length of vehicles waiting to enter the parking lot. "Interval-first Search", where each vehicle chooses a parking spot at least a minimum distance away from the one chosen by the vehicle that came before it, when operating with one lane between parking spaces, leads to the best parking time performance. However, if the objective is to minimize the entrance queue length, then the best approach would be to have 2 lanes between parking spaces and allocating vehicles using "Farthest-first Search", where vehicles choose the parking spot furthest away from the entrance [46].

2.2 Novel AV exclusive parking lots

Previous studies [7, 13], focusing on a novel parking lot model where vehicles are parked in a grid as closely as possible, show that by relying on the vehicles own source of energy for parking lot management and efficiently organising the vehicles inside the parking layout, parking lot space efficiency can be doubled when compared to traditional parking lots.

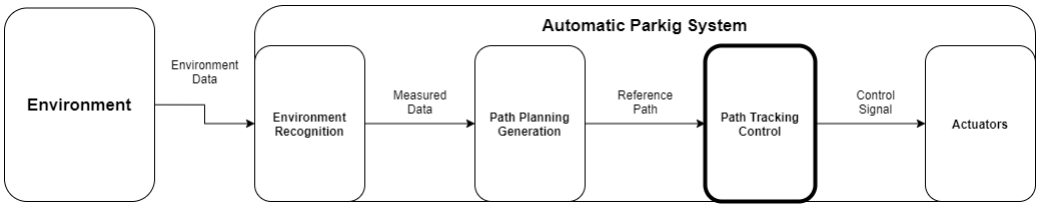

Figure 2.2: Modules that constitute an AV parking system

In these novel parking lot models, represented in Fig. 2.3 without any unoccupied traversing aisles in the parking lot, when the facility is at least partially full, most vehicles are dependent on others in order to move towards their desired location. This creates two mutually dependent challenges:

- The vehicle allocation problem
- The multi-agent path planning problem.

In order to move vehicles between parking spaces four planning and control strategies were developed in order to determine where each vehicle should be placed inside the parking lot; the first two strategies do not rely on future system knowledge (i.e. only rely on the current distribution of cars on the grid), attributing each vehicle a new place in the grid based on: (1) Smallest Length Stack (SLS), (2) Smallest Length Stack Platoon (SLSP). And additionally the following strategies which rely on future system knowledge were tested, based on: (3) Conditional Order on Arrival (COA), and (4) Order on Arrival (OoA). For both wide and deep parking grid layouts it was concluded that COA achieves the best performance when it comes to travel distance, number of maneuvers inside the park and removal time.

Innovative parking lots like the one presented in [13] can also benefit from their vehicles being able to closely follow a predefined path in a confined space, specially when considering a central authority able to compute multi-agent collision free path planning solutions that are then predictably followed by the vehicles, ensuring highly effective synchronous multiple vehicle maneuvers.

2.3 Path Planning

Path planning involves the creation of a path that connect two points, while respecting a set of movement constraints. This is a subject that has been heavily studied in recent years, from aerial vehicles [2], to terrestrial [55] and maritime vehicles [50]. In the next section, a brief explanation of the complexities of path planning for wheeled vehicle is presented.

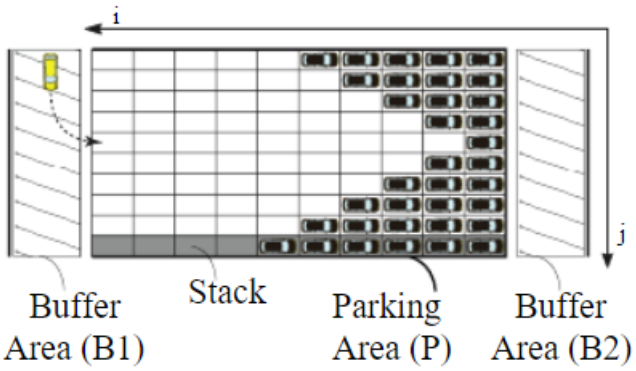

Figure 2.3: High-density parking lot with n interacting stacks and two buffers areas (B1, B2). Buffer area B2 is optional [13]

2.3.1 Path Planning for Wheeled Vehicles

In addition to finding complex paths in obstacle dense environments, path planning algorithms for wheeled vehicles also need to take into account the movement constraints of the vehicle. According to [23], to this end, four main components need to be considered: (1) A motion planning algorithm able to build a valid path, (2) steer functions that respect the vehicle's movement limitations and constitute path sections, (3) post-smoothing methods and (4) collision checkers.

Previously studied steering functions for driving with direction switches include Reeds-Shepp curves [43], capable of computing paths made up of line segments and circular arcs, offering optimal results with respect to path length, at the cost of smooth direction transitions. An example of a path generated using this type of curve can be observed in Fig 2.4. Continuous Curvature steer [17] computes paths made up of line segments, circular arcs and clothoid arcs. Although it does not ensuring minimal length, Continuous Curvature computes, in about the same time as Reeds and Shepp, paths whose length is close to the length of the optimal paths, and at the same time ensures smoother direction transitions with continuous curves. A representation of a general Continuous Curvature turn is illustrated in Fig 2.5. Recently, Hybrid Curvature Steer [8] was introduced. Like Continuous Curvature steer, Hybrid Curvature steer also computes paths made up of line segments, circular arcs and clothoid arcs, but it allows curvature discontinuities at direction changes. This brings its path length closer to the ones registered by Reeds-Shepp's paths. Overall, Hybrid Curvature steer, when compared to Continuous Curvature steer, achieves a higher degree of success in shorter planning times, and produces paths with less direction switches, leading to more practical results. This type of path is illustrated in Fig 2.6.

Motion Planning Techniques for automated vehicles have been investigated in [20], describing four different types of approaches: (1) Graph search, (2) random/deterministic sampling, (3) interpolating curves, and (4) numerical optimization. Graph search referring to approaches where discrete state and action spaces are used to create a graph that is then searched for the optimal solution. Sample based approaches are similar to graph search approaches but particularly draw

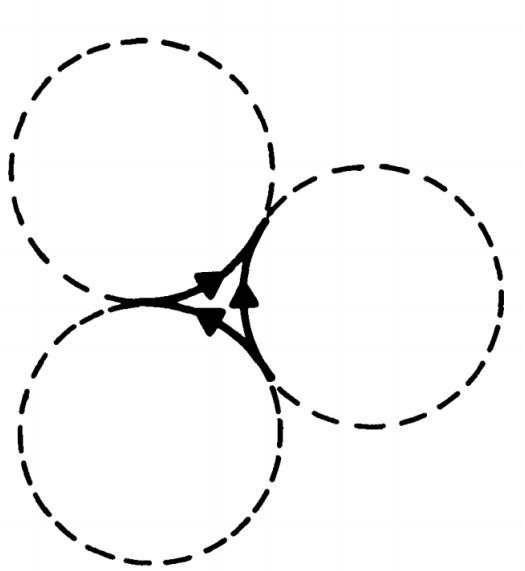

Figure 2.4: Illustration of Reeds-Shepp curves in a path which returns to the initial point but in the opposite direction [43]

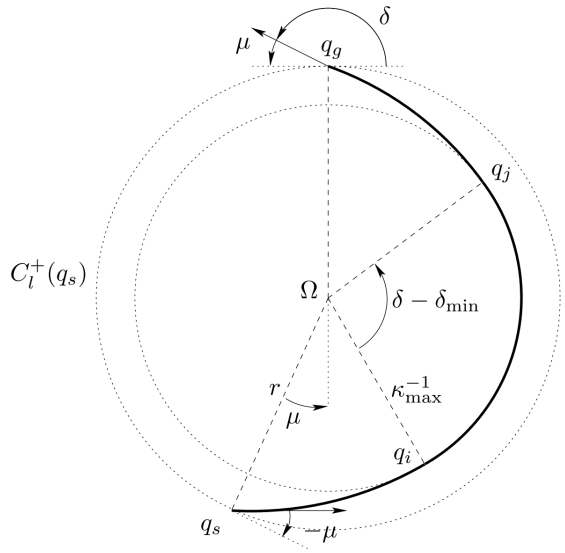

Figure 2.5: A general Continuous Curvature turn using a clothoid [17]

their samples either randomly, or from a discrete set of actions. Interpolating curve planners concatenate a set of curves, lines, circles, and clothoids, and plan a feasible path. This approach is particularly fast, but inflexible, and completeness can not be guaranteed. Optimization-based planners try to formulate the path planning problem as an optimization control problem, then solving it through numerical optimization.

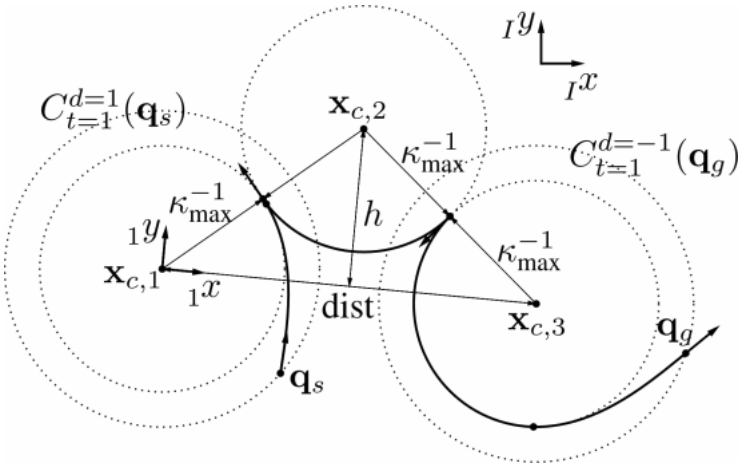

Figure 2.6: Example of Hybrid Curvature Steer path [8]

2.3.2 Reinforcement Learning for Path Planning

The capabilities of reinforcement learning have also been investigated in [4], with grid path planning. An agent trained through a neural Q-learning algorithm is shown to achieve its goal in relatively small maps that include obstacles, being able to navigate maps it has never been trained on as well, and, when the agent is able to find a path towards its goal, it is often the shortest path. However it is proven that in this type of simple path finding, the A* algorithm is still more reliable and efficient. However, for more complex environments, reinforcement learning has been shown to be a very useful tool. In [33] it is shown that through a DDQN algorithm, a trained agent is able to resolve local path planning tasks in a dynamic environment, as well as perceiving the movement of obstacles in advance through lidar data.

2.3.3 Multi Agent Path Planning

Single agent path finding in static environments is a relatively simple task, as in graph traversal the A* algorithm achieves optimal results in a fast and efficient way. However when other agents and moving objects are introduced into the environment, this turns into a complex and demanding problem that has been proven to be PSPACE-hard [27]. Until the mid 2000's, a simple Local Repair A* (LRA*) approach was the widely used standard in the video-games industry. In this family of algorithms each agent calculates the best route to its destination while ignoring every other agent, except when these other agents are in current neighbouring positions where they can block the intended passage of other agents. When a collision in imminent, the agent recalculates the remainder route, escaping the problematic area. This approach is not very efficient due to its lack of cooperation between agents, and achieves far from optimal results when it comes to total distance traveled by all agents, where cycles in the path taken are common, specially in crowded bottleneck regions, as described in Fig 2.7.

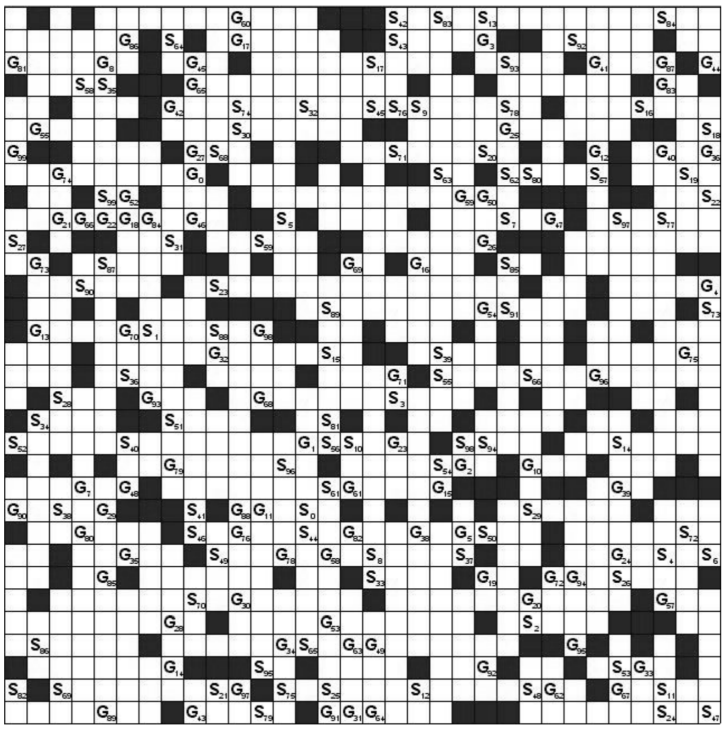

Figure 2.7: Example of a multi agent path planning problem environment, agents must navigate from their Source S_i to their Goal G_i [27]

2.3.3.1 Cooperative A*

Cooperative A* (CA*) [47] takes a three dimensional space-time approach, where every agent's planned routes are marked in a three dimensional reservation table, with two dimensions for space and one for time, these reserved paths being considered impassable for other agents in subsequent searches. However this approach is still not capable of solving certain problems where a greedy solution for one agent blocks all possible solutions for another one, this being illustrated in Fig. 2.8. In this example, the ideal solution would be for either agent to travel to the available cell on the left side of the corridor, while only afterwards the other agent would pass the middle of the corridor and make the intended trajectory to its final destination, followed by the final movement on the initial agent's part towards its destination. However, with CA* both agents would meet up in the middle of the corridor and none would be able to predict that moving towards the adjacent left cell would provide the best solution, remaining stuck in a deadlock.

2.3.3.2 Hierarchical Cooperative A*

Hierarchical Cooperative A* (HCA*) [47] is based on Holte's Hierarchical A* [25], where distances in abstract domains are computed on demand while tying to reuse previous search data in this abstract domain for better performance. Hierarchical Cooperative A* ignores both the time

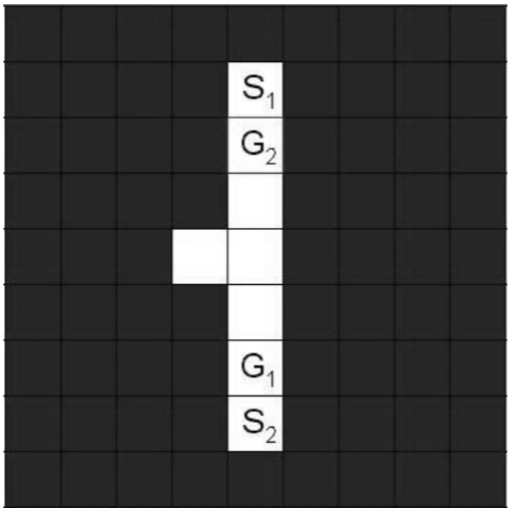

Figure 2.8: Problem not solvable through Cooperative A*. Agents must travel from S_i to their respective G_i [47]

dimension and the reservation table described in CA* for calculating these abstract distances, and uses a simple domain abstraction consisting of the original map with no other agents in it. This way, in an initial planning phase, abstract distances can be viewed as perfect estimates of the shortest path to every agents' destinations, and these can then be altered to avoid collisions in the plan execution phase. HCA* works in a similar way to CA*, but uses Reverse Resumable A*, which searches for the optimal path between two points but starting on the destination and making its way back towards the origin instead, this way, as explained in [40], for each node N expanded between the destination and the origin, the optimal path from N towards that destination is already calculated and can be stored for future reference. HCA* uses RRA* to calculate the abstract distances on demand. If the shortest path between an agent and its destination is clear of other agents' paths, then that path is taken. But otherwise, HCA* pushes the agent away from the shortest path into another free node and the RRA* search continues in concentric rings from this shortest path until the required node has been found. This algorithm achieves excellent results but is very computationally demanding and not suitable for some real time applications, and thus the following alternative approach was developed.

2.3.3.3 Windowed Hierarchical Cooperative A*

Windowed Hierarchical Cooperative A* (WHCA*) [47] works in a similar way to HCA* but after the initial RRA* planning phase in the abstract domain, during plan execution WHCA* is limited to a windowed search for replanning trajectories in case of interference from other agents. This window is shifted and the search is performed at regular intervals. For these intermediate windowed searches, considering a window of depth w, the objective is to find the best path from the agent's current location to the window's desired terminal edge w timesteps from there. For each

windowed search, other agent's trajectories in the reservation table are considered and can also be limited to w timesteps ahead. This approach can also spread processing time across all agents, when appropriately scheduled, and the RRA* search results can be reused for each consecutive window. Bigger windows are more computationally demanding but tend to produce results that are more similar to HCA*, with a higher rate of successes and paths with less cycles when compared to smaller windows, that tend to achieve results that are more similar to Local Repair A*, but perform faster and are easier to execute in real time.

2.3.3.4 Flow Annotation Replanning

Other less computationally demanding approaches have also been explored, such as FAR (Flow Annotation Replanning) [54], where an efficient method for multi-agent path planning on grid maps has been proposed. This method has been inspired by road networks, and was designed so that each row or column of the grid, on an initial planning phase, is restricted to only one direction, and this way head-to-head collisions are prevented. On this initial planning phase a directional search graph is constructed from the initial grid map, with connectivity constraints that ensure that every two locations reachable on the original map, are also reachable on the constructed directional search graph, in both directions. The execution phase is carried utilizing individual A* trajectory planning with additional constraints aiming to prevent collisions, such as the preference for straight lines, as fewer turns reduce the chance for collisions in unidirectional grids. Cooperation is also implemented through temporarily reserving the next k path cells prior to each of the agent's movements. Deadlock resolving mechanisms were also explored for situations where mutual dependencies occur, these situations being resolved through local repair procedures where an agent temporarily diverges off its planned trajectory and enables others to continue their initial trajectories. Through these approaches, FAR is able to resolve multi agent path planning problems in a quicker and less memory demanding way than WHCA*, while achieving very similar total traveling distances.

2.4 Path Following

At its core, path following is a control problem consisting of getting an agent to follow a predetermined path in the correct orientation while minimizing cross-track error, the distance from the path. This needs to be done while considering the vehicles movement constraints. Several approaches for a variety of vehicle types regarding path following methods have been proposed in the past, some of them involving reinforcement learning [32, 38, 30], where an agent learns the optimal policy only by interacting with the vehicle's controls, and other approaches opting for classical control theory approaches [3].

One of the first reinforcement learning steering control models was proposed in [32], showing how RL can be applied to control tasks like steering control for autonomous vehicles, with the learned policy achieving comparable results to then state of the art state space controllers. Achieving these results while learning the policy from scratch and without learning by imitation how to

obtain an initial controller. The initial agent is trained from a blank slate just by interacting with the vehicle's controls, proving that RL can also be applicable to this type of problem.

In [38] a methodology for path following in a maritime environment under unknown oceanic currents using DRL with the DDPG [36] algorithm is proposed. The training process is described as being much more efficient when employing transfer learning [58]. In an initial phase the agent is trained to follow a straight line in this type of environment, and then, in a second phase, the agent adapts the previously learned policy to tackle more complex curved paths. This approach can greatly reduce training time in order to achieve virtually indistinguishable results when compared to a policy that was trained from scratch in the target environment with curved paths.

In [30] a path tracking RL algorithm for a car-like mobile robot is presented, where the goal of the agent is to follow a drawn path line in the ground through images it captures of the upcoming path, utilizing a convolutional neural network (CNN) as image embedder for feature extraction. This approach is proven to guaranty a smooth vehicle control with velocity adaptation according to the shape of the path that is perceived ahead.

2.5 Deep Reinforcement Learning

2.5.1 Introduction

Deep reinforcement learning is the combination of deep neural networks and reinforcement learning, two fields of study that have seen an outstanding progress in recent years. In Reinforcement Learning agents learn an optimal policy by interacting with the environment through trial and error, as represented in Fig. 2.9. We are able to teach an agent whether or not the actions performed were good or bad by rewarding or punishing it for its behavior during training, making it more or less likely to repeat that behavior in the future. When combining deep neural networks with reinforcement learning, this enables another level of possibilities like automatic feature engineering and end-to-end learning, meaning that domain knowledge can be very reduced or virtually non-existent and the agent is still able to map state-action pairs to expected rewards and figure out the optimal policy in order to extract reward from the environment, under the right conditions even achieving superhuman performance in timed challenges.

Deep Reinforcement learning typically has the following components:

- Agent - The entity that makes decisions and takes action
- Actions - A set of actions is available to the agent, which can choose the most appropriate one at any given time, based on its environment observation.
- Environment - The world with which the agent interacts. It takes the agent's current state and action and returns to the agent its next state and the reward achieved by choosing that particular action in that state.
- State - The state is the perception that the agent has from the environment it interacts with, its input to produce a decision regarding the next action to execute.

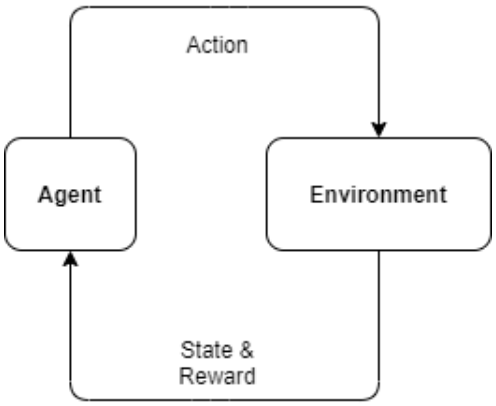

Figure 2.9: Representation of an agent choosing an action for its interaction with the environment, receiving a reward and moving on to the next state

- Reward - The feedback that the agent receives by taking an action in a given state, which serves as the agents source of information in order to optimize its decision making process.
- Policy - The strategy that the agent develops in order to determine the next action to make based on its current state. It is adjusted overtime with the goal of guaranteeing a maximum total cumulative reward.

Markov Decision Processes. Reinforcement learning problems are modeled as Markov Decision Processes, meaning that future states depend exclusively on the current state and action. This is an important factor for the agent's learning progress, as otherwise it would receive unpredictable rewards and advance to an unpredictable next state while being on the same apparent state and taking the same action. When the problem is modeled as an MDP, the agent is able to explore a consistent an predictable action/space environment, learning how to act accordingly in order to get the most return.

State spaces. The set of possible states where the agent can be in represents every possible observation that the agent can encounter. In most problems with a minimum degree of complexity, the agent will not encounter every possible state during its training, even after prolonged training sessions, but in a good training environment it should encounter and learn from all the possible situations where it might be tested on, to ensure it is prepared to handle all of those particular situations. Some environments might be too complex to feed completely into a state, so instead we can resort to just providing the agent with the useful information needed to ensure the Markov Decision Processes is not compromised.

Action spaces. The action space represents every action that an agent is able to perform in a given state, which can either be discrete or continuous, and can also have multiple dimensions, for example the steering angle and the acceleration of a vehicle.

Reward function The reward function is a very important piece in DRL, as it is the primary source of information available to the agent, telling it how good the action is for that particular state. Through the values returned by the reward function the agent learns which actions lead to better outcomes and which ones don't.

Return In most cases we do not want the agent to just pick the action that immediately brings it the highest reward value. Always picking the action that immediately brings the highest reward does not mean that in the long run the agent achieves great results, as this action might lead the agent to very bad states in the long run, where no more reward can be extracted. This is to say that we do not just want the biggest immediate reward, but the biggest return expected from the total cumulative reward. The most common way to define the best performance of an agent based on reward (r) is though a discount factor in the return, described in equation 2.1.

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 \gamma r_{t+3} + ... \tag{2.1}$$

By trying to maximize return (R), the discount factor (γ) ensures that the predicted return for the next time steps is considered, and the further in time the reward expected from each time step is, the less it accounts for the total value of the predicted return, as they are increasingly more unpredictable the further way they are from the current time step. A value between 0 and 1 is chosen for the discount factor, the most appropriate value for γ depending on what is the desired outcome. The bigger the value of γ the farther way in time the agent will try to maximize the return, even if it means receiving a lower immediate reward in the next time step. Typically values close to 1 are used.

Model-Based vs Model-Free. A significant characteristic of every reinforcement-learning algorithm is whether it operates under a model-free or model-based basis. Model-Based reinforcement learning algorithms, like AlphaZero and I2A, explore the environment and learn by constructing a model of the state transitions and how they affect the environment, essentially building a model of the environment. After the training phase the agent can then search the constructed model, and plan the best next action to take accordingly. In Model-Free reinforcement learning algorithms, like DQN and SAC, the agent tries to directly set up the optimal policy, ignoring a world model. While the agent is training, the policy is adjusted and a value is attributed to each state, or state/action pairs, ideally representing accurately the future utility that each state can offer. Compared to Model-based algorithms, Model-Free methods tend to be less efficient because, rather than directly trying to deduce the environment model, the policy is built by combining new information from the environment with other not necessarily accurate previous estimates about state values from previous experience [11]. A non-exhaustive, but useful taxonomy of RL algorithms created by OpenAI [41] is presented in Fig.2.10

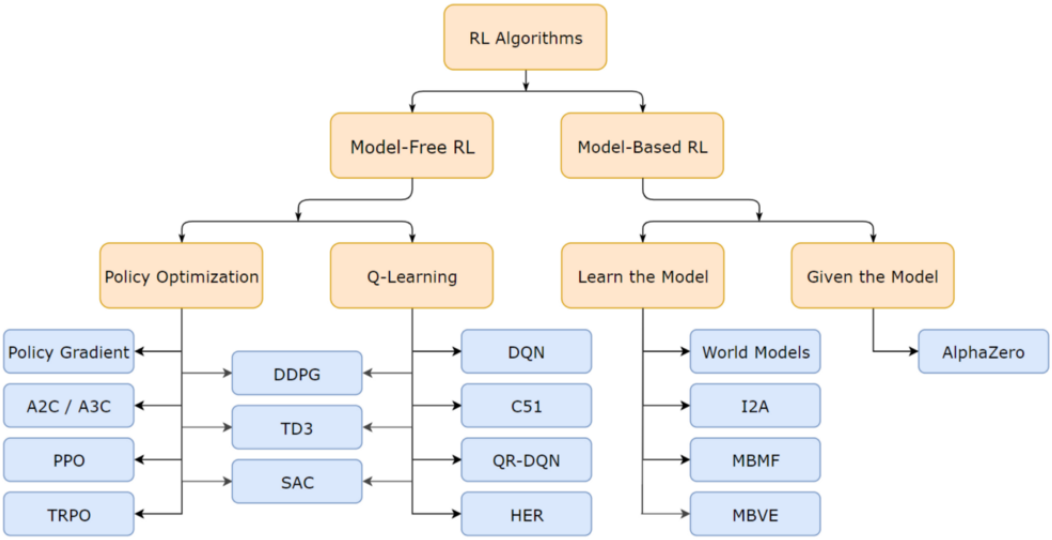

Figure 2.10: Taxonomy of RL algorithms created by OpenAI [41]

Off-Policy vs On-Policy. Another significantly distinctive characteristic of RL algorithms is whether they learn on-policy or off-policy. Off-policy algorithms like Q-learning, DDPG, or TD3 learn how to behave on the target policy, from data generated on the behaviour policy, that refers to another way of selecting actions. The idea behind having two different policies is that the target policy should be kept as close as possible to the optimal policy, while the behaviour policy is able to have more freedom to explore and find new ways to improve, using experience replay buffers to improve sample efficiency, however, and the convergence rates can be very dependent on the algorithm's hyperparameters configuration. For situations where the agent does not explore much, off-policy learning can improve the final target policy even in situations that differ from the ones the agent was trained on.

On-policy algorithms like SARSA, TRPO, PPO or A3C try to find the optimal policy while using the same learned policy for their actions, which means that their sample efficiency is generally lower than off-policy algorithms, needing new samples after each policy update, but they can still be useful for situations when the objective is to optimize the value of an agent that is exploring [52].

2.5.2 Algorithms

In this subsection, the three algorithms considered in this thesis are presented in more detail. SAC, DDPG and TD3 where considered because they are compatible with continuous action spaces, which is the case in this experiment, and also because all of them combine the strengths of Q-learning and policy gradient, learning both the policy function that maps state to action and also the action-value function which produces a value for each action. These DRL algorithms have also been applied to other path following tasks in the past with a vast degree of success [30, 38, 57]

Soft Actor Critic (SAC). Soft Actor Critic is a model-free, off-policy reinforcement learning algorithm that, besides trying to maximize reward, also tries to maximize the entropy of the policy. This entropy maximization approach has the goal of encouraging exploration, attributing equal probabilities to actions with similar Q-values and not assigning high probabilities to any range of actions. As the name suggests, it is an actor-critic method, taking a hybrid approach between Q-learning and policy optimization, having two independent memory structure representations for the policy and the value function. The policy represents the actor, that is used to select the most appropriate action, and the value function represents the critic, criticizing the decisions made by the actor [21, 22]. SAC concurrently learns a policy and two Q-functions, in order to not overestimate Q-values. SAC receives a batch of observations as input, and optionally also a batch of actions, that are then used to train a stochastic policy, maximizing the expected future return and also the expected future entropy of the policy, returning two outputs: the mean and the log standard deviation that are then used to get the desired action. Pseudo code for SAC is available in Fig. 2.11

Deep Deterministic Policy Gradient (DDPG). Similarly to the previously described SAC, Deep Deterministic Policy Gradient is also an actor-critic, model-free, off-policy reinforcement learning algorithm. It is used to learn continuous action spaces, and the "deterministic" part of its name refers to the fact that its actor computes the action directly, instead of a probability distribution over the action space. It is based on the original DPG [48] and DQN, and adds two more techniques to improve training stability and sample efficiency. (1) It uses two target networks to improve stability. This way it is able to slowly update the target networks, giving them more time to consider recent actions' influence in the network instead of updating the target network every time, and providing the opportunity to experiment with new changes while finding a more robust model before it is used to make actions. (2) The use of Experience Replay, where a list of tuples consisting of state, action, reward and next-state information is sampled during learning, instead of only relying in recent experience. DDPG concurrently learns a policy and a Q-function [36]. DDPG receives a batch of observations as input, and optionally also a batch of actions, that are then used to train a deterministic policy, progressively adjusted to maximize the expected return. The output of the algorithm is a specific action, instead of a probability distribution over actions. Pseudo code for DDPG is available in Fig. 2.12

Twin Delayed DDPG (TD3). Twin Delayed Deep Deterministic Policy Gradient, as its name suggests, is based on the previously mentioned DDPG algorithm, and it is also an actor-critic, model-free, off-policy reinforcement learning algorithm. However it addresses some of the challenges that DDPG faces with respect to hyperparameter tuning. Standard DDPG may overestimate Q-values, which leads to a non-optimal policy and learning efficiency. To avoid this type of behaviour, Twin Delayed DDPG, also known as TD3, introduces three critical differences: (1) Learning two Q-functions instead of one, using the minimum value of the two when computing the Bellman error loss functions, hence the "twin" part of its name. (2) Updating the policy and

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence

Figure 2.11: Psuedo code for SAC [21]

target networks less frequently than the Q-function, accounting for the "delayed" in its title. (3) Adding clipped noise on each dimension of the action space, smoothing the action space's Q value and avoiding the policy's exploitation of errors in the Q-function. These three characteristics make TD3 perform substantially better in some environments when compared to standard DDPG. TD3 concurrently learns a policy and two Q-functions [18]. TD3, in the same way as DDPG, receives a batch of observations as input, and optionally also a batch of actions, that are then used to train a deterministic policy, progressively adjusted to maximize the expected return. The output of the algorithm is a specific action, instead of a probability distribution over actions. Pseudo code for TD3 is available in Fig. 2.13

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Figure 2.12: Psuedo code for DDPG [36]

2.5.3 Hindsight Experience Replay

Hindsight Experience Replay can be combined with an arbitrary off-policy RL algorithm to allow agents to learn from sparse reward functions, which can be very useful when there is not a clear way to evaluate the behaviour of the agent in the middle of performing the task, but a very clear way to indicate whether or not it has been completed successfully. In some cases the reward function may not be very representative of how close the agent is to achieving the desired goal, or it might punish exploration too much. In these cases a standard reinforcement learning algorithm might not be able to figure out by itself whether or not the agent's actions led it closer to the desired objective. HER enables the agent to learn from mistakes, even when the end goal is not achieved or its reward does not reflect evidently how close the agent's actions led it to the end goal. With HER, by adding a goal to the input space, the agent is capable of learning how each action can bring it closer to its current goal. This way, even if the agent fails to achieve its end goal, it gains

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{I}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

      end if
    end for
  end if
until convergence

Figure 2.13: Psuedo code for TD3 [18]

knowledge regarding how to get to other previously explored states that may be closer to the end goal, and can then serve as stepping stones in order to complete the task successfully [5].

2.6 Research Gap

The previously mentioned works regarding path following are focused on generally open spaces with continuous one way paths, with no cusps or overlapping sections. The methodologies described in these works were not envisioned with this type of path characteristics in mind and their agents are not able to perform the necessary maneuvers when encountering these types of paths. Most of the previously developed approaches are based on maintaining a fixed velocity during the whole trajectory of the vehicle [32] or rewarding a specific range of one way acceleration values

[30], both unfeasible methods for multidirectional paths. On top of this, the predominant method of guiding the vehicle with its reference point in the path being the closest existing one is not feasible when considering overlapping path sections, which are common in paths containing direction changes.

During the research conducted for this project no studies that discuss path following methodologies for paths with cusps or overlapping sections where found.

Creating a methodology for a vehicle to follow paths with these characteristics can open a new set of possibilities for vehicles moving in confined spaces that require backwards and forwards maneuvers, specially with such precise and efficient path planning methods already developed for the creation of this type of paths, specifically considering confined environments [43, 17, 8].

Chapter 3

Problem Approach

Deep Reinforcement Learning has shifted the way to approach complex problems in recent years. Due to the recent progress made in this field, the range of problems where reinforcement learning can be employed has increased immensely, from real-time decision making problems [51] to management and planning [26], revealing an overwhelming degree of success. For this reason, a DRL approach is employed in order to efficiently achieve the desired results.

The described approach presents the DRL method that achieved the best results among several variations experimented.

3.1 Path and Vehicle Characteristics

Each path to be followed by the agent consists of a sequence of points that are one meter apart, and each one is represented by its x and y coordinates as well as its angle of heading θ (Fig. 3.1), which corresponds to the optimal angle of heading that the vehicle can have while strictly following the path, whether in forwards or backwards maneuvers, not the angle of the vehicle's velocity vector while moving through the path. In this scenario only forward parking maneuvers where considered.

The vehicle used for the simulation is 5 meters long and 2 meters wide, with a maximum speed of $40m/s$. It is able to turn its front wheels up to $\pi/3$ radians in each direction and its maximum acceleration is 5 m/s^2, for both forwards and backwards acceleration. The path following agent controls two parameters of the vehicle, (1) the steering angle of the vehicles front wheels δ and (2) its acceleration a, forwards or backwards. The vehicle then computes the inputs given by the agent and reacts accordingly.

In order to determine if an episode was successful or not, at every time step a matrix representing the distance from the goal position is calculated, it considers the absolute difference between the vehicle's and the goal spot's location and heading (φ) $[\Delta x, \Delta y, cos(\varphi), sin(\varphi)]$. This distance from goal matrix is multiplied by a weights matrix determining how much weigh each component of the distance from goal matrix should have, the weights matrix having the following values $[1, 1, 0.02, 0.02]$. The first two unitary values from the weights matrix are based on highway-env's

[34] default parking success values, while the two later values where achieved through trial and error between the values of $[0.01, 0, 1]$, where the value 0.02 provides enough orientation error margin for the vehicle to still be considered correctly parked when it is appropriate, while still enforcing a correct orientation of the vehicle inside the parking space. If at any time the corresponding inner product from the matrices achieves a value lower than 0.1, the episode is concluded successfully. This value was determined through trial and error between the values of $[0.08, 0, 2]$, where values under 0.1 more accurately consistently portrayed a vehicle completely inside the parking space.

3.2 State Space

The observation that the agent retrieves from the environment at each time step, represented in equation 3.3, is comprised by 2 main components, (1) the vehicle's current state information and (2) information regarding the best path points to tack. From the vehicle, the agent observes its current location on the x and y axis, the vehicle's current velocity v, acceleration a, the heading of its front wheels δ ((Fig. 3.2)), and both sine and cosine of its yaw angle (α), or in other words, the direction it is currently heading (Fig. 3.1). The state representation of the vehicle is presented in equation 3.1.

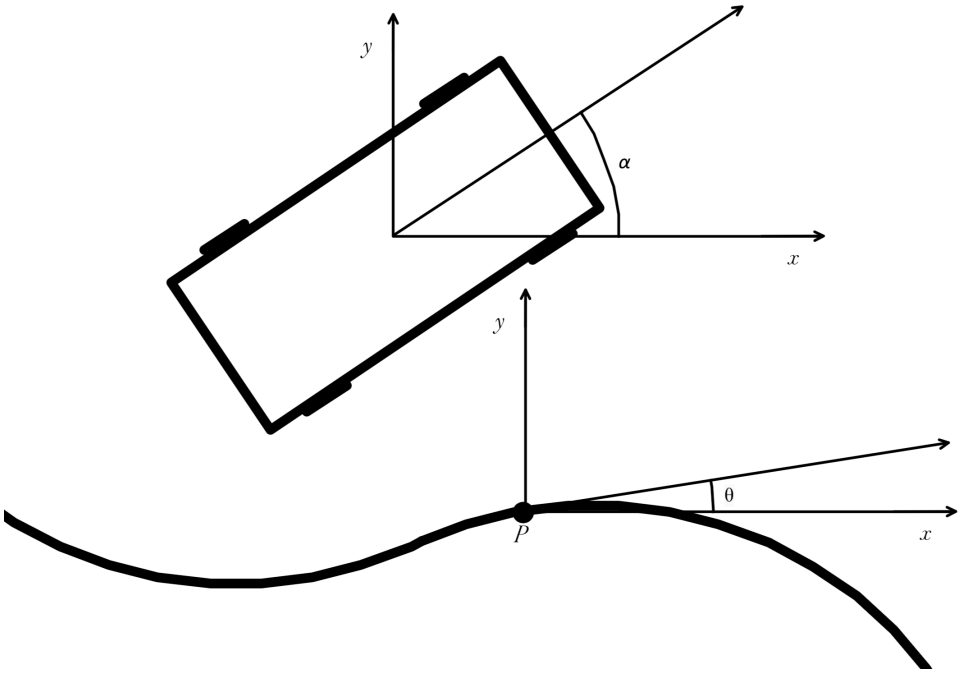

Figure 3.1: Visual representation of the vehicle's angle of heading (α) and the paths angle of heading (θ) at point P

The agent also observes the four next points it should traverse in the path, relative to its reference point in the path. From each of the four next points it retrieves the lateral distance relative to

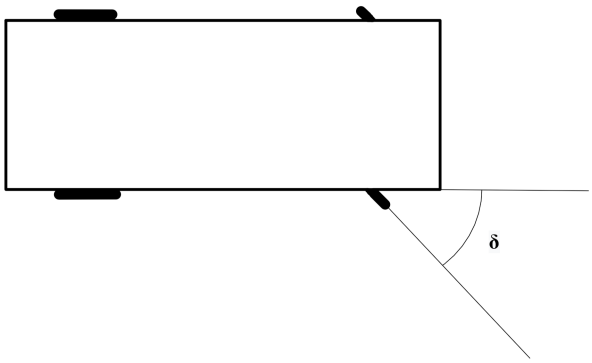

Figure 3.2: Visual representation of the vehicle's front wheels angle of direction δ

tangent frame of the path d_{Lat}, the longitudinal d_{Long}, illustrated in Fig. 3.5, and also both sine and cosine of the point's heading θ. The state representation of a path point is represented in equation 3.2. Through experimentation during the development of this project, state spaces considering one, two and three points in the path where attempted, but did not produce such reliable results during the first phases of training experimented. Observations containing four points proved to guarantee sufficient look ahead for the vehicle to plan the most appropriate actions.

$$s_{vehicle} = (x, y, v, sin(\alpha), cos(\alpha), a, \delta) \tag{3.1}$$

$$s_{pp} = (d_{Lat}, d_{Long}, sin(\theta), cos(\theta)) \tag{3.2}$$

$$s = (s_{vehicle}, s_{pp1}, s_{pp2}, s_{pp3}, s_{pp4}) \tag{3.3}$$

It is important to mention that the reference point of the vehicle, from which the distance to the path is calculated, is located in the center of its rear axis, so that according to the kinematic bicycle model [42, 29], the movement's center of rotation is the same when making turns both forwards and backwards.

3.2.1 How to Determine What Point of the Path the Vehicle Should Track?

Because in this confined scenario the vehicle's predefined trajectory may overlap itself at different points, the simple distance between the current vehicle location and the path points does not suffice in order to determine which point from the path is the best reference point. As such the agent needs to consider not only the absolute distance that separates it from the path, but also the angle difference between its current heading and the heading of the path. To achieve this, equation 3.4 attributes a score to each path point:

$$V_{pp} = d(1 + \sqrt{|\varphi|}) + 2.5\sqrt{|\varphi|} \tag{3.4}$$

Weighing in the absolute angle difference between the current vehicle heading and the angle of heading of the path point $|\varphi|$, described in Fig. 3.3, and the distance d from the path. When the vehicle is crossing an overlapping path section and is momentarily closer to a crossing part of the path, meaning that the absolute distance d between the vehicle and the crossing path point is shorter than the distance from the next point in the natural continuation of the planned path, the reference point will not shift, as this crossing part of the path will typically have a considerable heading difference relative to the vehicle, allowing for a reliable reference point in the path, that is not prone to jumps between overlapping path sections. After each path point has been evaluated, the one with the lowest score is chosen as the best candidate for a reference point in the path.

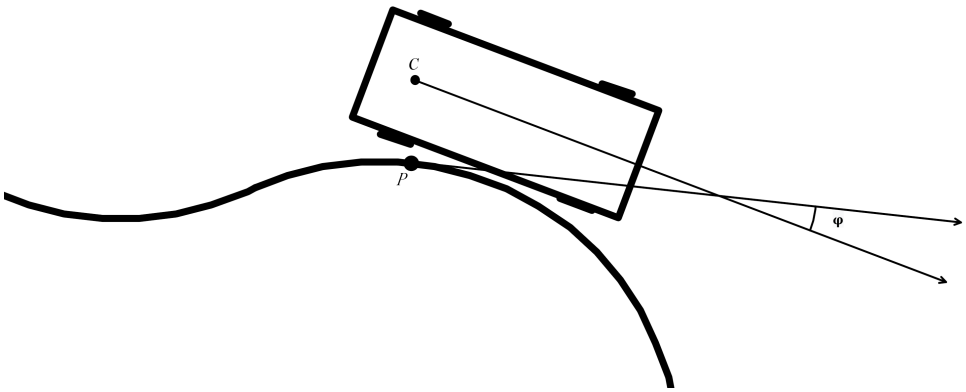

Figure 3.3: Representation of the angle difference between vehicle heading and path heading (φ) based on the vehicles tracking path point (P). With C representing the center of the vehicles rear axis

3.3 Action Space

The agent's control over the vehicle is equivalent to any standard automated vehicle's controls. The agent is able to control the acceleration (a) of the vehicle and the heading of its front wheels (δ). The steering angle's range is $[-\pi/3, \pi/3]$ radians, and the acceleration range is $[-5,5]m/s^2$, allowing the vehicle to perform both forwards and backwards maneuvers.

3.4 Reward Function

Several different approaches were tested for the reward function, rewarding good behaviour, penalizing bad behaviour, as well as mixed approaches. On an initial approach, a reward function based on the one utilised in [30] was tried in this type of paths with no success, leading to progressive alterations of the reward function as well as other factors of the reinforcement learning

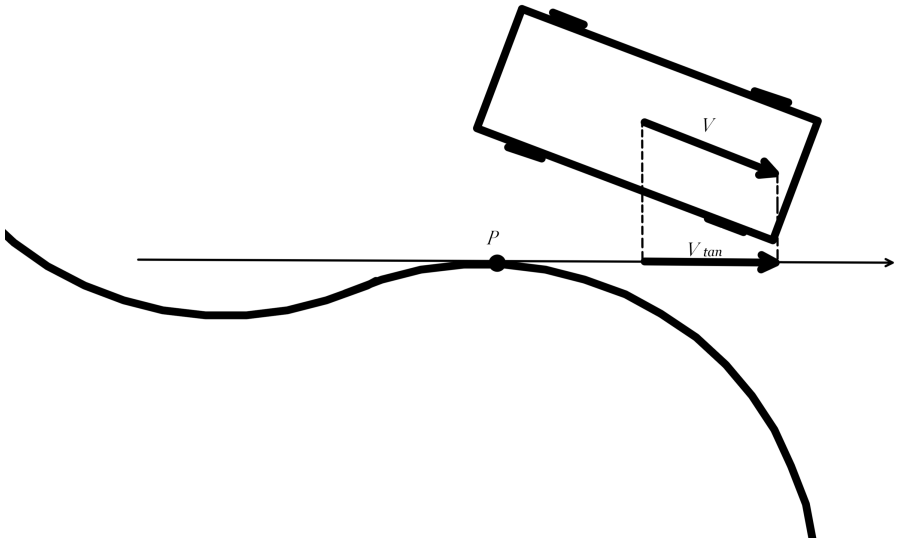

Figure 3.4: Visual representation of v_{tan}. P represents the point in the path that is being tracked by the vehicle, with the path's tangent having the same orientation as the path in point P. v_{tan} is a projection of the vehicle's velocity v in this tangent frame

process. In the end the one that achieved the best results and the fastest convergence was reward function 3.5:

$$r = \begin{cases} -1 & \text{if} \quad v_{tan} < 0 \\ 0 & \text{if} \quad |d_{Lat}| > 1 \vee |d_{Long}| > 2.75 \vee |\varphi| > \pi/2 \\ min(1,v)(1-d_{Lat}) & \text{otherwise} \end{cases} \tag{3.5}$$

The first condition penalises the vehicle's behaviour if its velocity projected on the tangential frame of the path's heading (v_{tan}) is negative, meaning that the vehicle is moving on the opposite direction of the path's heading, a behaviour that we want to avoid at every moment. A visual representation of v_{tan} can be observed in Fig. 3.4

The second condition neutralizes the reward for three types of situations:

- when the lateral distance to the path is greater than 1 meter, this value was determined through trial and error, where between $[0.5, 2]$, where the value 1 seems to offer a big enough lateral gap for the vehicle to adjust its policy and, after a short number of training time steps, it rarely surpasses this value;

- the absolute longitudinal distance from the next point in the path is greater than 2.75 meters, this length was determined through trial and error, where between $[1, 4]$, where the value 2.75 demonstrated to be a length that is long enough to not neutralize valid path following

rewards, but short enough so that it avoids rewards in states where the vehicle is aligned with the path tangent but far from the actual path;

- when the angle between the path's heading and the vehicle's heading (φ) is greater than $\pi/2$.

The values used in this second condition are tailored to the dimensions of a standard road vehicle and may need to be adjusted when considering different sized vehicles.

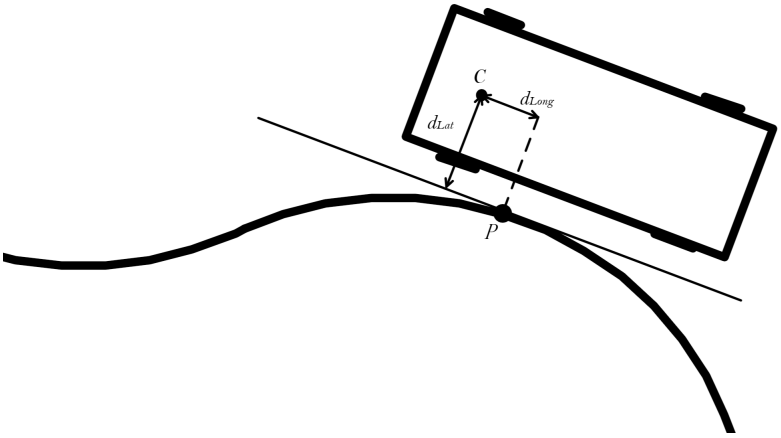

Figure 3.5: Lateral distance (d_{Lat}) and longitudinal distance (d_{Long}) from path point P, relative tho the vehicle's center of the rear axis C

The agent's main source of reward comes from how low its lateral distance (d_{Lat}) from the path is, while moving in the correct direction. The vehicle's lateral distance (d_{Lat}) is illustrated in Fig. 3.5. The vehicle's velocity component (v) is necessary, otherwise the agent would be rewarded just by standing immobilized on top of the path. To avoid this, the reward that comes from path closeness is multiplied by the vehicle's absolute velocity when it is lower than 1 m/s.

Generally, as long as the vehicle keeps moving through the path within a small lateral distance from it, the agent is rewarded. The proposed reward function is intentionally simple and does not consider the angle between the path's heading and the vehicle's heading (φ) as a discount factor in the reward attributed to the agent, as sometimes when the agent distances itself from the path, a small φ value can be beneficial to the vehicle's trajectory. Moreover, as reinforcement learning tries to maximize the long-term reward of the agent, it is expected that the agent learns the relation between the current error present in φ, and its steering angle δ without explicitly needing to immediately penalize it for φ error.

Description	Value
Learning rate	0.001
Replay buffer size	1,000,000
Discount factor	0.95
Batch size	256
Simulation frequency	15

Table 3.1: Hyperparameters' values used for the DRL algorithms' learning progress

3.5 Deep Reinforcement Learning Architecture and Algorithm Parameters

In order to learn how to perform in a continuous action space, a DRL algorithm capable of handling a continuous control problem is needed. To this end, three DRL algorithms where picked for the purposes of this research: SAC [21], DDPG [36], and TD3 [18], all of them using Hindsight Experience Replay [5].

The structure used for the neural network is comprised of 3 hidden layers of 256 neurons each, shared between the policy network and the value network.

The hyperparameters used for the training configuration of the DRL algorithms are presented in table 3.1

Additionally, the maximum episode duration varies according to the different training phases. In the initial phases of training, with only one column of parking spaces and paths that tend not to exceed 35 meters, the maximum episode duration is restricted to 10 seconds, providing the agent with enough time to learn how to follow a simple path for long enough without wasting too much processing power on the initial episodes, while the policy is still far from optimal and the agent tends to move far away from the path, ending up "lost".

When the agent has completed the initial phases of training and is capable of following paths with a minimum degree of reliability, the number of columns in the parking environment is increased, also increasing the distance between the agent's initial position and the goal parking spots for some initial/final position pairs. For this reason the maximum episode duration is extended to 20 seconds, giving the agent enough time to complete both shorter and longer paths used during this training phase. The training phases are further described in section 4.3.

Chapter 4

Used tools and Results

In this section the tools used for the simulation of the environment and data collection are presented, followed by an analysis of the results obtained from the training and testing phases of the agents.

4.1 Used Tools

The following third party tools were used during the development of this thesis:

4.1.1 Environment Simulation

For simulating the environment the autonomous driving simulation tool Highway-env [34] was used. It is based on Open AI's Gym environment toolkit for developing and comparing reinforcement learning algorithms [9], and it provides a collection of environments for autonomous driving, as well as the tools to build a custom environment according to our specific needs. It renders a 2D environment, top-down view, and accurately simulates automobile behaviour with front wheel steering through the Kinematic Bicycle Model [42, 29]. Every vehicle is rendered according to their x and y coordinates, θ angle of its heading, and δ for the steering angle. And in every time step, through the inputs of velocity v and steering rate φ, the next state of each vehicle is simulated, respecting the physics of their bodies. In this project a modified version of the parking environment provided by Highway-env was used. Some of the additions made to the environment generation include dynamic and configurable generation of different sized parking lots with varying number of parking spot rows and columns, varying parking spot length and width, adjustable corridor width, easily toggled generation of stationary vehicles in parking spots and barriers surrounding the parking lot.

4.1.2 RL Algorithms Library

Stable Baselines [24], the set of improved implementations of reinforcement learning algorithms based on OpenAI Baselines [12], was used in this experiment. It provides a set of State of the art

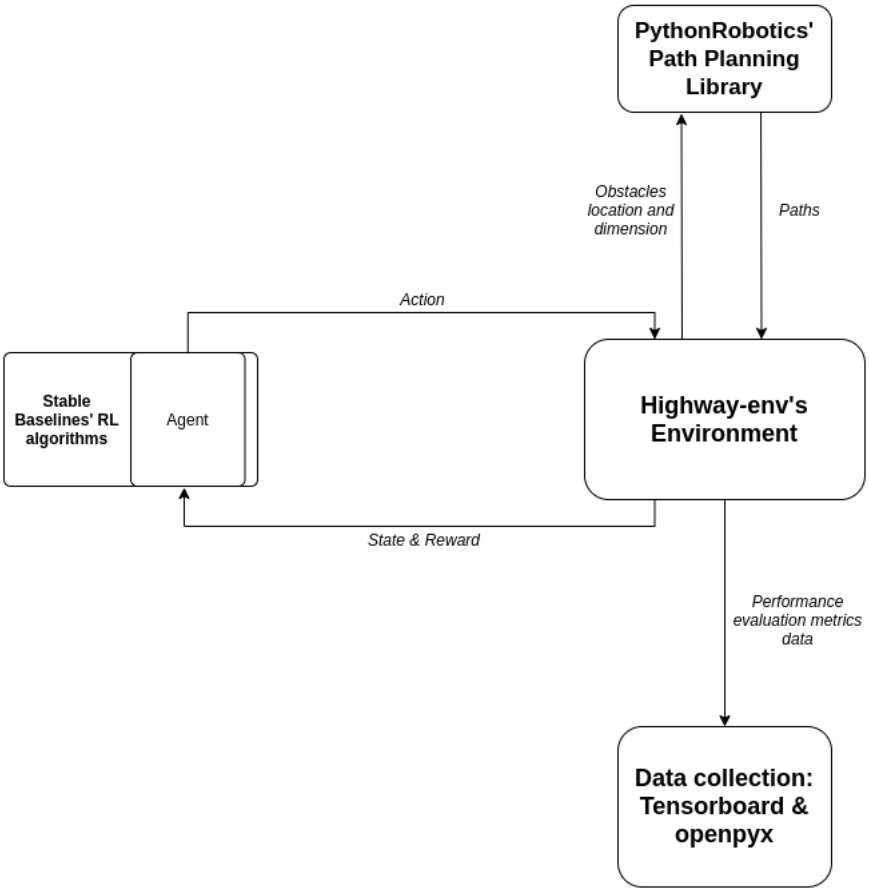

Figure 4.1: Graphical representation of third party libraries used and the data shared between them

RL algorithms in a simple and uniform interface, with detailed documentation.

4.1.3 Path Planning Library

The PythonRobotics library [44] is used in order to build the paths to be followed. It provides a collection of robotics algorithms, including the Rapidly-exploring random tree (RRT*) with Reeds-Shepp path creation algorithm that was used in this project to calculate possible paths using Reeds-Shepp curves [43]. The algorithm receives an initial and final position, consisting of x and y coordinates and heading, in order to create a path that avoids a provided list of circular obstacles. This list of obstacles is dynamically created from the four corners of each parked vehicle in the environment with a radius of 3 meters as well as the barriers at the park's edges with an 8 meters of radius. The algorithm samples a random configuration in the obstacle free space, connects it with a collision free, minimum-cost extension of the existing tree, and locally rewires it, repeating this process in order to converge into an optimal solution, with a computational budget, so that the algorithm does not run indefinitely, this meaning that the paths generated may not be optimal, even when generated with a large computational budget. Despite not being the most precise path

planner for this type of tight environments when it comes to collision checking, the algorithm provides a fast and valid trajectory from the center of the vehicle's rear axis to the desired goal parking spot, one that may include cusps and overlapping sections when necessary. The paths returned consists of a list of points, each represented by its x and y coordinates as well as its angle of heading θ, and each point is one meter apart.

4.1.4 Data Collection

For training data collection and visualisation the TensorFlow's [1] visualization toolkit Tensorboard is used, which provides an easy to read and detailed information visualization platform for machine learning experimentation, tracking and visualizing metrics such as the policy loss and mean reward. The success rate, mean lateral distance from path, vehicle velocity and other simulation metrics calculated during training and testing are collected and then stored using openpyxl [19], a library to read/write Excel 2010 xlsx/xlsm files.

4.2 Evaluation metrics

Throughout training and testing, three main metrics are used to evaluate the performance of the agents: (1) total cumulative reward extracted from the environment, (2) average lateral distance from the path registered in each episode (d_{Lat}), and (3) success rate that the policy achieves, limited to the last 100 episodes during training. Each of the metrics is defined formally in the following:

- **Total cumulative reward.** The reward that the agent manages to extract from the environment is a good indicator of how well the agent is able to follow its path, as the reward function takes into account its proximity to the path and its velocity, the two main indicators of how good the path following trajectory is. It also reveals whether or not the agent is effectively improving its policy during training, as this is the metric that the DRL algorithms are trying to maximize through policy optimization.

- **Average lateral distance from the path (d_{Lat}).** It indicates how close to the previously calculated path the agents are maintaining their trajectory. This is an important metric because if the agents do not maintain a short distance from the path, created taking obstacle collision avoidance into consideration, they risk colliding with surrounding obstacles increases, an outcome that is not desirable.

- **Success rate.** This serves as a more practical metric to this experiment, determining whether or not the end goal is achieved, with high success rates validating the policy of the agents and the proposed DRL approach to the problem, and low success rates refuting them.

Additionally, the absolute angle error between the vehicle's heading and the corresponding path section's heading ($|\varphi|$), and the velocity (v) of the vehicle are also taken into consideration in order to achieve a better understanding of the vehicle's behaviour during test runs.

- **Absolute angle error between the vehicle's heading and the path's heading ($|\varphi|$).** $|\varphi|$ indicates if there is a big disparity between the vehicle's trajectory orientation and the initially planned heading created when computing the path.

- **Vehicle's velocity (v).** The measurement of the vehicle's velocity helps to understand if the vehicle is moving through the path at an unwanted pace, or making unnecessary stops, providing a better understanding of how efficient the agent's policy is. When analysing the vehicles velocity throughout its trajectory, exceedingly slow or dangerously quick maneuvers can also be identified.

4.3 Training

Following a similar approach as the ones utilized in [38], in order to improve the training efficiency of our agents a series of training phases are employed through transfer learning [58], transferring knowledge acquired from initial simple tasks to accelerate the learning process in subsequent tasks. Through this curriculum learning approach the agent starts to explore the vehicles controls from scratch in a simple parking environment, and once it has mastered how to follow a specific set of paths, its capabilities are trained further with a bigger and more diverse set of paths. The paths from every initial position to every goal parking spot are computed and stored in memory in the beginning of every training phase, being then used throughout the episodes. This approach was chosen because the process of computing each individual path from initial to final position while avoiding obstacles is costly, and calculating a new path for each new episode would drastically increase the training time. In this manner the collection of paths to be used in each training phase can be calculated and stored beforehand, and during the actual training phase the corresponding path from initial to final position can quickly be loaded at the beginning of each episode. It is important to note that, despite being trained with paths that start and end with equal initial and final positions, at the beginning of the training processes of each of the three DRL algorithms, new paths are calculated for each training process, and because the paths are generated through a rapidly-exploring random tree (RRT*), each of the paths generated for each of the DRL algorithm's training may vary in their shape and length despite having equal initial and final positions, with slightly longer paths being able to offer a slightly bigger total cumulative reward than shorter ones. Despite this, the average length of the paths used in each of the agent's training is consistent across multiple runs, and as such it does not significantly impact the comparison of total cumulative reward between algorithms.

4.3.1 Training Phase 1

In the initial training phase the agent has no knowledge of how its interactions affect the vehicle's behaviour, it is therefore trained in a simple environment, a parking lot with one aisle that is 40 meters long and 9 meters wide, and only one column of goal parking spaces, resulting in a total of two parking spots, one on each side of the aisle, that are randomly sampled as the end goal in each

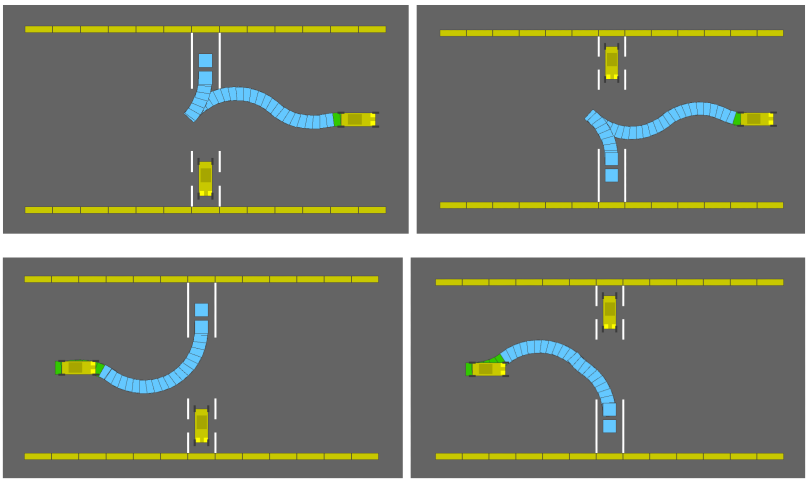

Figure 4.2: Four examples of paths generated for training phases one and two

training episode. This environment is depicted in Fig 4.2. The vehicle starts from two possible starting positions at both ends of the parking lot aisle, with its initial orientation set to zero ($\alpha = 0$), heading east on both occasions, creating a total of 4 different initial/final position pairs, and paths. When the vehicle's initial position is set on the east side of the parking lot, the predefined path towards the goal parking spot must include at least one cusp, as from that position it is impossible to achieve a feasible path without a direction change in these conditions, ensuring that this type of maneuvers are always included in the vehicle's training from the beginning. Each episode is limited to 10 seconds or the vehicle's success or crash against any obstacle. The training lasts 400000 time steps, which in our experiments corresponds to approximately 3500 episodes, depending on the frequency of crashes and successes during training. The details regarding training phase one's environment simulation are summarized in table 4.1.

Description	Value
Number of parking lot columns	1
Number of parking lot rows	2
Total number of parking spots	2
Number of vehicle initial positions	2
Total number of paths	4
Time limit per episode (s)	10
Training duration (Time steps)	400000

Table 4.1: Training phases one and two environment simulation parameters

4.3.2 Training Phase 2

Training phase two is carried in a similar environment to phase one, but with newly calculated paths from initial positions to the goal parking spots, as represented in Fig 4.2. This second training phase serves as a countermeasure to overfitting before advancing into a more demanding environment in the next phase, as previously each agent has only trained with four distinct paths, which is not a representative sample of every possible path that the agent might encounter, leading the agents to have adjusted their policy to work specifically with the previous four trajectories. It also serves as an observation of the training performance of the agents in a simple environment after having already learned path following from a restricted sample of paths in the past. The details regarding training phase two's environment simulation are summarized in table 4.1.

4.3.3 Training phase 3

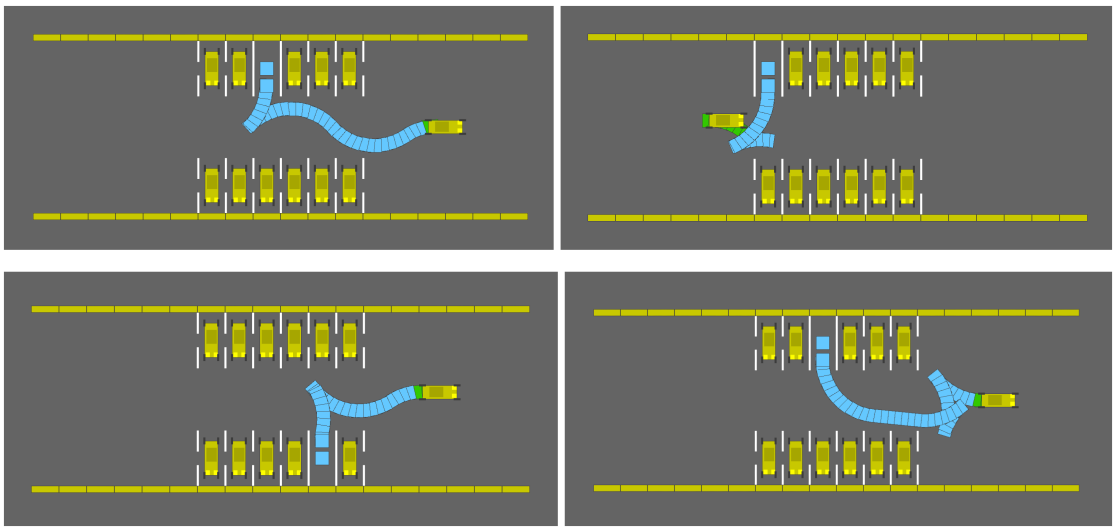

Figure 4.3: Four examples of paths generated for training phase three and the testing phase

In this phase the agent advances to a more realistic parking lot layout. The environment consists of an aisle, 40 meters long and 9 meters wide, and a row of parking spaces on each side. Each row has 6 columns of parking spots, for a total of 12 in the parking lot. This environment is illustrated in Fig 4.3. The agent has 12 possible initial positions, all of them starting with the vehicle pointed in the east direction ($(\alpha = 0)$), 6 of them near the east side of the aisle, and the other 6 on the west side. This brings the total number of paths to 144 possible path trajectories during training, ensuring a diverse and complete pool of possible paths from which the agent can train on, including overlapping sections, cusps, straight lines and tight turns. Here we can observe how each agent evolves when presented with a broader and more diversified set of paths, aiming to train them to be able to handle the any type of valid path between two positions. Due to this path diversity, the time limit of each episode is extended to 20 seconds, to accommodate

Description	Value
Number of parking lot columns	6
Number of parking lot rows	2
Total number of parking spots	12
Number of vehicle initial positions	12
Total number of paths	144
Time limit per episode (s)	20
Training duration (Time steps)	1500000

Table 4.2: Training phase three environment simulation parameters

longer paths with direction changes that take more time to be completed. This training phase lasts 1500000 time steps, corresponding to approximately 12000 episodes when there is a high rate of success, or 7500 episodes if the predominant episode ending condition is the time limit. The details regarding training phase three's environment simulation are summarized in table 4.2.

4.4 Training Phases Results Analysis

In this section, the results obtained from each training section are analysed, trying to get some insight to how each agent handles each training phase and and the learning progress throughout.

4.4.1 Training phase 1 results analysis

The learning evolution during training phase one of each of the three agents being trained through SAC, DDPG and TD3 is described in Fig. 4.4, 4.5 and 4.6, namely regarding total cumulative reward, average lateral distance from the path, and success rate.

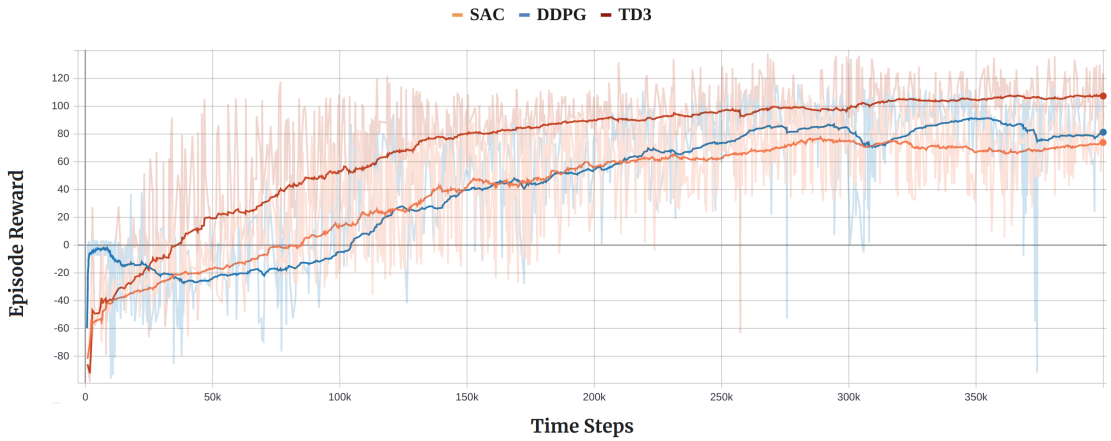

Figure 4.4: Reward achieved in each episode of training phase one by each of the three DRL algorithms

In Fig. 4.4 we can observe that through all three DRL algorithms the agent is able to learn how to extract reward from the environment, increasing their reward collection throughout training. In this initial phase TD3 is the fastest and most stable learner, being quickly able to end each episode with a positive reward. In just 100000 time steps TD3 is already averaging rewards close to 60 values, while SAC is still barely able to extract positive rewards in each episode, and DDPG is still averaging negative rewards from occasionally being penalised for traveling in the wrong path direction. In its initial steps the DDPG agent seams to take a very conservative approach, hardly exploring the environment, with the agent barely moving away from its initial position before crashing against an obstacle or running out of time, avoiding being penalised for moving in the wrong direction, a behaviour that is not present in the initial steps of SAC or TD3, where negative rewards are explored and episodes with values of -80 are recorded. This can also be observed in 4.5, where for the first 708 episodes the DDPG angent barely explores actions that lead the vehicle away from the path. With further training evolution, TD3 maintains a consistent lead when it comes to reward extraction, being the fastest to converge towards the optimal policy, and at the end of the training phase, TD3 is able to achieve results with an average reward of more than 25 values above both SAC and DDPG.

Fig. 4.5 represents the average lateral distance (d_{Lat}) from the center of the vehicle's rear axis to the path, on each episode of the first training phase.

Here we can observe that through the SAC algorithm, the agent is able to progressively achieve a lower average lateral distance from the path, indicating that it follows the predefined paths more closely, and at the end of the training phase, considering the last 50 episodes, it averages lateral distances of less than $0.25m$ with a variance of $0.00675m$.

The DDPG agent remains very conservative for the first 708 episodes, not exploring actions that lead it farther away from the path. Only then the agent explores other action ranges and progressively tends to follow a trajectory closer to the predefined path, although its learning process is much more unstable than that of SAC or TD3, not being able to achieve such consistent results. DDPG's instability may be derived from the lack of the stability features found in TD3, leading to the overestimation of Q-values and disturbing the decision making process of the agent, previously described in section 2.5.2, as its twin delayed counterpart achieves much more stable results. In the last 50 episodes DDPG averages distances of $0.21m$ with a variance of $0.00311m$.

TD3 achieves the closest vehicle trajectories of the three DRL algorithms, consistently averaging results better than the best results obtained by SAC and DDPG during the whole training phase in under 1250 episodes, and finishing training phase one while averaging only $0.05m$ of average lateral distance from the path, with a $0.00026m$ variance.

Fig. 4.6 presents the success rate achieved by each DRL algorithm considering the past 100 episodes in training phase one. A different number of episodes is run for each algorithm because this training phase runs for 400000 time steps regardless of the number of successes or collisions verified, which immediately terminate the episode and as such decreases the average number of time steps recorded in each episode. In this metric we can observe that TD3 once again outperforms SAC and DDPG, although all three DRL algorithms show improved performance

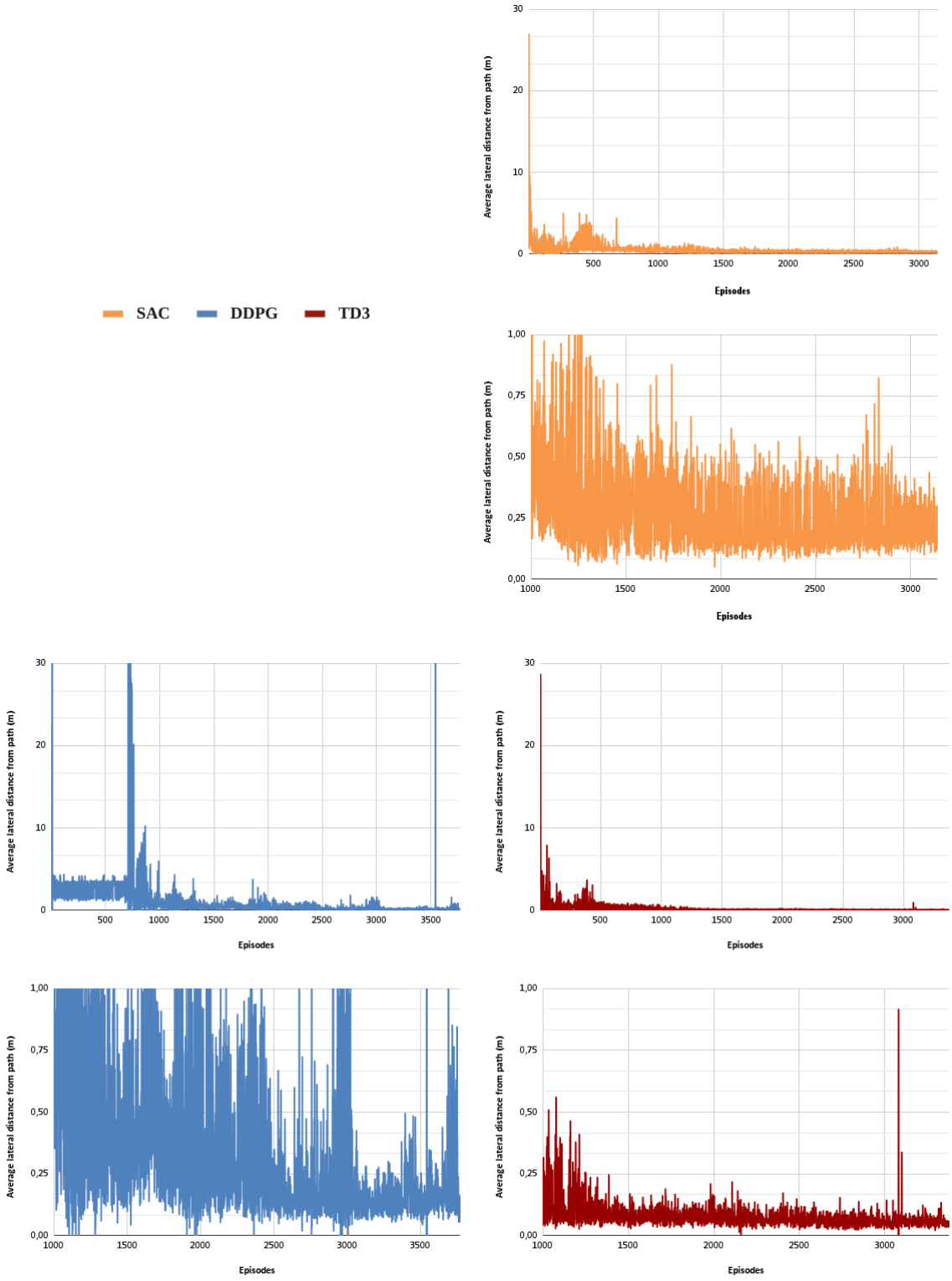

Figure 4.5: Average lateral distance from path by episode of phase one of training using the SAC, DDPG and TD3 algorithms

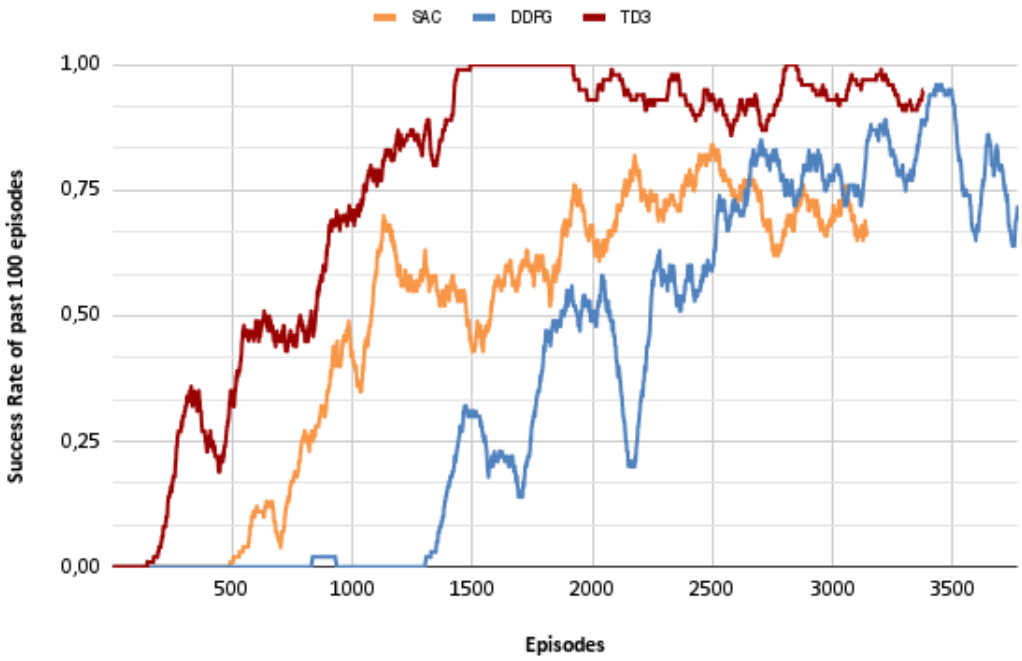

Figure 4.6: Success rate considering the past 100 episodes during phase one of training, using the SAC, DDPG and TD3 algorithms

throughout training.

The agent being trained through the SAC algorithm achieves its first successfully complete trajectory after 490 episodes, and shows a rapidly improving success rate right after. Despite not being able to achieve a perfect success rate, the agent shows improvement in its policy, recording a maximum success rate of 84% after 2500 episodes.

The DDPG agent records the most unstable success rate of the three DRL algorithms throughout training. It is also the last one to reach its first successful trajectory after 830 episodes, while only being able to surpass 25% success rate after 1440 episodes. Despite this, the DDPG agent manages to achieve a maximum success rate of 96% after 3429 episodes, before rapidly coming back down to values floating around 75% success rate while trying to optimize its policy.

TD3 reaches its first successful trajectory after only 145 episodes, and quickly improves its success rate right after, going from 0% success rate to 100% in just a 1345 episode interval, at episode 1490. It is then able to maintain a high success rate while optimizing its policy, recording a minimum of 86% after initially achieving 100 consecutive successful episodes.

4.4.2 Training Phase 2 results analysis

The learning evolution during training phase two of each of the three agents being trained through SAC, DDPG and TD3 is described in Fig. 4.7, 4.10 and 4.11

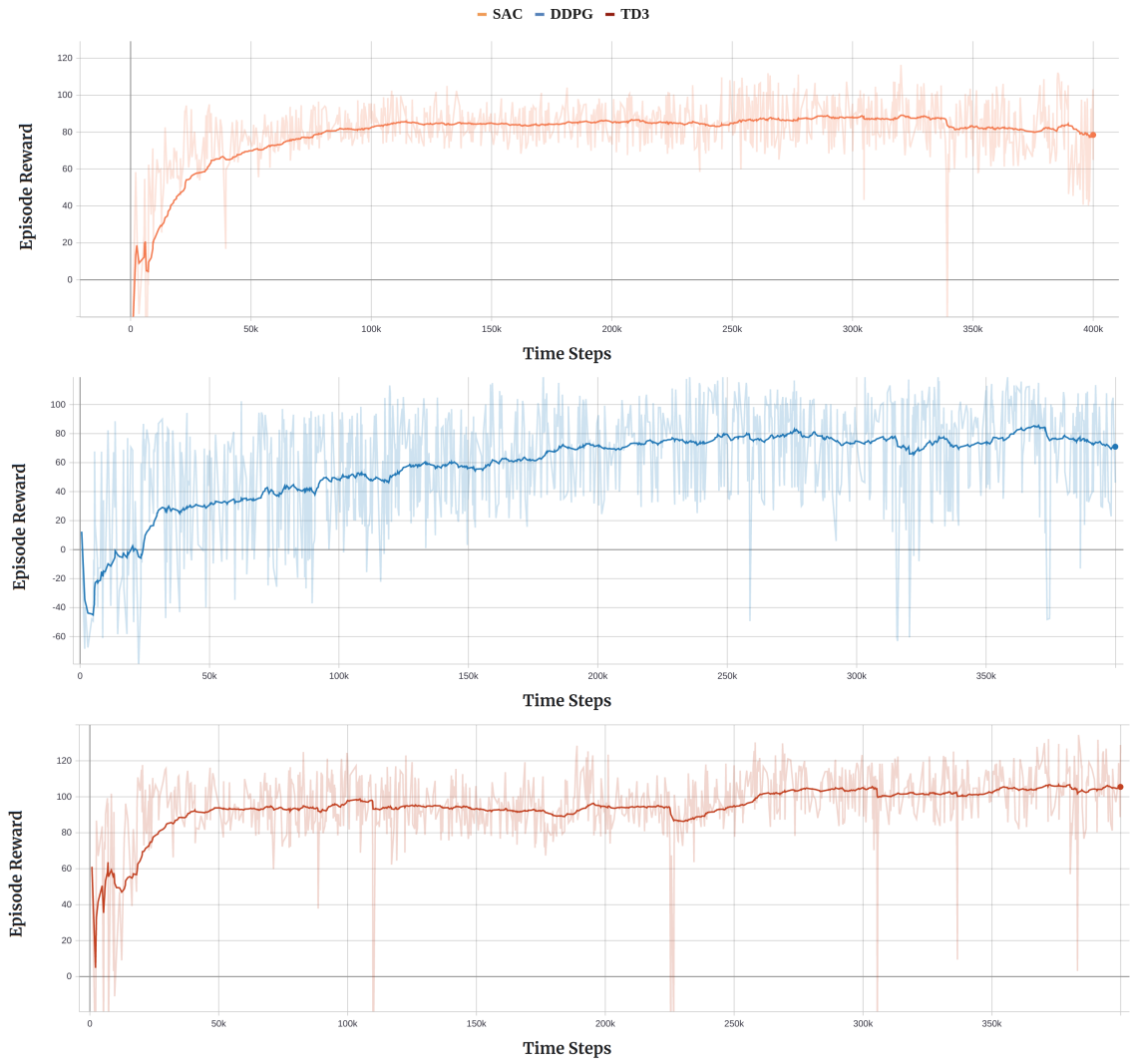

Figure 4.7: Reward achieved in each episode of training phase two by each of the three DRL algorithms

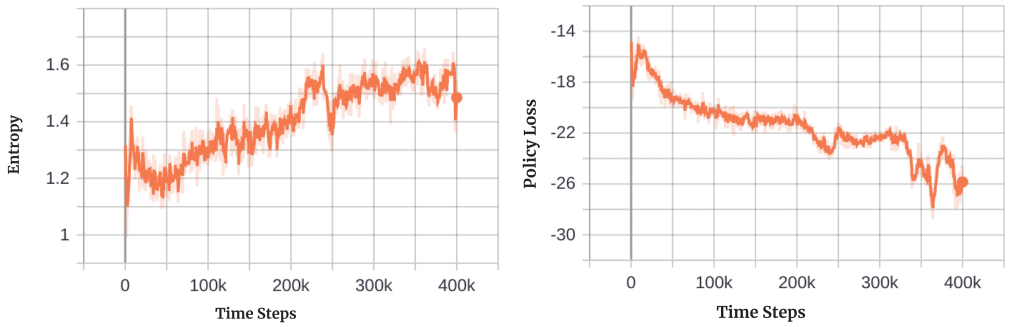

Figure 4.8: Entropy and Policy Loss parameters of SAC algorithm throughout training phase two

In Fig. 4.7 we can observe the total cumulative reward achieved by the agents in each episode throughout training in this similar environment with new paths. We can observe that both SAC and TD3 algorithms are able to quickly adjust their previously learned policies to extract a greater reward in these new paths, SAC reaches a consistent average reward extraction of more than 80 values in 100000 time steps, and the TD3 agent is able to consistently extract an average of 95 values from the environment in less than 30000 time steps. However, this quick adaptation to the new environment paths is not reflected in the DDPG agent, where difficulties in reward extraction during the initial steps and a slow and unstable progressive evolution in reward extraction is verified.

The agent being trained through SAC hits a stagnation point in its reward extraction improvement roughly between time steps 125000 and 200000, this stagnation being also verified in the policy loss recorded in the same time step interval, presented in Fig. 4.8, which indicates that the process for deciding actions is not changing significantly. This later leads to a spike in the entropy of the learning process at around the 210000 time steps mark, also presented in Fig. 4.8, increasing the randomness of the agent's actions while training, for a broader exploration of how the agent can be able to improve its policy. Until the end of training phase two the entropy value does not decrease significantly, revealing difficulties on how to improve the policy further. This increase in the exploration factor on the later half of SAC's agent's training can also be verified in its more extreme total cumulative reward extraction in Fig. 4.7, achieving both higher and lower rewards than previously, the behavioural instability recorded in its average distance to the path in Fig. 4.10 and its varying success rate in Fig. 4.11. In the end of training phase two, the SAC agent is averaging a total cumulative reward of approximately 78 values, although before the increase in the entropy factor it was able to maintain a consistent average of 87 values.

DDPG's agent seams to be the one that has most trouble adapting to the newly defined paths. In its initial episodes DDPG has difficulties regaining the reward collection abilities acquired with the initial paths in the first training phase, averaging negative rewards for the first 25000 time steps and achieving the most unstable reward collection progression of the three algorithms, with consecutive training episodes achieving differences in total cumulative reward of more than 80 values, while SAC and TD3's tend to be half as polarizing. Similarly to what was found in the first training phase, DDPG's reward collection capabilities seem to once again reach a maximum average of 80 values, in the second training phase after 250000 time steps, and with further training DDPG is not able to improve its policy in a consistent manner, finishing training phase two achieving an average cumulative reward of 71 values.

TD3 once again is the fastest to converge and the best performer of the three algorithms at extracting reward from the environment. It is quickly able to adjust the previously learned policy, trained using the paths from the previous training phase, to achieve great and consistent results with new paths. After being able to consistently achieve average reward values greater than 95, close to the 30000 time steps mark, TD3 tries to improve its policy through small adjustments, as can be verified from the stabilization of the recorded policy loss in Fig. 4.9, achieving a stable and reliable improvement of its policy. TD3's agent finishes training phase two achieving an average

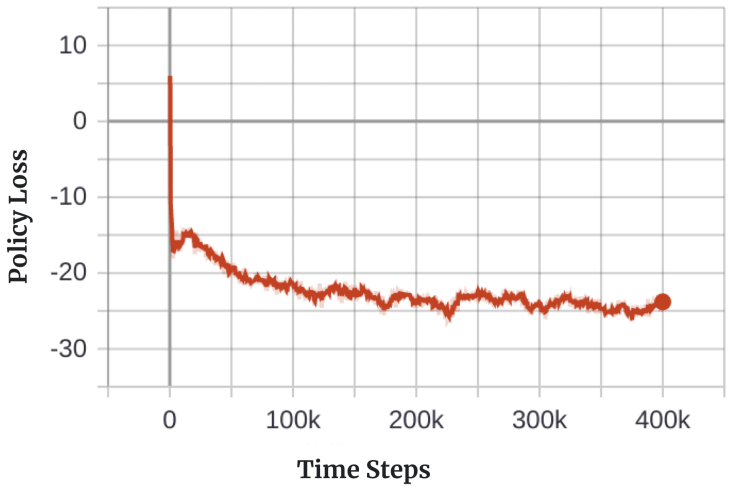

Figure 4.9: Policy Loss parameter of TD3 algorithm throughout training phase two

total cumulative reward per episode of 105 values.

When it come to the lateral distance from the path, observable in Fig. 4.10, TD3 once again surpasses both SAC and DDPG. Despite this, SAC and DDPG's agents both manage to decrease the average distance from the path recorded in the first training phase. At the end of training phase two, SAC is averaging a lateral distance from the path of $0.12m$ and a variance of $0.00159m$, compared to an average of $0.20m$ and variance of $0.00675m$ in the final 50 episodes of training phase one. Moreover, before its entropy increase around episode 2100, it was averaging a stable distance of just $0.10m$ from the path, with a variance of just $0.00094m$. DDPG on the other hand starts the second training phase with some episodes where the average distance from the path is greater than $30m$ and even $40m$, before achieving a more stable average of $0.11m$ with a variance of $0.00636m$ in its final 50 episodes. Even with SAC's entropy increase, DDPG still registers as the most unstable learner when considering its distance from the path, averaging distances of more than $3m$ even after 2250 episodes of training on the same four paths. TD3 maintains the closest trajectories to the path, averaging $0.05m$ from the path with a variance of $0.00034m$ in its last 50 episodes, an equal result to its first training phase trajectories, less than half the distances averaged by SAC and DDPG.

Similarly to the other performance metrics, DDPG also achieves the worst performance when it comes to success rate, presented in Fig. 4.11, and because of its lower success rate, it also averages longer episodes since a larger part of them terminate via the time limit of the episode and not the success condition, resulting in fewer training episodes. SAC is able to achieve 100 consecutive successful episodes in these 4 new paths in just 328 training episodes, and TD3 follows shortly after at episode 541. Meanwhile DDPG struggles to achieve a success rate higher than 75%, never reaching more than 81 successful episodes. After reaching high levels of success rate, SAC maintains the most stable success metric, adjusting its policy while barely deteriorating the success of its training episodes, only dropping bellow 95% at the very end of training while

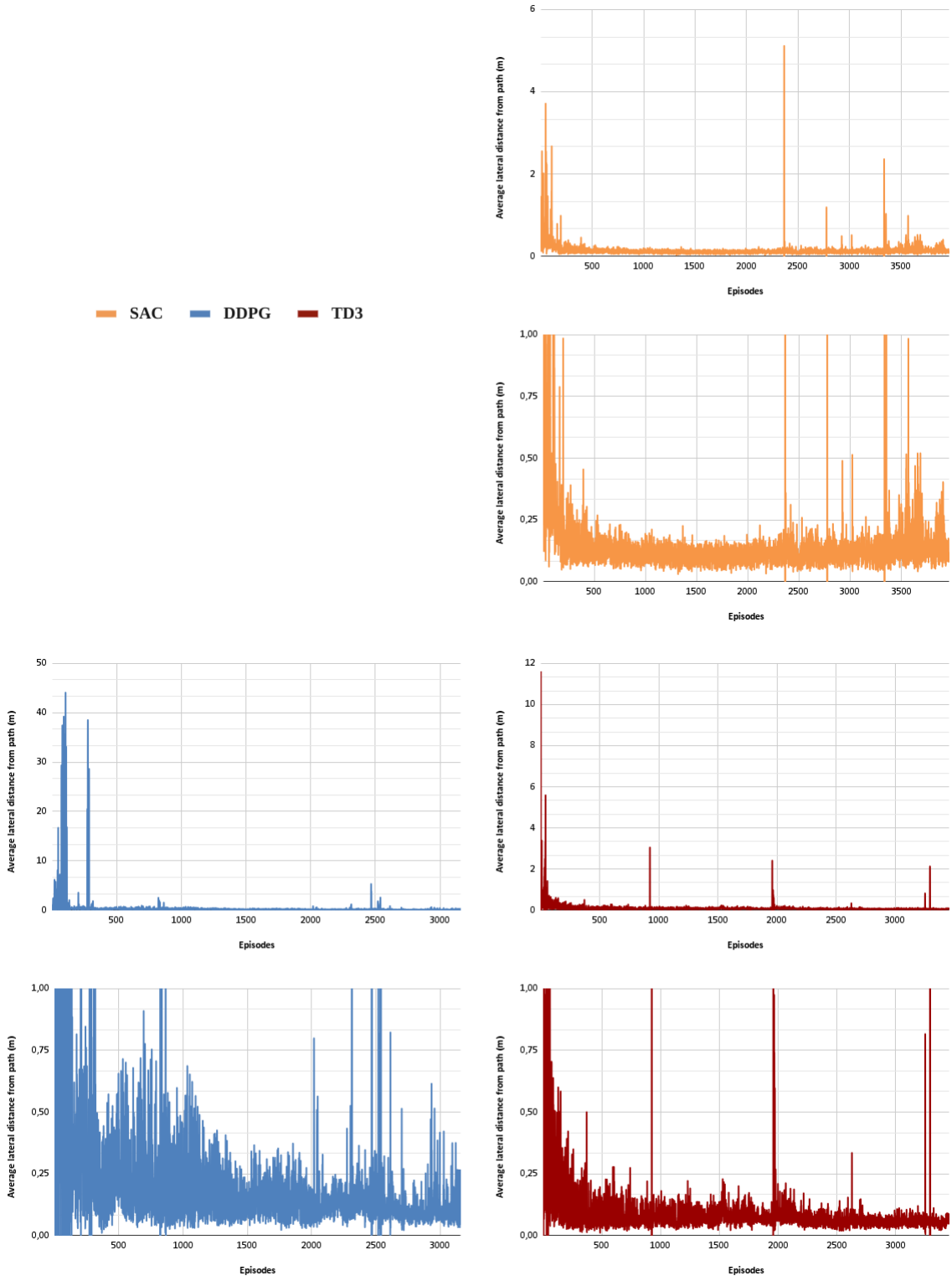

Figure 4.10: Average lateral distance from path by episode of phase two of training using the SAC, DDPG and TD3 algorithms

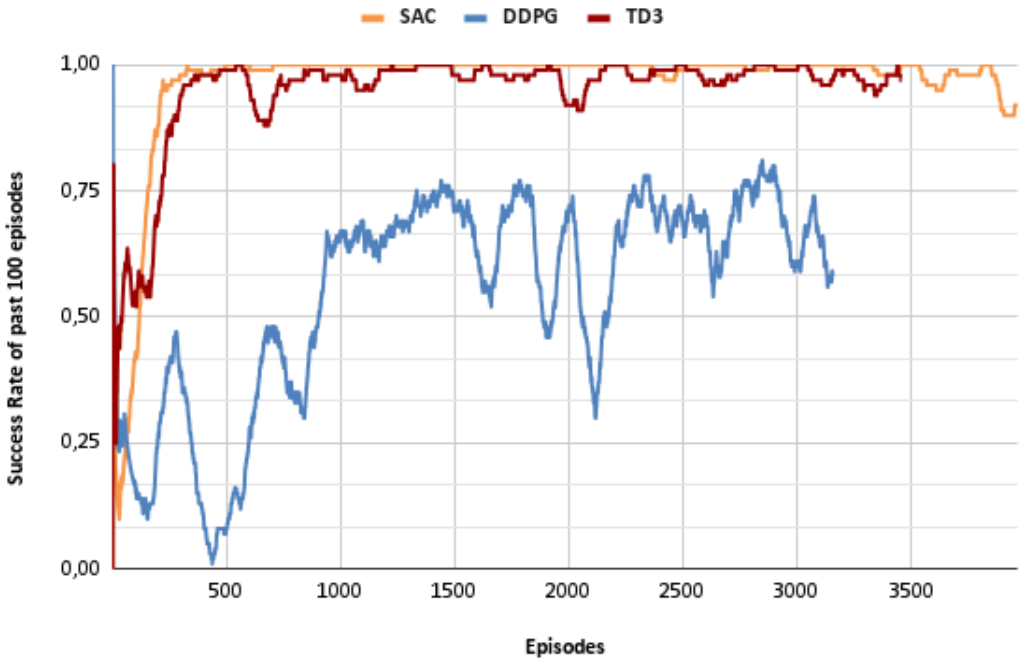

Figure 4.11: Success rate considering the past 100 episodes during phase two of training, using the SAC, DDPG and TD3 algorithms

its entropy was registering the highest recorded values of the whole training phase. TD3 also measures a stable success rate after the initial episodes, only dropping to a minimum of 91% while improving its policy. On the other hand, DDPG reaches maximum success rate values even lower than the ones registered in the initial training phase, 81% when compared to a maximum 96% registered in training phase one. Having its success rate float between 55% and 81% in the final 1000 episodes, possibly due to a more complex set of paths computed in the second training phase, but also to DDPG's unstable learning process in this environment.

4.4.3 Training phase 3 results analysis

The learning evolution during training phase three of each of the three agents being trained through SAC, DDPG and TD3 is described in Fig. 4.12, 4.13 and 4.14

Fig. 4.12 presents the total cumulative reward extracted by each of the DRL algorithm's agents throughout training phase three. In this more realistic environment the differences between the three DRL algorithms verified in the previous training phases are accentuated even further. Although it takes more time to train a single policy to handle a set of paths so diverse, all three algorithms show policy improvements throughout training, with varying degrees of success. It is also natural that the divergence in total cumulative reward registered between consecutive episodes is amplified, as some paths have lengths as short as $10m$ while others extend for more than $40m$, providing a much higher reward extraction potential. When beginning training in this diverse pool

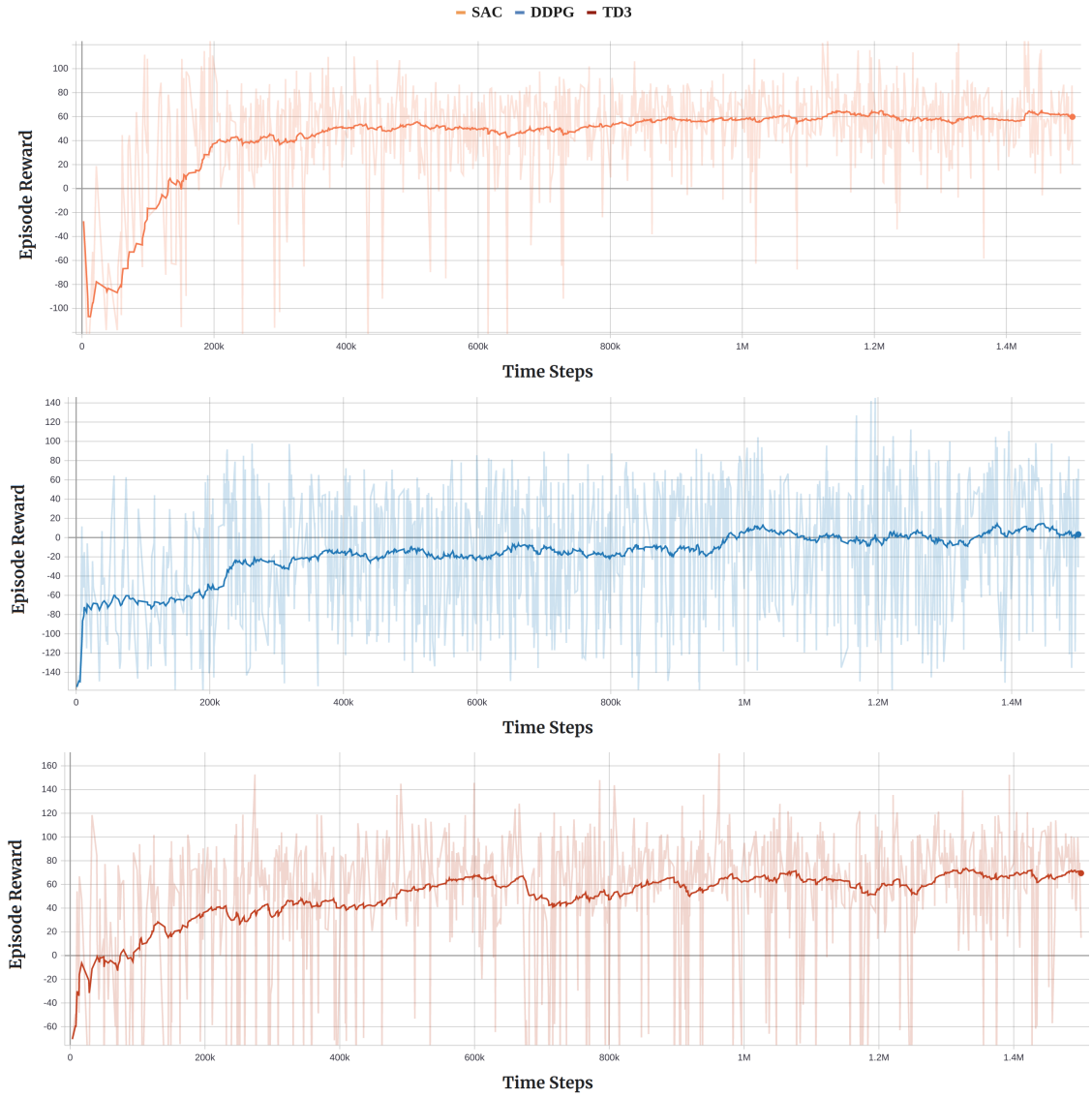

Figure 4.12: Reward achieved in each episode of training phase three by each of the three DRL algorithms

of paths all three algorithms show varying degrees of difficulties when extracting reward, as completely new path segments are introduced to the agents for the first time. SAC's agent requires approximately 200000 episodes before achieving a consistent positive reward, averaging 40 values at this point, this being also where the biggest policy improvement is verified. After the initial 200000 time steps SAC performs more subtle improvements , finishing training phase three while averaging a total cumulative reward of 60 values. DDPG is also challenged by the new and diverse set of paths, but contrary to SAC and TD3's agents, DDPG's agent improves its reward extraction capabilities at a much lower pace, only averaging positive rewards after almost 1000000 time steps and barely improving past this point, finishing training phase three while averaging a total

cumulative reward of 3 values. TD3 once more is able extract the most reward of all three DRL algorithms, quickly averaging positive rewards in just under 100000 time steps, and progressively improving its policy until it was able to average rewards of more than 70 values.

In Fig. 4.13 we can observe the average lateral distance from the path of the three DRL algorithms, recorded throughout training phase three. Here SAC and TD3 show a clear improvement throughout training, while DDPG's improvement is much more modest and unstable. SAC's agent consistently reduces its average lateral distance from the path throughout training, averaging, in this diverse set of paths, a distance of 0,28m with a variance of 0.01930m in the last 100 episodes recorded. DDPG's agent on the other hand is much more inconsistent, regularly averaging distances of more than 5m or even 10m in some episodes, including instances in the last episodes of training, showing an inability to closely follow a diverse set of predefined paths, even after 1500000 time steps of training. DDPG averages a distance of 2,01m with a variance of 217.8953m from the path in the last 100 episodes of its training, with the vast majority of its recorded values fluctuating between 0,25m and 1,50m. TD3 once again averages the shortest distance from path, from the beginning of the training phase until its last steps. Despite recording some instances where its average distance is greater than 0,50m, even including some in the last training episodes, the vast majority of its final episodes record an average distance between 0,30m and 0,05m from the path, with the last 100 episodes average recorded distance from path registering only 0,19m and variance of 0.02230m, a promising result considering the diversity and complexity of the paths utilized for this training phase.

When it comes to successful path runs, TD3 maintains its dominance while DDPG falls short of its objective, with SAC closely following TD3's accomplishments. DDPG continues averaging the longest episodes, resulting in more time steps spent per episode and a lower number of total training episodes. SAC is able to quickly achieve a success rate of 50% in its first 493 episodes, progressively increasing its success rate throughout training, until its improvements stagnate after around 7500 episodes, achieving a maximum of 94% while tending to float between 83% and 93% during its final training episodes. DDPG tends to improve its success rate during its first 5000 training episodes in training phase three, reaching a peak success rate of 68%, but it is unable to improve its policy further with more training, unstably balancing its success rate between 20% and 60% during its last 2500 training episodes. TD3 rapidly averages success rates above 75% within just 272 episodes of training with this vast set of paths, and seems to be the only one capable of consistently improving its success rate during the 12139 episodes of its training, showing a new maximum success rate after each couple of hundred episodes. TD3 reaches a maximum of 99% success rate, consistently staying above 93% success rate during its final episodes. The small percentage of paths where the agent does not achieve successful runs may be due to some particularly tricky and not well optimized paths generated, where overlapping sections of the path extend for prolonged distances with the same heading but inverted directions, making it impossible for the agent, with its current observation, to distinguish which way it should move. This type of path is illustrated in Fig. 4.15

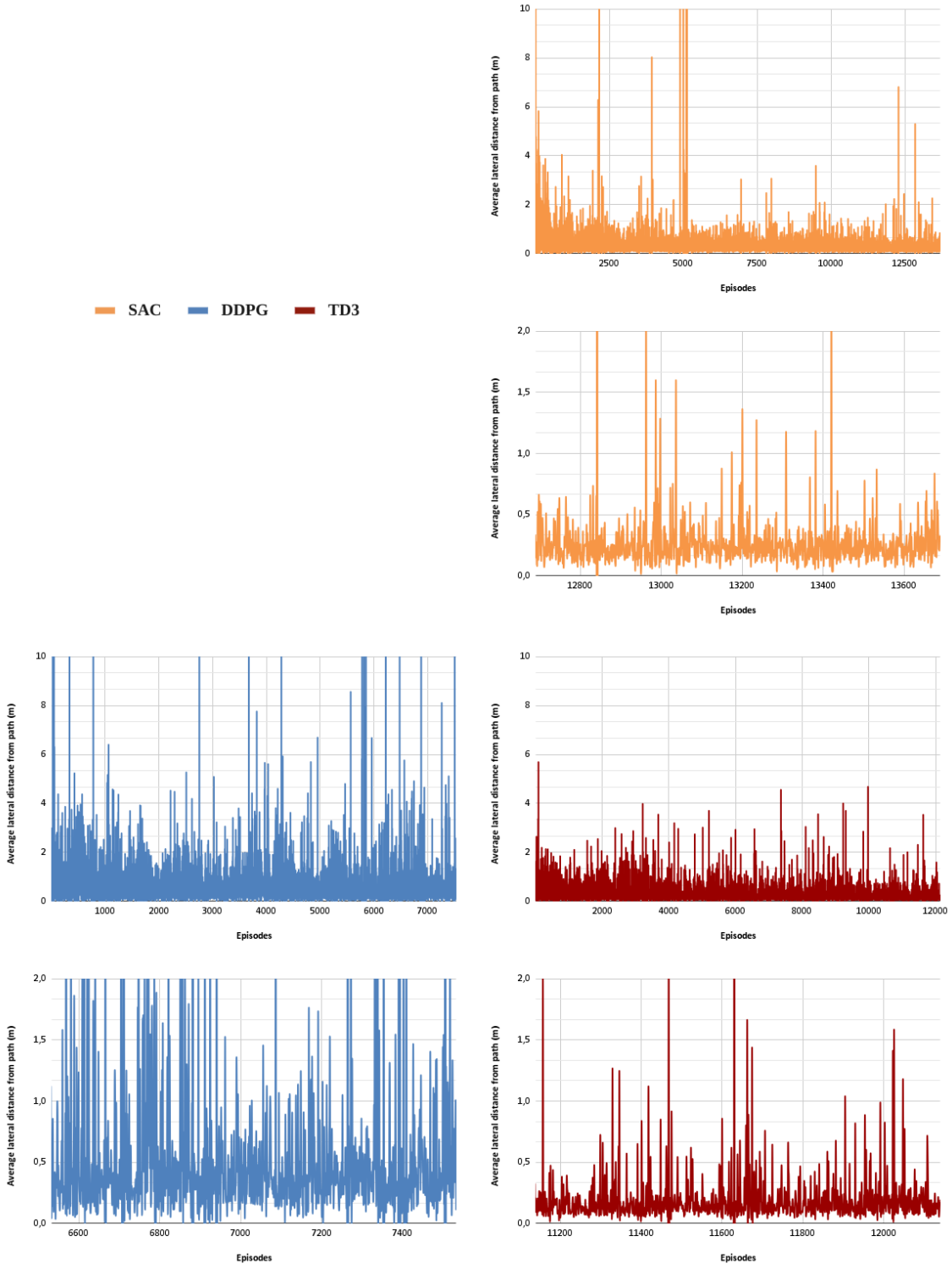

Figure 4.13: Average lateral distance from path by episode of phase three of training using the SAC, DDPG and TD3 algorithms

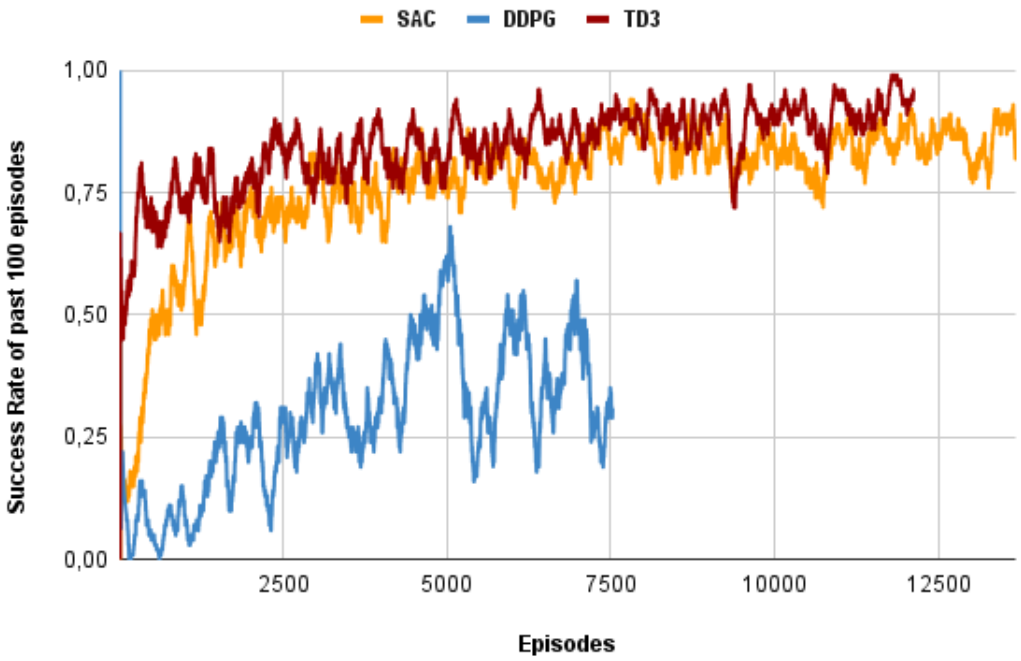

Figure 4.14: Success rate considering the past 100 episodes during phase three of training, using the SAC, DDPG and TD3 algorithms

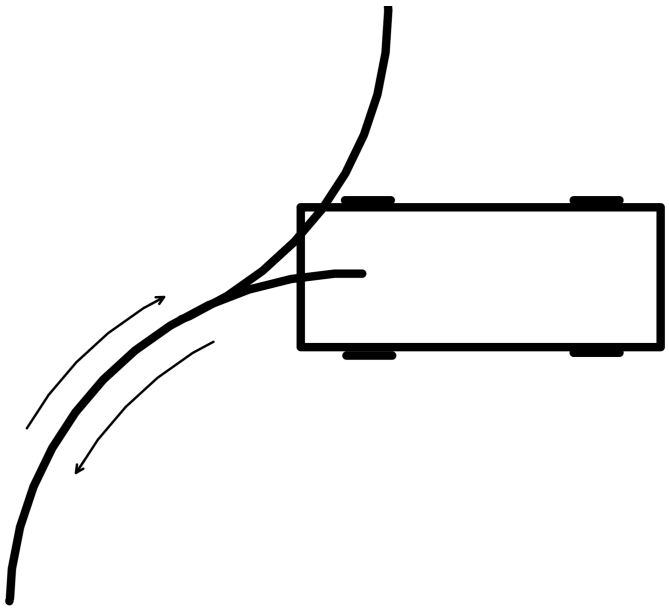

Figure 4.15: Challenging type of path with prolonged overlapping sections with the same heading but inverted directions

Description	Value
Number of parking lot columns	6
Number of parking lot rows	2
Total number of parking spots	12
Number of vehicle initial positions	12
Total number of paths	144
Time limit per episode (s)	20
Testing duration (Time steps)	50000

Table 4.3: Testing phase environment simulation parameters

4.5 Testing Phase Results

4.5.1 Testing environment

The testing phase was conducted in a similar environment to that of training phase three, depicted in Fig 4.3. The same 40 meters long aisle, 9 meters wide, with a row of parking spaces on each side is used to generate 144 new paths that the agent has never had any prior experience with, from which tests are conducted during 50000 time steps, which may lead to a different total number of episodes being recorded for each agent due to varying average duration of the episodes when crashing or being successful. Details describing the testing environment are summarized in 4.3

4.5.2 SAC's agent performance review

After completing the three training phases, when SAC's agent encounters new paths, it achieves a somewhat consistent path following behaviour, rarely moving more than $1m$ away from the predefined paths introduced in the testing phase. As we can observe in Fig. 4.16, in most of its runs, the agent trained by SAC averages lateral distances from the path of less than $0{,}30m$, with only a few rare exceptions distancing over $0{,}80m$. In this previously unknown set of paths, the vehicle controlled by SAC's agent averages a lateral distance of $0{,}34m$ with a variance of $0.26792m$.

SAC's agent also achieves a success rate of 84% during testing, following the path closely in most of its runs, but being disoriented in some rare occasions, ending up distancing the vehicle from the path and eventually colliding with nearby obstacles. Throughout its testing episodes SAC's agent averages a total cumulative reward of 56 values. The results obtained by SAC's agent during the testing phase are summarized in table 4.4

A successful run performed by SAC's agent is detailed in Fig 4.17, representing the vehicle's lateral distance from the path (d_{Lat}), the absolute angle error between the vehicle's heading and the corresponding path section (φ), and its velocity (v) throughout the trajectory. Here we can observe that despite some instances where the vehicle progressively drifts away from the trajectory, reaching lateral distances in the range between $0{,}25m$ and $0{,}40m$, the agent manages to recenter the vehicle's rear axis back on the path, as spends most of its trajectory within $0{,}15m$ from the

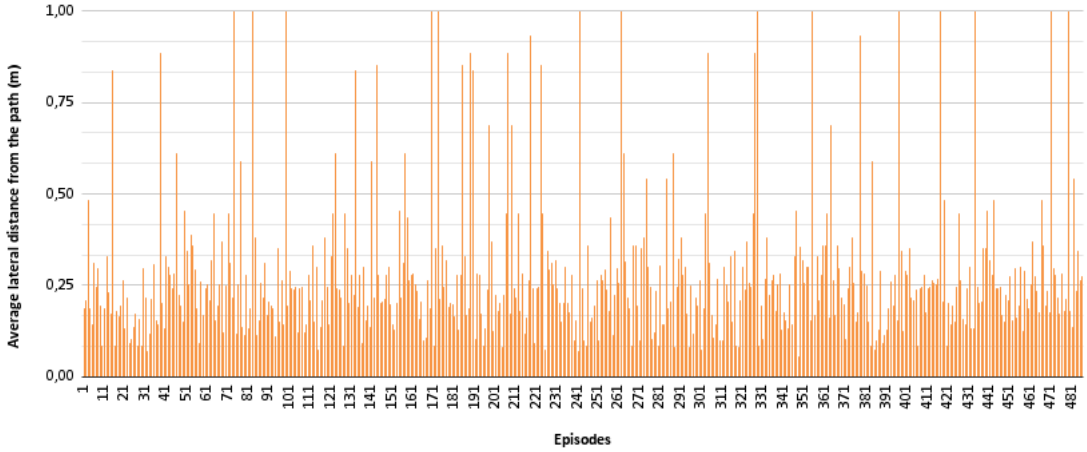

Figure 4.16: Average lateral distance recorded in each episode of testing with SAC's agent

path. The unstable nature of the recorded angle difference between the vehicle and the path's heading is due to the constant shift from the current reference point in the path, to the next, where the agent detects an increase in the angle difference and rapidly corrects it before advancing to the next reference point. It is consistently kept below 0,2 radians during cornering shifts in path point references and close to 0 when in straight lines. The vehicle's velocity also consistently surrounds $1m/s$, velocity above which the agent is able to extract the maximum amount of reward, with the only exception being when there is the need to stop and change direction at the cusp in the path.

4.5.3 DDPG's agent performance review

DDPG's agent is not able to perform reliable maneuvers when encountering new paths, recording the largest lateral distance from the paths and sometimes even drifting away from the path by several meters, as can be verified in Fig. 4.18. The average lateral distance from the path recorded by the DDPG's agent is 1,74m and its variance is 106,2410m, with most of its episodes recording an average distance of more than 0,30m from the path, a considerable distance that does not ensure collision avoidance with nearby obstacles and does not guaranty predictable path following behaviour, resulting in a success rate of only 21% and an average total cumulative reward collection of -31 values. The results obtained by DDPG's agent are summarized in table 4.5

Description	Value
Success rate	84%
Average lateral distance from path (m)	0,34
Average total cumulative reward	56

Table 4.4: Results obtained by SAC's agent during testing phase

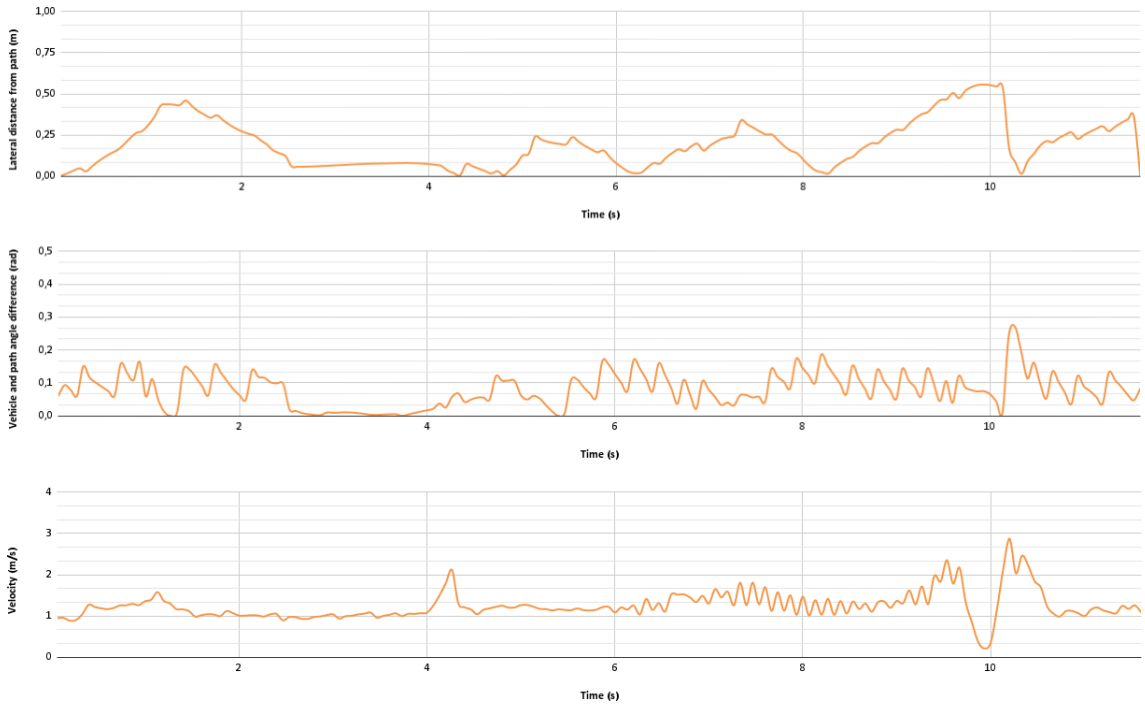

Figure 4.17: Lateral distance from the path (d_{Lat}), absolute angle error between the vehicle's heading and the corresponding path section (φ), and velocity (v) of the vehicle throughout a successful episode of testing with SAC's agent

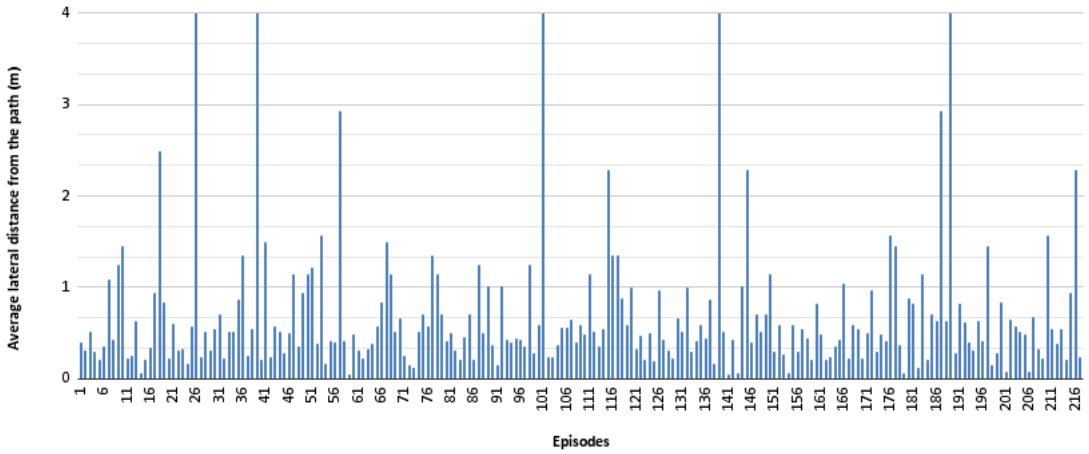

Figure 4.18: Average lateral distance recorded in each episode of testing with DDPG's agent

The worst results achieved by DDPG could be explained by the overestimation of its Q-values. Since DDPG only uses one Q-function and directly retrieves the Q-values from it, it has a tendency to overestimate its Q-values. SAC and TD3 solve this problem by utilizing two Q-functions and using the minimum of the two for policy updates.

Description	Value
Success rate	21%
Average lateral distance from path (m)	1,74
Average total cumulative reward	-31

Table 4.5: Results obtained by DDPG's agent during testing phase

An example of DDPG's agent trajectory is presented in Fig 4.19. The agent is able to maintain a relatively close path trajectory during the first second of the simulation, before the vehicle's angle of heading starts to distance itself from the path heading angle, where a sudden jump between path reference points occurs, which leads to a drastic increase in the vehicles lateral distance to the path. This type of behaviour is not desirable as it can disorient the vehicle and lead it through a non desirable trajectory. In this case however, the vehicle is able to regain the ability to get closer to the path before on its final moments starting to slowly drift away from the path and increasing its velocity before colliding with a stationary obstacle in the environment. This example run reflects the unpredictable behaviour expected from DDPG's agent, where the overestimation of Q-values may be the most disturbing factor in the decision making of the agent, following a non-optimal policy and leading to an unreliable path following methodology.

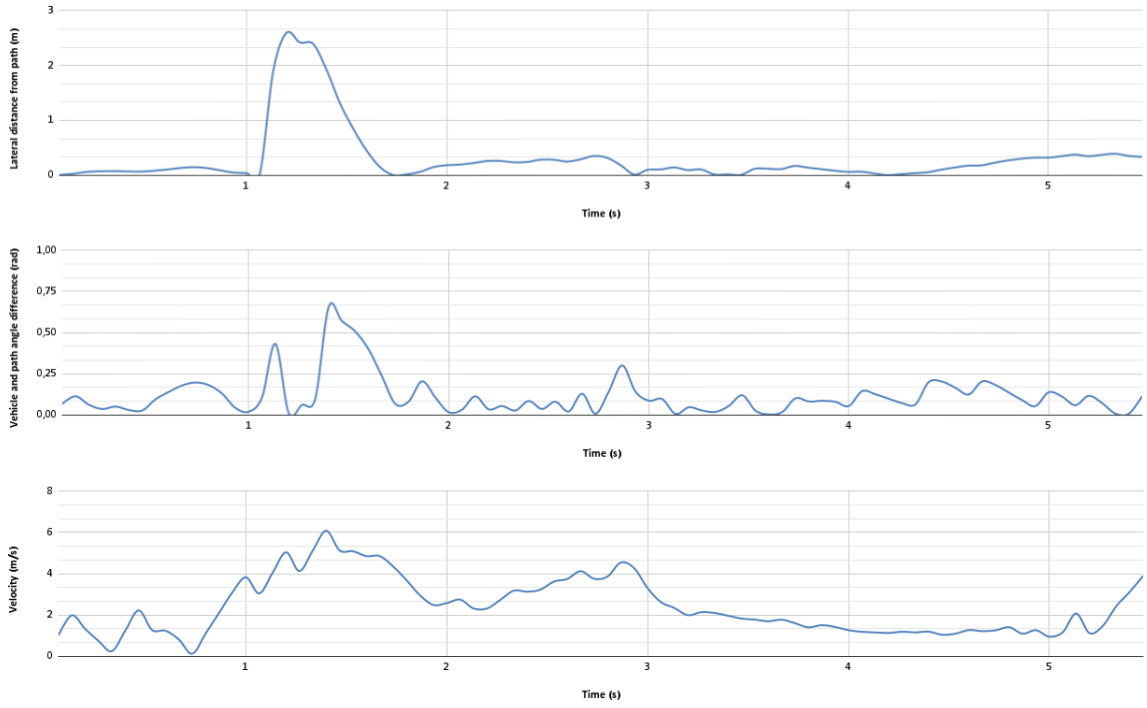

Figure 4.19: Lateral distance from the path (d_{Lat}), absolute angle error between the vehicle's heading and the corresponding path section (φ), and velocity (v) of the vehicle throughout a successful episode of testing with DDPG's agent

Description	Value
Success rate	93%
Average lateral distance from path (m)	0,20
Average total cumulative reward	63

Table 4.6: Results obtained by TD3's agent during testing phase

4.5.4 TD3's agent performance review

When TD3's agent encounters the newly created test paths, it is able to maintain a similar behaviour to the one observed when running on the training paths, although with a slightly less precise behaviour. TD3's agent maintains a small lateral distance from the path, as it can be seen in Fig. 4.20, averaging a distance of just 0,20m between the center of its rear axis and the path, with a variance of 0.02939m, and the vast majority of its testing episodes registering an average lateral distance of less than 0,15m, and some instances, approximately 7% of the episodes reaching averages of more than 0,50m.

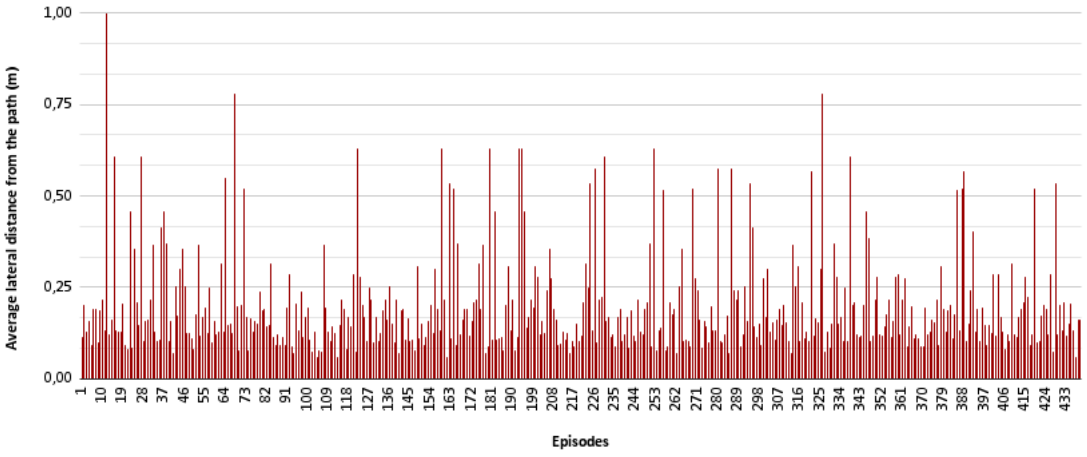

Figure 4.20: Average lateral distance recorded in each episode of testing with TD3's agent

TD3's agent is able to maintain an success rate of 93% in these newly drawn paths, with the majority of its failed attempts happening because of the specific type of unoptimized overlapping paths represented in Fig. 4.15, where the agent is not able to decide whether to move backwards or forwards from its observations. When encountering other sections of the path, such as straight lines, curves or cusps, the agent seems to have a similar performance between all of them, only accumulating less reward during cusps because of the need to reduce its velocity and change direction, staying at an average consistent distance from the path during cusps or other path sections.

Throughout its testing runs, TD3's agent also averages 63 values of total cumulative reward collection. The results obtained by TD3's agent are summarized in table 4.6

A successful run performed by TD3's agent is detailed in Fig 4.21. Here we can observe that the vehicle's maximum lateral distance from path, recorded at 0,41m, is verified while its velocity is at its peak (3,50m/s), before being promptly corrected closer to the path. During the cusp section of the path, registered during second 6 of the episode, where the agent must stop and change direction, the vehicle remains as close to the path as in previous sections, registering a maximum of 0,12m from the path during this maneuver. When it comes to the angle difference between the vehicle and the path's heading, TD3's agent consistently maintains it below 0,2 radians throughout the episode, the maximum recorded value, happening right after progressing to the next reference path point. As we can see from the recorded vehicle velocity, the only time it stops is in order to perform the direction change in the cusp of the path, 6,3s into the episode, reducing its velocity when approximating the cusp, and advancing in the opposite direction afterwards. Between other path sections the vehicle's velocity is kept between 1m/s and 3,5m/s.

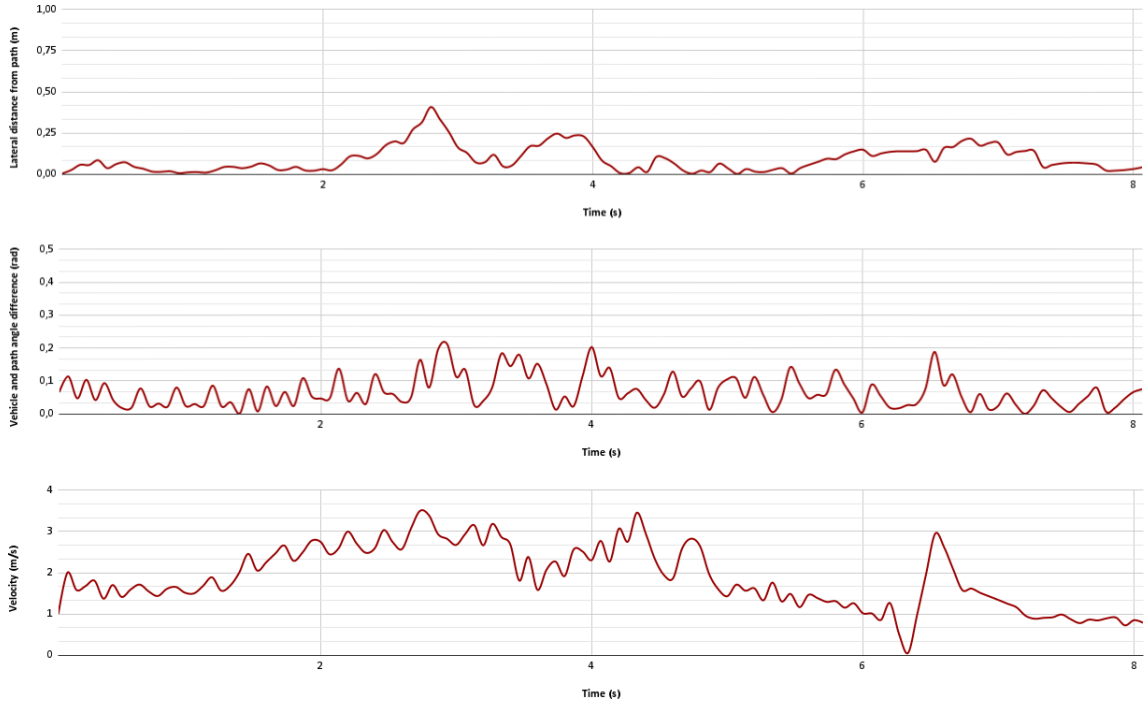

Figure 4.21: Lateral distance from the path (d_{Lat}), absolute angle error between the vehicle's heading and the corresponding path section (φ), and velocity (v) of the vehicle throughout a successful episode of testing with TD3's agent

Chapter 5

Conclusions

It was established that automated vehicles can drastically improve the parking routine and overall well being of people living in densely populated cities around the world, and the development of agents capable of following predefined paths inside parking lots has the potential to be the first step towards achieving a fully automated parking system where people are able to leave their vehicle on the parking lot entrance and let the vehicle go park on its own, later summoning it back when needed to.

During the development of this thesis it was concluded that the multidirectional characteristics of this type of paths lead previously developed path following methodologies based on optimal unidirectional acceleration or velocity values to unsuccessful results when encountering cusps in the path. Moreover, the traditional method of choosing the closes path point to the vehicle as its reference point in the path is not feasible when the paths involve overlapping sections, requiring the angle of the path to be taken into consideration in order to reliably progress through the path in these sections.

Throughout development, a great understanding of how DRL works, which types of algorithms are better suited for different tasks, the intricacies of the learning process and how it can be managed in order to achieve a fast convergence rate and improved results, were achieved.

In this use case, TD3 shows the fastest convergence rate out of the three reinforcement learning algorithms during all three training phases, and when compared directly to DDPG, the metrics recorded suggest that the improvements made in TD3, namely learning two Q-functions instead of one, updating the policy and target networks less frequently than the Q-function and adding clipped noise on each dimension of the action space, three characteristics not present in the original DDPG, do drastically improve the reliability and the overall learning ability of the agent.

TD3 also proves to generate the most reliable policy out of the three tested RL algorithms, closely following the predefined path and being able to complete 93% of the testing paths it is presented with, with the majority of the remaining failed attempts being due to unoptimized paths. SAC also shows respectable learning capabilities when presented with the same task. DDPG on

the other hand, at least in this experiment and without further fine tuning of its hyperparameters, lacks the learning consistency and efficiency shown by the other RL algorithms.

The vast majority of path traversals do not show any specific type of section that suggests a worst performance by the agent than in other sections, and cusps are not an exception, with the agents being able to complete this type of path formation as expected and with similar levels of performance as straight or other curved path sections. The only notable exception, that could be avoided with better path optimization, is present in paths that involve prolonged overlapping sections with the same heading and opposite directions, a type of section that should not exist in optimal paths.

Following paths with cusps through well defined layouts is proven to be feasible and a promising solution for both completely autonomous parking lots and also mixed parking lots with manually driven vehicles seamlessly driving alongside AVs, provided collision detection and vehicle synchronization methods are integrated alongside it.

The potential of this methodology for autonomous robot locomotion is not restricted to automobiles in parking scenarios. It can also be extend to mobile robots working inside warehouses or nautical vehicles moving through confined environments, with the scenario of a boat docking in a harbour being a relevant example. In order to make this conversion the proper adjustments regarding the vehicle's specific movement characteristics would need to be considered.

Overall, path following with cusps and overlapping sections does prove to be a challenging problem, requiring creative solutions to the specific challenges originated by this type of path, and a well adjusted training process in order to efficiently achieve a good policy, but the vast ability of deep reinforcement learning algorithms to adapt to a large variety of problems is extended to the particularities of this one.

The main objectives of this research where achieved and serve as its main contributions: the creation of a valid methodology for the training of an agent capable of traversing these types of paths, as well as a deep understanding of how the three reinforcement learning algorithms used in this study perform when using this methodology.

5.1 Future work

The results obtained from this experiment suggest that path following in confined spaces through these types of paths is possible and DRL is an efficient method to achieve a policy capable of guiding a vehicle from one point to another, even when requiring direction changes. However, when considering parking lots and specially traditional ones accessible by people, where the safety of everyone involved is crucial, collision prevention methods need to be integrated in order to ensure the safety of the environment. When dealing with policies trained through reinforcement learning it is very difficult to guarantee the predictability of the agents actions, and a single potentially dangerous action is enough to have dramatic consequences, a realistic possibility when the agent expects a greater reward in the future, that needs to be taken into consideration.

In order to not risk human lives or damaged goods, distances recorded by proximity sensors around the vehicle could be added to the agents observation in future work, allowing it to be aware of its surroundings while following a predefined path. Investigating the evading capabilities of the agent when moving near movable obstacles could enhance collision avoidance and greatly improve the safety standard of the agent.

Another interesting topic to be investigated is a collision prediction method that forces the vehicle to slow down or even brings it to a complete stop when it detects that a collision is eminent.

In this work the path planning library utilized, PythonRobotics [44], creates paths through Reeds and Shepp's curves and avoids circular obstacles when creating paths. By opting for a more precise path generation method, one that takes into account the precise dimensions and shapes of the actuating vehicle as well as those of the surrounding obstacles, the existing space present in these confined spaces could be more efficiently used when generating feasible paths. The use of smother curvature types, for example continuous curvature paths [17], could also alleviate the stress enforced in the vehicle when performing its maneuvers. Paths with both forward and backwards parking maneuvers should also be considered in future work, to confirm that the inclusion of backwards parking maneuvers do not alter the path following behaviour of the agents.

Future work should also consider paths generated through multi agent path planning, as trajectories created in confined spaces tend to have a high number of overlapping sections between paths, and one of the potential benefits of having multiple autonomous vehicles moving efficiently at once through these confined spaces include drastically reduced traversal times when comparing to a sequence of single vehicle movements. It would also allow, in a parking lot model made exclusively for autonomous vehicles, similar to the one presented in [7], for the dynamic reallocation of vehicles when opening a pathway for the trajectory of a summoned vehicle that is blocked by others, in order to exit the parking lot, through an efficient multiple vehicle reallocation strategy.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Shubhani Aggarwal and Neeraj Kumar. Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges, 2020.

[3] Adeel Akhtar, Christopher Nielsen, and Steven L. Waslander. Path following using dynamic transverse feedback linearization for car-like robots. *IEEE Transactions on Robotics*, 31(2):269–279, 2015.

[4] Konstantin S. Yakovlev Aleksandr I. Panov and Roman Suvorov. Grid path planning with deep reinforcement learning: Preliminary results, 2017.

[5] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2017.

[6] Anonymous Author(s). Robust auto-parking: Reinforcement learning based real-time planning approach with domain template. 2018.

[7] J. Azevedo, P. M. D'orey, and M. Ferreira. High-density parking for automated vehicles: A complete evaluation of coordination mechanisms. *IEEE Access*, 2020.

[8] H. Banzhaf, L. Palmieri, D. Nienhüser, T. Schamm, S. Knoop, and J. M. Zöllner. Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8, 2017.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[10] Graham Cookson and Bob Pishue. The impact of parking pain in the us, uk and germany, 2019.

[11] Peter Dayan and Yael Niv. Reinforcement learning: The good, the bad and the ugly. 2008.

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

[13] Pedro M. d'Orey, José Azevedo, and Bob Pishue. Automated planning and control for high-density parking lots, 2017.

[14] D. Edwards. Cars kill cities. Available at https://progressivetransit.wordpress.com/2012/01/25/cars-kill-cities/.

[15] Eurostat. Statistics explained: Passenger transport statistics. Available at https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Passenger_transport_statistics#Modal_split_, December 2019.

[16] M. Ferreira, L. Damas, H. Conceicão, P. M. d'Orey, R. Fernandes, P. Steenkiste, and P. Gomes. Self-automated parking lots for autonomous vehicles based on vehicular ad hoc networking. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 472–479, 2014.

[17] Thierry Fraichard and Alexis Scheuer. From reeds and shepp's to continuous-curvature paths, 2005.

[18] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[19] Eric Gazoni and Charlie Clark. openpyxl v3.0.7, 2021.

[20] David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles, 2016.

[21] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[22] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018.

[23] Eric Heiden, Luigi Palmieri, Kai O. Arras, Gaurav S. Sukhatme, and Sven Koenig. Experimental comparison of global motion planning algorithms for wheeled mobile robots, 2020.

[24] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[25] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently, 1996.

[26] Ishai Menache Hongzi Mao, Mohammad Alizadeh and Srikanth Kandula. Resource management with deep reinforcement learning, 2016.

[27] J. Hopcroft, J. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: Pspace-hardness of the warehouseman's problem, 1984.

[28] J. van Balen J. Timpner, S. Friedrichs and L. Wolf. K-stacks: highdensity valet parking for automated vehicles. In *Proc. IEEE Intell. Vehicles Symp. (IV)*, page 895–900, June 2015.

[29] Georg Schildbach Jason Kong, Mark Pfeiffer and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. 2015.

[30] Danial Kamran, Junyi Zhu, and Martin Lauer. Learning path tracking for real car-like mobile robots from simulation, 2019.

[31] S. Klaudt, A. Zlocki, and L. Eckstein. A-priori map information and path planning for automated valet-parking. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1770–1775, 2017.

[32] Martin Lauer. A case study on learning a steering controller from scratch with reinforcement learning. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 260–265, 2011.

[33] Xiaoyun Lei, Zhian Zhang, and Peifang Dong. Dynamic path planning of unknown environment based on deep reinforcement learning, 2018.

[34] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

[35] Yuxi Li. Deep reinforcement learning: An overview. 2018.

[36] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[37] Xiyan Mao, Xianjin Huang, Yaya Song, Yi Zhu, and Qichuan Tan. Response to urban land scarcity in growing megacities: Urban containment or inter-city connection? *Cities*, 96:102399, 2020.

[38] Andreas B. Martinsen and Anastasios M. Lekkas. Curved path following with deep reinforcement learning: Results from three vessel models. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–8, 2018.

[39] Silver D. Mnih V., Kavukcuoglu K. Human-level control through deep reinforcement learning. 2015.

[40] N. J. Nilsson. Principles of artificial intelligence, 1980.

[41] OpenAI. Openai spinning up, introduction to rl.

[42] Brigitte d'Andrea-Novel Philip Polack, Florent Altche and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? 2017.

[43] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards, 1990.

[44] Atsushi Sakai, Daniel Ingram, Joseph Diniusc, Karan Chawla, Antonin Raffin, and Alexis Paques. Pythonrobotics: a python code collection of robotics algorithms, 2018.

[45] X. Shen, I. Batkovic, V. Govindarajan, P. Falcone, T. Darrell, and F. Borrelli. Parkpredict: Motion and intent prediction of vehicles in parking lots. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1170–1175, 2020.

[46] Xu Shen, Xiaojing Zhang, and Francesco Borrelli. Autonomous parking of vehicle fleet in tight environments. 10 2019.

[47] David Silver. Cooperative pathfinding, 2005.

[48] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms, 2014.

[49] Maddison C. Silver D., Huang A. Mastering the game of go with deep neural networks and tree search, 2016.

[50] Yogang Singh, Sanjay Sharma, Robert Sutton, Daniel Hatton, and Asiya Khan. A constrained A* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents, 2018.

[51] Erotokritos Skordilisa and Ramin Moghaddass. A deep reinforcement learning approach for real-time sensor-driven decisionmaking and predictive analytics, 2020.

[52] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction, 2018.

[53] U.S. DEPARTMENT OF TRANSPORTATION. State motor-vehicle registrations - 2019. 2020.

[54] Ko-Hsin Cindy Wang and Adi Botea. Fast and memory-efficient multi-agent pathfinding, 2008.

[55] Mohd. Nayab Zafar and J. C. Mohanta. Methodology for path planning and optimization of mobile robots: A review, 2018.

[56] Peizhi Zhang, Lu Xiong, Zhuoping Yu, Peiyuan Fang, Senwei Yan, Jie Yao, and Yi Zhou. Reinforcement learning-based end-to-end parking for automatic parking system, 2019.

[57] Xueyou Zhang, Xian Guo, Yongchun Fang, and Wei Zhu. Reinforcement learning-based hierarchical control for path following of a salamander-like robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6077–6083, 2020.

[58] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey, 2020.