FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Exploratory study of DRL with applications to Motion Planning in roundabouts for Autonomous Driving

André Santos

Master in Informatics and Computing Engineering

Supervisors: Filipa Ramos & Rosaldo Rossetti

July 19, 2024

Exploratory study of DRL with applications to Motion Planning in roundabouts for Autonomous Driving

André Santos

Master in Informatics and Computing Engineering

July 19, 2024

Abstract

The interest in autonomous vehicles has been significantly increasing in recent years. The reasons are various, ranging from general driver comfort to safety or even social status. Considering this, many brands have been equipping their fleet with driverless software, mainly for highway driving or parking. Despite the overall progress in these sub-areas, other urban scenarios such as roundabout manoeuvres still remain a challenge, in this case, both for manual and autonomous vehicles, given the high traffic accident rate in this context. Moreover, the fact that roundabouts are usually placed on intersections with an intense car flow will add complexity to the validation and safety of algorithms that operate under these conditions. In some cases, it may also create space for work on multi-agent problems.

In this sense, a few existing studies have explored the application of reinforcement learning or other machine learning algorithms in path planning and motion control tasks in these scenarios. However, there is still a gap for other advanced algorithms that have proven to be efficient and precise when employed in other areas. Reinforcement Learning is a methodology where an agent learns how to behave in an environment by performing actions and receiving rewards or penalties as a consequence. The agent's objective is to learn a policy, a mapping that maximizes the reward of the agent's actions. Deep learning is a sub-field of machine learning that uses deep neural networks (multiple layers) to solve complex tasks.

Using the combination of the two concepts, Deep Reinforcement Learning is born. In this field, deep neural networks are used to approximate the optimal policy or value function in RL problems. This allows the handling of high-dimensional spaces from some real-world environments with complex representations. Building upon this, this research will focus on experimenting how a Deep Reinforcement Learning algorithm behaves when applied in the calculation of an optimized path and decision-making in a roundabout driving scenario, as well as in the definition of a novel reward function capable of attending to all the variables and constraints present in the task

In order to evaluate the obtained results, a set of restrictive metrics will be used, since autonomous vehicles are a critical system that carries high risk with it. This system will be accompanied by a simulation environment which will enable the visualisation of the car behaviour on the road and the management of the intervenients on the scene.

Resumo

O interesse em veículos autónomos tem aumentado significativamente nos últimos anos. As razões são várias, variando desde o conforto geral do condutor até à segurança ou mesmo ao status social. Tendo isso em consideração, muitas marcas têm vindo a equipar a sua frota com software autónomo, principalmente para condução em autoestradas ou para manobras de estacionamento. Apesar do progresso geral nestas subáreas, outros cenários urbanos, como a condução em rotundas, continuam a ser um desafio, tanto para veículos manuais como autónomos, dado o elevado índice de acidentes de trânsito neste contexto. Além disso, o facto de as rotundas estarem geralmente localizadas em interseções com um fluxo intenso de carros adiciona complexidade à validação e segurança dos algoritmos que operam nestas condições. Em alguns casos, isso pode também criar espaço para trabalho em problemas com múltiplos agentes.

Nesse sentido, alguns estudos existentes têm explorado a aplicação de *Reinforcement Learning* ou outros algoritmos de *Machine Learning* em tarefas de planeamento de trajetórias e controlo de movimento nestes cenários. No entanto, ainda existe uma lacuna para outros tipo de algoritmos avançados que se têm mostrado eficientes e precisos quando usados noutras áreas. O *Reinforcement Learning* é uma metodologia onde um agente aprende como se comportar num ambiente ao realizar ações e receber recompensas ou penalizações como consequência. O objetivo do agente é aprender uma *política*, um mapeamento que maximize a recompensa das ações do agente.

O *Reinforcement Learning* é um subcampo de *Machine Learning* que usa redes neurais com (múltiplas camadas) para resolver tarefas complexas. Utilizando a combinação dos dois conceitos, nasce o *Deep Reinforcement Learning*. Neste campo, redes neurais profundas são usadas para aproximar uma política ótima ou função em problemas de *Reinforcement Learning*. Isto permite lidar com espaços de alta dimensão de alguns ambientes do mundo real com representações complexas. Com base nisso, esta pesquisa focar-se-á em experimentar como um algoritmo de *Deep Reinforcement Learning* se comporta quando aplicado no cálculo de uma trajetória otimizada e na tomada de decisões num cenário de condução em rotunda, bem como na definição de uma função de *reward* inovadora capaz de incluir todas as variáveis e restrições da tarefa

Para avaliar os resultados obtidos, serão utilizados um conjunto de métricas restritivas, uma vez que os veículos autónomos são um sistema crítico que carrega alto risco. Este sistema será acompanhado por um ambiente de simulação que permitirá a visualização do comportamento do carro na estrada e a gestão dos intervenientes na cena.

Acknowledgements

To all the people without whom I wouldn't be able to stand here today, this is for you.

First and foremost, I'd like to express my sincere gratitude to my supervisors, Filipa Ramos and Rosaldo Rossetti, for their support and guidance throughout this master's thesis. Their knowledge and challenging approach were inspiring, and their consistent assistance and feedback were fundamental to the execution of this project.

I would like to extend my heartfelt thanks to my family for their presence, support, and unwavering belief in me. The blind trust I have received from them since I was young has been instrumental in shaping who I am today, and I wouldn't be the person I am without it. Their support in every aspect of my life has been indispensable, and I am deeply grateful for it.

I would also like to express my gratitude to FSFEUP for sparking my passion for Autonomous Driving and always pushing the engineer within me further.

I also wish to express my gratitude towards Rita for her presence throughout these years, for celebrating and sharing all my small victories, for accompanying me through all the ups and downs, and for being my home.

To all friends who have been present in any way during this journey, I'd like to express my immense gratitude for the pleasure of having you by my side and for all the memories we shared.

Lastly, I am indebted to my course colleagues and friends for their help and presence in every aspect of the way. From everyone who helped me or whom I helped with the courses, to every person who hosted me and made this experience what it was. They have been a fundamental presence, and completing this stage wouldn't have been possible without them. This has been an adventure. But I couldn't finish without highlighting the close family FEUP gave me. To Edgar, for being with me on the lane all time; to João, for being an inspiration to everyone around; to Maria, for mediating every situation and being a warm presence; to Rodrigo, for being an example in everything and carving the essence of this faculty into my head; to Sérgio, for always caring, going to the supermarket, and for the 005; and to Sofia, for showing me that coffee cups are also made for breaking.

I will never be able to repay what I have received. The most heartfelt thank you,

André Santos

"If everything seems under control,
you're just not going fast enough"

Mario Andretti

"Alone, its just a journey,
Now adventures, they must be shared"

The Stranger, LOTR

.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

AD	Autonomous Driving
AS	Autonomous System
CR	Collision Rate
DNN	Deep Neural Network
DQN	Deep Q-Networks
DDQN	Double Deep Q-Networks
D3QN	Dueling Double Deep Q-Networks
DRL	Deep Reinforcement Learning
LSTM	Long-Short Term Memory
ML	Machine Learning
NN	Neural Network
POMPDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RRT	Rapidly exploring Random Trees
SAC	Soft Actor-Critic
SR	Success Rate
TD	Temporal Difference
TTC	Time to Collision

Chapter 1

Introduction

1.1 Motivation

The driving scene has been the target of a major paradigm shift in recent years. With the introduction of electric-powered vehicles in the car fleet and the creation of driving aid mechanisms, such as autonomous driving, the automation of vehicles is becoming a reality. As these mechanisms become increasingly sophisticated, the utilization of this software in daily life is already underway. The application of these systems to highway driving tasks such as overtaking and ramp merging has shown promising results, both in performance and safety, as well as in user experience feedback. While this is now widely spread across various recently released vehicles, the transition from highway driving to urban environments introduces a whole set of challenges whose solutions are yet to be fully explored. The issues raised by high traffic, complex roads, and unpredictable environments make the implementation of autonomous systems in these scenarios a difficult task. There has been a concern about making advances in these types of scenarios, despite the still lacking performance. The results obtained have been improving, but are still far from being considered safe and reliable for a critical system, whose nature requires highly advanced decision-making capabilities.

As such, the main focus of this work is to perform an exploratory study on the roundabout maneuvering scenario, a task with amplified problems as far as motion planning is concerned. While there are many variables and major tasks in the Autonomous Driving pipeline, the highlight will be on path definition and following. This problem incorporates concepts from different sub-issues: lane changing, efficient path planning, exit and entrance management, etc. To achieve this, a Deep Reinforcement Learning algorithm will be employed in the solution of the task. These algorithms have been achieving state-of-the-art results for different tasks in recent years and are yet to be explored in the aforementioned context. As such, their application to a roundabout maneuvering scenario will be studied and evaluated based on different parameters.

1.2 Problem Definition

The challenge presented consists of guiding an autonomous vehicle through a roundabout, from a specific entrance to a specific exit, in the most efficient way, following traffic rules and causing no collisions.

Using a set of perception ground truth data, the vehicle is to define its path and manage its velocity and position through it to avoid causing collisions, merging tasks from Path Planning and Vehicle Control. This is one of the fundamentals behind the task definition, since it is intended for the vehicle to have full control over its decisions, which means having no path or control references. The vehicle should complete the task by managing its velocity and position in the state space, with the goal of traversing the roundabout in the least amount of time. This will require the car to manage its timings, so that it can smoothly enter, cross, and exit the roundabout.

In order to learn how to perform the task, the agent will learn through repetition, receiving a reward that varies depending on the speed variations through the crossing, so that the regular and smooth behaviour is transmitted as the desired behaviour. As the training progresses, the policy will be updated through a Neural Network to better simulate reality in each action outcome. A Deep Reinforcement Learning methodology will be the chosen paradigm to deal with the problem.

The scenarios will vary in the type, number, and behaviour of cars which will augment the unpredictability and, in consequence, the adaptability of the vehicle to uncertainty.

1.3 Aims & Goals

The main aim of this work is to define a reward function and develop a model around it, maximizing an agent's capabilities in performing a roundabout traversal smoothly and without collisions, with respect to motion planning. In order to do so, four main goals can be defined:

- Definition of a reward function that encompasses all aspects and variables of the task, balancing all components.
- Development of a DRL framework to train the agent within the environment.
- Analysis of the decisions regarding the reward function and algorithm performance.
- Analysis of the viability of integrating a similar system into a real-life scenario.

1.4 Structure

his report is divided into three more chapters. In Chapter 2, a review is conducted on the state of the art for the path planning and motion planning subtasks for different driving scenarios. The main focus is on Reinforcement Learning and Deep Reinforcement Learning approaches. In Chapter 3, the structural definitions behind the work and selected approach are described. As for Chapter 4, the different experiments that support, analyze, and explain the final decisions taken regarding

both the model and reward function are explained and discussed, as well as the results obtained from the work. Finally, the final chapter will provide a review of the conducted work, providing some discussion on the positive takeaways, the main issues found, and perspectives for the future regarding the problem intrinsically and generally.

Chapter 2

State of the Art

This work will focus on the Path Planning and Decision Making tasks of an Autonomous System, also known as Motion Planning. As such, other parts of the autonomous vehicle pipeline, as the processing of the perception data from the sensors, state estimation or how the actuators will induce the vehicle to follow the desired path will not be studied. The information from the environment will thus be provided to the system previously, and assumed to be precise. However, in order to make the system congruent with the reality, only information from the vehicle surroundings will be considered, and the other vehicle routes and decisions will not be predictable. These constraints will make the problem a Partially Observable Markov Decision Process (POMDP). In this chapter, similar problems will be studied in depth, gathering the particular details from each and thus providing information on the general behaviour and performance of the autonomous vehicles in some AD scenarios. The algorithms will be split mainly into Path Planning, tasks where, statically, the most efficient path is calculated for a certain environment, and Decision Making, where the vehicle behaviour and movement is incorporated with the path. Some information on the mentioned theoretical concepts will also be provided in the Background Section.

2.1 Background

In this section, the main concepts about Deep Reinforcement Learning will be explained and summarized, along with its components. Autonomous Driving aspects such as the general tasks and the details of each scenario will also be covered.

2.1.1 Reinforcement Learning

Reinforcement Learning is one of the most researched artificial intelligence solutions in recent years. Its main advantage is that it is an algorithm which does not need any prior knowledge, contrarily to supervised and unsupervised learning approaches, depending only on feedback and reward signals from the agent's actions in the environment. According to these outputs, the agent behaviour is corrected so that it can gradually tend to the set of actions with maximum reward. The functioning of this algorithm is described in the schema below 2.1. The agent navigates around

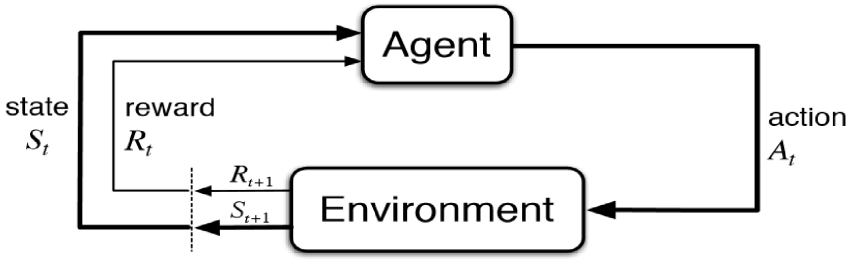


Figure 2.1: Reinforcement Learning Schema, taken from Zhao et al. (2021).

the environment through a continuous set of iterations where each action creates a reward that is taken into account for the next decision. Given the current state, the agent takes what is considered the best action. The environment receives this input, updates its state and updates the reward value using the previous decision.

Within the RL algorithm family, there are different approaches, depending on the task, usually varying considering the action space size and nature. While Q-Learning is usually used for discrete and smaller or medium-sized action spaces, policy gradient is mostly used for continuous and larger problems. Actor-critic approaches are frequently implemented on the second paradigm, but are more versatile and can be considered a mixture of both.

Traditional reinforcement learning methods have been effective in solving many different tasks, in spite of being known to be limited by the dimension of the sample and action spaces, resulting in an underwhelming output of actions in complex environments.

2.1.1.1 Q-Learning

Q-Learning is an RL algorithm that uses a Q-Values table, a set of states where each cell represents the expected future reward for taking an action that leads to that state.

The table is initialized with zeros in the standard algorithm, and it is iteratively updated using the Bellman equation. It contains the Q-function, which takes the state and action as inputs to output the q value of a determined movement. The exploration importance can be managed using the ε value when using a greedy strategy, for example: valuing initial exploration with random actions and gradually transitioning to a lower value to exploit the learned information. The equation describing the behaviour of the function can be seen below 2.1. The *Q(s,a)* component is the expected cumulative reward for taking an action *a* in the state *s*. It is what's pretended to update. The α is the learning rate and controls how fast new information overrides old one. A lower one is used for a conservative approach and a higher one for rapid incorporation. If not tuned, the function may converge too slow or overshoot the value calculation. The *R(s,a)* component is the immediate reward for the action - state pair. The γ is the discount rate and it measures how important future rewards are in comparison to immediate ones. A 0 value only considers the immediate reward, while a 1 value will attribute the same importance to both. The *max Q'(s',a')* is the maximum

Q-value among all possible actions in the next state, representing the highest expected cumulative reward the agent can still achieve from the state.

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{R(s,a)}_{\text{Reward}} + \overbrace{\gamma \max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)] \tag{2.1}$$
$$\text{Discount rate}$$

The training process is performed by following the pipeline: choose an action based on the Q-Table, updating it with the observed reward. These steps are constantly executed until the Q-Table converges to the pretended values.

The rewards of each step are usually: a small penalization for each step (to make the robot choose the smaller path), a great penalty for going to an undesired state, like a wall or an obstacle, a small reward for going through a certain place (robot recharge, collecting money, ...) and a big reward for reaching the final goal. These all depend on the problem context and needs Lamba (2018).

2.1.1.2 Policy Gradient

Deterministic Policy Gradient (DPG) is an RL algorithm whose main advantage is handling continuous action spaces. It can be considered an alternative to stochastic policy gradient algorithms, but is generally more efficient and simple to implement.

This algorithm replaces the action-state function with an approximation deterministic policy function. Since the action space is continuous, instead of taking the argmax action like in Q Learning, the policy is chosen as an approximation in the direction of the Q functions gradient, adjusting the parameters as time goes, ultimately converging to the optimized value expectation mapping.

The specific form of the policy depends on the chosen algorithm. This approximation can be performed using various solutions such as off-policy Q-Learning Divergence or Gradient Q-Learning deterministic Actor-Critic algorithms. These approaches vary a lot from work to work, and their advantages are exploited depending on the problem's nature, but can be described by the equation 2.2. Weng (2021). The $J\theta$ represents the objective function that maximizes the cumulative expected reward. When its gradient ∇_θ is instantiated, the variation of the function according to the parameters θ is the result to be calculated instead. The policy function is defined by $\mu_\theta(s)$. Its gradient ∇_θ maps the policy variation according to its parameters. The $\rho^\mu(s)$ is the probability of a certain state when following the respective policy, mapping the state distribution. The $Q^\mu(s,a)$ represents the expected return value from the action-value function following the referred policy. Accompanied by the gradient ∇_a, the way the function changes with action adjustments is

defined. The $\mathbb{E}_{s\sim\rho^{\mu}}$ denotes the expectation operator, computing the value of the function over the state distribution.

$$\nabla_{\theta}J(\theta) = \int_{\mathscr{S}} \rho^{\mu}(s)\nabla_{a}Q^{\mu}(s,a)\nabla_{\theta}\mu_{\theta}(s)\Big|_{a=\mu_{\theta}(s)} ds = \mathbb{E}_{s\sim\rho^{\mu}}\left[\nabla_{a}Q^{\mu}(s,a)\nabla_{\theta}\mu_{\theta}(s)|_{a=\mu_{\theta}(s)}\right] \quad (2.2)$$

2.1.1.3 Actor-Critic

The Actor-Critic approach can be seen as a middle ground between the aforementioned approaches, combining the benefits from both allowing a stable and effective learning even in large state spaces. It is a Temporal Difference (TD, Sutton (1988)) version of the policy gradient algorithm Karunakaran (2020).

This algorithm contemplates two networks: the Actor (decision-maker) and the Critic (evaluator). The first uses a policy gradient technique to decide on the actions, while the second uses a value function to evaluate these actions. It can be compared to a Generative Adversarial Network (GAN), Goodfellow et al. (2014), since this behaviour is similar to the discriminator and generator relation (the first generates artificial images, the second evaluates their authenticity).

The pipeline for this algorithm can be described as:

1. Sampling states and actions from the policy in the actor network.
2. Evaluating the advantage function using the critic network.
3. Updating the policy parameters.
4. Updating the critic weights.

This process is repeated until an optimal policy is found. The advantage function produced by the critic network is described in 2.3. The function itself represents how good an action is in a state s, when compared to the average value for the same state. The immediate reward for the same action-state pair is represented by r. Finally, the V function represents the expected cumulative reward in a certain state, in this case for s_t+1 and s_t from the state for the value function. All the functions are defined for the policy π_{θ} Karunakaran (2020).

$$A_{\pi_{\theta}}(s_t,a_t) = r(s_t,a_t) + V_{\pi_{\theta}}(s_{t+1}) - V_{\pi_{\theta}}(s_t) \quad (2.3)$$

2.1.2 Deep Reinforcement Learning

Deep Reinforcement Learning combines deep learning with reinforcement learning. The first is usually reliable for its pattern recognition ability and the capacity to process a large amount of data. The second is really effective in decision-making and the search for the best action sequence. Putting these two concepts together, DRL is created, resulting in a family of algorithms able to perform well in complex environments, usually improving the results over any of the two singular

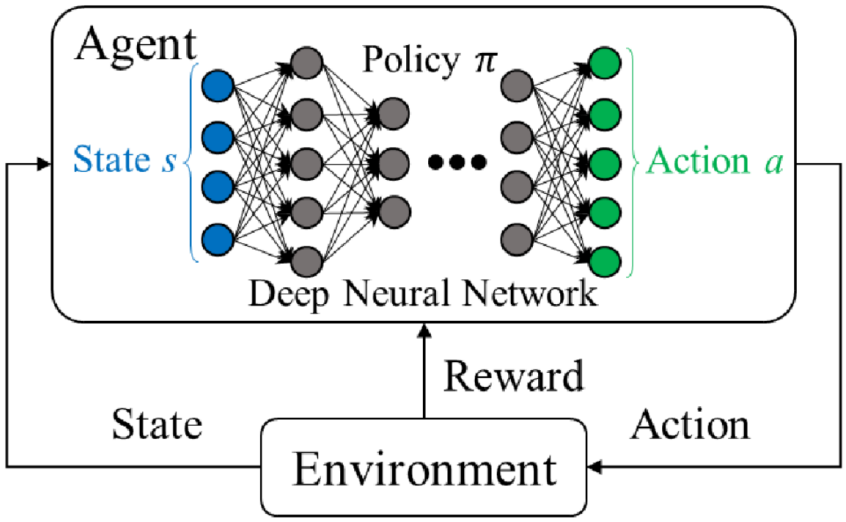


Figure 2.2: Deep Reinforcement Learning Schema, taken from Huang et al. (2019).

components. This approach allows the function to work with non-linear approximations, opening up the opportunity for more complex solutions.

Its functioning is pretty similar to regular RL, but a neural network is introduced to calculate the policy. Instead of using a table to map the action-state relations for example, the Deep Learning component replaces it. It still receives the state as input, containing every possibility for those in the first layer. The output of the network will contain every available choice in the action space, selecting the highest rewarding activation. The eventual outcome / reward of the chosen action is evaluated as the loss function of the NN and is back-propagated into it, updating the weights for the next iteration. The environment also communicates the new state to the agent, as in regular RL, so that this process can be repeated and continuously optimized.

This methodology is common across all algorithms, but the way the step is performed varies significantly depending on the RL paradigm chosen to handle the task. The process can be summarized generally by the schema 2.2.

2.1.2.1 Value-Function DRL

In this algorithm family, the DNN is used to approximate the value function. In every iteration, the parameters from the previous one are fixed and the loss function is optimized using stochastic gradient descent Mnih et al. (2013). The specific algorithms (Q-Learning, Temporal Difference Learning, etc) will then update it iteratively according to their definitions, aiding the robot in finding the optimized path.

In Mnih et al. (2013), the first Deep Q-Network (DQN), and the base for the most used methods at the moment, was proposed by Google DeepMind, marking the beginning of the exploration of the combination of DNN's with the already developed realm of reinforcement learning. The algorithm was applied to seven Atari 2600 games and ended up matching human expert players

performance. This was just the first pillar, so in the following years, different enhanced versions were published.

In Lei and Ming (2016), the DQN algorithm was applied to a path planning exploration task for the first time. His work revealed some issues with the regular method, as the known overestimation problem (the value of a particular action or state gets estimated higher than its optimal value) and algorithm instability. In response, another set of upgrades were introduced to the previous version.

In Anschel et al. (2017), the Averaged DQN (ADQN) algorithm was proposed. This solved the overestimation problem by averaging the Q-values from prior learning and defining more conservative value estimations. Evidently, this also opened a breach for underestimation issues. In Van Hasselt et al. (2016), a solution was also proposed, by suggesting the Double DQN (DDQN) algorithm. This introduction decoupled the action selection and evaluation, by using a Q-Network for each. This helped mitigate the issue, since the second NN will control any overestimation defect.

Once again, pulling the qualities of different algorithms, in Xie et al. (2017), the Dueling Double Deep-Q Network algorithm (D3QN) was published. It was able to not only reduce the overestimation problem, but to also effectively learn how o avoid obstacles easily in an unknown environment.

In Wen et al. (2020), another proposal is done to the overestimation drawback and enhance the algorithm performance at the same time. By separating the representation of the state values and the advantageous actions for each, in a Dueling DQN algorithm, it was able to correctly perform obstacle avoidance in a complex environment, while reducing the cost values when in comparison to the DDQN proposed by Van Hasselt et al. (2016) and on an even bigger scale when paired to the regular DQN.

The referred works are the most common iterations or enhancements of Deep Q-Networks used in the area and other references to their application will be explored in 2.2.4.1 and 2.2.4.1.

2.1.2.2 Policy Gradient & Actor-Critic DRL

The main advantage of using algorithms based on Policy Gradient is working with continuous or larger action spaces, since value-function based approaches are usually applied to discrete scenarios. While Actor-Critic algorithms include an extra layer on the critic side that uses a value function, they can be considered to similar tasks as Policy Gradient and thus will be described together in this segment.

In Mnih et al. (2016), an asynchronous Advantage Agent-Critic algorithm (A3C) made use of multi-threading to operate with multiple agents in parallel and obtain sample data. It is an approach that significantly reduces the time needed for training, despite still failing in overcrowded scenarios due to convergence issues. In Kartal et al. (2019), a Terminal Prediction (TP) task is introduced into this algorithm, aiding the training in converging efficiently, by predicting the distance to the terminal state. In Labao et al. (2021), another alternative (A3C-GS) is suggested by adding a Gradient Sharing mechanism (GS) to improve the data exploration efficiency, reduce the need

for the entropy loss term, and the general behaviour in high dimensional state spaces. Different policies are shared, generating more heterogeneity, ultimately converging to an optimal policy.

In Lillicrap et al. (2015), the Deep Deterministic Policy Gradient algorithm is proposed, following the success of DQN, and in this case still using it to estimate the value function. It is a common approach in AD since, and has been used to solve many tasks 2.6. It uses the concept from Deterministic Police Gradient (DPG) and employs the Neural Network component. Many iterations of this algorithm have been proposed during the course of the years. Given the application of this algorithm to high dimensional state spaces, in order to improve stability, in Du et al. (2019), an algorithm that computes the gradients of the target Network uses it as a regularizer in the critic loss function, to help fitting the network, utilizing stored Jacobian Information (D3PG). A differentiable simulation engine is fundamental here to allow the backpropagation through the state-action pairs of the Markov Decision Process (MDP). In order to have an efficient gradient optimization strategy, an appropriate step size should be used.

In order to approximate an adequate step size, various algorithms are also an option. In Schulman et al. (2015), a Trust Region Policy Optimization (TRPO) based approach was suggested. This algorithm can be considered conservative since it works with policy updates inside its trust region to control large changes. While it was a positive addition to the pipeline, it was prone to errors in large environments. In Zhang et al. (2019), an enhancement in the ability to learn information from different targets was implemented and diminished the previously mentioned error. Since this algorithm is somewhat complex and too dependent on the strategy and environment relation or interaction, making it deviate from the global optimal path, other solutions were proposed. In Schulman et al. (2017), a Proximal Policy Optimization (PPO) algorithm was proposed based on TRPO. It introduced changes by sampling data through the interaction with the environment and using a random gradient function instead of a regular one.

2.1.3 Driving Scenarios

While driving on a road is an activity in which every second, an infinite set of actions is possible, the scenarios that the DRL implementations try to solve are more general and pretend to represent a multitude of road sections within a single instance/task. The main goal is not to project a path for the car to follow or stay within the limits efficiently, but to make it choose an ideal set of actions time-wise and lane-wise. Considering that, a set of different situations on the road formalize different scenarios which require different needs and problem definitions.

2.1.3.1 Lane Change

The lane change scenario is one of the most simple. The settings depend on the number of lanes, but the usual is 2 or 3 2.3. The main goal in these tasks is usually just avoiding collisions, by either changing the car speed or changing lane. The reward attributed to each will then determine the car behaviour. The state space in this scenarios is often represented by the car position and speed, besides sometimes defining the lane as the lateral position indicator. The action state is usually

Figure 2.3: Lane Change in 3-lane highway example scenario, taken from Zhao et al. (2020)

composed by an acceleration / deceleration command and a lane change action to one of the sides, if applicable. Common used variables are the time to collision Wang et al. (2019) or distance to vehicles Zhao et al. (2020).

2.1.3.2 Overtaking

The overtaking scenario is really similar to the Lane Change one, and could almost be considered an instance of it. The main difference relies on the mission attributed which includes more actions than just changing the lane, since the car needs to perform the overtake safely. In these scenarios, besides the position and speed, other parameters, such as the orientation angle are also included . The vehicle also needs to consider the surrounding vehicles with greater attention, since its a task with more variables and collision risk. The existence of intermediate position values between the lanes is more important here. In some works, the lane concept is discarded 2.4. The distance to other vehicles is a more popular metric in this scenario, Kaushik et al. (2018), Chen et al. (2019b), Lu et al. (2023)).

2.1.3.3 Ramp Merge

Ramp merging can also be considered similar to the previous described scenarios. The concept is identical, but the complexity of the task is on a different detail. In Lane Change, the orientation and angles are generally not necessary, however, in ramp merging tasks, these are fundamental to the problem definition. The traffic intensity is usually higher and the timing of entrance requires way more precision. The vehicle does not have more than one lane to change, nor it can avoid the collision by simply breaking, since other vehicles can follow the leading car behind. In sum,

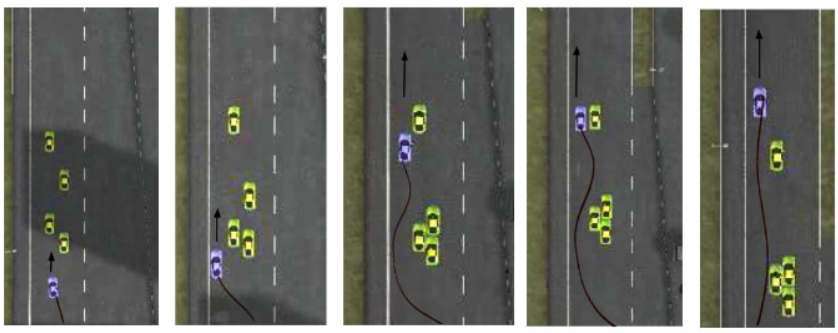


Figure 2.4: Overtaking scenario. Kaushik et al. (2018)

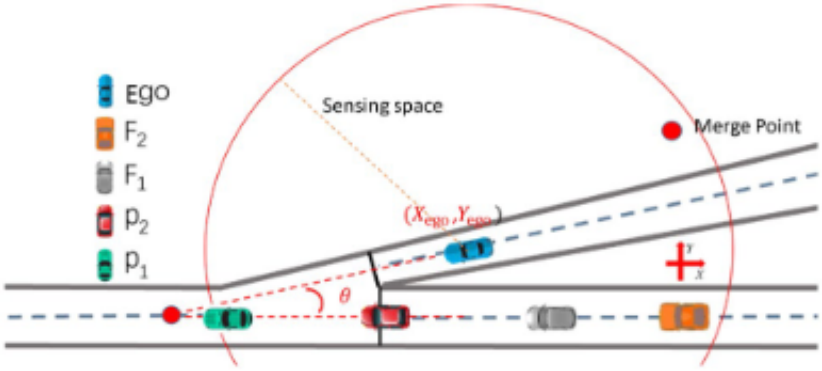


Figure 2.5: Ramp Merge Scenario, taken from Li et al. (2023a).

there are less actions possible, but its management is way more critical. The distance between the vehicle positions and time to collision are also very common variables Li et al. (2023a). In 2.5, the situation described makes reference to the mentioned conditions. The angle of entrance has direct influence in the distance and time for the merge. The ego vehicle will have to manage these to find the preferable entrance timing between the set of cars.

2.1.3.4 Intersections

Regarding the Intersections scenario, the constraints are more case specific when compared to the rest. While the previous relied a lot on the longitudinal controller, this task also employs a heavy burden on the lateral controller, since it needs to perform actual 90ž curves. Besides the usual state variables, it also needs to employ the concept of priority, and define well the possible path range and traffic rules, especially for vehicles turning-left in an intersection. These restraints will make the problem more complex, so more collisions are expected depending on the behaviour from the other vehicles. More reward parameters are also considered. The time and distance until the crossing are variables commonly utilized for comparison with the other vehicles Wang et al. (2020).

2.1.3.5 Roundabouts

Similarly to the Intersections, Roundabout maneuvering also requires greater care in timing management and lateral control than the first three. The concept of priority is also used, being even simpler to define. However, the traffic rules are harder to map in this case. Besides this, the path creation is also way more complex and it escalates with the number of lanes. In order to perform a roundabout correctly, the vehicle needs to follow a set of rules of when to change lane depending on its pretended exit and the number of lanes. This makes it way more difficult to manage than the other tasks, since so many variables are in consideration. Roundabouts are also more prone to have different designs, making the problem unpredictable. As such, many times, it needs to be simplified. Examples of this are one-lane roundabouts, Gritschneder et al. (2016), or predefined routs, García Cuenca et al. (2019), as shown in the figure 2.7.

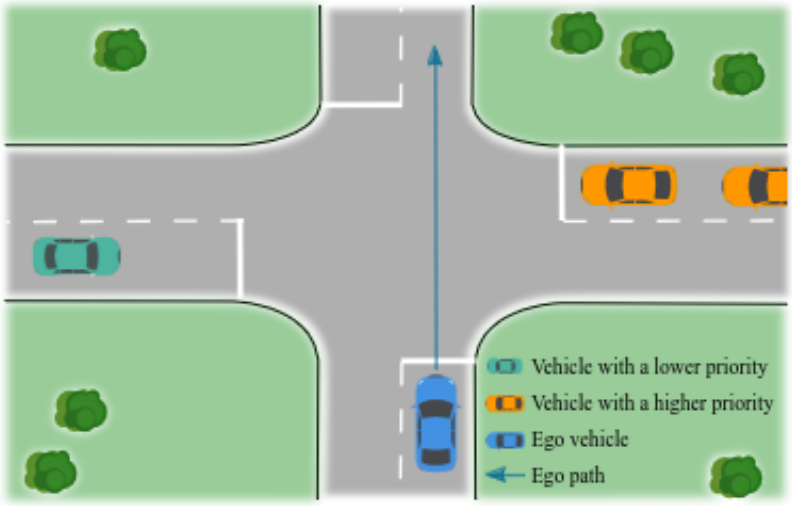


Figure 2.6: Intersection Scenario, taken from Zhang et al. (2022a).

2.1.4 Roundabout Restrictions

The categorization of a roundabout is not a trivial task, considering the different parameters that can be changed between each situation. The traffic flow or the available space are variables that vary significantly depending on the case, which leads to the existence of roundabouts with all kinds of shapes, lane numbers, among others. In this section, the good practices and general details behind designing a roundabout will be further studied, accompanied by a dive on the rules and recommended patterns to traversing one .

2.1.4.1 Design

Since roundabouts are a complex road structure, the topics covered regarding the design will be limited to the general concepts that would most affect the vehicle behaviour in the Motion Planning task. As the usage of sensors will not be integrated, the visibility of the roundabout will not be accounted for, despite being one of the most important aspects.

In aus, a few design principles are defined to rule the most important parameters of concern:

- It is recommended for the number of legs to be limited to four (although up to six could be carefully used in a single-lane roundabout) and for them to intersect at approximately 90ž with the circulating lanes.
- Entry curvature should be used to limit entry speed. This value is established by considering the demand and the different types of users.
- Exits should allow the vehicles to depart efficiently.
- The periphery of the roundabout must be large enough to integrate all required entries and exits with separation between critic / conflict points.

Figure 2.7: Roundabout Scenario, taken from García Cuenca et al. (2019).

- The circulatory roadway should be wide enough and have clearance kerbs to accommodate the swept path of the different vehicles.

In 2.8, the geometric elements that compose a roundabout are defined. The terminologies will be used in this section.

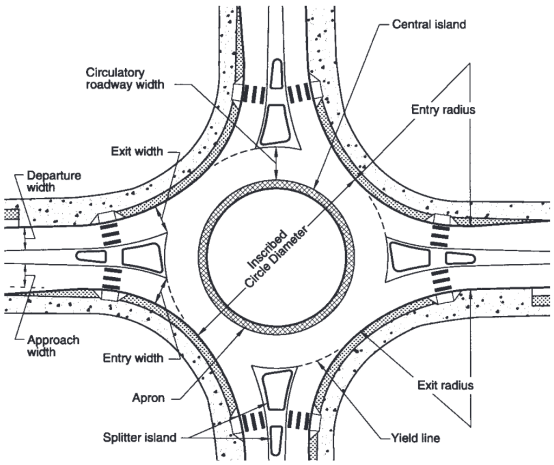


Figure 2.8: Geometric elements of a Roundabout, taken from Robinson et al. (2000).

One of the most general aspects regarding defining a roundabout is its general shape. Overall, there are 2 main possibilities regarding the circulating section form: radial or tangent, exeplified in 2.9. In aus, the authors make remarks about binary look, mentioning the first are more common in Europe and the latter in Oceania's faster roads. Both have their advantages. The radial shape provides a slower traffic flow, which can create a struggle in the vehicle circulation, but their safety rate is considerably higher, specially for bicycles and pedestrians. The tangential approach is preferable in scenarios where a higher vehicle velocity is not a concern. Examples of this are low traffic intensity or roads in remote areas.

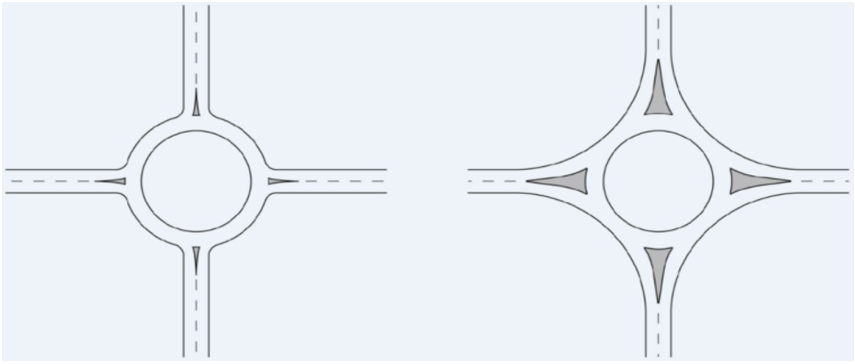


Figure 2.9: Comparison between radial (left) and tangential (right) roundabout shapes, taken from aus.

An important detail regarding the macro design of the roundabout is the number of lanes and legs. The choice between these values is mainly based on the amount of flow expected in the intersection. In Souliman (2016), a study was conducted on the accident rate comparison between single and double lane roundabouts. While increasing the number of lanes can help the traffic flow, the number of collisions also raises with the amount of lanes in the roundabout, making roundabouts with more than 2 lanes generally not recommended. The lane change is a move that can generate conflict in any type of road, but is amplified by the many attention needs in this case. In aus, further information is provided on the relative number of lanes between entry / exit and circulating lanes. The general rule is that the number of circulating lanes cannot be lower than the other two, since this this would generate conflicts on the merge. This is amplified in the entry scenario. Regarding the number of legs, in aus, the authors also state that roundabouts with more than four legs should be avoided when in multiple lanes, since drivers cannot identify the correct lane and exit so easily and the conflict zones are more and closer. For the same reasons, the recommended angle is 90 degrees in 4-exit roundabouts, and if the number of legs increases, the angle between them, should be regular.

In Kennedy et al. (2005), an extensive review was made on the different international guidelines for the different components of the roundabout. Some of those are constraints relevant for the definition of a RL problem. Among the most important are:

- Regarding the diameters of the circles, in the inscribed circle, the key points to account for are: a too large roundabout will increase the car speeds to dangerous levels and a too small will contain the entries / exits and the possible conflicts in a narrow space and increase the accident rate. The recommended values vary generally between 25 and 45 meters for single-lane and 25 and 55 meters for double-lane. As far as the central island is concerned, because of the same reasons of the outer circle, it leaves about 6m away each-side for the circulatory carriageway when there is only 1 lane, and around 10m when there are 2. The central island may also include a truck apron to further augment the ability of large vehicles to maneuver.
- The entry angle is related to the relation between the orientations of the roundabout circulating vehicles and the car about to enter (vector parallel to road guidelines). This can measure

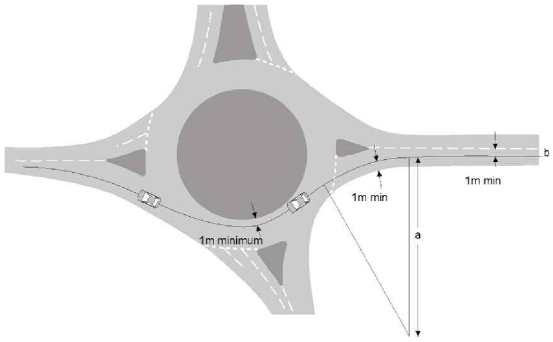


Figure 2.10: Representation of the entry path radius measurement. *a* represents the smallest best fit radius. *b* represents the beginning of the path. Taken from Kennedy et al. (2005).

how easily a vehicle can join a roundabout without changing direction or speed. If the angle is too sharp, the joining car will need to over-brake and steer excessively, and if the opposite happens it will enter the roundabout with a higher speed than it should with residual turning. Examples of this are shown in 2.11. The recommended range is between 30 and 40 degrees.

- By using both the circle diameters and the entries / exits locations and orientations, a metric is created called "entry path curvature". This refers to the curvature or radius of the path that the vehicles entering the roundabout take from the approach road to the circulatory part. The path should be drawn considering the fastest possible route from the entrance, through the roundabout and until the exit, while giving away at least 1m to any non-road geometry. The entry path radius metric is the smallest radius of the path on the bend to the roundabout before joining the circular roads. A figure representing this description is referenced in 2.10. It is useful to measure how easy a vehicle can cross the roundabout while speeding. An appropriate curvature will force the car to manage its velocity, while not reducing it unnecessarily.

- Roundabouts can be design to better fit the presence of cyclists, by either providing them a specific path, or integrating them in the roundabout outer lane. If the latter is present, the desired vehicle speeds should be less, and the previous parameters should be changed according to it. The same happens in the presence of pedestrians.

Other components regarding road sizes and orientations as the entry and exit widths, flares and kerb ratios are also relevant details in the roundabout design and can change both the driveability

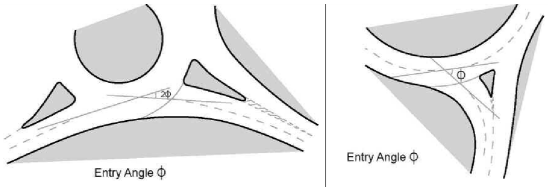


Figure 2.11: Examples of too low (left) and too high (right) entry angles in a roundabout, taken from Kennedy et al. (2005).

and collision rate significantly, but are not a major concern regarding the training of an autonomous vehicle. For the same reasons, other physical aspects as the crossfal or the introduction of splitter islands were also not further investigated.

2.2 Planning Algorithms

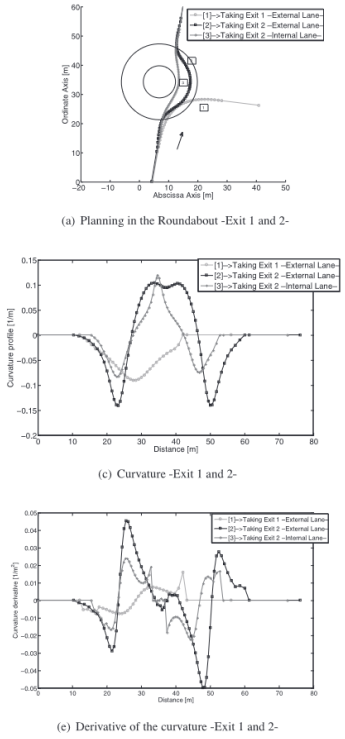
The amount of articles on the application of path planning algorithms to roundabout maneuvering is different depending on the selected approach. Some of them, usually the most traditional, already have a solid research background on the topic to back the concepts, so the literature review will focus mainly on this AS scenario. As for other algorithms, despite some having a significant load of investigation on top for some AD tasks, there is a gap on the roundabout one, so some of the sources will be based on these. The following subsections will review these approaches, ranging from the general definition of parametric curves to the usage of machine learning, specially with RL or DRL.

2.2.1 Parametric Curve Approaches

Parametric Curves have been used across many engineering and math disciplines and this is no exception. This approach can be used both for the creation of a path or for the optimization of a predefined one. In Pérez et al. (2018) there is a study on the application of Bézier (de Casteljau (1959)), B-splines (Schoenberg and Sharma (1973)), NURBSs (Piegl and Tiller (1995)), and rational Bézier curves (Pratt and Faux (1979)) to path planning problems. All of these are closely related to each other, but vary in some details as in whether their control is local or global or how the interpolation is performed between the control points.
In Pérez et al. (2018) and González et al. (2017), the authors state that Bezier curves are usually the preferable and most popular option in path planning tasks. In González et al. (2017), the roundabout is split in three sections, the entrance, the circulatory roadway and the exit. This approach simplifies the lane changing by reducing it to the first and last parts. The behaviour in the middle section is correlated with the curve tangent. At the entrance and exit it can be described as *k(t)*. Bézier curve generation, defined in 2.4, is described by the function *B(t)*, where n is the degree of the polynomial, P_i are the control points and t represents the parameter defined between *[0,1]*. The computation of the curvature is given by *k(t)*, which uses the first and second derivatives of the Bézier curve, *B(t)* and *Bt't'(t)*.

$$\mathbf{B}(t) = \sum_{i=1}^{n} P_i \binom{n}{i} (1-t)^{n-i} t^i \tag{2.4}$$

$$k(t) = \frac{(\mathbf{B}'(t) \times \mathbf{B}''(t))}{\|\mathbf{B}'(t)\|^3} \tag{2.5}$$

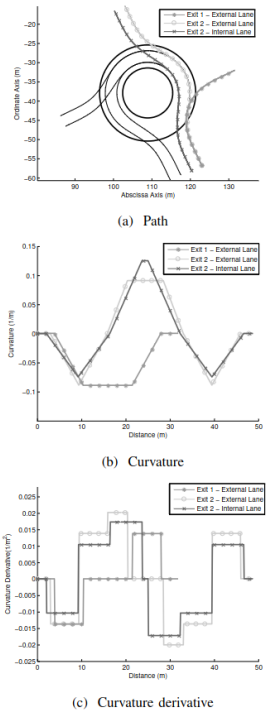


(a) Planning in the Roundabout -Exit 1 and 2-

(c) Curvature -Exit 1 and 2-

(e) Derivative of the curvature -Exit 1 and 2-

(a) Bézier curve profile, taken from González et al. (2017)



(a) Path

(b) Curvature

(c) Curvature derivative

(b) Clothoid curve profile, taken form Silva and Grassi (2017)

Figure 2.12: Curves behaviour in tested roundabouts from the original works. Represented by the curve function, the curvature (derivative of the curve) and the derivative of the curvature.

In 2.12a, the application of these curves in different roundabout approaches, along with the curvature profile. The peaks in the curvature represent the higher variation for the entrance and exit. The curvature derivative also correlates with this.

In another work, Silva and Grassi (2017), this task is solved by using Clothoid curves (also known as Euler Spirals, Cornu (1874a) and Cornu (1874b)). The main advantage of this approach relies on the transition smoothness between different radius curves, since they have a constant curvature rate as shown in 2.12b. Their mathematical definition involves Fresnel integrals, which are significantly more complex than Bézier curves and are defined by more equations.

Generally, these approaches present a solid solution to the problem without difficulty when discarding lane changes and what can be considered a correct car behaviour on a roundabout. If these constraints are applied to parametric curves, extra complexity is added to the definition of the curve segments, which become hard to map without other methods. As previously shown (2.12), this methods can provide smooth roundabout trajectories, nevertheless, path planning tasks often work with unknown environments, making the calculation of a full trajectory not feasible beforehand.

2.2.2 Standard Path Search Approaches

A common way to calculate the path between two points is using general path searching algorithms. These approaches are usually either graph-based, like Dijkstra (Dijkstra (1959)) and A* (Hart et al. (1968)), or tree-based like Rapidly exploring Random Trees (RRT, LaValle and Kuffner Jr (2001)).

As far as RRT is concerned, many variations of the algorithm have been introduced and used in autonomous driving. In Ma et al. (2014), a FAST-RRT approach is suggested and applied to an overtaking scenario, reducing the number of nodes by 90% or more in various tests. Also, in Li et al. (2023b), an approach merging the RRT* and RRT-Connect (LaValle (1999)) variations is introduced, reducing the number of states by 19.7%, 29.3% and 1% in different test cases for a straight road with obstacles. In Zhang et al. (2022b), a BI-RRT approach is suggested and manages to match RRT* in the path length, but reduces the number of states by 47.5% in a curve scenario and 88.6% in a straight road one.

Graph based algorithms may also be used to solve these tasks. In Meng et al. (2023), an altered version of the Hybrid A* algorithm is used in an AD parking scenario, with the addition of both safety and efficiency heuristics. This change allowed the searching period to be reduced by 74.1% and the number of states by 68.5% in a test run. This family of algorithms is also versatile since heuristics can be introduced to adapt the problem to the requirements, as in a roundabout.

One of the advantages of using these approaches when comparing to the works in the previous subsection relies on the fact that it is easier to configure the state space and induce the car/traffic behaviour by restraining some point clusters in the graph or in the exploration range.

When only considering the path planning task, the roundabout scenario won't require a huge set of points, so it will not be an issue to have an optimized path as a result. The main bottleneck will be the lane change restraints. Despite everything, since the trajectory will always be composed by straight lines between points, their smoothness will always be less than in a parametric curve approach, which will adjust the curve to the control points.

2.2.3 Reinforcement Learning Approaches

The introduction of RL algorithms on Path planning allowed for the creation of a new approach on the exploration of unknown environments. Policies introduce a new level of flexibility in these tasks, since they open space to the addition of many restrictions, changeable in real time. That's the case, for example, for the inclusion of other robots or vehicles and the avoidance of collisions with these. The scientific community has then pushed forward to experiment how different RL algorithms behave when solving problems of the sort.

Q-Learning is arguably the most used RL technique across different disciplines and Path Planning is no exception. Following the success of this methodology, many researchers have tried to tune the algorithm and deliver improved versions of it. In Maoudj and Hentout (2020), a more efficient approach is suggested by initializing the Q-table differently and providing the robot with prior knowledge of the environment. The strategy definition takes an approach that pretends to

speed the environment exploration and result in a rapid convergence. In a set of environments defined by the authors the path length is reduced from 22% to 34% and the computation time from 88% to 96% when comparing to a regular Q-Learning approach. In Low et al. (2023), an Improved version of the algorithm is proposed, in which the success rate of the path test for a set of 14 maps is 100% in nearly all, despite falling off in some cases for the travel distance and time taken. A Kolmogorov-Smirnov Predictive Accuracy (KSPA) metric Hassani and Silva (2015) is used to compare the algorithms. In Hao et al. (2022), a different approach was used on the definition of the Q-table. The rewards were mapped using a potential function to map the distance to the robot target. In the results section the algorithm is compared to other techniques, such as RRT, which was mentioned previously, having the edge when measuring the path length.

In another work using Monte Carlo Search Trees, Dam et al. (2022), an experimental study on the appliance of the algorithm to the path planning task tests out different state of the art exploration strategies, generally achieving better results than RRT* when solving POMDP problems.

To our best knowledge, other techniques, such as Policy Gradient and Actor-Critic have not been through plenty of research on the topic, however, when combined with Neural Networks, the paradigm is different as it will be explained in the next section.

2.2.4 Deep Reinforcement Learning Approaches

RL approaches were resulting in advances and achieving SOTA results, so the researchers took advantage of this emergence and introduced Neural Networks into the algorithms, building DRL structures. Generally, in AD, the DRL approaches include the decision making task, so in this section the discussed works will frequently be robot-focused and not restrained to driverless vehicles.

2.2.4.1 Deep Q-Learning

As in normal RL, Q-Learning is a popular algorithm when employed with Neural Networks (Deep Q-Networks). Other variations are used such as Double DQN (DDQN) or Dueling Double DQN (D3QN).

In Raajan et al. (2020), a DQN approach is suggested to evaluate how a robot would behave planning a route in a totally known environment with obstacles in static and dynamic scenarios. The tests were performed in a 2D Pygame environment. The results were compared to various variants of the RRT and Fast Marching Trees (FMT, Overton and Biros (2013)) algorithms, both with and without noise, as shown in 2.1. When comparing the Path Cost, the DRL algorithm performed better than the rest in the noise iterations, with a mean value lower than the other minimums. When no noise is in the equation, DRL still has a lower mean value, which is positive, but it cannot go as low as the rest as seen in the minimum values. In the Path Duration variable, the values differ 2 orders of magnitude in average when compared to DRL, which is a significantly larger improvement, proving the edge of the algorithm.

In Yin et al. (2023), many different DQN algorithms are compared, with and without the double network, the dueling, adding an n-step method and improving the greedy exploration.

Table 2.1: Path quality comparison. First table values without noise. Second table values with noise. Taken from Raajan et al. (2020).

Parameters		**RRT***	**FMT***	**RT-RRT***	**Deep-RL**
Path Cost	Mean	646.78	522.03	521.03	514.26
	SD	63.87	40.10	19.50	-
	Min	492.02	491.38	495.38	514.26
	Max	687.11	591.78	551.78	514.26
Path	Mean	13.37	13.01	9.91	0.151
Duration(s)	Min	12.79	12.76	8.76	0.128
	Max	13.90	13.33	11.93	0.176

Parameters		**CC-RRT***	**Robust-FMT***	**Deep-RL**
Path Cost	Mean	726.92	657.290	550.97
	SD	113.68	64.35	21.8
	Min	565.04	567.41	519.12
	Max	912.22	797.99	584.16
Path Duration(s)	Mean	20.05	18.45	0.162
	Min	19.02	18.01	0.147
	Max	20.90	19.08	0.184
Maximum Risk	Mean	0.0006	0.0007	0.038
	SD	0.0002	0.0002	0.031
	Min	0.00001	0.00003	0.0002
	Max	0.0010	0.0014	0.096
Accumulated risk	Mean	0.0016	0.0018	0.091

These scenarios are simulated in Gazebo, Open Source Robotics Foundation. In every step the robot has 5 possible actions: left, right, front and the respective diagonals. The proposed D3QN with the n-step and the greedy reward optimization improves 745%, 313% and 610% over the regular version and 173%, 65% and 61% over the D3QN approach for the 3 testing stages, as shown in 2.2.

2.2.4.2 Deep Deterministic Policy Gradient

Another popular alternative is employing the Neural Networks to Policy Gradient methods. These are usually the best option to work on a continuous state space. In Bhuiyan et al. (2023), the path of a robotic arm is planned using DDPG. The environment is 3D and different tasks are experimented, one including a moving target, which can correlate to an AD scenario. The results are compared with 2 RRT variations in 2.3. When comparing the execution time, DRL achieves results that are

Table 2.2: Average number of successful arrivals.Yin et al. (2023)

Models	**DQN′**	**DQN**	**Double DQN**	**Dueling DQN**	**D3QN**	**ND3QN**	**RND3QN**
stage 2	0	1.1	2.7	3.3	3.4	7.8	**9.3**
stage 3	0	0.8	0.9	1.2	2.0	2.8	**3.3**
stage 4	0	0.1	0.1	3.4	3.8	5.7	**6.1**

Table 2.3: Mean values of 30 runs each for 7 different Path Planning experiments related with the movement of a robotic arm, taken from Bhuiyan et al. (2023).

	execution time in [s]			**path length in [m]**		
	Planner					
Experiment	DRL	RRT	NC-RRT	DRL	RRT	NC-RRT
1	0.84	3.38	1.97	0.45	3.88	2.27
2	2.15	6.59	4.54	0.85	6.65	3.08
3	3.05	/	/	1.18	/	/
4	1.29	22.38	6.76	0.91	1.45	1.29
5	1.29	25.34	6.36	0.93	1.34	1.32
6	1.30	25.58	5.46	0.95	1.35	1.36
7	1.33	21.59	4.70	0.98	1.30	1.12

in average twice as better in the most simple tasks and can go as good as six times better for more complex tasks, than the best RRT iteration. The path length metric is also improved for at least 1.5 times in the harder tasks and 4 times in the simpler. This reveals that the tested algorithm is way more robust to interference and uncertainty.

In Chen et al. (2022), the same technique is compared to a policy gradient-based Soft actor-critic(SAC) algorithm, since both are used in continuous state spaces. The testing was performed in a PyBullet physics engine, Erwin Coumans and accompanied by the RL toolkit Gym, OpenAI. Both algorithms achieve more than satisfactory results, but SAC ended up achieving the 100% safe-rate, while the DDPG never hit it, despite being above the 99% rate. In the accuracy, SAC also got the edge by a few percent, and can be considered more stable, as seen in 2.13.

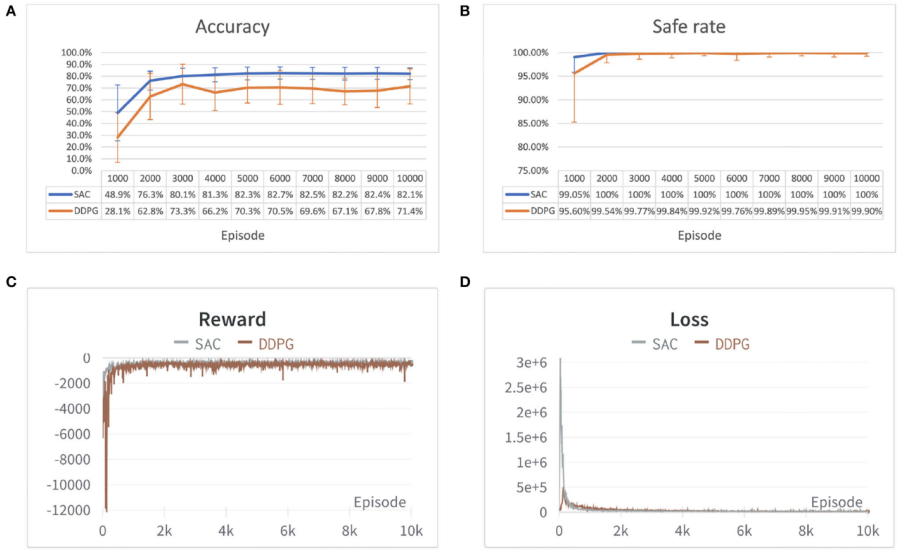


Figure 2.13: Accuracy and Safe-Rate comparison between SAC and DDPG across different episodes in a Path Planning task optimizing the movement of a robotic arm, taken from Chen et al. (2022).

In Ren et al. (2022), a different approach was taken by enhancing the DDPG algorithm. First, a Dynamic Programming step was took in order to generate optimal training data by creating

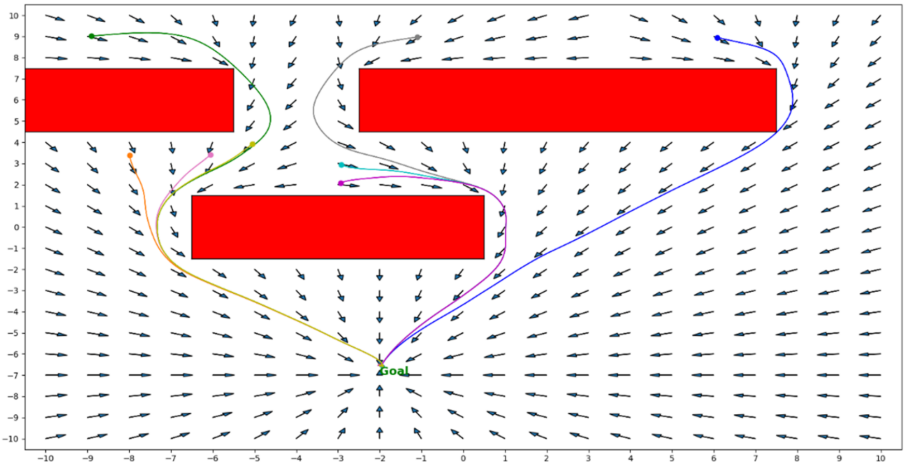


Figure 2.14: Sample trajectories after DRL fine-tuning and the respective optimized vector field, taken from Ren et al. (2022).

an efficient, exact and analytical algorithm. Afterwards, an Extreme Learning Machines method was used to initialize the parameters of the neural network to a near-optimal solution in order to improve the learning performance. This allowed the algorithm to converge and find an efficient solution substantially faster. The work defines its problem by using a different data structure which refers to a generated field of optimized vectors that point to the most efficient direction in each point, almost like in an eletro-magnetic field. Different trajectories calculated using this state space are shown in 2.14.

2.3 Decision-Making

The AD Decision-Making task has different nuances than the path planning one. The latter is usually performed to be a static output, which means there will be no interference with the path. Decision-Making will introduce the choice of moving along the path quicker, slowly, or stopping to avoid collision. This impedes the usage of some previously mentioned algorithms, like parametric curves or the regular graph and tree based approaches, while opening more alternatives to the DRL application.

2.3.1 Reinforcement Learning

The investigation in AD has been somewhat recent, so the introduction of DRL also accompanied the evolution of these systems. This means there will be more exploration on this topic rather than simple RL, but a few works will still be studied.

In Wang and Chan (2018), a **Q-Learning** approach is used to solve a **Ramp Merge** task. The key concept explored in this paper, is the usage of a quadratic Q-function, whose parameters are defined by a Neural Network. This is not considered a DRL approach, since the NN is used for a different purpose. This technique is what allows the autors to work with a continuous space in a

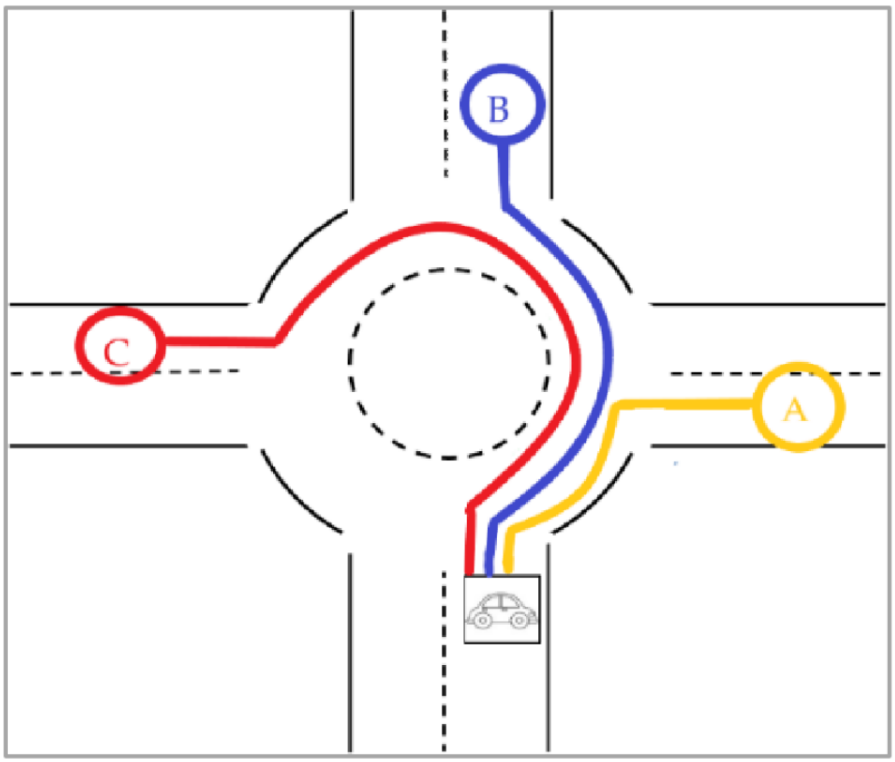


Figure 2.15: Roundabout Schema with the 3 possible routes, taken from García Cuenca et al. (2019)

QL approach. The results are not compared to any other algorithm, but using 4 variables as reward: speed, acceleration and distances to front and back cars, the loss stabilizes and the vehicle is able to perform this task with a high reward level. The action space is composed by the acceleration, deceleration and steering.

Focusing on the pretended **Roundabout Maneuvering** tasks, there has been more investigation using regular RL than DRL, contrarily to the other tasks. In García Cuenca et al. (2019), a **Q-Learning** model works with 7 state variables: x, y, velocity in x-axis, velocity in y-axis, distance to the next state in the path and two binary variables indicating the existence of a left or right lane from the current lane. Speed and wheel angle control the movement on the roundabout. The roundabout routes are 3, considering the different exits in a 4-exit roundabout. The different options are shown in 2.15. The states are points belonging to these routes. The vehicles were trained on 96 hours of manual driving in the Carla Simulator, CARLA Team, the same used for the algorithm. The agent was trained for 10000 episodes, 100 samples each, if no collisions would occur. It was trained on traffic-less roundabouts. The results are not compared to other algorithms, but the evolution of the speed, reward, distance and deviation is shown.

In Gritschneder et al. (2016), a **Q-Learning** algorithm is also experimented in a **Roundabout Maneuvering** scenario. The state space is defined by 4 variables: x, y, velocity value and velocity angle. Multiple lane approaches were not considered. The possible actions are: approach (reduce entrance velocity), go (maximum speed value target), stop (speed target is 0) and uncertain (safe

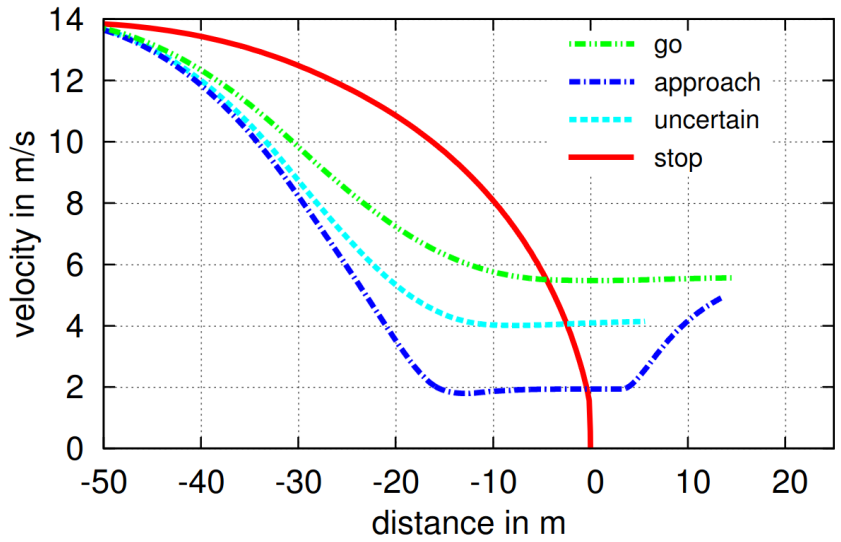


Figure 2.16: Velocity profiles for 4 different roundabout entrance approaches, taken from Gritschneder et al. (2016).

speed close to 0). These different profiles are described in 2.16. These actions were given different rewards, stop being the most penalizing and approach the better one, to try to achieve an ideal driving pattern. No simulation information is provided. The results were not compared to other algorithms, since the main goal was to make a vehicle go through the roundabout the most efficient way, balancing the referred actions.

2.3.2 Deep Reinforcement Learning

As previously stated, the introduction of Deep Learning into Reinforcement Learning will make the agent able to roam on more complex environments, augmenting the general number of studies for different AD tasks. The two main core families of algorithms are Deep Q-Networks and Deep Deterministic Policy Gradient. The first is usually used for discrete state spaces and the second for continuous, as previously explained.

2.3.2.1 Q-Learning

In Hossain (2023), the authors use QL to explore an **Overtaking** task. The used state space for each vehicle includes: the position with x, y values and the velocity with x-axis and y-axis components. The available actions contain longitudinal movement (accelerate, decelerate), lateral movement (right lane, left lane) and staying idle/maintaining the velocity and lane. The results show a great improvement, comparing to the untrained version of the model, but the collision rate is still high in some situations. The testing was performed using Carla Simulator for 7 different scenarios. Two different weather sets are used in the experiment.

When applying the algorithm to **Lane Changing** tasks, in Wang et al. (2019), a rule-based policy is proposed with the goal of reducing the collisions and increase the safety rate. The state

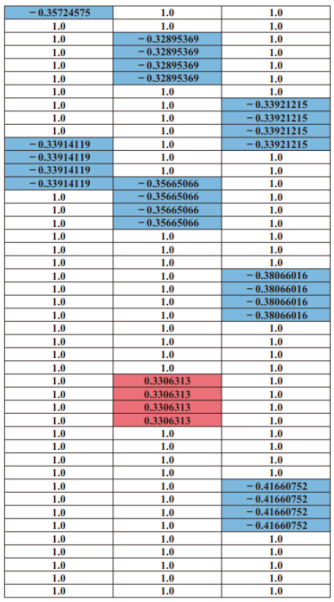


Figure 2.17: Lane Change Discrete State Space. Red cells represent the AV and blue represent the other vehicles. Taken from Wang et al. (2019).

space defined by 45 positions for each of the 3 lanes is defined (60 meters to the front, 30 to the back), with every car within this area occupying 4 positions of the lane. A schema representing the state space is shown in 2.17. The possible actions are simply: changing to the left lane, to the right and maintaining the position. This model is tested using Udacity's Self-Driving simulator, Team (b). The introduction of the rule with the DQN manages to achieve 80% tests with no collisions and reducing the time to change a lane significantly, despite only augmenting the average vehicle speed for a small value.

In Zhao et al. (2020), a DDQN model is used. It decouples the action selection and its evaluation to solve the overestimation error. The state space includes the position (x, y) and the velocity (x, y). The action space has more alternatives than the usual with 10 different possible choices. Regarding the position the car can either maintain itself on the lane or change to either the left or right lane. Regarding the car speed, it can either accelerate, reduce or maintain its speed. By running the permutations between the 3 lane choices and the 3 acceleration commands, 9 different actions are available. Besides these, another action was introduced that maintains the lane and hard brakes the car with a higher deceleration than the other actions. This work was tested using SUMO, Center. An ε-greedy policy is used. The results don't compare the algorithm with other previously implemented ones, but it is clearly shown the performance of the model in solving the task. The speed variations are represented in the figure 2.18. Before the 1000 episodes, some simulations were stopped due to collision, and the minimum speed is 0 m/s, which means hard-breaking is used. After this mark, the collision rate drops to 0% and the speed average increases as the training continues, eventually achieving an average of 25m/s.

In Li et al. (2022), a DDQ approach is employed with different nuances. The state space is a complex and composed structure. A vehicle state is composed by the (x, y) position and the

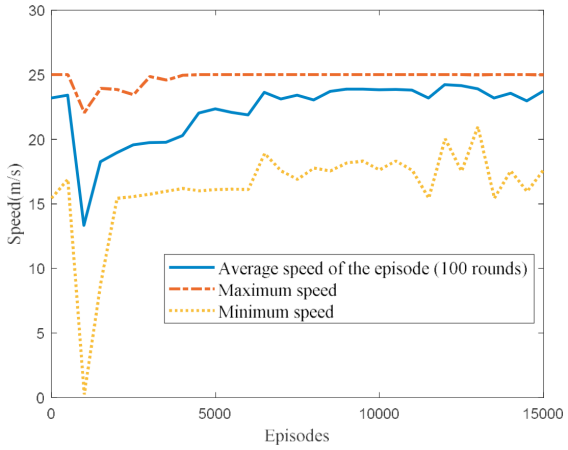


Figure 2.18: Speed Range and Convergence in a Lane Change scenario, taken from Zhao et al. (2020).

speed value v. By grouping many vehicle states, a new structure called vehicle history is created with its trajectory. The state space for the pretended vehicle is finally achieved by grouping the different histories of the surrounding vehicles and its own. The allowed actions, contrarily to other approaches, only contain the lane changes. The authors use a different methodology since they believe adding a negative reward for collisions will slow the convergence. Their solution is restricting the actions. When there are no cars in front of the vehicle, no lane change is allowed. Besides, a safe trajectory planning function is defined and if there is a collision chance, the action is also removed from the set of actions, not allowing it to happen. The proposed model effectively manages to converge way quicker to the desired speed target than MOBIL (General Lane-Changing Model for Car-Following Models), as shown in the histogram 2.19. To simulate the model, the authors define their own environment by employing a trajectory planning model that generates either lane-changing or lane-keeping trajectories, accompanied by the velocity profiles of the vehicles along their route.

As for **Ramp Merging** tasks, other solutions are suggested. In Wang and Chan (2017), an innovative approach is taking by introducing a Long short-term memory (LSTM, Hochreiter and Schmidhuber (1997)) block before the DQN. This helps it understanding sequential patterns in the historical driving information and to feed the Q-Network compact and fixed-size data. The state is composed by the vehicle speed, position, heading angle, and distances to the left and right road markings. The 2 available options are acceleration commands or steering angle changes. The model was trained using real-life data. There's no information on simulation. This work lacks information on results, but shows a different approach that could be experimented. In Mahabal et al. (2022), a mixed approach that uses DQN to the lateral control (lane merging) and DDPG for the longitudinal control (acceleration). While this work does not separate the results obtained in the 2 different algorithms, it brings novelty by splitting the two tasks and is an idea to be further explored as well. The results are satisfactory, since the model converges and stabilizes in around 300 episodes.

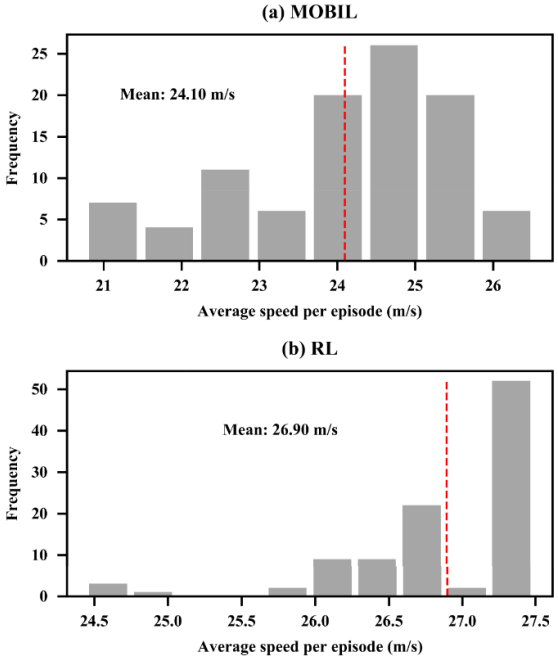
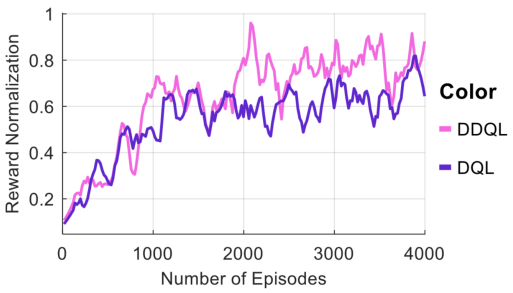


Figure 2.19: Comparison between Model SpeedsLi et al. (2022)

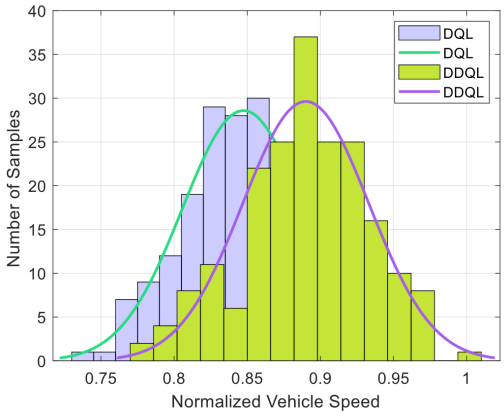
Other common task is the management of **Intersections**. In Wang et al. (2020), in order to solve the sub-task of turning left in a 4-side intersection, both DQN and DDQN algorithms are experimented and compared. The state space is defined by the longitudinal and transversal positions (x, y), the longitudinal and lateral speeds (vx, vy) and the sin and cos components of the heading angle. A boolean value also identifies if a car is considered existent in the scene. The DRL algorithm only controls the longitudinal speed, and has 3 acceleration possibilities: positive, neutral and negative. The lateral movement is controlled by another algorithm further explained in the paper. The two algorithms are compared for iterations using the same parameters. As expected, DDQN has the edge generally, achieving a 6% collision rate after 40 episodes, against a 15% for the regular version. The enhanced version manages to be superior when comparing the rewards, represented in 2.20a and average speed values, represented in 2.20b.

2.3.2.2 Deterministic Policy Gradient

Overtaking tasks have been one of the main targets for investigation on DDPG algorithms. In Kaushik et al. (2018), human-like behaviour is tried to be simulated the most similar way possible. Besides the normal architecture, a layer with curriculum learning is also added to help the model in converging by giving it a clearer guideline in the beginning and reduce the randomness. This is later confirmed to be a huge difference in the results section. A complex vehicle state space is composed by 65 variables (angle between the car and the axis of the track, 19 distances to the track edges in 10 degree spanned measurements in front of the car, normalized distance between car and track axis, longitudinal speed, lateral speed, vertical speed for bumpiness, 4 wheel spin velocities, rpm's and 36 distances to nearest opponents in each 10 degree spanned measurements



(a) Comparison between Model Rewards



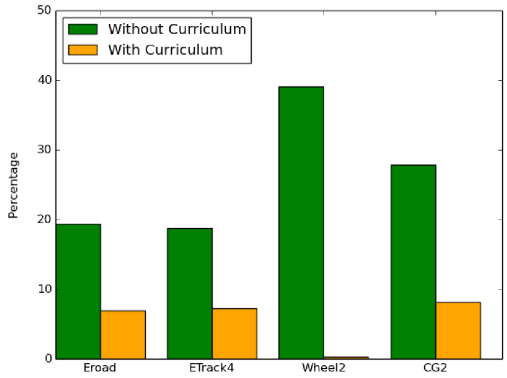
(b) Comparison between Model Speeds

Figure 2.20: Results comparison between DQN and DDQN in an Intersection Left Maneuver Scenario, taken from Wang et al. (2020).

around the whole car). The action space is composed by 3 variables: steering, acceleration and breaking commands. The simulation tool used was a modified version of TORCS, Team (a), Gym-TORCS, Networks. The authors also tried to see how the vehicle would learn differently by removing some layers or reward function parameters, which validated the performance evaluation and comparison. In 2.21a, the table shows the model performance in a 4 vehicles scenario and in a more complex one with 9. In the first, it manages to go through all the vehicles almost every time, having many tasks with a 100% rate of episodes with all overtakes concluded. In the more complex iterations, the vehicle manages to overtake all cars in about 55% of the episodes, but in average still managing to succeeding in 90% of the overtakes. In 2.21b, the authors also show the effect of the Curriculum Learning component in the decreasing of the collision rate.

Track Name	Avg no. of cars overtaken		% of colliding timesteps		% of episodes where agent overtook all cars	
	4 cars	9 cars	4 cars	9 cars	4 cars	9 cars
wheel2	3.95	7.55	0.23	0.23	100	50
Forza	4	7.8	25.835	9.64	100	40
CG2	4	8.45	7.01	8.135	95	65
CG3	3.05	6.15	25.64	39.7	30	35
Etrack1	4	8.35	7.08	1.05	100	80
Etrack2	3.55	7.8	26.41	0	65	60
Etrack3	4	6.35	7.99	2.36	100	40
Etrack4	4	8.5	0	7.23	100	70
Etrack6	3.65	7.55	26.13	10.5	90	60
ERoad	4	8.05	3.25	6.9	100	75
Alpine1	4	8.55	17.45	0.67	100	80
Alpine2	3.9	7.95	7.57	0.71	85	50
Olethros	4	7.1	7.3	18.84	100	30
Spring	3.8	7.8	3.87	8.05	95	45
Ruudskogen	3.95	7.65	2.21	12.29	100	40
Street1	3.95	8.55	4.07	6.19	100	80
wheel1	4	8.5	0	10.38	100	50
CG-Speedway1	3.85	7.95	5.62	7.53	95	50

(a) Comparison between Model Speeds



(b) Curriculum Learning Effect on Collision Rate

Figure 2.21: Result evaluation of a DDPG with Curriculum Learning algorithm on an Overtaking task, taken from Kaushik et al. (2018).

In Chen et al. (2019b), while grouping other autonomous system tasks, such as perception

processing through an attention mechanism, an extra Recurrent Neural Network is used to process temporal information in the form of an LSTM. The authors employ the DDPG algorithm in a actor-critic approach. Using a simple action space with acceleration, brake and steering commands, ruled by a higher hierarchical layer with the left, right lanes change or its maintenance. The rewards set is slightly more complete, containing different types of parameters, including the position, speed, the offset to lane center and yaw angle to the road definitions. The simulation is performed using TORCS. Every enhancement has a direct effect on the results and raise the values obtained in the chosen metrics. The positive change introduced by each iteration can be observed in 2.22.

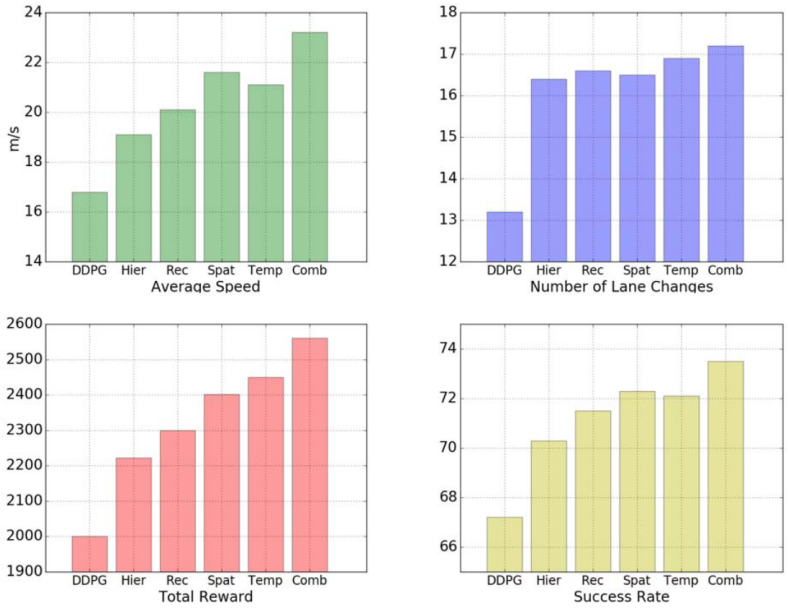


Figure 2.22: Comparison between base model (DDPG), DDPG w/ hierarchical actions (Hie), DDPG w/ recurrence (Rec), DDPG w/ spatial attention (Spat), DDPG w/ temporal attention (Temp) and DDPG w/ all enhancements combined (Comb). Taken from Chen et al. (2019b)

In Li et al. (2020), an actor-critic approach is also taken, using sensor inputs, with relative coordinates. DDPG and DQN are put to comparison. The state variables are represented by the relative distance, distance information to surrounding obstacles, relative orientation angle, velocity of the leader vehicle and velocities of the ego vehicles. The action space is controlled by the forward speed and angular velocity. The simulation was performed in a laboratory with experimental vehicles. The results show a better performance for the DDPG algorithm when compared to the DQN. The algorithms start to converge at similar stages of training, but the reward values region of the DDPG is significantly higher. At the 25 thousand episodes it is also noticeable a big difference in the collision rate, since the DDPG algorithm manages to converge to a residual number. Overall, it excelled in the different metrics, despite having a similar convergence rate. In 2.23, the finals results show DDPG's slight edge on the overtaking time and distance to obstacles, as well as the major reduction on the collisions.

Algorithm	Collision rate	Overtaking time	Minimum distance from obstacles
DDPG	**1.56%**	**20.1 s**	**0.3094 m**
DQN	7.74%	22.9 s	0.3926 m
Fuzzy control	5.92%	31.8 s	0.7323 m

Figure 2.23: Comparison between DQN, DDPG and Fuzzy Control algorithms in the solution of an Overtaking task, taken from Li et al. (2020).

In Ashwin and Naveen Raj (2023), another overtaking scenario is also studied, including obstacles in the car path. The authors put heavy importance on the theoretical study gathered from similar papers. The proposed state space is the velocity, the normalized horizontal offset between the vehicle and centre of the lane, the departure angle between the lane and agent, the angle between the lane's edge and agent and the distances between the agent and the obstacles ahead and sideways. The action state utilises the common approach with steering, acceleration and breaking. The simulation is also performed on TORCS. Different tasks are performed regarding obstacle avoidance, lane keep and the overtaking issue. The results are not compared with other algorithms, but within the model. Different scenarios that gather the different possibilities and the reward and speed variations according to those are shown and explained.

In specific **Ramp Merge** scenarios, other works have also been developed. In Mahabal et al. (2022), DDPG is used for the longitudinal control (acceleration) component of the AD task, while DQN is used for the lateral control (merge). This may be a good alternative since the acceleration component is defined in a continuous space, providing more liberty in the action selection, while maintaining a discrete lateral speed controller.

In Li et al. (2023a), new metrics are introduced in an attempt to improve the algorithm results for this scenario. The ego vehicle state space is defined by its speed and by its (x, y) distances to the merge point and to the final destination point, 200m ahead in the lane. The only action input type is acceleration, which can be negative for breaking. The simulation tool utilised was SUMO. In order to insure a successful merge, the Time to Collision (TTC) metric is used, configuring whether a car is in eminent risk of crashing following its behaviour. To further improve the safety parameter, the authors designed a new metric TDTM. TTC is not really effective when both cars are travelling at similar speeds. This new metric allows the car to compare itself with other in the main lane, and establishing a reward for the distance between the arrival of the two to the merge point, rewarding the security distance left afterwards. The usage of these metrics was evaluated for different methods and is described in 2.4. Despite having a lower success rate without the 2 metrics, its superiority is clear (near the 100%) when these are employed, proving to be useful in the problem solution.

Table 2.4: Comparison of the ramp-merging and overall success rates in a Ramp Merge task between different algorithms (DDPG, SAC, TD), with and without the usage of the TTC and TDTM metrics in each. Taken from Li et al. (2023a).

Method	On-ramp merging success rate (%)	Final success rate (%)
DDPG + TDTM + TTC	99.96	98.56
DDPG + TTC	97.22	92.99
DDPG	86.93	78.35
SAC + TDTM + TTC	94.35	92.68
SAC + TTC	92.09	89.01
SAC	90.60	86.10
TD3 + TDTM + TTC	97.45	96.70
TD3 + TTC	95.71	93.54
TD3	90.22	86.37

2.4 Comparative Analysis

In order to summarize all the revised works and have a general view of the state-of-the-art, a comparative analysis was performed by listing all methods together, along with the respective algorithms and metrics employed. The main patterns found were also highlighted, and in the end, a discussion on the existing research gap is presented.

2.4.1 Path Planning

As previously mentioned, path planning algorithms have been designed using different methodologies over the years. Parametric curves have shown that they can perfectly fit a path to a simple task within a road. However, the constant reception and variation of information from the environment makes this alternative less feasible since it is usually applied in static scenarios. As for the graph and tree based algorithms, they've also delivered results in path planning tasks successfully. Complemented by a curve approximation algorithm, they could be considered an alternative, but since the AD realm may include an extraordinary large space of action, as the real world, the amount of nodes could escalate pretty quickly and the execution time would be too slow. There would also be required the integration of obstacles and unexpected changes into the structure, which would not be ideal. The usage of reinforcement learning has then been explored since, while not too related to AD in path planning problems. As such, a list containing the different RL and DRL algorithms has been grouped into Table 2.5, to provide an easier comparison structure, specially between the different metrics. Overall, the referenced works are related with robotics and their movement planning. The first 3 entries are regular Reinforcement Learning, then there is an entry for a Monte-Carlo approach and the latter 5 are Deep Reinforcement Learning based. Among the various works the metrics who stood out were:

Table 2.5: Planning Algorithms Summary

Work	Algorithm	Metrics
Maoudj and Hentout (2020)	Efficient Q-Learning	Path Length, Computation Time
Low et al. (2023)	Improved Q-learning	Path Length, CR, SR, Computational Time + KSPA
Hao et al. (2022)	Potential and Dynamic Q-Learning	Path Length, Turning Angle, Computational Time
Dam et al. (2022)	Monte-Carlo	Computation Time, Collisions, SR
Raajan et al. (2020)	DQN	Path Cost, Duration, Risk
Yin et al. (2023)	RND3QN	Reward, SR
Bhuiyan et al. (2023)	DDPG	Reward, Path Length, Computation Time
Chen et al. (2022)	DDPG & SAC	Reward, Distance, Accuracy, Safe Rate, Loss
Ren et al. (2022)	DDPG + ELM	Visualisation

CR	Collision Rate
ELM	Extreme Learning Machines
KSPA	Kolmogorov-Smirnov Predictive Accuracy
RND3QN	D3QN with n-step and greedy reward
SR	Success Rate

- **Path Length**: Evaluates how short and efficient the calculated path is.
- **Computation Time**: Evaluates how efficient the algorithm is performance-wise.
- **Reward**: Evaluates the relative success of the algorithm comparing to its training so far.
- **Risk**: Evaluates how risky a path is comparing it to obstacles, borders or space center.

Among these, other metrics were employed, depending on the algorithm needs as shown in the table as: path cost, collision and safe rate, loss or regular graphic visualisation.

2.4.2 Decision Making

The decision making brings some variables relative to the vehicle movement into the problem, such as speed management. As so, the standard algorithms used in path planning cannot be applied here. By introducing velocity, distances and angles into the state space and acceleration and steering into the action space, the motion planning or decision making tasks can be defined. Other variables are also utilised depending on the work. The studied algorithms will then be summarized through a table, allowing the variables, metrics and Simulators to be compared between the different works. Among the different Metrics, the most popular were:

- **Reward**: Evaluates the relative success of the algorithm compared to its training so far.
- **Speed**: Evaluates the speeds in which the vehicle can perform a task.
- **Distance**: Evaluates the efficiency of the path as well as the safety distance to other vehicles.
- **Number of Successes**: Evaluates the discrete number of Lane Changes, Overtakes... in a scenario to compare its overall success with other proposals.
- **Collision and Success Rate**: Evaluates how safe or not the vehicle behaviour was in the different episodes and its general success.

Table 2.6: Decision Making Algorithms Summary

Work	Scenario	Algorithm	S	A	Sim	Metrics
Wang and Chan (2018)	Ramp Merge	Q Learning	4	3	-	Reward, Loss
García Cuenca et al. (2019)	Roundabout	Q Learning	7	2	CARLA	Reward, Speed, Distance, Deviation
Gritschneder et al. (2016)	Roundabout	Q Learning	4	4	-	Speed
Hossain (2023)	Overtaking	DQN	4	2	CARLA	Reward, Speed, CR
Wang et al. (2019)	Lane Change	DQN w/RC	45	3	USDS	Speed, Lane Changes, SR
Zhao et al. (2020)	Lane Change	DDQN	4	10	SUMO	Reward, Speed, CR
Li et al. (2022)	Lane Change	DDQN w/RC	t*	2 ... l	Self-Defined	Speed, CPU, Lane Changes, Overtakes, States
Wang and Chan (2017)	Ramp Merge	DQN + LSTM	5	2	-	-
Mahabal et al. (2022)	Ramp Merge	DQN ‖ DDPG	-	-	-	Reward, Speed, Distance
Wang et al. (2020)	Intersections	DQN & DDQN	6	3	-	Reward, Speed, Length, Loss, CR, Actions
Kaushik et al. (2018)	Overtaking	DDPG + CL	65	3	Gym-TORCS	Reward, Overtakes, CR
Chen et al. (2019b)	Overtaking	DDPG+	4	3	TORCS	Reward, Speed, Lane Change, Success
Li et al. (2020)	Overtaking	DQN & DDPG	5	2	Lab	Reward, Distance, CR, Overtaking Time
Ashwin and Naveen Raj (2023)	Overtaking	DDPG	8	3	TORCS	Reward, Speed, Angle, Position
Li et al. (2023a)	Ramp Merge	DQN w/TDTM	4	1	SUMO	Speed, Distance, Success

+	Different Enhancements
A	Number of Actions in Action Space
CL	Curriculum Learning
CR	Collision Rate
RC	Rule Constraint
S	Number of States in State Space
Sim	Simulation Tool
SR	Success Rate
RND3QN	D3QN with n-step and greedy reward
SR	Success Rate
TDTM	Time and Distance Merge Metric

3t* State space for vehicle defined by the number of time intervals in its trajectory, each containing *x, y, v*.

Besides these, other possibly useful metrics have been used to evaluate the algorithms such as: the vehicle positions, speeds and accelerations compared to the target values (similar to visualisation), the execution percentage and probability of the different actions, the model loss and the CPU performance during the algorithm running time.

2.4.3 Additional Considerations

The different algorithms were found by searching them on scientific tools and databases, as well as by utilising data gathered in different surveys. Such as: Kiran et al. (2020), Elallid et al. (2022) and Teng et al. (2023).

Among the different papers, various metrics were marked as well-aligned to evaluate the studied AD task, as seen in the tables, but other reward formulas or algorithm definitions were also considered and documented. On this matter, the Time to Collision (**TTC**) or **TDTM** are propositions that were employed in at least one paper and are useful in the RL algorithms definition. The Kolmogorov-Smirnov Predictive Accuracy test (**KSPA**) was also a different evaluation metric noted separately than the rest, useful for the path validation.

The different works have mainly explored and evaluated the RL algorithms within their own problem, mainly because to compare them with others, the same simulation scenarios and problem definitions should have been used. Nevertheless, with different levels of depth, the general **results** are sufficient as a know-how of what should be expected regarding the performance and safety of these algorithms. The **problem definitions** also vary a lot, which ends up being useful for the consideration of different environments and problem approaches, including the main disparities between continuous and discrete state spaces and the different parameters and importance

given to some values and metrics in the reward definition. The **simulation tools** are various and depend a lot on the explored problem. Some papers did not refer the usage of any, but there is still a wide-range of options between all. Some works provided information on the differences between DQN and DDPG, the two main explored **algorithms**, but these are not sufficient to make a statement on their performance, since it is highly dependent on the chosen environment and the added enhancements.

2.4.4 Research Gap

After searching on the different AD tasks and the existence of RL and DRL algorithms studying the behaviour of AS's, there is a clear difference between the amount of investigation for some scenarios. Regarding the Roundabout Maneuvering task, some RL works have been published, always for a specific scenario within the problem, since this road element has many possible designs and forms, augmenting its complexity. It is understandable that some AD tasks are simpler to solve, and that reflects on the lack of investigation regarding this issue.

Regarding the main DRL algorithms, they have been employed in different forms and with various enhancements, to all the other tasks, so there is a wide range of options that could be applied to the Roundabout scenario.

The general understanding, is that there is a substantial lack of investigation regarding the application of DRL into AD in a roundabout, which reflects a big research gap on the topic. Besides the application of a DRL algorithm base version to a simple roundabout, many iterations using algorithm enhancements, different algorithms, different roundabout types and road and weather conditions are also yet to be explored.

2.5 Summary

This chapter reviewed the state of the art in the application of Deep Reinforcement Learning to autonomous driving problems. The two main focuses regarding this were Path Planning and Decision Making tasks. Each of these areas were researched thoroughly, always with the intent of exploring different scenarios (lane changes, intersections, roundabouts...) and using distinct approaches regarding important factors, such as the state definition, the reward functions and the evaluation metrics. This in-depth analysis of the various algorithms culminated in the collection of a broad set of information which will serve as a base for the experiments regarding the main problem. A comparative analysis of the principal methods, assessing their key parameters and metrics was also performed, demonstrating the general overview of the State of the Art and all known possible approaches. By identifying these challenges, this chapter sets a solid base to not only sustain the carried work on this topic but also to guide future research efforts towards addressing these gaps and enhancing the overall field.

Chapter 3

Methods and Materials

Choosing the appropriate algorithm and parameters necessitates a thorough consideration of the overarching goal. From the two primary branches of techniques, a pivotal decision revolves around opting for either a discrete or continuous space. While a discrete space would simplify the learning process and enhance overall stability, the primary objective of this experiment is to grant the agent complete autonomy in action selection, enabling it to maneuver and control its position using any valid controls. Hence, the preferred approach remains a continuous state space coupled with the implementation of a DDPG structure. Moreover, this approach has been relatively less explored, particularly in tasks involving roundabout maneuvering, as discussed in Section 2. The cost associated with this architectural decision will be substantial, introducing complexity to other aspects of the design and potentially leading to initial volatility in training and challenges in defining the reward function. This complexity represents a significant hurdle in achieving positive results, alongside the parameter tuning process. Evaluating how various modifications impact the agent's learning process and overall training performance with respect to reward will constitute the primary contribution and research focus of this work. Consequently, all definitions and decisions outlined in this chapter are framed within this approach. The specifications of the DDPG network serve as the foundation and rationale for many of these decisions.

3.1 State Space

Concerning the state space fed to the network, the different works address their problem based on factors such as the input size, the availability of data or the general approach regarding the Reinforcement Learning and the reward calculation ideas. The most common option is using a camera sensor, as in Chen et al. (2019b). The advantages are set with the direct application of the received images into the neural network that always receives the same type of input. By using this approach, a full pipeline of AD is implemented, since it starts with the perception of the sensors in the network entrance, and it outputs an action to control the car movement. Since there is no overhead between sub-tasks and can train a fully autonomous system, making it the preferred paradigm. The main issue with it is that the size of the input will be larger than other

approaches, slowing the training substantially. Reducing the image size and quality can be a positive pre-processing, but it is capped by the loss in the ability of the network to capture the desired features. Other works, generally related with simple to map tasks, as lane changes (Li et al. (2022)) or overtakes (Kaushik et al. (2018)) on a straight road, can reduce the size of the input, by just utilising the useful features in the scene, as the distance to the side or to other cars. This reduces the complexity of the information fed to the network, but when the state space contains many variables and irregular shapes, it is really difficult to define a standard. It also depends on the number of agents, which amplifies the variance of the input. Therefore, sensing is not directly integrated into the DRL architecture. Using similar information retrieval methods, or a set of camera perspective transformations, the input can also be in the form of a bird's-eye view. The advantage of this approach is that it can condense the information from a 3d camera into a 2d view, which can be further reduced into a smaller input while still providing all the desired information. It can be considered as a mix of the previous two, and is employed in some works such as Chen et al. (2019a) Considering the objective of focusing on the Motion Planning task, the usage of images was discarded, so that the controls and route planning would be the focus and no extra innacuracy was introduced in the solution of the issue. Also, given the shape of the roundabout (the entrances, exits, curved sidewalks...), the other approach would be really hard to map into direct variables. As such, the bird's-eye view input was the selected one. In order to avoid noise and issues with the perception and processing of the environment, all information was directly retrieved from the simulation and later run through a set of transformations to create a valid scene the network can learn from.

In this iteration of the work, the multiple lane restraints and requirements were not considered.

The roundabout scenario demanded a 100x100m area, which includes both the circular road and margins with some space for the entrances and exits. The state space choice relies on how detailed and precise this representation should be. The best option is to find a middle ground between precision and the input size. The value attributed was 0.5x0.5m per square. It balances out since 0.5m is a short distance that can track a collision or near miss between cars until an acceptable extent, while maintaining the input with a 200x200 shape, which is not significantly large. This map is then filled with the entities located within its range. In order for the model to work, we need to feed it three different important state variables: velocity, orientation and type. These will eventually induce the model to take actions based on its values. The type is divided in the 4 categories that need to be distinguished: empty space, ego vehicle, obstacles (walls and other vehicles) and goal. To send all these informations to the network, the type, velocity and orientation are set to have a map each, where the object's respective tiles or squares are filled with the mentioned values. Non moveable objects have null velocity, and a negative standard orientation value. As such, the input becomes a 200x200x3 matrix containing all data. In Wang et al. (2019), the authors utilize the same methodology of the first layer.

The generation of the goal and vehicle spawn point was done randomly, so that the vehicle would train the path to farther exits and from different points of view.

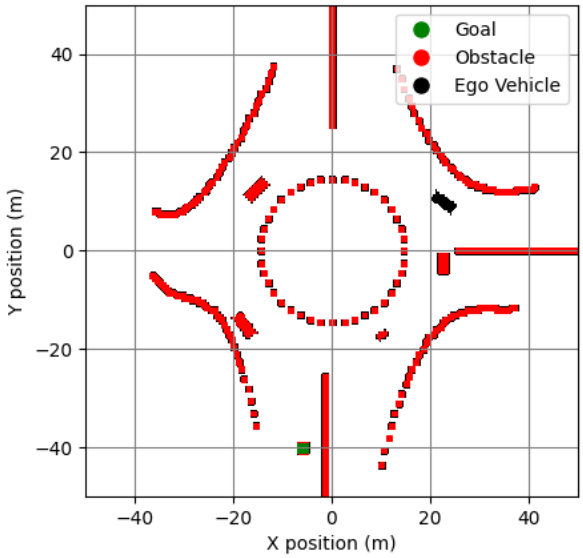


Figure 3.1: State matrix with the object type representation for a random scenario.

The type matrix contains the locations of the objects within the environment, and can be visually represented by mapping a different color for each intervenient. An example of this is shown in Figure 3.1. In the illustration, various types of actors can be observed: the goal is denoted by a green square, the ego vehicle by black, and all obstacles, encompassing external cars, walls, and sidewalks are represented in red, as depicted within the roundabout shape and traffic scene.

Given the layers capturing the three state variables, the state space is defined by three key matrices. Each matrix represents the values for object type, tile velocity, and tile orientation respectively. The comprehensive state space S, encompassing all current conditions, is structured as follows:

$$S = S_1, S_2, S_3 = T, V, O \tag{3.1}$$

where:

- T represents the Object Type matrix.
- V represents the Tile Velocity matrix.
- O represents the Tile Orientation matrix.

Each element $S_{i,j}$ of the state space S can be described as:

- $T_{i,j}$: Object Type at tile (i, j)
  - Values: $\{\text{ego}, \text{Obstacle}, \text{Empty}, \text{Goal}\}$
- $V_{i,j}$: Tile Velocity at tile (i, j)

  - Values: $[0, \text{speed\_limit}]$

- $O_{i,j}$: Tile Orientation at tile (i, j)
  - Values: $\{-1\}$ (if no car) or $[0, 360[$ degrees

The full state space S can be described by three separate matrices as follows:

$$S = \begin{bmatrix} T_{1,1} & T_{1,2} & \cdots & T_{1,200} \\ T_{2,1} & T_{2,2} & \cdots & T_{2,200} \\ \vdots & \vdots & \ddots & \vdots \\ T_{200,1} & T_{200,2} & \cdots & T_{200,200} \end{bmatrix}, \begin{bmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,200} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,200} \\ \vdots & \vdots & \ddots & \vdots \\ V_{200,1} & V_{200,2} & \cdots & V_{200,200} \end{bmatrix}, \begin{bmatrix} O_{1,1} & O_{1,2} & \cdots & O_{1,200} \\ O_{2,1} & O_{2,2} & \cdots & O_{2,200} \\ \vdots & \vdots & \ddots & \vdots \\ O_{200,1} & O_{200,2} & \cdots & O_{200,200} \end{bmatrix}. \tag{3.2}$$

3.2 Action Space

As for the action space it is defined by two variables. The first variable is acceleration, a, which controls both forward speed and braking (represented by negative values). Negative acceleration corresponds to braking, maintaining the vehicle's current velocity when at a standstill. The second variable is steering angle, , which manages the vehicle orientation and, consequently, position adjustment based on current velocity.

As mentioned earlier, DDPG utilizes a continuous action space where the control commands are constrained within the interval [-1, 1] for both action variables. These bounds represent the maximum acceleration (or braking) and steering angle for directional control.

This can be expressed by the following equation:

$$a_{vehicle} = (a, \delta) \tag{3.3}$$

3.3 Rewards

The reward function responsible for evaluating the vehicle's actions will have different sub-functions, each aimed at teaching the agent a different condition of the environment on its way to the goal state. As the roundabout scenario is complex within the AD domain, these equations are not trivial and will thus include several branches.

The first component is associated with the reward for safety. Its primary function is to penalize illegal or collision causing actions. This includes crashes with other vehicles or walls, driving in the opposite direction, or navigating onto sidewalks, grass, and narrowly avoiding other entities. While only crashes result in tangible losses (injuries, vehicle damage, etc.), the others are also considered invalid actions and are therefore penalized equally with a negative constant (1000), leading to termination of the episode. An additional penalty is imposed for early crashes, calculated as the difference between the time limit (*max(t)*) and the elapsed time, Δt, due to the invalid action, discouraging the vehicle from choosing this over more severe penalties during the entire driving sequence. This relationship is described by Equation 3.4.

$$r_{\text{safe}} = -1000 + (\max(\text{t}) - \Delta\text{t})^2 \times 6 \tag{3.4}$$

The second element is the reward for prevention. This integrant of the function was based on the work of Li et al. (2023a), which utilises a Time to Collision (ΔTTC) metric in a Ramp Merge scenario to verify whether 2 vehicles are in the imminence of crashing by calculating their immediate future positions using their speed, orientation and current location. The reasoning behind it is assessing the likelihood of a potential crash and subtly discourage risky maneuvers. Its emphasis is on prevention and is parameterized by the Equation 3.5.

$$r_{\text{prevention}} = \begin{cases} -30 & \text{if } 0 \leq \Delta\text{TTC} \leq 0.5 \text{ seconds} \\ -15 & \text{if } 0.5 < \Delta\text{TTC} \leq 1 \text{ second} \end{cases} \tag{3.5}$$

Next comes the speed reward. The objective is to complete the path as quickly as possible, so the velocity values v will be positively rewarded. A penalty is also imposed when the vehicle is stopped to discourage it from remaining stationary to avoid failure or crashes. Equation 3.6 describes this behavior, outlining the expected return for both scenarios

$$r_v = \begin{cases} 2v & \text{if } v \geq 1 \\ -100 & \text{if } v < 1 \end{cases} \tag{3.6}$$

Succeeding it, there is the reward stability component responsible for the smoothness of the vehicle trajectory. It aims to penalize harsh steers, δ, braking, or accelerations, a. The ideal run would be a smooth drive around the roundabout, so the larger the absolute value of the control, the greater the penalty. The values are cubed to increase the penalty for higher command values, as seen in Equation 3.7. Another approach for this reward could focus on rewarding stable decisions instead of penalizing unstable driving.

$$r_{\text{stability}} = -((5 \cdot a)^2 + (5 \cdot \theta)^2) \tag{3.7}$$

The reward for distance d is included to propel the vehicle forward and complete the roundabout trajectory. Since in every episode the entrance and targets will be randomly defined, the vehicle can be aiming for any of the roundabout exits. This implies the distance to the goal would

be different in every iteration, and the distance to complete a full turn around maneuver would be very small, despite the need for the vehicle to traverse the entire roundabout. This miscalculation confuses the agent unnecessarily. Therefore, the distance traveled is quantified by the percentage of the length already covered, *d%*, along a predefined path from start to finish, adhering to roundabout rules. This metric ensures consistency across different traffic scenarios. The closer the agent is to the goal, the greater the benefit. Additionally, to encourage vehicle movement from the outset, an initial bonus component is added. This supplementary parameterization, combined with the overarching function that maps the distance reward, is described in Equation 3.8.

$$r_{\text{distance}} = \begin{cases} (\text{d}\%/4)^2 & \text{if d}\% \geq 20 \\ (\text{d}\%/4)^2 + 5 \times (1 - \text{d}\%/20) & \text{if d}\% < 20 \end{cases} \tag{3.8}$$

Finally, the reward for success is a constant value given when the vehicle reaches its destination within a distance threshold *dt*. An additional bonus is awarded for completing the trajectory in less time, calculated using δt and *max(t)*. This function is described by Equation 3.9.

$$r_{\text{success}} = 1000 + \left(\frac{max(t)}{\delta \text{t}}\right)^4 \text{ if dt} < 2 \tag{3.9}$$

The different functions contribute to the overall reward definition, which evaluates the value of an action taken by the agent and influences its training to navigate the roundabout towards the goal point. The complete reward function consists of all these components, as described in 3.10.

$$reward = r_{\text{safe}} + r_{\text{prevention}} + r_v + r_{\text{stability}} + r_{\text{distance}} + r_{\text{success}} \tag{3.10}$$

3.4 Parameters

The parameterization of the models involves numerous conditions that can contribute positively or negatively to performance, depending on execution quality. The number of variables does not permit an exhaustive brute-force search for optimal parameters, so decisions were made based on domain knowledge, while others underwent experimental validation.

The networks were initialized with a ResNet-18 architecture (He et al. (2015)). While other options were considered, it struck a balance between depth and computational efficiency. It is sufficiently deep to capture complex patterns while remaining computationally feasible and relatively fast to train and evaluate compared to deeper models. Its widespread adoption and well-understood performance also facilitate comparison and benchmarking with other methods.

Regarding learning rates, different values were chosen for the actor and critic networks. The actor network had a lower value (1e-3) to avoid large and unstable updates, ensuring a stable learning process. The critic network used a slightly higher rate (1e-2) to facilitate faster learning of action values and provide timely and accurate feedback to the actor network. Both rates were decayed to increase gradient variance initially and promote stability later, based on optimal performance observed through experimentation.

The discount factor γ, balances consideration of future rewards versus immediate rewards. It was set to 0.99, striking a balance where the agent learns to make decisions with a long-term perspective while still valuing immediate gains.

The soft update factor τ, controls the rate at which target networks update toward current networks, influencing algorithm stability. Its value was set to 1e-3, ensuring smooth updates that contribute to stable learning.

An ε-greedy policy was employed to ensure the agent focuses on exploration during initial training stages and gathers sufficient information about the environment before exploiting learned information. A minimum value for the decay factor ensures an exploration component is always present.

The replay buffer size, which stores past experiences, was set to 1000. A sufficiently large buffer ensures a diverse set of experiences, which helps break correlations between consecutive samples.

The batch size, representing the number of experiences trained simultaneously, was set to 128. This value balances computational efficiency without compromising stability and learning by processing too large a set of samples.

Regarding the environment, the number of vehicles included per experience varied between 6 and 10. This range ensures sufficient traffic to challenge the agent without overwhelming its ability to navigate.

Each episode had a timeout of 20 seconds, allowing the vehicle to carefully cross the roundabout. If no action was taken within this timeframe, the episode reset.

The agent was set to train over 10,000 episodes initially, with flexibility for fewer test runs to simplify performance measurements.

The general range of values experimented, determined through domain knowledge and empirical experimentation to strike a balance between learning efficiency and stability, is summarized in Table 3.1.

3.5 Summary

This chapter explores the methods and materials used in the study, focusing on the foundational concepts of Reinforcement Learning, network architecture, and parameterization essential for training an autonomous vehicle to navigate a roundabout using DRL. The state space choice of a bird's eye view and the utilization of a continuous action space were crucial design decisions, that despite the complexx environment, enable the agent to perceive and act effectively. The reward function was tailored according to the state, to prioritize safety, speed, stability, and successful navigation towards the goal, integrating all elements within a comprehensive function that will guide the agent's learning process effectively. The ResNet-18 architecture was chosen for its depth and computational efficiency, facilitating effective pattern recognition during training and evaluation. Key parameters such as learning rates, discount factors, and soft update rates were meticulously

Table 3.1: Hyperparameter value ranges used in the DDPG experiments

Parameter	**Range**
Learning Rate Actor	[1e-4, 1e-3]
Learning Rate Critic	[1e-3, 1e-2]
Learning Rate Decay	[0.9999, 0.99999]
Learning Rate Minimum	[5e-5, 1e-4]
γ	[0.99, 0.999]
τ	1e-3
ε-decay	[0.9999, 0.999975]
ε-minimum	[0.05, 0.1]
Replay Buffer Size	[1000, 10000]
Batch Size	64, 128
Number of Vehicles	[6, 10]
Episode Timeout	[15, 20]s
Number of Episodes	[1000, 10000]

selected through a combination of domain knowledge and empirical testing to enhance learning stability and efficiency.

Chapter 4

Experiments & Results

In this chapter, various experiments regarding the parameterization of the reward function and general Reinforcement Learning definitions are presented, which will allow an analysis on the viability of the motion planning task in the mentioned environment and on the most relevant features in the task. The tools that enabled the execution of the experiments will also be further explained.

4.1 Used Tools

Several tools played essential roles in conducting the experiments, with the simulation environment standing out as the most crucial. This environment was responsible for defining the state and executing actions for all participants, ensuring the experiments ran smoothly. In addition to the simulation environment, a number of key libraries were incorporated into the code, each contributing to different aspects of the experiment's functionality. These libraries will be discussed in detail to highlight their importance. Furthermore, the hardware specifications used in the experiments are outlined in this section, providing a comprehensive overview of the technical resources that supported the experimental process.

4.1.1 Simulation

CARLA (Car Learning to Act) is an open-source simulator designed for the development, training, and validation of autonomous driving systems, CARLA Team. It provides a set of highly flexible and realistic urban environments containing a large variety of road layouts. As such, its main qualities dwell exactly on the training of AV's in these urban scenarios.

Despite this work not requiring the existence of a visual input from perception sensors (one of CARLA's best predicates), this simulator contains unique features that proved to edge other options, such as the inclusion of pedestrians and different sized vehicles or the high precision modelling of the vehicle movement and general mobility. The exclusion of graphics was also useful since it removed the need for rendering, and in consequence the GPU could be fully exploited for the model training.

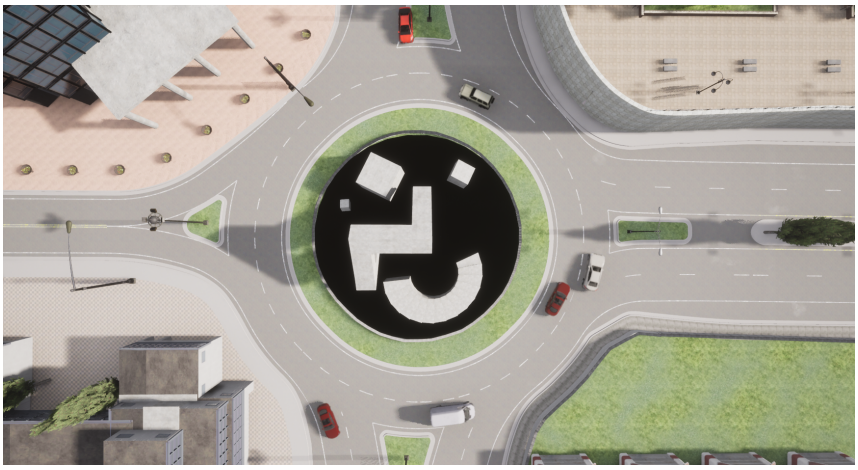


Figure 4.1: Top view from CARLA's map 3 roundabout that serves as basis for the state space and environment definition.

The extent of CARLA's API provided all sensing information required in the state space. This includes the locations and bounding boxes of the actors and its speeds and orientation. The static obstacles had an issue regarding their definition and length, which left many walls or other objects with undefined sections in between. This issue was solved with the usage of Spline Curves that approximate the wall shape to fill it afterwards in the environment matrix.

The simulator contains an urban map with a two-lane central roundabout structure with different sized exits. This provided the necessary background for the training and was the main core of the training environment.

All the vehicle movement patterns were simulated using the pre-defined autopilot feature. The ego vehicle's actions were carried out using the simulator API, which executes the car movement with precision given a certain command.

An example of the scenario is shown in Figure 4.1.

4.1.2 Hardware

There were a few critic specifications regarding the hardware used to train the model, with special focus in the GPU. Carla is an Unreal Engine based simulator, which makes it a heavy software. The minimum requirements suggested for the gpu is having 8gb of VRAM. After a few runs, even this value proved to be unsufficient causing crashes frequently. Having more than 30gb disk space available is also necessary with the package installation. The source version would occupy 5 to 6 times more.

With the removal of the visual requirements, the graphic card needs from the simulator were also withdrawn. Despite this, training the model still requires a high performance GPU, along with a solid pack of components that does not cap the overall execution, namely the CPU and the memory RAM. Considering the referred needs, the setup listed in Table 4.1 was selected as the main host of all the experiments described in this chapter. In this list, the referred requirements

Table 4.1: Hardware Specifications

Component	**Specification**
Processor	Intel Core i7-13700K "Raptor Lake" 16-Core 2.5GHz with Turbo 5.4GHz, 30MB Cache, Socket 1700
Graphics Card	NVIDIA RTX A6000 48 GB GDDR6
Motherboard	ATX MSI MPG Z790 Carbon WiFi
Memory	G.SKILL Trident Z5 RGB 64GB (2x32GB) DDR5-6000MHz CL36, Black
SSD	M.2 2280 WD_Black SN850X 4TB 3D NAND NVMe PCIe Gen 4.0x4

can be found, namely a 64GB GPU, an SSD disk with 3TB of storage and a high performance processor with 16 cores.

4.1.3 Libraries

Several essential libraries were utilized to facilitate the development and evaluation of autonomous driving algorithms. All code was developed using Python.

NumPy enables efficient handling of large-scale numerical data and matrices (Harris et al. (2020)). While working with all kinds of operations in a matrix like input, this library enhanced the efficiency in all searches, copies, updates, etc, ultimately resulting in better data manipulation and mathematical operations.

PyTorch, one of the most known deep learning framework, was employed in the construction and training of the neural network architectures (Paszke et al. (2019)). Specifically, the usage of the ResNet model in the Actor and Critic Networks was enabled by this library. It contains all functions and capabilities required in the training and testing pipelines and is responsible for all the Reinforcement Learning related processing.

TensorBoard, integrated through PyTorch, complements it by providing visualization tools for monitoring and analyzing the training process (Abadi et al. (2016)). It enabled the tracking of metrics like loss functions and accuracy over the episodes, thereby facilitating in-depth performance analysis and model optimization.

Matplotlib is also a visualization tool wich enriched the analysis component by offering a diverse tools, specially the large variety of plots (Hunter (2007)). All relevant statistical data and metrics were translated into graphics from this library, which aided a lot in the illustration of the simulation results.

Besides these, an important and fundamental tool related with the libraries was CARLA's Python API, which supported all integration between the simulator and the RL component, by establishing all comunication protocols between client and server and generating, updating and delivering the states used in the training and providing information useful for the reward function and generall evaluation of the agent behaviour.

Collectively, these libraries synergized in support of the experiment training and testing setups, enabling the study of the topic in focus.

4.2 Evaluation Metrics

The Path Planning and Decision Making problems differ moderately in the evaluation assessments. While Path Planning is characterized more through path length, smoothness and cost, the latter relies on other type of metrics, such as collision rate or speed. This was previously discussed in Section 2.4.

With the inclusion of both subproblems in the study, these quantification tools also converged to be a combination of both, however, always with a more defined presence of the second.

Condering everything, a set of metrics was defined and is able to evaluate different behaviours and training stages of the agent:

- From a general point of view, the vehicle can achieve 3 outcomes. It can collide, succeed, or run out of time without doing the previous 2. This result can be seen as the most global product of a run, which makes the **Collision Rate (CR)** and the **Success Rate (SR)** really useful estimates to assert the overall fulfilment of the task.

- As much as these 2 are simple, the performance of the vehicle cannot be summarized by a binary result, since it may fail in driving all the path, while still trasversing 90% of it, for example. In consequence, as a second level of validation, the **Percentage of the Distance Run** was considered the most valid metric. The distance being measured in percentage is driven by the different distances depending on the pair entrance-exit of the episode and was further explained in the section 3.3. This specific computation is increasingly beneficial to evaluate unsuccessful runs, given that it directly maps how close or far the agent was from succeeding.

- In order to validate how stable and efficient was the run, two types of metrics were included. First, the **Average Speed**, is helpful in evaluating not only how well was a successful run executed, but also how confident the vehicle moved even if in failure. In second place, the control variables variations. Composed by the **Speed and Orientation Variations**, these keep track of how stable a run was, providing a score that translates how curvy the path was and how harsh and irregular the acceleration and braking were employed.

- As a success-only comparison value, the **Time to Conclusion** was considered to validate how quickly the path was traveled, similarly to the speed, but focused on the finished episodes.

- Finally, the overall **Reward** was contemplated aswell, as a direct assessment of how well the vehicle performance across all parameters and available measurements.

Table 4.2: Main relevant hyperparameter values used in the Reward Standardization Experiment.

Parameter	**Model 1**	**Model 2**
Reward Normalization	No	Yes
Hard Updates	No	
Learning Rate Actor	1e-4	
Learning Rate Critic	1e-3	
Number of Episodes	10000	

4.3 Experiments and Training

As was introduced, the main outcome intended to be derived from the experiments is the direction in which both the model parameters and the reward definitions should evolve to extract the most from the training and achieve better results. In this section, descriptions of different experiments carried out with the purpose of comparing specific components of the overall DRL definitions will be presented. Accompanying the results and reasons for each choice, there will be a review of the aftermath of the experiments and a discussion on the possible fundamentals behind the demonstrated behavior.

4.3.1 Reward Normalization / Standardization (Model)

One of the first aspects encountered at the beginning of the experiments was related to the reward scale and order of magnitude. Considering the number of parameters and variables present in the reward function, pre-tuning the range of values to a stable setup was not trivial, especially since deviating from real-life order of magnitude values makes the comparison between different function components harder. This meant velocity and distance-related values would stay in the dozens range, as all other small rewards and penalizations, in order to maintain balance. Collision and goal-related rewards would be between 1 and 2 orders of magnitude above these to sustain their impact.

The issue with this type of approach is the increased probability of having a loss explosion. When working with high values, it is not uncommon for a certain set of actions in an episode to escalate the reward, interfering with the loss calculation and disrupting the current evolution. When this scenario occurs, it is difficult for the model to return to normalcy. As such, the normalization or standardization of the reward is a common approach in Reinforcement Learning and is generally recommended to prevent these scenarios. The process used to transform the reward was a Z-Score normalization.

To evaluate the impact these two approaches had on the training, a few initial iterations were run without standardizing the obtained reward values. Two models with synchronized configurations but differing in the reward scale were tested. The first did not include any standardization (Model 1) while the second one did (Model 2). The set of parameters is stated in Table 4.2.

This experiment compares the actor and critic losses for both models. The order of magnitude of the reward function influences the loss values as well. As such, the evolution of each run cannot

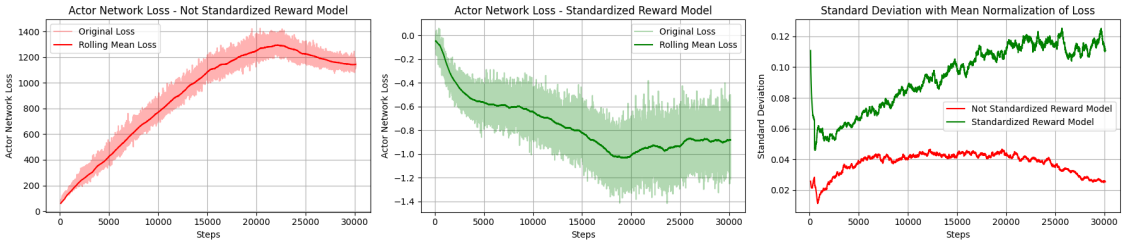


Figure 4.2: Actor Network Loss Comparison between Models with and without Reward Standardization

directly be compared given the range of values in each of the loss graphics. However, a review of the general function behavior, visualized in the plot shape, is still possible. In order to have a more direct comparison between both, the functions were normalized using their mean value. This method creates a new list with relative values, transforming all values into a similar order of magnitude. Using this, the comparison between the standard deviations of both is possible, and a few conclusions can be drawn from it. Considering all, the Actor Network Loss Plots in Figure 4.2 and the Critic Network Loss Plots in Figure 4.3 were generated. The first two graphics represent the loss for the standardized (shown in green) and non-standardized (shown in red) models, and the last represents the deviation comparison between both.

The differences in loss's overall range of values are noticeable from the start. This was expected considering the difference in reward scaling, which is approximately 3 orders of magnitude apart. The same relationship is reflected in the actor network loss, which exhibits a similar scale in Figure 4.2. As for the Critic Network, the disparity gets bigger, with the ratio between both scaling up to 5 orders of magnitude in Figure 4.3.

With the wider reward ranges in the non-standardized model, training became susceptible to loss explosions, potentially disrupting the run and causing the loss to fluctuate too much. After conducting tests, two opposing trends were observed. In the actor loss, which is responsible for learning and updating the policy, the non-standardized version ended up achieving a lower standard deviation, as seen in Figure 4.2. The stability and variance of this metric were similar across both models. The difference can be attributed to the use of mean normalization, which can lead to a mismatch in ranges. Additionally, there is also information regarding the loss variation. Ob-

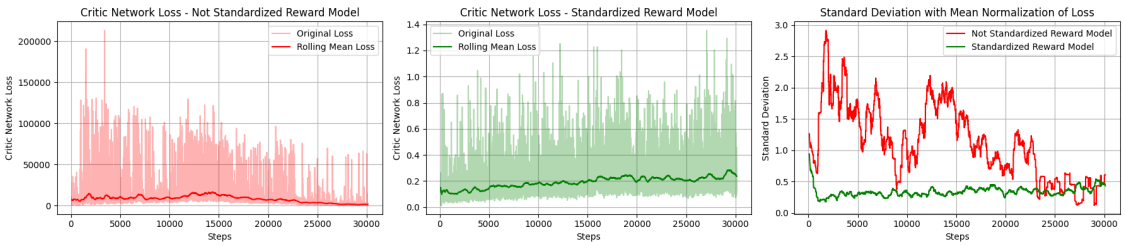


Figure 4.3: Critic Network Loss Comparison between Models with and without Reward Standardization

Table 4.3: Main relevant hyperparameter values used in the inclusion of Hard updates experiment.

Parameter	**Model 1**	**Model 2**
Hard updates	Yes	No
Reward Normalization	Yes	
Learning Rate Actor	1e-4	
Learning Rate Critic	1e-3	
Number of Episodes	6000	

serving the linear behaviour of the function, the standardized loss, despite having a greater range (indicated by the larger area), shows less abrupt average fluctuations. While it may exhibit more ups and downs, it does not escalate as quickly as the other. In the comparison of critic network losses, the paradigm shifts as expected. When comparing deviations, the non-standardized model not only has a higher value, but its instability is also significant, as depicted in Figure 4.3. The variance in loss across all steps shows many ups and downs, ranging from similar values to up to 10 times bigger in a few steps. This behavior can be explained by the previously mentioned issue. When rewards are sparse or highly variable, with potential loss explosions, the critic network may find it challenging to accurately predict the long-term value of actions and evaluate the agent's performance, contributing to instability during its training.

The comparison of other metrics did not reveal any major differences. The overall conclusion from this experiment is that the standardization of rewards initially seemed over-emphasized, given the similar performance across most metrics. However, the instability observed in the critic network loss raises concerns, suggesting potential issues later on. Therefore, the chosen approach is to standardize the inputs using Z-Score.

4.3.2 Soft & Hard Updates (Model)

Another significant decision considered was whether to include hard updates in the network architecture. Soft updates involve gradually updating the target network parameters to slowly track the parameters of the main network, whereas hard updates involve completely replacing the target network parameters with the learned network parameters at certain intervals.

The standard practice in DDPG is to frequently use soft updates, and the occasional introduction of hard updates is less common. The first facilitate a stable and gradual learning process where the target network parameters are interpolated with those of the main network. This approach is widely favored because it prevents rapid changes in the target networks, which can lead to instability and suboptimal performance. However, during the initial phases of training, the agent's policy and value functions evolve rapidly. Introducing hard updates occasionally can assist the target networks in catching up quickly, reducing lag and ensuring they accurately reflect the current policy and value estimates.

Therefore, a study on the impact of occasionally using hard updates was deemed necessary, with an expectation that differences in loss convergence would shed light on the optimal approach.

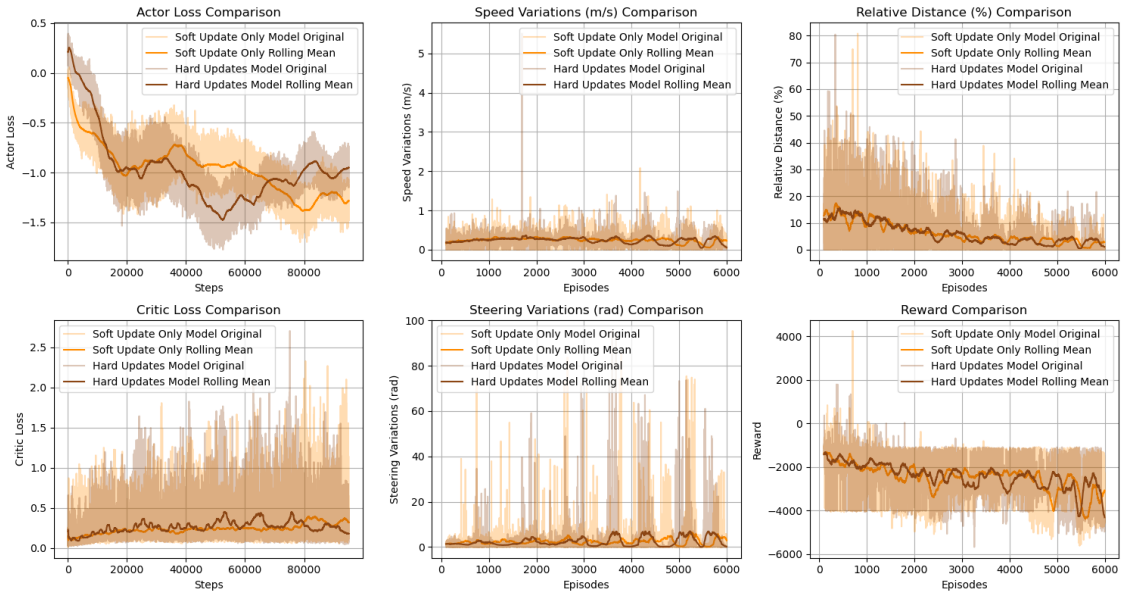


Figure 4.4: Hard updates vs only Soft updates model comparison.

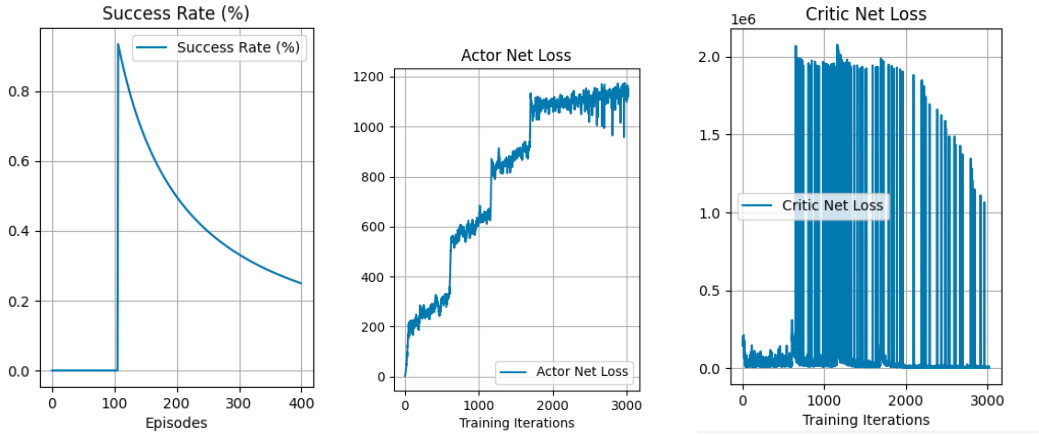
Thus, one model included hard updates every 200 steps (Model 1), while another model had no hard updates at all (Model 2). The experimental parameters used are detailed in Table 4.3.

To visually assess the performance of both models, the losses, velocity and steering variances, reward, and relative distances were plotted and are shown in Figure 4.4.

Across the different metrics, the values align with each other and do not deviate too far from the same range. This may dispel any initial expectation that one model would significantly outperform the other. One of the considerations and reasons for implementing hard updates in the networks was to expedite the convergence process. However, examining the loss plots in Figure 4.4, the trend shows no conclusive evidence to support that claim. Around the 40000 steps, the model with hard updates slightly diverges from the trajectory of the regular model, but they eventually converge to similar ranges. The most prominent pattern in the model with hard updates is its irregularity, stemming from the updates themselves. With a full update occurring every 200 steps, the vehicle's behavior undergoes drastic changes, clearly reflected in the highs and lows that differ from the stable evolution observed in the soft update-only model. This variation is evident in the reward and relative distance plots, and in the data for the later episodes of the speed variation graph. However, the most pronounced effect is seen in the steering variation plot, where it manifests as a weaving behavior.

The risk of loss explosions due to the usage of hard updates was mitigated by the reward standardization used in this experiment. A previous experiment, depicted in Figure 4.5, illustrates the complete disruption of the model caused by an early random success case, as observed in the Critic Loss plot.

This experiment did not clearly favor either approach. The potential advantages and disadvantages of including hard updates did not emerge from the experimental results. Given the similar



(a) Success Rate (b) Actor Loss (c) Critic Loss

Figure 4.5: Loss explosion in a non normalized reward model with hard updates.

performance but greater stability demonstrated by the soft update-only model, this approach ultimately became the preferred in comparison to other training experiments.

4.3.3 Convergence & Vehicle Exploiting Extreme Controls

This experiment can be considered more of a series of trials and will be the most extensive of all. In this subsection, the central theme explored is the primary challenge in training the agent: convergence. Across various variations in model parameters and reward functions, there was a consistent issue with the convergence of the loss. This persisted due to several factors and remained an active concern affecting the overall results. To address the multifaceted nature of this issue, different approaches were employed.

Many of the iterations resulted in one of two recurring behaviors. Either the vehicle would accelerate and steer directly into a wall, mistakenly believing it to be beneficial, or it would remain stationary to avoid actions that incurred penalties. It is evident that these behaviors stemmed from the agent getting trapped in undesired local minima. However, solutions to mitigate these behaviors could involve various adjustments:

- Updating rewards to alter how bonuses or penalties influence vehicle behavior.
- Modifying the model to encourage convergence away from these specific minima.

These two categories of solutions, along with the rationale behind each decision, will be further explained in the following sections.

4.3.3.1 Control Variables & Stability (Reward)

In order to deal with the previously mentioned issues, to define a balanced reward function, and to increase the chances of success, a study was made on the dominance of the various reward components, using distinct functions. There were different objectives regarding the stability of

Table 4.4: Equations for the different proposals of the reward components, along with an identification tag

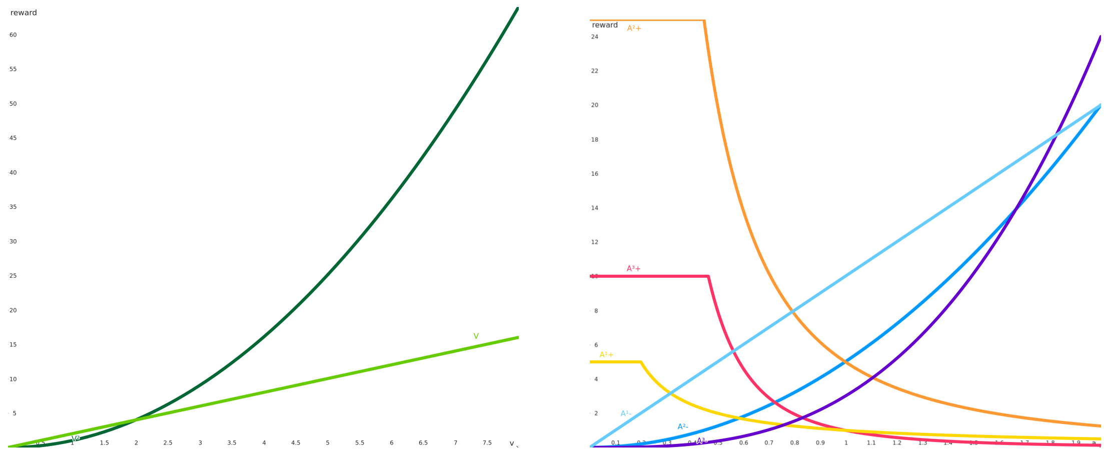
Tag	**Equation**
V	$2 \cdot \text{ego\_v}$
V^2	ego_v^2
$A+$	$\frac{1}{\max(\lvert a \rvert,0.1)+\max(\lvert \theta \rvert,0.1)}$
A^2+	$\frac{5}{\max(\lvert a^2 \rvert,0.1)+\max(\lvert \theta^2 \rvert,0.1)}$
A^3+	$\frac{1}{\max(\lvert a^3 \rvert,0.05)+\max(\lvert \theta^3 \rvert,0.05)}$
$A-$	$-(10 \cdot \lvert a \rvert + 10 \cdot \lvert \theta \rvert)$
A^2-	$-((5 \cdot a)^2 + (5 \cdot \theta)^2)$
A^3-	$-((3 \cdot \lvert a \rvert)^3 + (3 \cdot \lvert \theta \rvert)^3)$

the vehicle and the management of its controls. An example could be the desire of avoiding large acceleration requests, while still prioritizing a reasonable speed value. The end goal was finding a middle ground where the vehicle would not prefer either maximizing the commands for a speed reward or minimizing them for stability reward. To achieve this, the stability component needs to be balanced out so that no specific component overpowers the final value, and the controls in the middle end up being more positive overall than the extreme cases.

The experiment carried out to find the ideal balance between the velocity and control components involved iterating over different reward proposals and varying value intervals to determine how much one value outweighed the other.

The velocity reward proposals were only considered as positive values, despite the fact that an approach where lower velocity could incur greater penalization could also be explored. The two functions related to this variable were linear and squared, as shown in Table 4.4. The first would adjust the coefficient to yield higher returns for lower values but less than the squared function for the highest values. Regarding the control component, a similar methodology was used, with the introduction of a cubic function. Additionally, a negative approach was also considered, penalizing higher absolute commands instead of rewarding lower ones. These are also represented in Table 4.4. Altogether, there were 2 velocity reward proposals and 6 for commands, totaling 12 combinations to choose from. Each function was assigned a tag indicating either action, a, or velocity, v, along with the sign and exponent, all for simplicity.

The mapping of each equation in function of the variable is visually represented by their tags in Figure 4.6.



(a) Velocity Reward Functions (b) Action Reward Functions

Figure 4.6: Graphical representation of the proposed reward components, identified by the respective tags

To evaluate how each pair behaved, they were put to test using data from the training. The combinations of the acceleration and steering sum with the velocity were split into a few intervals that could translate varying vehicle patterns without increasing the number of permutations too much. The pairs of values taken from the agent history were added into each combination of intervals or cells. For each, the dominance index, d_i, was calculated using the action, a, and velocity, v, values, as defined in Equation 4.1. These values come in percentage and represent how large the control reward component is when compared to the velocity one. A 50% would mean both are equally representative.

$$d_i = \frac{a}{a+v} \times 100 \tag{4.1}$$

The average of all values within an interval was calculated and filled in a table represented in Figure 4.7. For visualization purposes, the percentages are colored using an Heatmap, where the greener it gets, the more dominant the controls are for a combination of values.

As previously mentioned, the main goal is to find a balance that weights the interval of each

	Action Reward	A+					A^2+					A^3+				
Velocity Reward	Intervals	0,0 - 0,4	0,4 - 0,8	0,8 - 1,2	1,2 - 1,6	1,6 - 2,0	0,0 - 0,4	0,4 - 0,8	0,8 - 1,2	1,2 - 1,6	1,6 - 2,0	0,0 - 0,4	0,4 - 0,8	0,8 - 1,2	1,2 - 1,6	1,6 - 2,0
V	0,0 - 2,0	73%	59%	54%	38%	40%	94%	91%	87%	73%	67%	87%	82%	73%	46%	42%
	2,0 - 4,0	38%	24%	16%	8%	9%	81%	75%	63%	35%	34%	64%	57%	39%	10%	10%
	4,0 - 6,0	27%	16%	11%	8%	5%	73%	66%	52%	36%	24%	52%	45%	28%	12%	6%
	6,0 - 8,0	23%	12%	8%	6%		66%	58%	47%	33%		44%	37%	25%	12%	
V^2	0,0 - 2,0	80%	68%	64%	52%	51%	96%	94%	91%	81%	75%	91%	87%	80%	59%	52%
	2,0 - 4,0	30%	18%	12%	4%	7%	75%	68%	54%	21%	28%	55%	48%	31%	6%	8%
	4,0 - 6,0	14%	8%	5%	4%	2%	53%	46%	32%	21%	12%	32%	26%	15%	6%	3%
	6,0 - 8,0	9%	4%	3%	2%		37%	31%	21%	13%		19%	16%	9%	4%	

	Action Reward	A-					A^2-					A^3-				
Velocity Reward	Intervals	0,0 - 0,4	0,4 - 0,8	0,8 - 1,2	1,2 - 1,6	1,6 - 2,0	0,0 - 0,4	0,4 - 0,8	0,8 - 1,2	1,2 - 1,6	1,6 - 2,0	0,0 - 0,4	0,4 - 0,8	0,8 - 1,2	1,2 - 1,6	1,6 - 2,0
V	0,0 - 2,0	68%	79%	86%	88%	92%	55%	77%	88%	93%	96%	40%	62%	82%	92%	96%
	2,0 - 4,0	30%	49%	60%	64%	77%	17%	46%	66%	79%	89%	5%	28%	54%	78%	89%
	4,0 - 6,0	22%	38%	49%	60%	66%	12%	35%	56%	75%	82%	4%	20%	44%	73%	82%
	6,0 - 8,0	14%	31%	38%	49%		6%	29%	42%	61%		1%	16%	29%	53%	
V^2	0,0 - 2,0	75%	84%	90%	91%	94%	63%	83%	92%	95%	97%	46%	70%	87%	94%	97%
	2,0 - 4,0	24%	41%	51%	47%	70%	13%	38%	57%	65%	84%	4%	22%	45%	65%	85%
	4,0 - 6,0	11%	21%	30%	41%	46%	6%	19%	36%	57%	66%	2%	10%	26%	56%	66%
	6,0 - 8,0	5%	12%	15%	23%		2%	12%	18%	32%		0%	6%	11%	26%	

Figure 4.7: Heatmap conatining the weight the command related reward has over the velocity-based one, depending on the state and control intervals and reward functions.

value and the general return. As such, the preferable match would be two functions where the regular actions within the middle of the spectrum are fairly balanced in terms of absolute values, and where the edges and extreme values have a slight advantage over the other, both for positive and negative outcomes.

In the positive action components, the action edge is bigger throughout the lower/left side, while on the negative action reward, it's bigger on the higher/right side. Looking at the proposed solutions, it is clear the A+ is not adequate for the velocity reward. The opposite happens for the A^2+. The A^3 also shows some concerns when the action values are in the lower intervals. The missing value for the highest velocity and commands pair is not there because no scenarios occurred where the values were in both intervals at the same time. This happens because when this type of commands occurr, the vehicle is already near collision and not driving at higher speeds.

In the decision process, the positive action rewards were the first to be considered inadequate. Not only would balancing their relative influence be difficult, but deducting rewards for the erratic decisions the agent has demonstrated so far is the most sensible option. Among the options that include action penalties, the A^3- seems to vary too intensively, suggesting it might be too volatile. On the contrary, since the A- and V pair do not have any exponential component, the values do not fluctuate sufficiently. The three remaining pairs are: V with A^2-, V^2 with A^2-, and V^2 with $A-$. These combinations differ slightly in behavior but all meet the requirements of balancing, as observed in the value dispersion.

To make a decision regarding the most promising options, a test was conducted with 150 random episodes for all options. The episodes could use the same generated data for comparison reasons, but previous experiments already covered that type of assertion. Therefore, this test will use random data generated by the simulator and libraries. The different reward evolutions are shown in Figure 4.8.

The best-performing function over time varies considerably. This is due to the randomness of the generated episodes. The conclusion drawn is that none of the three experimented functions can be considered outliers, since the general range of values is maintained. Upon closer examination of the data from the Heat Map, the V with A^2- pair was preferred, although there were no particularly significant differences.

4.3.3.2 Masking, Capping & Learning Rate (Model)

The reasons for convergence issues may also lie not only in the reward function but can also be driven by various components of the model, ranging from the learning rate to the actual state and action spaces. In an effort to mitigate the problem related to the agent becoming stuck on repeatedly performing negative actions, several options were considered. Among these were the following:

- **Priority Replay Buffer**: Using this approach would mean increasing the probability of "important" actions being included in the batches sent for training. From the large buffer size, a certain number of steps are generated and delivered together for training. However, given

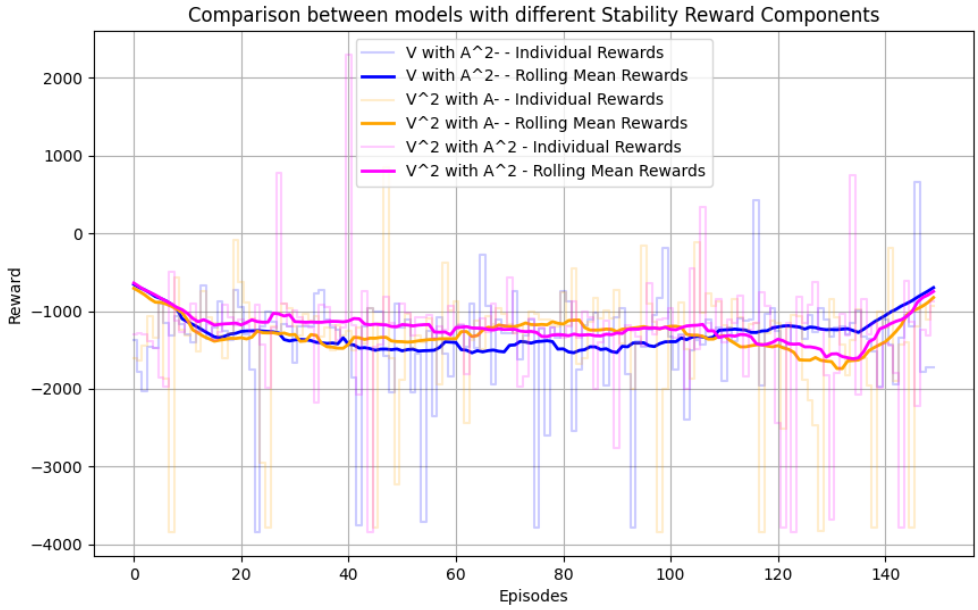


Figure 4.8: Reward comparison between the 3 most promising stability components for a random set of 150 episodes.

the large number of steps within an episode, the most relevant actions (e.g., penalties for hitting a wall) can become underrepresented in the large pool of returned values. This could be an option to improve how the agent learns the outcomes of these actions. A drawback of this approach would be prioritizing the referred actions too much and losing some of the information that would normally be retrieved from other actions.

- **Clipping or Capping the Actions**: Although not ideal, restricting the set of available actions for the agent could help encourage it to follow preferred ones, since generally less positive actions would be prohibited. The main issue with this approach lies in the principles and goals of training. One constraint related to the specific research problem was allowing the agent to freely choose its path and actions. Any intervention similar to this could be seen as deviating from the initial principles and changing the paradigm.

- **Masking the Actions**: A masking mechanism does not directly intervene in the agent's set of actions but rather restricts certain behaviors within the available range of possibilities. Given the current issue, it would help prevent the function from getting stuck in a local minimum by preventing frequent repetition of certain actions. If the agent is stuck and keeps repeating the same choices, the masking mechanism registers this repetition within a threshold and generates another action. However, this can pose issues regarding the agent's learning, as the training process may begin to lack iterations related to specific actions, resulting in an incomplete learning process that limits the possibilities for learning from those actions.

Table 4.5: Main relevant hyperparameter values used in the Convergence Experiments.

Parameter	**Model 1**	**Model 2**	**Model 3**
Mask Similarity Threshold	0.2	0.1	0.1
Learning Rate Actor	1e-3		
Learning Rate Critic	1e-2		
Decaying Learning Rate	No	No	Yes
Number of Episodes	10000		7000

- **Increasing the Learning Rate**: Boosting the learning rate might be an option to address convergence issues, as it can help overcome undesirable local minima by taking larger steps in the parameter space, thus exploring more widely. It also enables the network to explore different regions of the loss landscape more quickly. However, the risk here is mainly related to the potential for instability and divergence if set too high, potentially overshooting the optimal region.

All of these can be valid countermeasures to resolve convergence issues. However, implementing them all together may overly specialize the learning process to a specific set of conditions that may not be generalizable enough.

Among the proposed solutions, the selected approach was to use a mask. The degree of application of the mask remains a topic for discussion, as it can offer various levels of control over actions. Different runs were conducted with similarity thresholds. The first (Model 1) applied to all actions within a 0.2 similarity threshold, and the second (Model 2) used 0.1. The mask utilized an action buffer of size 10, with a repetition limit of 3 actions. These values need to balance between a short buffer that does not count actions executed too long ago or casual repetitions, while still preventing repetitive behavior. The learning rate was also increased. The parameters that were used in the experiment are listed in Table 4.5.

In order to compare both variations, the two models were put side to side across different evaluation metrics. This results for this test are represented in Figure 4.9

The primary difference observed when analyzing the graphs is the evolution of the loss. While relatively stable in the initial iterations, the 0.2 mask version quickly begins to exponentially increase. At 20,000 steps, the actor network loss is already 10 times larger than the 0.1 mask model, and by 100,000 steps, the difference is approaching five orders of magnitude, as shown in Figure 4.9. A similar trend is also evident in the critic loss. This can be attributed to the size of the mask. Due to its largeness, many values are constrained, leading to the discarding and random generation of numerous actions. This results in the model failing to learn certain patterns and the loss escalating exponentially. This frequent random generation is also reflected in the graphics of relative distance and reward. The initial average achieved by exploring randomly generated actions is sustained throughout the entire period. Additionally, a change in the variance of speed is observed for both models. The average decrease observed corresponds to the same episode range at which the loss begins to increase in both models. In the 0.1 mask loss plots, despite lower values, the absolute increase in loss is also present, for similar reasons, albeit with less intensity.

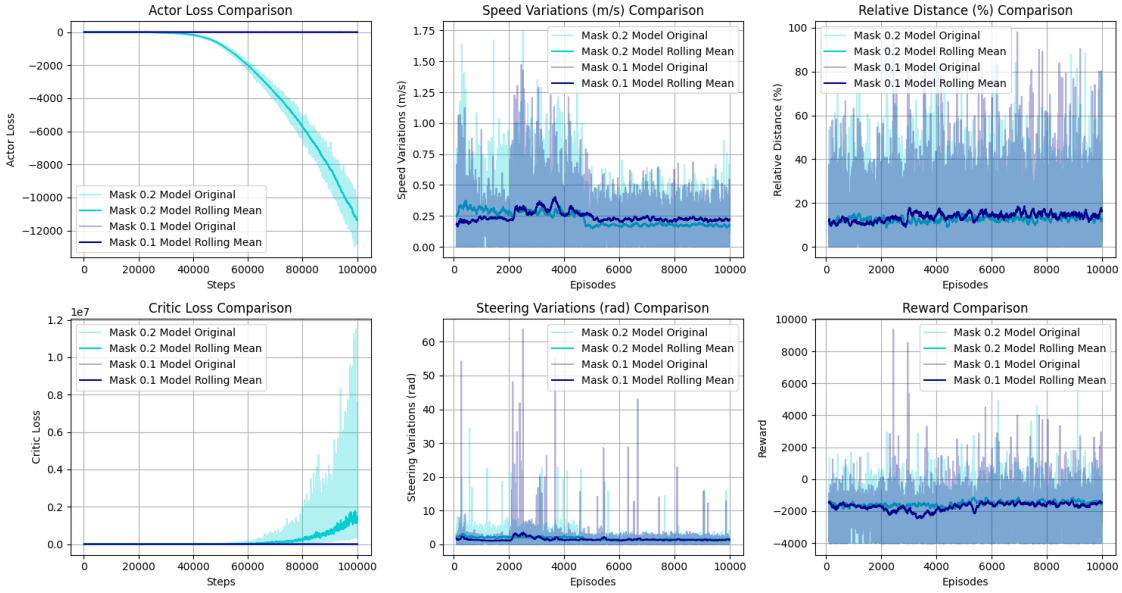


Figure 4.9: Comparison between Models using Masking Mechanism for similarity thresolds of 0.1 and 0.2

This is further exacerbated by the increase in learning rate, which may have introduced instability. The introduction of the mask generally assisted the model in maintaining higher absolute values of reward, enhancing exploration when stuck. With careful monitoring of the loss evolution, it can serve as a beneficial addition to the learning process.

After reaching an optimal solution regarding masking, another potential intervention that could assist the model into convergence to other local minima is initially increasing the learning rate and adjusting it for later iterations. This approach would help the model float more initially, thus avoiding convergence to undesirable repeated actions and achieving faster convergence to a more diverse range of runs. The learning rate would be decayed over time to stabilize training more quickly.

To compare the effectiveness of a decaying learning rate with a fixed one, the best model from the previous test (0.1 mask threshold) was selected as the baseline for this comparison. The decaying learning rate was implemented, and the model was trained with the same parameters as before (Model 3, as detailed in Table 4.5). The same metrics were used to compare both iterations. The behavior of both is visually represented in Figure 4.10.

In contrast to the previous test, the loss of these two models remains within the same range. After initial fluctuations, the values for both actor networks deviate in a similar manner. Around 2,000 episodes, the stable learning rate starts to show signs of change, moving towards the variable one, while the latter remains consistent. By approximately 4,000 episodes, the variable learning rate, decreasing over time, begins to converge towards lower loss values, ultimately stabilizing around 2. Conversely, the fixed learning rate continues to diverge further from a neutral loss. This can be attributed to the larger learning rate of the non-decaying version in later stages, which increases instability and makes it more prone to divergence. The decaying learning rate emerges

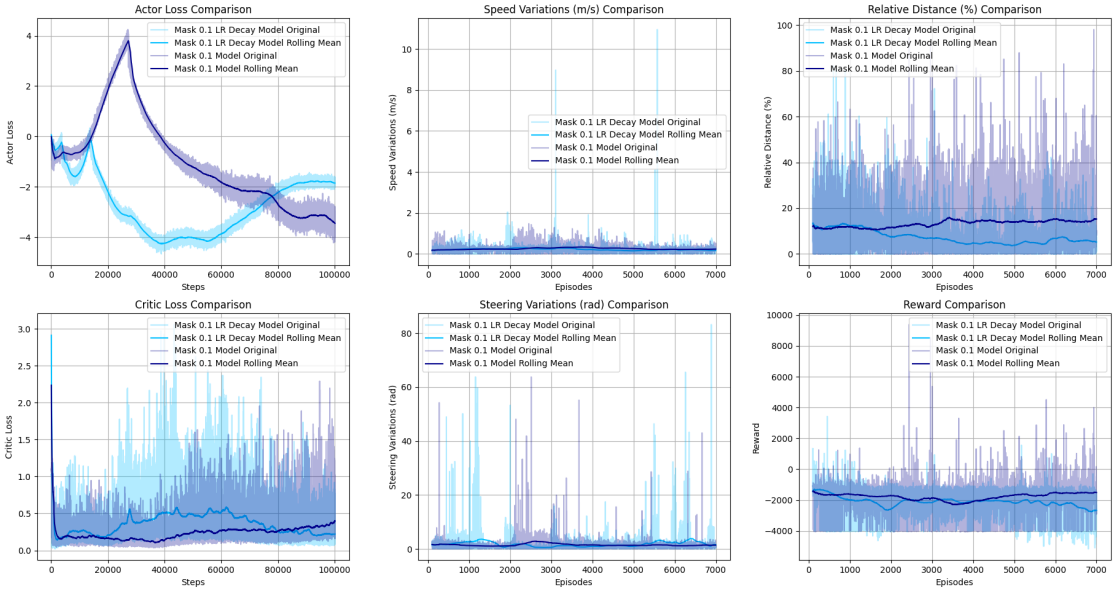


Figure 4.10: Comparison between Models using Masking Mechanism for similarity thresholds of 0.1, with or without Decaying Learning Rate

as the preferred choice due to its stability. A similar trend is observed in the critic network, where the decaying LR initially causes higher loss, but reverses this trend a few episodes later. The higher learning rate appears to introduce more randomness, as evidenced by the reward and relative distance remaining consistent throughout the episodes. The decaying learning rate shows an increasingly selective trend, though it has not yet managed to improve the reward to higher values. There are no significant differences in speed and steering variations between the two approaches, besides random highs and lows.

In all tests, the introduction of a light masking mechanism extends exploration and, consequently, the possibility of learning new features, while helping to steer the gradient away from undesired local minima. However, this can create impediments to loss convergence. Adding a decaying learning rate alongside the mask can mitigate this issue and potentially improve the overall training progress.

4.3.4 Distance (Reward)

Regarding the reward function that drives the vehicle forward, a similar approach was taken in assessing its role.

The primary challenge with roundabouts is the variation in distances between different pairs of entrances and exits. The distance required for a vehicle to exit at the first opportunity versus completing a full circuit around the roundabout is considerably different. In this scenario, there would be a significant reduction in the distance bonus component when dealing with longer distances. While this might seem like a straightforward condition for the agent, it was deemed important that

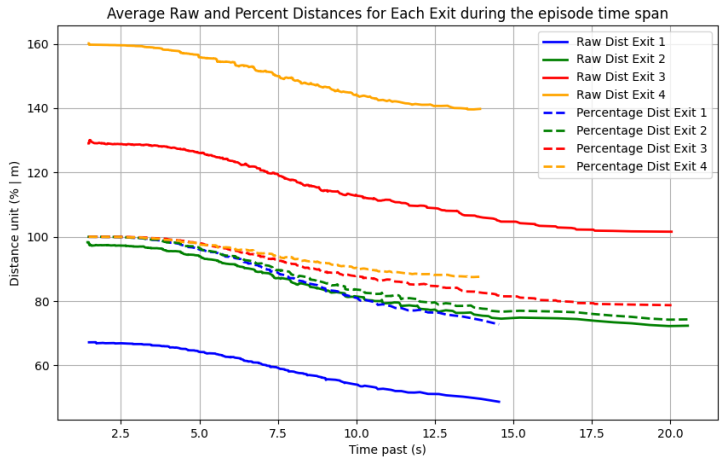


Figure 4.11: Average Raw and Percent Distances for each Exit during the episode time span

rewards across all possible routes follow a consistent pattern. To address this, a method was developed to normalize the distance progress along the entire path. This involved mapping the total path distance against the current distance traveled, yielding the percentage of the path completed. Thus, the remaining distance to the goal can be assessed in a normalized manner across paths of various lengths. The path distance comprised the sum of distances from the entrance to the central circle, the distance from the central circle to the exit, and the counter-clockwise distance around these points (following the circular shape of the road).

To assess the effectiveness of this approach, data was gathered from several simulation runs, calculating the average distance variations throughout an episode for all types of exits, with and without percentage normalization. A graphical comparison of these data can be seen in Figure 4.11. The expected outcome was that percentage distances would show less variation compared to raw distances.

As expected, the distance normalized by percentage ends up falling within the same range of values. While the raw values range from 60m for the first exit to 160m for a full turn-around, the percentage values maintain similar patterns, allowing for a more balanced calculation. For example, the Exit 1 data, shown in green, ends up being very similar in both versions, despite the values being unrelated, with an average starting distance around 100m, creating a false sense of correlation with the percentage value. Despite the usage of this method, the distance travelled by a vehicle in one step will always be less for longer exit paths, since the percentage distance decreases less.

Regarding the calculation of the distance reward component, a squared function was adopted as the paradigm. This choice was motivated by the desire to achieve exponential growth as the vehicle progresses towards the exit, thereby increasing the reward for longer distances. To address the initial meters of the run, where the percentage of the path covered was still low, a bonus was introduced up to 20% to mitigate this issue. The function is defined in Equation 4.2 and can be visually observed in Figure 4.12, where both the standard and the bonus iterations are depicted as

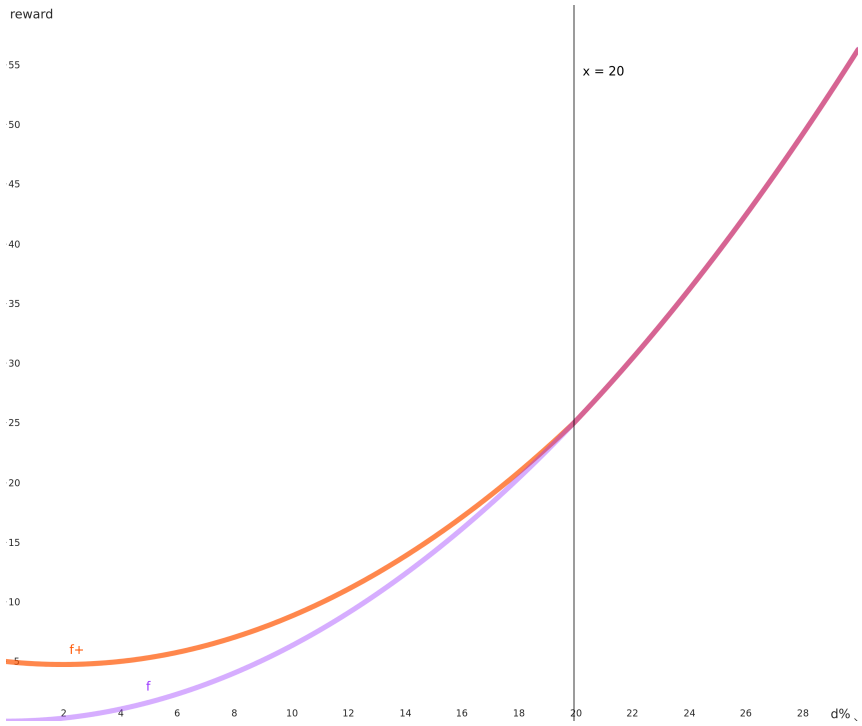


Figure 4.12: Distance reward function graphical representation

f and $f+$ respectively.

$$\text{reward}+= \left(\frac{\text{d}\%}{4}\right)^2 + \begin{cases} 5\left(1-\frac{\text{d}\%}{20}\right) & \text{if d}\% < 20 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

In an attempt to compare the reward functions in training, two models were evaluated: one without a bonus in the first meters and the other with a bonus. Both were tested over 150 episodes, as shown in Figure 4.13. The average values remain within the same range but vary considerably depending on the generated random data. The model without the bonus performs better in the initial episodes, a trend that later reverses, depending on the number of high and low sequences. The examination under these conditions does not allow for definitive conclusions, but the already demonstrated function definition justifies the increase of the general reward for distance in the first meters of a run.

4.3.5 Time to Collision (Reward)

In Li et al. (2023a), a Time to Collision metric is used as part of a reward function to pass information to the vehicle on imminent collisions. This is used in a Ramp Merge task in which the ego vehicle needs to manage its entry timing to avoid collision with other vehicles.

This was considered a relevant feature since the entrance of a vehicle in a roundabout is a similar problem, with some differences in the curve trajectories and more freedom of movement.

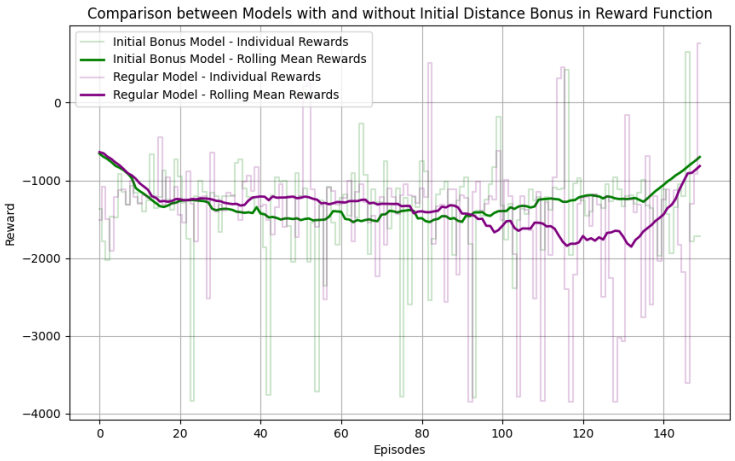


Figure 4.13: Comparison between Models with and without the reward distance component bonus for the initiall meters of the path.

To calculate this metric, the ego vehicle is compared with all other cars, by estimating the future position of both, based on its current position, speed and orientation. This process is iterated for all intervals of time, using various degrees of antecipation. If their predicted paths are expected to intersect in the selected time frame, a penalization is attributed. This allows for the existence of an incremental and preventive warning to the vehicle, aiding in the reduction of collision cases.

Introducing this metric for too long intervals of time could bring issues, since the agent gets penalized for a possible collision that is still unlikely to happen. To evaluate how frequent this scenarios would happen for different intervals, 200 episodes were run and a count was kept to track the occurrencies. This information is shown in Figure 4.14.

After tracking all cases and assessing the correlation between them, a certain connection between the counts was found. Some of the counts were added in the same episodes, since the vehicles trajectory did not change and the imminent collision was counted for more than one interval as their positions got closer.

The larger intervals were slightly more frequent, but it was decided that they should not be added into the final model, because with such large time gap, it is possible that the agent is getting penalized unfairly, since it still had time to correct its trajectory and manage the crossing.

When driving at lower speeds, the vehicle has more time to stop and avoid a collision, so the threshold selected was 1 second. This is a balance between conservative and naive approaches. The reward will penalize imminent collision when the estimated time is 1 second, and double the value when in half a second.

4.4 Testing & Evaluation

:

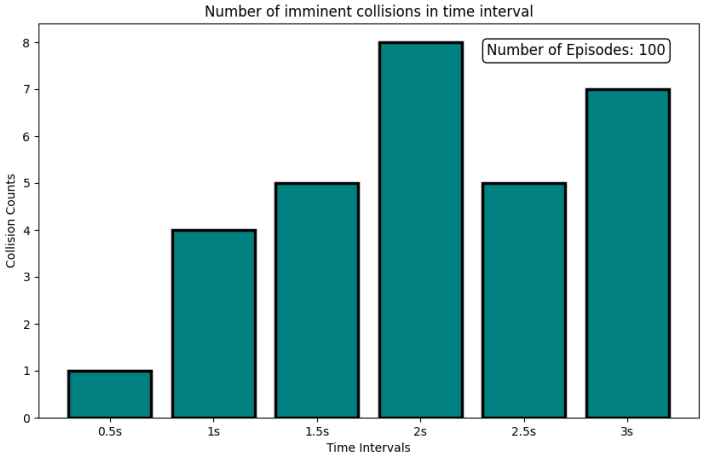


Figure 4.14: Number of imminent collisions in different Time to Collision (TTC) intervals.

The different experiments mentioned in the previous section led to the optimization of the model and reward function. All positive introductions were bundled together into a final iteration, which was the best-performing setup. The used parameters are stated in Table 4.6.

The evaluation of the agent is enabled by the predefined metrics, whose evolution throughout the training is shown in Figure 4.15.

The different metrics were selected with the intent of measuring and assessing distinct components of the task. However, given the existing difficulties for the agent to learn sufficient information that enables it to conclude the task, a few metrics were left as less relevant. This is the case for the Collision and Success Rates. Considering the lack of successful episodes, neither of the ratios can transmit an evaluation of the vehicles performance. A similar thing happens with the Time to Finish, which is only considered when the goal is achieved, and even the Average Speed may be disregarded, considering there is no point for it to be high if it is directly causing collisions.

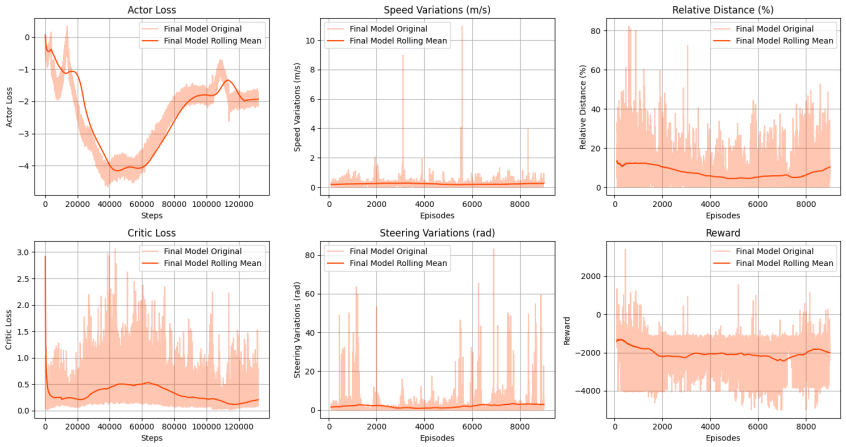


Figure 4.15: Metrics throught training for Final Model.

Table 4.6: Hyperparameter value ranges used in the Final Model.

Parameter	**Value**
Learning Rate Actor	1e-3
Learning Rate Critic	1e-2
Learning Rate Decay	0.99995
Learning Rate Minimum	1e-4
γ	0.999
τ	1e-3
ε-decay	0.999975
ε-minimum	0.05
Replay Buffer Size	10000
Batch Size	128
Number of Vehicles	[6, 10]
Episode Timeout	20s
Number of Episodes	9000

The speed and steering variations are not regular throughout the training process. The behavior the ego vehicle adopts in a certain local minimum influences this general trend, as can be clearly seen in the Steering Variations plot. In the first episodes, the agent is sending high steering values, until around episode 2000, where this behavior ceases, restarting at the 5000 episode mark. The metrics that reveal the most about the model are the losses and the reward and relative distance.

Looking at the Actor Loss, there is a clear irregular pattern as well. The masking mechanism contributed to the unblocking of a stale agent, but it also caused some floating in the loss function, which would converge and diverge in an unpredictable pattern. The variations in the Learning Rate were also responsible for this. Throughout the training, there were three major shifts in the loss function. The first has the greatest decline, which is expected since the training is still in its early stages. In the middle, the model converges around the 50,000 steps, but ends up lowering its absolute value later. In the latter stages, the learning rate changes possibly induced the model to a late spike.

Reward and Distance end up being the variables that carry the most information about the overall performance. In the first stages, the values are relatively high and random due to the ε-greedy policy motivating exploration. After some time, the values go lower, accompanied by the second loss deviation. In the latter stages, these metrics rise again and an improvement in the performance can be observed.

To assess the final results of the model, a battery of simulations was run. The relative distance, which is the metric that enables the evaluation of the agents performance in unsuccessful runs, was measured and split into the average for the different number of exits in the simulation. The rewards were also noted and the results are shown in Figure 4.16.

The results were quite revealing and aligned with expectations. Exit 1 consistently topped the graph, achieving the highest distance values. On the other hand, the full turn-around maneuver in exit 4 scored the lowest for most episodes. Interestingly, when comparing the rewards, exit 2 paths

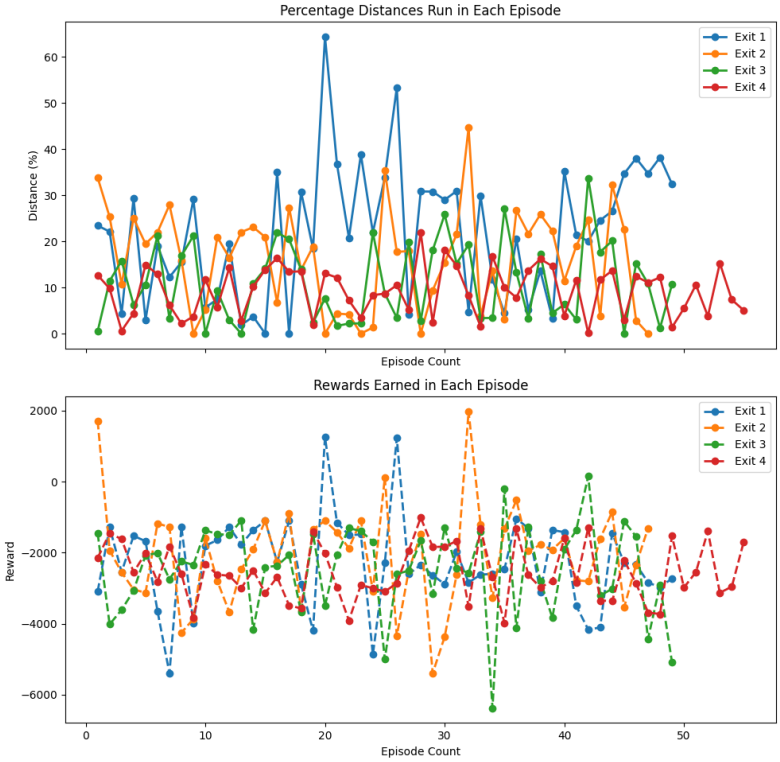


Figure 4.16: Graphical view of the Reward and Distance achieved for a set of 200 runs, depending on the required exit, with the final model.

surprisingly outperformed exit 1. This could be because the trajectory for exit 2 requires fewer steering commands, having a more straightforward path.

Overall, the results obtained from the tests are summarized in Table 4.7.

As predicted at the outset, the chosen paradigm required strict constraints, making the agent's training challenging. Despite a carefully defined reward function, the problem's nature necessitated the introduction of enhancements and brute-force mechanisms into the network. Although there was initial skepticism, the combination of all features successfully taught the agent some key aspects, boosting its performance in the later stages of training. While successful episodes were rare, the increase in relative distance values with the elimination of random actions demonstrated that the reward function effectively guided learning. Future research is expected to offer further improvements, and these findings provide a helpful foundation.

Table 4.7: Reward and Distance Results for the different Exits

Metric	Exit 1	Exit 2	Exit 3	Exit 4
Max Distance	64.4%	44.7%	33.7%	21.9%
Avg Distance	22.0%	16.2%	11.1%	9.4%
Avg Reward	-2250	-2068	-2466	-2540

4.5 Summary

In this chapter, various experiments were conducted to address multiple tasks and issues related to the algorithm. First, there is a review of the tools used and simulation issues encountered. Model-related comparisons were made regarding reward normalization, the use of hard updates, and the application of masking mechanisms. On the reward function side, tests assessed the efficiency and relevance of the proposed components, focusing on stability, distance, and the Time to Collision metric. Finally, the different enhancements were consolidated, and a discussion on the final results was presented.

Chapter 5

Conclusions

The roundabout intersection was established from the beginning as an extraordinarily complex scenario within Autonomous Driving. The number of constraints and variables related to the task is larger than usual, creating a significant challenge for an autonomous vehicle to learn.

The expectations from the outset were realistic, acknowledging the difficulty of achieving high success rates. The nature of the Motion Planning task makes it challenging for the vehicle to navigate the roundabout even on an empty road, let alone with the inclusion of other participants. The execution of full turn-around maneuvers was deemed improbable, but it was anticipated that some paths to the first exit could be successful in the absence of obstacles. The choice for Deep Deterministic Policy Gradient (DDPG) was driven by the desire for a continuous action space and the need to experiment with a Deep Reinforcement Learning algorithm. Considering the set of controls, using a discrete action space would have simplified the problem immensely, as the precision required for the commands was not overly restrictive. Despite having an expected lower performance, DDPG remained the preferred approach due to the nature of the task.

The observed behavior closely matched the expectations. The various solutions experimented with, as mentioned in Chapter 4, addressed many issues in the training process, enhancing the conditions for successful runs. This ultimately resulted in the agent learning some patterns to increase the relative distance traveled, despite being insufficient for consistently successful runs.

The reward function components were thoroughly studied and proven to be well-balanced in the tests. The main obstacles impeding the learning of the tasks appeared to be model-related, and most of the potential performance enhancements would relate to that. One of the main issues regarding the learning difficulty was the paradigm itself. The design of this work proved to be a significant challenge for the current state of the art. This conclusion emerged from the various experiments related to the convergence struggles mentioned earlier. Another setback was the simulation tool itself. CARLA is a very comprehensive simulator, encapsulating many urban scenarios for autonomous driving along with an excellent set of tools. However, this simulator still has ongoing problems related to entity management and parallel processing. While setting up the environment, the spawning and destruction of entities created issues related to time management (delayed and asynchronous spawning) and memory. These problems occasionally caused

the system to generate Segmentation Fault errors or simply crash. The high-performance hardware requirements were also an issue.

Despite these obstacles, the main goals of the study were completed successfully. At the beginning of this work, the defined requirements aimed at creating and analyzing a novel reward function that would meet the current needs for research in similar scenarios. In this regard, the objectives were achieved, and the proposed solution offers new ideas that can be relevant and applied in future works.

Overall, the main outcomes extracted from the work are:

- Reward Function Design: The reward function was carefully designed and tested for different scenarios. It is an innovative contribution given the inclusion of all types of components and its integration with an also complex task of crossing a roundabout. The results perceived from the study showed promising results in guiding the agent's learning process, despite the overall challenge of the task.
- Continuous DDPG Implementation: The use of Deep Deterministic Policy Gradient (DDPG) allowed for a continuous action space, allowing total freedom in the action selection and providing valuable insights into the applicability of this type of deep reinforcement learning algorithms for these driving scenarios.
- Enhancements and Mechanisms: The introduction of enhancements such as masking mechanisms and varied learning rates demonstrated potential improvements in training performance and a distinct approach on Reinforcement Learning as other works.
- Simulation Challenges: The usage of CARLA simulator further explored an expanding tool, and contributed by delivering solutions to some issues the simulator still has up to this date.

While the incorporation of these findings in real-life scenarios is still not feasible due to the still insufficient success rate, extensive validation and safety requirements in the industry, the experiments developed in this work provide a solid foundation for further investigation. This is particularly significant for roundabout scenarios, where research is still relatively sparse, and it represents a relevant contribution by introducing a new set of perspectives on a Reinforcement Learning paradigm for Autonomous Driving.

5.1 Future Work

This work opened opportunities for further research on multiple fronts due to the enormous number of variables and options that are still yet to be explored.

Despite the constraints defined for this work leaning the investigation towards Deep Reinforcement Learning, namely the DDPG algorithm, there is still a wide range of options to experiment with, both for discrete and continuous action spaces. The usage of other algorithms would benefit the validation of results as well, as it can clarify some discussion topics and may achieve state-of-the-art results. Additionally, in this study, the sensing and state estimation data was assumed as

ground truth and retrieved directly through the simulation tool. Nevertheless, there are still solid options for this, such as the common DRL option of using input from a camera sensor as the state fed to the network or the processing of LIDAR and odometry data through SLAM algorithms to retrieve positional and movement-related data.

To simplify the problem, an option could be removing the Path Planning component and focusing solely on the Decision Making part. This way, the vehicle would only need to follow a reference path, aiding in the optimization of its movement. Conversely, if the goal is to approximate the problem to reality, the introduction of the concept of lanes and the inclusion of pedestrians could be relevant features. However, they would also elevate the complexity levels of the problem, especially the lane change parameterization. Finally, experimenting with custom roundabout shapes, by increasing the number of legs or lanes, for example, could help the training to be generalizable and adaptable to more scenarios.

All in all, there are many branches requiring further investigation, and the exploration of different aspects related to the Autonomous Driving Roundabout task would be very relevant and enhance the overall conditions known to date.

References

AGRD04B-23-ed3.2 laustroads. URL https://austroads.com.au/publications/road-design/agrd04b.

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning, 2016. URL https://www.tensorflow.org/tensorboard.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. jul 2017. URL https://proceedings.mlr.press/v70/anschel17a.html.

S. H. Ashwin and Rashmi Naveen Raj. Deep reinforcement learning for autonomous vehicles: lane keep and overtaking scenarios with collision avoidance. *International Journal of Information Technology*, 15(7):3541–3553, oct 2023. ISSN 2511-2104. doi: 10.1007/s41870-023-01412-6. URL https://link.springer.com/10.1007/s41870-023-01412-6.

Teham Bhuiyan, Linh Kästner, Yifan Hu, Benno Kutschank, and Jens Lambrecht. Deep-reinforcement-learning-based path planning for industrial robots using distance sensors as observation. *arXiv*, 2023. doi: 10.48550/arxiv.2301.05980. URL https://arxiv.org/abs/2301.05980.

CARLA Team. CARLA: An open urban driving simulator. https://carla.org/.

German Aerospace Center. Sumo - simulation of urban mobility. URL https://eclipse.dev/sumo/.

Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2765–2771. IEEE, oct 2019a. ISBN 978-1-5386-7024-8. doi: 10.1109/{ITSC}.2019.8917306. URL https://ieeexplore.ieee.org/document/8917306/.

Lienhung Chen, Zhongliang Jiang, Long Cheng, Alois C Knoll, and Mingchuan Zhou. Deep reinforcement learning based trajectory planning under uncertain constraints. *Frontiers in neurorobotics*, 16:883562, may 2022. doi: 10.3389/fnbot.2022.883562. URL http://dx.doi.org/10.3389/fnbot.2022.883562.

Yilun Chen, Chiyu Dong, Praveen Palanisamy, Priyantha Mudalige, Katharina Muelling, and John M. Dolan. Attention-based hierarchical deep reinforcement learning for lane change behaviors in autonomous driving. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1326–1334. IEEE, jun 2019b. ISBN 978-1-7281-2506-0. doi: 10.1109/{CVPRW}.2019.00172. URL https://ieeexplore.ieee.org/document/9025386/.

M. A. Cornu. Sur une nouvelle espèce de courbe plane à double courbure. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, 79:4345, 1874a.

M. A. Cornu. Sur la détermination rigoureuse de la longueur des arcs de toutes courbes. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, 79:11711172, 1874b.

T. Dam, G. Chalvatzaki, J. Peters, and J. Pajarinen. Monte-carlo robot path planning. *arXiv*, aug 2022. URL https://arxiv.org/abs/2208.02673.

P. de Casteljau. Courbes et surfaces à pôles. *Bulletin de la Société Mathématique de France*, 87: 57–93, 1959.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269–271, 1959.

Tao Du, Yunfei Li, Jie Xu, Andrew Spielberg, Kui Wu, Daniela Rus, and Wojciech Matusik. D3PG: Deep differentiable deterministic policy gradients. sep 2019. URL https://openreview.net/forum?id={rkxZCJrtwS}.

Badr Ben Elallid, Nabil Benamar, Abdelhakim Senhaji Hafid, Tajjeeddine Rachidi, and Nabil Mrani. A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving. *Journal of King Saud University - Computer and Information Sciences*, 34(9):7366–7390, oct 2022. ISSN 13191578. doi: 10.1016/j.jksuci.2022.03.013. URL https://linkinghub.elsevier.com/retrieve/pii/S1319157822000970.

Erwin Coumans. PyBullet: Physics simulation for robotics, deep learning, reinforcement learning. https://pybullet.org/.

Laura García Cuenca, Enrique Puertas, Javier Fernandez Andrés, and Nourdine Aliane. Autonomous driving in roundabout maneuvers using reinforcement learning with q-learning. *Electronics*, 8(12):1536, dec 2019. ISSN 2079-9292. doi: 10.3390/electronics8121536. URL https://www.mdpi.com/2079-9292/8/12/1536.

David González, Joshué Pérez, and Vicente Milanés. Parametric-based path generation for automated vehicles at roundabouts. *Expert systems with applications*, 71:332–341, apr 2017. ISSN 09574174. doi: 10.1016/j.eswa.2016.11.023. URL http://linkinghub.elsevier.com/retrieve/pii/S0957417416306583.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Franz Gritschneder, Patrick Hatzelmann, Markus Thom, Felix Kunz, and Klaus Dietmayer. Adaptive learning based on guided exploration for decision making at roundabouts. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 433–440. IEEE, jun 2016. ISBN 978-1-5090-1821-5. doi: 10.1109/{IVS}.2016.7535422. URL http://ieeexplore.ieee.org/document/7535422/.

Bing Hao, He Du, Jianshuo Zhao, Jiamin Zhang, and Qi Wang. A path-planning approach based on potential and dynamic q-learning for mobile robots in unknown environment. *Computational intelligence and neuroscience*, 2022:2540546, jun 2022. doi: 10.1155/2022/2540546. URL http://dx.doi.org/10.1155/2022/2540546.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with numpy, 2020.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Hossein Hassani and Emmanuel Silva. A kolmogorov-smirnov based test for comparing the predictive accuracy of two sets of forecasts. *Econometrics*, 3(3):590–609, aug 2015. ISSN 2225-1146. doi: 10.3390/econometrics3030590. URL http://www.mdpi.com/2225-1146/3/3/590.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Jumman Hossain. Autonomous driving with deep reinforcement learning in CARLA simulation. *arXiv*, jun 2023. URL https://arxiv.org/abs/2306.11217.

Xuefei Huang, Seung Ho Hong, Mengmeng Yu, Yuemin Ding, and Junhui Jiang. Demand response management for industrial facilities: A deep reinforcement learning approach. *IEEE access : practical innovations, open solutions*, 7:82194–82205, 2019. ISSN 2169-3536. doi: 10.1109/{ACCE\SS}.2019.2924030. URL https://ieeexplore.ieee.org/document/8742652/.

John D. Hunter. Matplotlib: A 2d graphics environment, 2007.

Bilal Kartal, Pablo Hernandez-Leal, and Matthew E. Taylor. Terminal prediction as an auxiliary task for deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 15(1):38–44, oct 2019. ISSN 2334-0924. doi: 10.1609/aiide.v15i1.5222. URL https://ojs.aaai.org/index.php/{AIIDE}/article/view/5222.

Dhanoop Karunakaran. The actor-critic reinforcement learning algorithm, 2020. URL https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14. Medium.

Meha Kaushik, Vignesh Prasad, K Madhava Krishna, and Balaraman Ravindran. Overtaking maneuvers in simulated highway driving using deep reinforcement learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1885–1890. IEEE, jun 2018. ISBN 978-1-5386-4452-2. doi: 10.1109/{IVS}.2018.8500718. URL https://ieeexplore.ieee.org/document/8500718/.

J V Kennedy, J Peirce, and I Summersgill. International comparison of roundabout design guidelines. In *3rd International Symposium on Highway Geometric Design*, page 16, Chicago, Illinois, United States, 2005. Transportation Research Board, Transportation Research Board. Compendium of Papers CD-ROM.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *arXiv*, 2020. doi: 10.48550/arxiv.2002.00444. URL https://arxiv.org/abs/2002.00444.

Alfonso B Labao, Mygel Andrei M Martija, and Prospero C Naval. A3C-GS: Adaptive moment gradient sharing with locks for asynchronous actor-critic agents. *IEEE transactions on neural networks and learning systems*, 32(3):1162–1176, mar 2021. doi: 10.1109/{TNNLS}.2020.2980743. URL http://dx.doi.org/10.1109/{TNNLS}.2020.2980743.

Akshay Lamba. An introduction to q-learning: Reinforcement learning by akshay lamba on freecodecamp, 2018. URL https://medium.com/free-code-camp/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc. Published on Medium.

S. M. LaValle. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Natick, MA, 1999.

S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

Tai Lei and Liu Ming. A robot exploration strategy based on q-learning network. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 57–62. IEEE, jun 2016. ISBN 978-1-4673-8959-4. doi: 10.1109/{RCAR}.2016.7784001. URL http://ieeexplore.ieee.org/document/7784001/.

Guofa Li, Weiyan Zhou, Siyan Lin, Shen Li, and Xingda Qu. On-ramp merging for highway autonomous driving: An application of a new safety indicator in deep reinforcement learning. *Automotive Innovation*, 6(3):453–465, aug 2023a. ISSN 2096-4250. doi: 10.1007/s42154-023-00235-2. URL https://link.springer.com/10.1007/s42154-023-00235-2.

Jin Li, Chaowei Huang, and Minqiang Pan. Path-planning algorithms for self-driving vehicles based on improved RRT-connect. *Transportation Safety and Environment*, 5(3), jun 2023b. ISSN 2631-4428. doi: 10.1093/tse/tdac061. URL https://academic.oup.com/tse/article/doi/10.1093/tse/tdac061/6955827.

Shurong Li, Chong Wei, and Ying Wang. Combining decision making and trajectory planning for lane changing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):16110–16136, sep 2022. ISSN 1524-9050. doi: 10.1109/{TITS}.2022.3148085. URL https://ieeexplore.ieee.org/document/9726894/.

Xiaoxiang Li, Xinyou Qiu, Jian Wang, and Yuan Shen. A deep reinforcement learning based approach for autonomous overtaking. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–5. IEEE, jun 2020. ISBN 978-1-7281-7440-2. doi: 10.1109/{ICCWorkshops49005}.2020.9145279. URL https://ieeexplore.ieee.org/document/9145279/.

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. *arXiv*, 2015. doi: 10.48550/arxiv.1509.02971. URL https://arxiv.org/abs/1509.02971.

Ee Soong Low, Pauline Ong, and Cheng Yee Low. A modified q-learning path planning approach using distortion concept and optimization in dynamic environment for autonomous mobile robot. *Computers & Industrial Engineering*, 181:109338, jul 2023. ISSN 03608352. doi: 10.1016/j.cie.2023.109338. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835223003625.

Jinxiong Lu, Gokhan Alcan, and Ville Kyrki. Integrating expert guidance for efficient learning of safe overtaking in autonomous driving using deep reinforcement learning. 08 2023.

Liang Ma, Jianru Xue, Kuniaki Kawabata, Jihua Zhu, Chao Ma, and Nanning Zheng. A fast RRT algorithm for motion planning of autonomous road vehicles. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1033–1038. IEEE, oct 2014. ISBN 978-1-4799-6078-1. doi: 10.1109/{ITSC}.2014.6957824. URL http://ieeexplore.ieee.org/document/6957824/.

Chinmay Mahabal, Hua Fang, and Honggang Wang. On-ramp merging for connected autonomous vehicles using deep reinforcement learning. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 56–61. IEEE, aug 2022. ISBN 978-1-6654-5417-9. doi: 10.1109/{iThings}-{GreenCom}-{CPSCom}-{SmartData}-Cybermatics55523.2022.00048. URL https://ieeexplore.ieee.org/document/9903132/.

Abderraouf Maoudj and Abdelfetah Hentout. Optimal path planning approach based on q-learning algorithm for mobile robots. *Applied soft computing*, 97:106796, dec 2020. ISSN 15684946. doi: 10.1016/j.asoc.2020.106796. URL https://linkinghub.elsevier.com/retrieve/pii/S1568494620307341.

Tianchuang Meng, Tianhong Yang, Jin Huang, Wenrui Jin, Wei Zhang, Yifan Jia, Keqian Wan, Gang Xiao, Diange Yang, and Zhihua Zhong. Improved hybrid a-star algorithm for path planning in autonomous parking system based on multi-stage dynamic optimization. *International Journal of Automotive Technology*, 24(2):459–468, apr 2023. ISSN 1229-9138. doi: 10.1007/s12239-023-0038-1. URL https://link.springer.com/10.1007/s12239-023-0038-1.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv*, 2013. doi: 10.48550/arxiv.1312.5602. URL https://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv*, 2016. doi: 10.48550/arxiv.1602.01783. URL https://arxiv.org/abs/1602.01783.

Preferred Networks. Gym-TORCS: Torcs environment for openai gym. https://github.com/ugo-nama-kun/gym_torcs.

Open Source Robotics Foundation. Gazebo: An open source robotics simulator. http://gazebosim.org/.

OpenAI. OpenAI Gym: A toolkit for developing and comparing reinforcement learning algorithms. https://gym.openai.com/.

M. J. Overton and G. Biros. A fast method for factoring the distance matrix in phylogenetic tree reconstruction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 10(1):21–34, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

L. Piegl and W. Tiller. *The NURBS Book*. Springer, 1995.

M. J. Pratt and I. D. Faux. *Computational Geometry for Design and Manufacture*. 1979.

Lucía Hilario Pérez, Marta Covadonga Mora Aguilar, Nicolás Montés Sánchez, and Antonio Falcó Montesinos. Path planning based on parametric curves. In Rastislav Róka, editor, *Advanced path planning for mobile entities*. InTech, sep 2018. ISBN 978-1-78923-578-4. doi: 10.5772/intechopen.72574. URL http://www.intechopen.com/books/advanced-path-planning-for-mobile-entities/path-planning-based-on-parametric-curves.

Jeevan Raajan, P V Srihari, Jayadev P Satya, B Bhikkaji, and Ramkrishna Pasumarthy. Real time path planning of robot using deep reinforcement learning. *IFAC-PapersOnLine*, 53(2): 15602–15607, 2020. ISSN 24058963. doi: 10.1016/j.ifacol.2020.12.2494. URL https://linkinghub.elsevier.com/retrieve/pii/S2405896320332171.

Jing Ren, Xishi Huang, and Raymond N. Huang. Efficient deep reinforcement learning for optimal path planning. *Electronics*, 11(21):3628, nov 2022. ISSN 2079-9292. doi: 10.3390/electronics11213628. URL https://www.mdpi.com/2079-9292/11/21/3628.

B. W. Robinson, L. Rodegerdts, W. Scarborough, W. Kittelson, R. Troutbeck, W. Brilon, L. Bondizio, K. Courage, M. Kyte, and J. Mason. Roundabouts: An informational guide. Technical report, Kittelson & Associates; Queensland University of Technology; Ruhr-Universitat Bochum; University of Florida; United States. Federal Highway Administration, June 1 2000. URL https://rosap.ntl.bts.gov/view/dot/15382.

I.J. Schoenberg and A. Sharma. Cardinal interpolation and spline fucntions v. the b-splines for cardinal hermite interpolation. *Linear Algebra and its Applications*, 7(1):1–42, 1973. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(73)90034-7. URL https://www.sciencedirect.com/science/article/pii/0024379573900347.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *arXiv*, 2015. doi: 10.48550/arxiv.1502.05477. URL https://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. doi: 10.48550/arxiv.1707.06347. URL https://arxiv.org/abs/1707.06347.

Junior A. R. Silva and Valdir Grassi. Path planning at roundabouts using piecewise linear continuous curvature curves. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, pages 1–6. IEEE, nov 2017. ISBN 978-1-5386-0956-9. doi: 10.1109/{SBR}-{LARS}-R.2017.8215288. URL http://ieeexplore.ieee.org/document/8215288/.

Beshoy Souliman. Effect of roundabouts on accident rate and severity in arizona, March 2016. A Thesis Presented in Partial Fulfillment of the Requirements for the Degree Master of Science.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3 (1):9–44, 1988.

TORCS Team. Torcs - the open racing car simulator, a. URL https://sourceforge.net/projects/torcs/.

Udacity Team. Udacity self-driving car simulator, b. URL https://github.com/udacity/self-driving-car-sim.

Siyu Teng, Xuemin Hu, Peng Deng, Bai Li, Yuchen Li, Yunfeng Ai, Dongsheng Yang, Lingxi Li, Zhe Xuanyuan, Fenghua Zhu, and Long Chen. Motion planning for autonomous driving: the state of the art and future perspectives. *IEEE Transactions on Intelligent Vehicles*, pages 1–21, 2023. ISSN 2379-8904. doi: 10.1109/{TIV}.2023.3274536. URL https://ieeexplore.ieee.org/document/10122127/.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), mar 2016. ISSN 2374-3468. doi: 10.1609/aaai.v30i1.10295. URL https://ojs.aaai.org/index.php/{AAAI}/article/view/10295.

Feng Wang, Dongjie Shi, Teng Liu, and Xiaolin Tang. Decision-making at unsignalized intersection for autonomous vehicles: Left-turn maneuver with deep reinforcement learning. *arXiv*, aug 2020. URL https://arxiv.org/abs/2008.06595.

Junjie Wang, Qichao Zhang, Dongbin Zhao, and Yaran Chen. Lane change decision-making through deep reinforcement learning with rule-based constraints. *arXiv*, 2019. doi: 10.48550/arxiv.1904.00231. URL https://arxiv.org/abs/1904.00231.

Pin Wang and Ching-Yao Chan. Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, oct 2017. ISBN 978-1-5386-1526-3. doi: 10.1109/{ITSC}.2017.8317735. URL http://ieeexplore.ieee.org/document/8317735/.

Pin Wang and Ching-Yao Chan. Autonomous ramp merge maneuver based on reinforcement learning with continuous action space. *arXiv*, mar 2018. URL https://arxiv.org/abs/1803.09203.

Shuhuan Wen, Yanfang Zhao, Xiao Yuan, Zongtao Wang, Dan Zhang, and Luigi Manfredi. Path planning for active SLAM based on deep reinforcement learning under unknown environments. *Intelligent Service Robotics*, 13(2):263–272, apr 2020. ISSN 1861-2776. doi: 10.1007/s11370-019-00310-w. URL http://link.springer.com/10.1007/s11370-019-00310-w.

Lilian Weng. Policy gradient algorithms - lil'log, 2021. URL https://lilianweng.github.io/posts/2018-04-08-policy-gradient/. Accessed on 2023-12-31.

Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv*, jun 2017. URL https://arxiv.org/abs/1706.09829.

Yan Yin, Zhiyu Chen, Gang Liu, and Jianwei Guo. A mapless local path planning approach using deep reinforcement learning framework. *Sensors (Basel, Switzerland)*, 23(4), feb 2023. doi: 10.3390/s23042036. URL http://dx.doi.org/10.3390/s23042036.

Chi Zhang, Kais Kacem, Gereon Hinz, and Alois Knoll. Safe and rule-aware deep reinforcement learning for autonomous driving at intersections. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2708–2715. IEEE, oct 2022a. ISBN 978-1-6654-6880-0. doi: 10.1109/{ITSC55140}.2022.9922164. URL https://ieeexplore.ieee.org/document/9922164/.

Hanbo Zhang, Site Bai, Xuguang Lan, David Hsu, and Nanning Zheng. Hindsight trust region policy optimization. *arXiv*, 2019. doi: 10.48550/arxiv.1907.12439. URL https://arxiv.org/abs/1907.12439.

Xiao Zhang, Tong Zhu, Lei Du, Yueqi Hu, and Haoxue Liu. Local path planning of autonomous vehicle based on an improved heuristic bi-RRT algorithm in dynamic obstacle avoidance environment. *Sensors (Basel, Switzerland)*, 22(20), oct 2022b. doi: 10.3390/s22207968. URL http://dx.doi.org/10.3390/s22207968.

Junwu Zhao, Ting Qu, and Fang Xu. A deep reinforcement learning approach for autonomous highway driving. *IFAC-PapersOnLine*, 53(5):542–546, 2020. ISSN 24058963. doi: 10.1016/j.ifacol.2021.04.142. URL https://linkinghub.elsevier.com/retrieve/pii/{S240589632100272X}.

Yanwei Zhao, Yinong Zhang, and Shuying Wang. A review of mobile robot path planning based on deep reinforcement learning algorithm. *Journal of Physics: Conference Series*, 2138(1): 012011, dec 2021. ISSN 1742-6588. doi: 10.1088/1742-6596/2138/1/012011. URL https://iopscience.iop.org/article/10.1088/1742-6596/2138/1/012011.