Technical University of Crete

School of Electrical and Computer Engineering

Deep Reinforcement Learning Reward Function Design for Lane-Free Autonomous Driving

Diploma Thesis

Athanasia Karalakou

Thesis Committee:

Supervisor : Georgios Chalkiadakis, Associate Professor

Committee Member : Michail G. Lagoudakis , Professor

Committee Member : Ioannis Papamichail, Professor (School Of P.E.M.)

July 2022

Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Σχεδίαση Συναρτήσεων Ανταμοιβής Μεθόδων Βαθιάς Ενισχυτικής Μάθησης για Αυτόνομη Οδήγηση χωρίς τη Χρήση Λωρίδων Κυκλοφορίας

Διπλωματική Εργασία

Αθανασία Καραλάκου

Επιτροπή:

Επιβλέπων : Γεώργιος Χαλκιαδάκης, Αναπληρωτής Καθηγητής

Μέλος Επιτροπής : Μιχαήλ Γ. Λαγουδάκης, Καθηγητής

Μέλος Επιτροπής : Ιωάννης Παπαμιχαήλ, Καθηγητής (Σχολή Μ.Π.Δ.)

Ιούλιος 2022

Abstract

Lane-free traffic is a novel and challenging research domain, in which vehicles do not adhere to the notion of lanes, but are rather able to be located at any lateral position within the road boundaries. This constitutes an entirely different problem domain for autonomous driving compared to lane-based traffic, as vehicles consider the entirety of the two-dimensional space available, and their decision-making needs to adapt to this concept. There is no leader vehicle or lane-changing operation to adjacent lanes, therefore the observations of the vehicles need to properly accommodate the lane-free environment without carrying over bias from lane-based approaches. In addition, each vehicle wishes to maintain a (different) desired speed, therefore creating many situations in which vehicles need to perform overtaking and react appropriately to the behavior of others.

At the same time, Deep Reinforcement Learning (DRL) has already been used in a variety of applications, while the fact that it can handle high dimensional state and action spaces makes it suitable for controlling autonomous vehicles. Existing studies, however, have not employed Reinforcement Learning (deep or otherwise) in the lane-free traffic domain.

Against this background, this diploma thesis initiates the study of the application of (Deep) Reinforcement Learning to lane-free traffic environments. To this end, we put forward a Markov Decision Process formulation for the problem of Lane-Free Autonomous Driving, by addressing all its elements. We consider the two-dimensional continuous action space, along with a discretized form, as well as the state space. Our main focus is on designing an effective reward function, as the reward model is crucial and determines the overall efficiency of the resulting policy.

Specifically, we construct different components of reward functions tied to the environment at various levels of information. Then, we combine and collate the aforementioned components and focus on attaining a reward function that results in a policy that manages to both reduce the collisions among vehicles, and also address their requirement of maintaining a desired speed.

Additionally, we study the performance of two quite popular DRL algorithms—namely Deep Q-Networks (enhanced with some commonly used extensions), and Deep Deterministic Policy Gradient (DDPG). Our experimental results indicate that DDPG has an overall better performance, and confirm that our DRL-employing autonomous vehicles are able to gradually learn effective policies in environments with varying levels of difficulty.

Abstract in Greek

Η οδική κυκλοφορία σε δρόμους χωρίς λωρίδες είναι ένας νέος τομέας έρευνας με ιδιαίτερες προκλήσεις, όπου τα οχήματα δεν περιορίζονται από την έννοια των λωρίδων, αλλά διαθέτουν τη δυνατότητα να βρίσκονται σε οποιαδήποτε πλευρική θέση εντός των ορίων του δρόμου. Αυτό συνιστά έναν εντελώς διαφορετικό τομέα προβλημάτων για την αυτόνομη οδήγηση σε σύγκριση με την κυκλοφορία που βασίζεται σε λωρίδες κυκλοφορίας, καθώς τα οχήματα λαμβάνουν υπόψη τους τον πραγματικά διαθέσιμο δισδιάστατο χώρο, ενώ και η διαδικασία που ακολουθούν για τη λήψη αποφάσεων πρέπει να προσαρμοστεί σε αυτήν την ιδέα. Επιπλέον, η έλλειψη των «οχημάτων-οδηγών» και της λειτουργίας αλλαγής κυκλοφοριακής λωρίδας σε παρακείμενες λωρίδες, συνιστούν αναγκαία τη σωστή προσαρμογή των παρατηρήσεων των οχημάτων στο περιβάλλον χωρίς λωρίδες, δίχως να υπάρχει μεροληψία από προσεγγίσεις που βασίζονται στην ύπαρξη κυκλοφοριακών λωρίδων. Ακόμα, κάθε όχημα αποσκοπεί να διατηρήσει μια (διαφορετική) επιθυμητή ταχύτητα, δημιουργώντας έτσι πολλές καταστάσεις όπου τα οχήματα πρέπει να προσπεράσουν και να αντιδράσουν κατάλληλα στη συμπεριφορά των άλλων.

Ταυτόχρονα, η Βαθιά Ενισχυτική Μάθηση (Deep Reinforcement Learning) έχει ήδη χρησιμοποιηθεί επιτυχώς σε ποίκιλες εφαρμογές, ενώ το γεγονός ότι μπορεί να χειριστεί υψηλών διαστάσεων χώρους καταστάσεων και ενεργειών, την καθιστά ιδιαίτερα κατάλληλη για τον έλεγχο αυτόνομων οχημάτων. Ωστόσο, μέχρι σήμερα, δεν υπάρχουν εργασίες που να έχουν αξιοποιήσει την (Βαθιά ή μη) Ενισχυτική Μάθηση για την κίνηση οχημάτων σε δρόμους χωρίς λωρίδες.

Τούτων δοθέντων, η παρούσα διπλωματική εργασία ξεκινά τη μελέτη της εφαρμογής της (Βαθιάς) Ενισχυτικής Μάθησης σε περιβάλλοντα κυκλοφορίας χωρίς λωρίδες. Για το σκοπό αυτό, μοντελοποιήσαμε το πρόβλημα της αυτόνομης οδήγησης χωρίς λωρίδες ως μια Μαρκοβιανή Διαδικασία Λήψης Αποφάσεων (Markov Decision Process), λαμβάνοντας υπόψη όλα τα επιμέρους στοιχεία της. Εξετάσαμε την αναπαραστάση του δισδιάστατου χώρου ενεργειών, θεωρώντας τον, είτε ως συνεχή, είτε ως διακριτό, ενώ ορίσαμε και το χώρο καταστάσεων. Το κύριο μέλημα μας, ωστόσο, υπήρξε η σχεδίαση μίας αποδοτικής συνάρτησης ανταμοιβής, καθώς το μοντέλο ανταμοιβής είναι ιδιαίτερης σημασίας και καθορίζει τη συνολική αποτελεσματικότητα της πολιτικής που προκύπτει.

Συγκεκριμένα, κατασκευάσαμε διαφορετικές συνιστώσες συναρτήσεων ανταμοιβής, οι οποίες συνδέονται με το περιβάλλον σε διάφορα επίπεδα πληροφορίας. Έπειτα, συνδυάσαμε και συγκρίναμε τις προαναφερθείσες συνιστώσες, με σκοπό στην εύρεση μιας αποτελεσματικής συνάρτησης ανταμοιβής, η οποία οδηγεί σε μια πολιτική που επιτυγχάνει ταυτόχρονα τη μείωση

των συγκρούσεων με άλλα οχήματα, αλλά και την διατήρηση μίας επιθυμητής ταχύτητας.

Επιπλέον, συγκρίνουμε δύο αρκετά δημοφιλείς και θεμελιακούς αλγορίθμους βαθιάς μάθησης, πιο συγκεκριμένα τον αλγόριθμο Deep Q-Networks (DQN), εφοδιασμένο με ορισμένες ευρέως χρησιμοποιούμενες επεκτάσεις, και τον αλγόριθμο Deep Deterministic Policy Gradient (DDPG). Τα πειραματικά μας αποτελέσματα υποδεικνύουν ότι ο DDPG έχει συνολικά καλύτερη απόδοση και επιβεβαιώνουν ότι τα αυτόνομα οχήματα που χρησιμοποιούν Βαθιά Ενισχυτική Μάθηση είναι σε θέση να μαθαίνουν σταδιακά όλο και πιο αποτελεσματικές πολιτικές σε περιβάλλοντα με διαφορετικά επίπεδα δυσκολίας.

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network. 12–14, 18, 34, 56

AV Autonomous Vehicle. 1, 2

DDPG Deep Deterministic Policy Gradient. 2, 18–20, 24, 33, 36, 48, 52, 57

DDQN Double Deep Q-Networks. 16, 17, 48, 52

DNA Dueling Network Architecture. 18, 48, 52

DQN Deep Q-Networks. 2, 15–19, 23, 33, 48, 52, 57

DRL Deep Reinforcement Learning. 12, 18, 34, 48, 56, 57

MDP Markov Decision Process. 2, 6, 10, 21, 22, 34, 58

NAF Normalized Advantage Function. 57

PER Prioritized Experience Replay. 17, 48, 52

PPO Proximal Policy Optimization. 57

ReLU Rectified Linear Unit. 14, 33

RF Reward Function. 31, 32, 38, 46, 48, 52

RL Reinforcement Learning. 1, 2, 6–9, 11, 12, 22, 24, 33, 48, 56–58

TanH Hyperbolic Tangent. 14

TD Temporal Difference. 17

Contents

Chapter 1

Introduction

Reinforcement Learning is a rapidly developing area of machine learning that significantly impacts the future of technology and everyday life. Nevertheless, its applications were limited to simple problems within constrained environments until recently. Nowadays, significant steps have been made toward making reinforcement learning more efficient. This fact allowed researchers to address many unsolved challenges such as autonomous driving, traffic signal control systems, and transportation systems overall.

In recent decades, such a challenging domain has been vehicular traffic, which has been the primary means of transporting both people and products, making life more comfortable and faster. However, nowadays, the ever-increasing vehicular traffic congestion is observed, imposing numerous complications on everyday life. In particular, traffic congestion causes an increase in harmful emissions, delays in travel time, reduced traffic safety, and excessive economic loss. Indisputably, taking measures to address this phenomenon is considered imperative. Thus, during the past years, the automobile industry, as well as various researchers, have made vast endeavors to alleviate this problem by investigating both autonomous driving vehicles and alternative traffic models.

1.1 Motivation

Applications of Reinforcement Learning (RL) in the field of autonomous driving are gaining a momentum in recent years [1] due to advancements in Deep RL [2, 3], giving rise to novel techniques [4]. Another important reason for this momentum is an increasing interest towards autonomous vehicles (AVs), as the current and projected technological advancements in the automotive industry can enable such methodologies in the real-world [5, 6].

As a result, novel traffic flow research endeavours have already emerged, such as *TrafficFluid* [7], which primarily targets traffic environments with 100% penetration rate of AVs (no human drivers). *Trafficfluid* examines traffic environments with two fundamental principles:

- (i) *Lane-free* vehicle movement, meaning that AVs under this paradigm do not consider lane-keeping, but are rather free to be located anywhere laterally. Lanes

emerged to simplify human driving, and when only automated vehicles exist, they may no longer be required, given the observational capabilities of an AV compared to that of a human driver.

- (ii) *Nudging*, where vehicles may adjust their behavior so as to assist vehicles on the back that attempt overtake. In lane-based traffic, this would involve a lane-change operation, and as such, nudging could not actually be considered in conventional traffic. By contrast, in lane-free settings, a vehicle can provide the necessary space for receding vehicles just by leaning towards an appropriate lateral direction.

In the context of lane-free driving, multiple vehicle movement strategies have already been proposed [7, 8, 9, 10]. To be more specific, previous research has focused on optimal control methods, such as model predictive control [9]. In addition, the application of a movement strategy based on heuristic rules that involve the notion of "forces" [7] and the deployment of a cruise controller has already been examined [10]. Finally, [8] investigates the utilization of the max-plus algorithm, and constructs a dynamic graph structure of the vehicles, considering communication among vehicles as well.

However, to the best of our knowledge, as of now no work that tackles the problem of lane free traffic with RL techniques, while there is an abundance of (Deep) RL applications for conventional (lane-based) traffic environments [1, 5, 4]. We believe that (Deep) RL could be a valuable asset to managing such a challenging domain, as it is capable of solving complex and multi-dimensional tasks with lower prior knowledge because of its ability to learn different levels of abstractions from data. Furthermore, (Deep) RL provides policies that automatically adjust to the environment, and thus there is no need for eithera centralized authority or explicit communication among vehicles. The efficiency of RL heavily relies on the design of the Markov Decision process and especially on the reward model. The design of a reward function, though, is a challenge for (Deep) RL.

1.2 Contributions

In this work, we view the problem of designing an RL agent that learns a vehicle movement strategy in lane-free traffic environments. As such, we design a Markov Decision Process (MDP) for lane-free autonomous driving, given a single agent and an environment with other vehicles adopting a lane-free driving policy. Regarding the MDP formulation, we examine both discrete and continuous action domains, as they are tied with different methodologies. Specifically, we examine Deep Q-Networks (DQNs) and some common extensions which require a discretized action domain, and compare with Deep Deterministic Policy Gradient (DDPG), which was designed to handle continuous action domains.

The reward design is crucial and determines the overall efficiency of the resulting policy [5]. Given the nature of the algorithms, their ability to properly learn only with delayed rewards and obtain a (near) optimal policy is uncertain, so we propose a set of different reward components, ranging from delayed rewards to more elaborate and therefore more

informative regarding the problem's objectives. The learning objectives in our environment are twofold, and include safety, i.e., collision avoidance among vehicles, and that our agent is able to maintain a desired speed of choice. One of the more informative reward components we introduce includes the notion of desired lateral placement. In essence, it is calculated by partitioning the road downstream into available lateral zones based on the downstream traffic, and the agent is guided towards the selected zone through the reward. However, during the experimentation process, we noted that this particular reward component guides the agent towards more specific solutions, as it adds bias to the optimization procedure and thus leads to performance inconsistencies. Contrariwise, while testing reward functions that do not predispose our agent, we noticed that the agent tends to tackle both objectives more efficiently, even in the most challenging environments.

Part of this work has already been published under the same title "*Deep RL Reward Function Design for Lane-Free Autonomous Driving*" [11], coauthored by Athanasia Karalakou, Dimitrios Troullinos, Georgios Chalkiadakis and Markos Papageorgiou and appears in the Proceedings of the *20th International Conference on Practical Applications of Agents and Multi-Agent Systems* (PAAMS 2022), L'Aquila, Italy, July 2022.

1.3 Thesis Outline

In Chapter 2 we provide the necessary theoretical background for this thesis. First, we give an overview of several concepts; specifically the terms of Markov Decision Process, Machine Learning, Artificial Neural Networks and the algorithms that we utilize for Deep Reinforcement learning . Next, in Chapter 3 we provide information about the Lane-Free traffic environment, discuss previous and related work, and also present our approach. Then, in Chapter 4 we present our training settings, demonstrate our experimental evaluations and discuss results and various trade-offs that emerge. Finally, in Chapter 5 we summarize this thesis and adress potential future work.

Chapter 2

Theoretical Background

In this chapter we discuss the Theoretical Background of topics regarding this Thesis. Discussion topics involve Machine Learning, Reinforcement Learning, Deep Neural Networks and Markov Decision Processes.

2.1 Machine Learning

Machine Learning [12] is a subfield of Artificial Intelligence [13], that provides systems with the potential to learn and improve from experience without being explicitly programmed. Machine learning focuses on developing computer programs that can access information and use it for independent learning. The learning process begins with observations or data, such as examples, direct experiences, or instructions, to look for patterns in the data and make better decisions in the future based on the examples that have been provided. It aims to allow computers to learn and to adjust actions automatically, without any human help.

Machine Learning was firstly introduced in 1959 by Arthur Samuel [12] and is mostly classified according to the way an algorithm learns to become more accurate in its predictions. There are four main approaches: supervised, unsupervised, semi-supervised, and reinforcement learning. It is worth mentioning that the type of algorithm that is used depends on the type of prediction data.

- Supervised Learning [14]: This approach includes algorithms that use defined variables, as well as both labeled input and output data as training data, to assess for correlations. Both the input and the output of the algorithm are considered known.

- Unsupervised Learning [15]: is the task of training based on unlabeled and unclassified input data, with the purpose of identifying patterns in data sets and information.

- Semi-supervised Learning [16]: This type of machine learning is a combination of supervised and unsupervised learning, meaning that typically combines a small amount of labeled data with a large amount of unlabeled data for training.

- Reinforcement Learning [17]: This is an alternative paradigm of machine learning which addresses decision-making problems, and is described in more detail below in Section 2.2.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of Machine Learning used for decision-making problems [17]. Reinforcement learning differs from both supervised and unsupervised learning, as well as from semi-supervised learning, in the sense that its main goal is to optimize the behavior of an agent in a specific environment, by maximizing the obtained rewards that correspond to the agent's decision-making.

In particular, Reinforcement Learning can be conceived as a trial and error type of learning, meaning that the agent explores its environment and is guided towards an action through a reward function, by aiming to find an optimal behavior that maximizes the total future rewards. Typically, in RL applications there are two distinct entities, the agent and the environment, which are usually formulated in a Markov Decision Process (MDP).

In the following Figure 2.1, a very basic reinforcement learning model and how the agent interacts with the environment are depicted. Specifically as studied in Sutton's and Barto's book titled "Reinforcement Learning: An Introduction" [18], the agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, ...^2$. At each time step t, the agent receives some representation of the environment's state, $S_t \in S$, where S is the set of possible states, and on that basis selects an action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in state S_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in R \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}.

At each time step, the agent has a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted as π_t, where $\pi_t(a|s)$ is the probability that agent will take action $A_t = a$ if $S_t = s$. RL methods specify how the agent learns a policy $\pi_t(a|s)$ as a result of its experienced transitions.

There are many different types of RL algorithms, and a typical taxonomy contains the following categories:

- ***Value-based learning*** [19] applies to cases that an agent learn the state or state-action value and then act by choosing the best action in the state, i.e., the one that maximizes the learned value function.

- ***Policy-based learning*** [20, 21] describes cases that an agent learns directly a (potentially stochastic) policy function that maps state to action (or probabilities for each available action).

- ***Actor-Critic learning*** [22] is a combination of the above-mentioned categories. An agent learns both a policy and a value function, with the learning part of both functions influencing one another. In more detail, the policy function (actor) learns

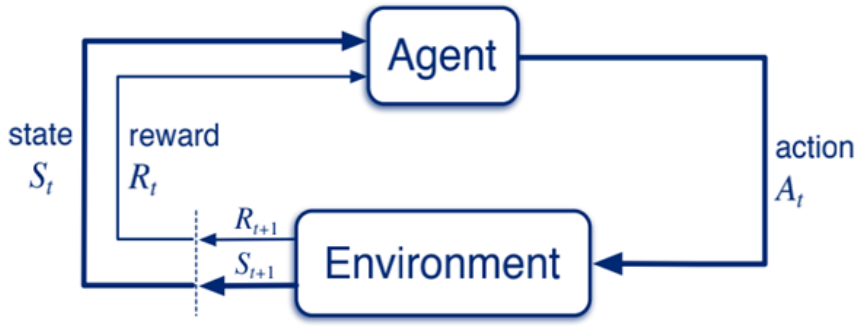


Figure 2.1: The agent–environment interaction in reinforcement learning

and proposes the most appropriate action (or probability of each action) given a state, while the estimated value function (critic) evaluates the actions taken by the actor given a specific state and the policy resulting by the actor.

RL algorithms can also be characterized by whether the environment dynamics (transition and reward functions) are explicitly learned or not. As such, we can also distinguish RL algorithms into two categories:

- ***Model-based learning*** [23, 24] is when the agent is designed to learn a model that describes how the environment works from its observations and then plan a solution using that specific model.

- ***Model-free learning*** [25] refers to an agent that can directly derive an optimal policy from its interactions with the environment without having to create a model of the environment (either the transition dynamics or the reward function)

2.2.1 Reward and Return

In RL problems, a reward signal is used to determine the goal. Given a state s, each action taken is associated with a reward, that evaluates the outcome of applying that particular action. As described in Sutton's and Barto's book [18], the purpose of a reinforcement learning agent is to maximize the expected cumulative sum of rewards. To define the total long-term reward of a trajectory after a time-step t, we use a metric referred to as the return G_t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

where γ is a value between $[0, 1)$, called the discount factor and is used not only to bound the return, but also to model the uncertainty about the future.

2.2.2 Markov Decision Processes

The concept of Markov Decision Processes was first introduced by R. Bellman in the 1950's [26]. It is a mathematical framework that completely describes the environment in a reinforcement learning problem. A Markov decision process is described by the 5-tuple:

$$(S, A, P, R, \gamma)$$

In detail:

- S describes the state space, a finite set of all possible states of the environment.
- A describes the action space, a finite set of all possible actions that the agent can choose.
- P describes the transition model, a matrix that defines transition probabilities from all states s to all successor states s_0 given the action taken a.

$$(P^a_{ss'} = Pr(S_{t+1} = s_0 | S_t = s, A_t = a)) \tag{2.2}$$

- R describes the reward model that defines the reward that the agent receives at each time step.
- The discount factor, described by γ, is used to define the return.

The state transitions of the environment are assumed to obey the Markov property, which means the probability of reaching a future state only depends on s, and not on the history of any earlier states. If the complete state of the environment is available to the agent through the state, the environment is fully observable. On the other hand, the environment is partially observable, if only a partial observation is available. The framework of Partially Observable Markov Decision Processes [27] addresses such environments, which however lies beyond the scope of this thesis.

Different environments allow different kinds of actions, and the set of all valid actions in a given environment is called the action space. The action space can be characterized as a discrete action domain, where a finite number of moves are available to the RL agent [18]. Other environments can alternatively incorporate continuous actions spaces, i.e., a real-valued action vector.

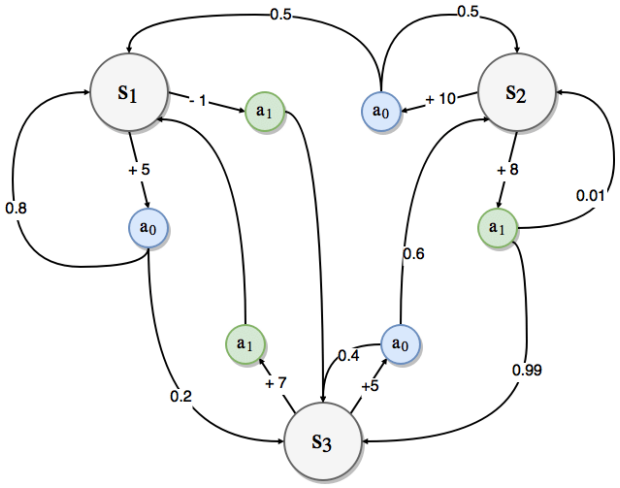


Figure 2.2: Representation of an MDP

2.2.3 Policy

A policy π dictates the behaviour of the agent. There are two different types of policies: they can be either deterministic and return a specific action a based on the state, $\pi(s)$, or stochastic, $\pi(a|s)$, i.e., define a probability distribution over the actions given a state.

2.2.4 Value Functions

The Value Function $v_\pi(s)$ evaluates how fitted it is for the agent to be in state s. This function is the expected discounted sum of rewards that the agent will receive while following a certain policy π a state s. The value function, $v_\pi(s)$ for a specific policy π is calculated as:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \forall s \in S \tag{2.3}$$

where π is the policy followed by the agent, and can be either deterministic or stochastic; hence the existence of the expectation operation. An action-value function $q_\pi(s, a)$ can also be defined. The action-value of a state is the expected return if the agent selects an action α and then follow policy π.

$$q_\pi(s, a) = E_\pi(G_t|s_t = s, a_t = a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a], \forall s \in S \ and \ \forall \alpha \in A \tag{2.4}$$

An important property of value functions used in RL is that they can be formulated recursively. The recursive form is based on the Bellman Equation [28] for v_π and has the

following form:

$$\begin{aligned} v_\pi(s) = E_\pi[G_t|S_t = s] &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma v_\pi(s')] \end{aligned} \tag{2.5}$$

In addition, the Bellman equation is applied to the action-value function, as follows:

$$\begin{aligned} q_\pi(s,a) = E_\pi(G_t|s_t = s, a_t = a) &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] \\ = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_a \pi(a|s) q_\pi(s', a')] \end{aligned} \tag{2.6}$$

The Bellman equation utilizes the structure of the MDP formulation to decrease this infinite sum to a system of linear equations [17]. The precise state values can then be determined by directly solving the equation.

2.3 Q-Learning

Q-Learning [29] is a popular off-policy value-based method It approximates the optimal action-value function $Q^*_\pi(s, a)$, used to evaluate action a in state s. The Bellman Equation is utilized for this reason:

$$Q^*_\pi(s, a) = r + \gamma \max_{a'} Q^*_\pi(s', a') \tag{2.7}$$

where r is the current reward, while the remaining part of the equation refers to a expected future (discounted) rewards. The approximated $Q(s, a)$ function is acquired through Q-Learning by minimizing the total expected loss function stated as the difference between the predicted reward at a state and the obtained reward. As such, Q-Learning constitutes a Temporal Difference Learning [30] method. The action-value update rule of Q-learning can be described as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.8}$$

where α denotes the learning rate. A pseudocode that describes Q-Learning, adapted by [30], is provided in Algorithm 1.

Algorithm 1: Q-Learning

Result: Off-policy control for estimating $\pi \simeq \pi_*$
Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a) \forall s \in S$, $\alpha \in A(s)$ arbitrarily.
Set $Q(terminal, \cdot) = 0$
for *each episode* **do**
  Initialize s
  for $step = 0$ ***to*** T **do**
    Choose a from s using policy derived from Q (e.g. ε-greedy);
    Take action a, observer r, s';
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$;
    $s \leftarrow s'$;
  end
end

2.4 Exploration vs Exploitation

The exploration-exploitation dilemma [31] has been a recurring theme in reinforcement learning and AI in general. The dilemma lies between exploiting the agent's current estimated value by choosing a greedy approach to get the most reward, and allowing the agent to improve its knowledge about each state/action which could lead to long-term benefit. Many policies exist that address this issue, and the ones that are relevant to our work are described below.

2.4.1 Epsilon-Greedy Action Selection

In epsilon-greedy action selection, the agent attempts to balance both exploitation and exploration. Essentially, the agent selects an exploratory action with probability ϵ and a greedy action (based on the agent's greedy policy) with probability $1 - \epsilon$. Epsilon-Greedy exploration is often used as a behaviour policy in several reinforcement learning models, to encourage exploration in training. We provide a pseudocode implementation of this exploration method in Algorithm 2.

2.4.2 Ornstein–Uhlenbeck process

The Ornstein Uhlenbeck process [32] is a stochastic process that satisfies the following stochastic differential equation:

$$dx_t = \theta(\mu - x_t)\, dt + \sigma\, dW_t \tag{2.9}$$

where x_t stands for a state in RL ,$\theta > 0$, μ and $\sigma > 0$ are parameters and W_t refers to the Wiener process [33].

Algorithm 2: Epsilon-Greedy Action Selection

Data: Q: Q-Table generated so far, ϵ : a value within 0 and 1, S: current state
Result: Selected Action
Function `SELECT-ACTION`(Q, S, ϵ)**:**
  $n \leftarrow$ uniform random number between 0 and 1 ;
  if $n < \epsilon$ **then**
    $A \leftarrow$ random action from the action space;
  else
    $A \leftarrow \max Q(S, .)$;
  end
  return selected action A;

The Ornstein–Uhlenbeck (OU) process is often used as stochastic process to incorporate noise for continuous action domains in Deep RL problems, specifically for action-selection in the Deep Deterministic Policy Gradient algorithm [34] (see Section 2.5.6 for more details). The noise is temporally correlated allowing to set a long-term mean μ. The process moves towards μ with a given standard deviation Σ at a rate θ and current value x_t over timesteps of the episode and is reset with an episode termination.

$$a_t = \mu(s_t|\theta^\mu) + OU(x_t, , , \theta, , , \mu, , , \Sigma) \tag{2.10}$$

where the action is marked as a_t, while the Ornstein-Uhlenbeck process, marked as OU, is used to generate temporally correlated exploration, meaning that it generates noise that is correlated with the previous noise.

2.5 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) [35, 36] is an aggregate of reinforcement learning and deep learning [37]. DRL typically pertains to the utilization of Deep Neural Networks as function approximators for value functions or policy in the context of RL.

Deep Neural Networks have already been used to extend a variety of RL methods [38]. In this work, we focus on Q-learning, and a specific actor-critic method that we discuss below in this section.

2.5.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) [39, 40], also mentioned simply as Neural Networks [41, 13], are computing systems modeled after the neurons in a biological brain. Resembling the neurons in the brain, ANNs also consist of nodes which are arranged in various layers, a typical Artificial Neural Network is composed of three layers [41]:

- ***Input layer***: a layer that receives the external data to perform pattern the training.

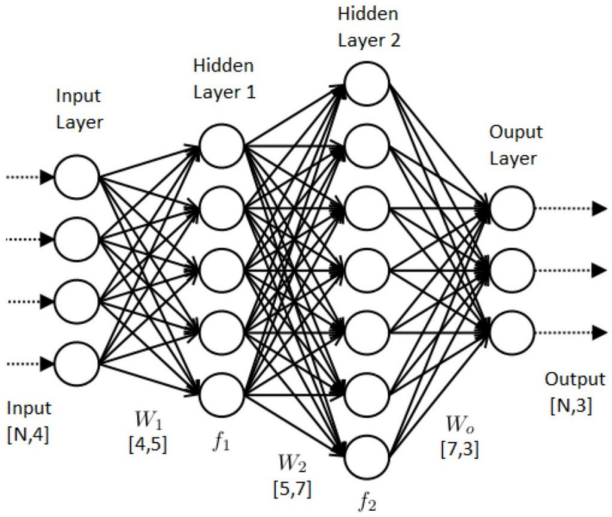


Figure 2.3: Representation of an Artificial Neural Network

- ***Hidden layer(s)*** or Intermediate layer(s): one or multiple layers that separate the input and output layers and where the computational process takes place.
- ***Output layer***: a layer that produces the results for the provided inputs.

In most cases, there will be multiple hidden layers in a neural network. In that case, the neural network is known as a Multi-Layer ANN or a Deep Neural Network. Moreover, each layer has a number of neurons that are connected to the corresponding neurons in another layer. The connections are assigned weights that are used for the unit's activation.

In order to perform activation, a weighted sum of its inputs is computed at each unit and then passed to an activation function to produce the layer's output value. For that purpose, we discuss a few commonly used activation functions [41, 42], namely the Sigmoid, Hyperbolic Tangent and Rectified Linear Unit.

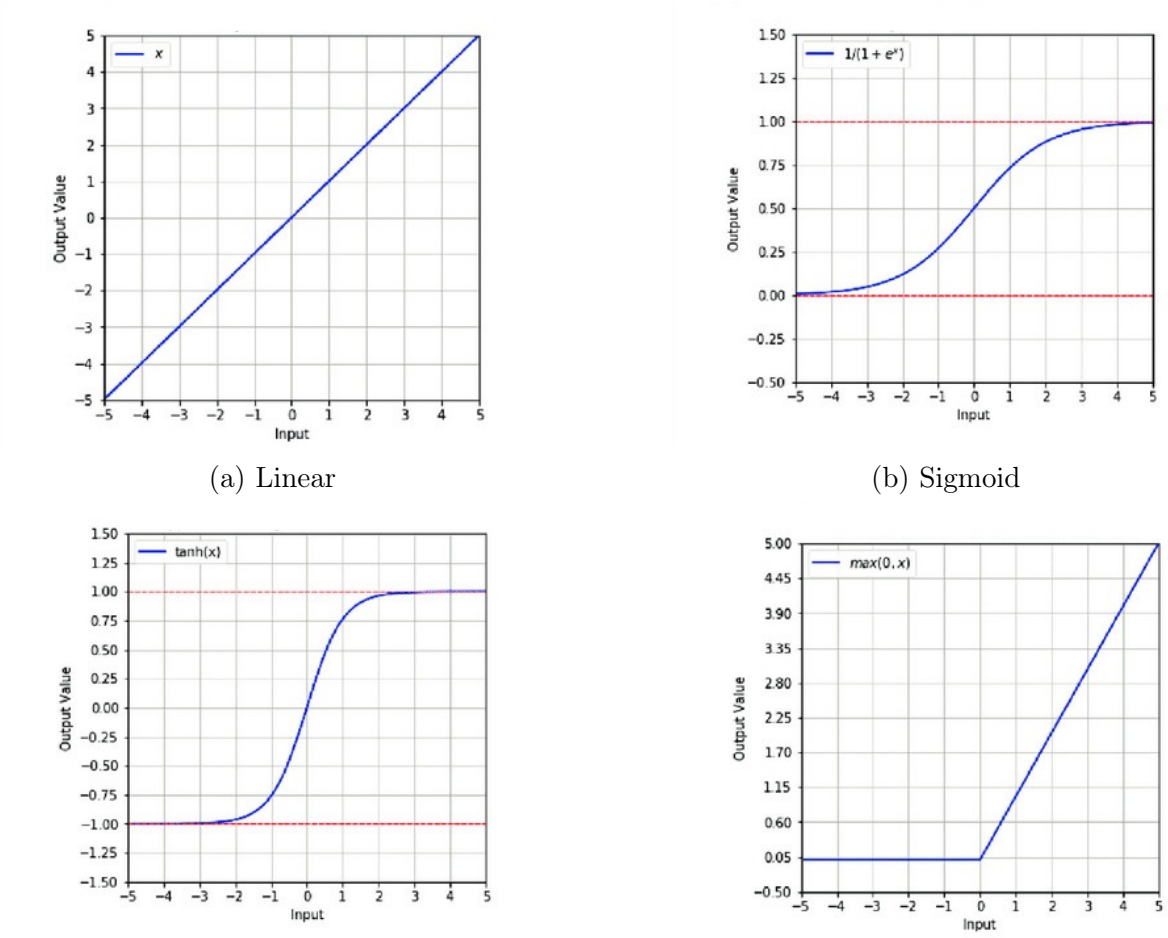
The ***Linear function***, illustrated in Figure 2.4a, is a function that produces an output which is proportional to the input according to a coefficient a, meaning that: $y = a \cdot x$. Assuming $a = 1$, a linear activation function has the form:

$$linear(x) = x \tag{2.11}$$

The ***Sigmoid function***, showcased in Figure 2.4b, is a math function with a sigmoid or a typical "S" curve. It maps the input value x between the range of 0 and 1. Large negative values become 0 and large positive values become 1.

$$sigm(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

The ***Hyperbolic Tangent function***, as shown in Figure 2.4c, resembles the sigmoid activation function, but it ranges from -1 to 1. It also shares a similar "S" form and is



(a) Linear

(b) Sigmoid

(c) TanH

(d) ReLU

Figure 2.4: Types of activation functions

differentiable.

$$tanh(x) = 2 * sigm(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{2.13}$$

The ***ReLU function*** [43], as depicted in Figure 2.4d, is the most popular activation function. As authors in [44] claim, the use of ReLU activation outperforms both TanH and Sigmoid functions in several applications, while sharing a similar computational cost.

$$ReLU(x) = max(0, x) \tag{2.14}$$

In addition, to actually train a neural network, we use *Optimizers* [45], in order to change the attributes of a neural network, i.e., the network's weights and even adjust the learning rate upon training. The most commonly used optimization functions are based on (Stochastic) Gradient Descent [45, 46, 47]. One of the most popular optimization algorithms used for ANN's training is Adam [48], which incorporates various optimization Specific types of artificial neural networks include:

- ***Feed-forward neural networks*** [49]: is the most common and simple architecture. In this framework, the data passes through the input layer to the hidden layers, while the output layer which computes the final output.

- ***Recurrent neural networks*** [50]: work on the principle of saving the output of a layer and feeding this back to the input to help in predicting the outcome of the layer. Here, the first layer is shaped in a similar way to the feed forward neural network with the product of the sum of the weights and the features. Once this is computed, the recurrent neural network process starts, which means that from one time step to the following, each neuron will store an amount of information it held in the previous time step.

- ***Convolutional neural networks*** [51, 52]: are inspired by the receptive field in the brain, that processes sensor input data and is sensitive to certain stimuli. They process large amount of input data efficiently and thus are popular in advanced approaches in the fields of Computer Vision and Machine Learning.

- ***Modular neural networks*** [53]: have a collection of distinct networks that act independently and contribute to the final output. Moreover, each neural network has a set of inputs that are unique compared to other networks constructing and performing sub-tasks. In completing the tasks, these networks do not interact or communicate with one another.

2.5.2 Deep Q-Network

DeepMind introduced Deep Q-Network (DQN), which adapts the Q-Learning algorithm [54, 29, 18] for function approximation using Neural Networks, utilizing Convolutional Neural Networks to obtain a graphical representation of the input state for an environment, and produce a vector of Q-values associated with each possible action.

The concepts of target network and experience replay, formally introduced in this work [2], are the two important methods that enable the use of deep learning for approximating the Q function and addressing the issues of network stability.

Specifically, the target network is identical to the approximated Q function, and is introduced to stabilize the learning process. Moreover, it computes a target value and is updated by the Q function at a regular rate. The Q-network is updated according to the following loss function:

$$L(\theta_t) = E_{(s_t,a_t,r_t,s_{t+1})}[(y_t^{DQN} - Q(s_t, a_t; \theta_t))^2] \tag{2.15}$$

where $y_t^{DQN} = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$ refers to the Q-Network's target value at iteration t, while θ_t and θ^- are the network's parameters at iteration t and at a previous iteration respectively, with the latter being the target network's parameters.

The loss function is minimized using stochastic gradient descent. The behaviour policy is usually an Epsilon-Greedy policy (see Section 2.4.1) to ensure sufficient exploration.

Algorithm 3: DQN Algorithm

Initialize replay memory $D = \emptyset$
Initialize action-value function Q with random θ
Initialize target action-value function $\overline{Q}$ with $\overline{\theta} = \theta$
for *episode* $= 1, 2....M$ **do**
  With probability ϵ select a random action a_t
  otherwise select $a_t = argmax_a Q(s_t, a; \theta)$
  Execute action a_t in emulator and observe reward r_t and state s_{t+1}
  $D \leftarrow D \cup (s_t, a_t, r_t, s_{t+1})$
  Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
  Set $y_j = \begin{cases} r_j, & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_a Q(s_{j+1}, a; \overline{\theta}), & \text{for non-terminal } s_{j+1} \end{cases}$
  Perform a gradient descent step on $J(\theta) = (y_j - Q(s_j, a_j; \theta))^2$
  Every C steps, set $\overline{\theta} = \theta$
end

The main idea of Experience Replay is to store the agent's experiences (the tuples (s_t, a_t, r_t, s_{t+1})) in a buffer. Then, in each training step, a batch of experiences is uniformly sampled from the buffer and fed to the network for training. Experience Replay ensures that old experiences are not disregarded in later iterations, and removes the correlations in the data sequences, feeding the network with independent data. This is suitable to DQN, since it is an off-policy method. We describe the complete DQN procedure, as explained in the original paper [2], in Algorithm 3.

2.5.3 Double DQN

A drawback of Q-learning, and consequently of DQN, is overestimation, occurring from to the use of the max operator in the Bellman equation to compute Q-values. Recently, Hado van Hasselt et al. [55] constructed a new Double Q-learning (see Hado van Hasselt, 2010 [56]) inspired algorithm called Double DQN (DDQN), that addressed this issue by decomposing the max operation in the target into action selection and evaluation.

The idea behind DDQN is that of Double Q-Learning [56], which includes two Q-functions, where one function selects the optimal action, while the other estimates the value function. As a consequence, Double DQN offers a faster training and more stable learning across many domains, as evident by the results in [56].

To be specific, DDQN uses DQN's target network as the second action-value function, i.e., it assesses the greedy policy according to the online network, while utilizing the target Q-network to calculate its value. The update is similar to DQN, but replacing the target y^{DQN} with:

$$y_t^{\text{DDQN}} = r_t + \gamma Q(s_{t+1}, \text{argmax}_{a'} Q(s_{t+1}, a'; \theta_t), \theta^-) \tag{2.16}$$

Algorithm 4: DDQN with Prioritized Experienced Replay

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T

Initialize replay memory $D = \emptyset, \Delta = 0, p_1 = 1$

Observe s_0 and choose $a_0 \sim \pi_\theta(s_0)$

for $t = 1$ ***to*** T **do**

  Observe s_t, r_t, γ_t

  Store transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in D with maximal priority $p_t = \max_{i<t} p_i$

  if $t \equiv 0$ *mod* K **then**

    for $j = 1$ ***to*** k **do**

      Sample transition $j \sim P(j) = {p_j}^\alpha / \sum_i {p_i}^\alpha$

      Compute importance-sampling weight $w_j = (NP(j)) - \beta / \max_i w_i$

      Compute TD-error

      $\delta_j = r_j + \gamma Q(s_j, argmax_a Q(s_j, a; \overline{\theta}); \theta) - Q(s_{j-1}, a_{j-1}; \theta)$

      Update transition priority $p_j \leftarrow |\delta_j|$

      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(s_{j-1}, a_{j-1}; \theta)$

    end

    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$

    From time to time copy weights into target network $\overline{\theta} \leftarrow \theta$

  end

  Choose action $a_t = argmax_a Q(s_t, a; \theta)$

end

The goal of DDQN is to translate the idea of Double Q-learning within the framework of DQN.

2.5.4 Prioritized Experience Replay

Prioritized Experience Replay (PER) [57] is an extension on Experience Replay, which quantifies the 'importance' of each collected transition, and does not rely on a uniform distribution for the sampling process.

Each experience tuple is now stored with an additional 'priority' value, so that experiences with higher priority have a higher sampling probability and therefore have the chance to remain longer in the buffer than others. As importance measure, the Temporal Difference (TD) error can be used. It is expected that if the TD error is high (in absolute value), the agent can learn more from the corresponding experience, because the agent behaved better or worse than expected. We present a pseudocode for DDQN with Prioritized Experienced Replay in Algorithm 4, as provided by the authors in [57] Furthermore, in the experimental evaluation of the original paper [57], the authors showcase a significant speed up the learning process when PER was used, compared to a conventional Experience Replay buffer.

2.5.5 Dueling Network Architecture

In the work of [58], authors introduce a novel algorithm for DRL with Dueling Network Architectures (DNA), where the neural network architecture decouples the value and advantage function. In general, the advantage function has the form:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.17}$$

DNA leads to improved performance in the paper's examined domains [58]. In detail, the ANN is constructed with two streams, one of them providing an estimate of the value function, while the other stream produces an estimate of the advantage function. Note that both of those streams use the same convolutional feature learning module, and are then combined to compute the state-action value function Q.

The Q-value function, according to the authors, is now given by:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \Big(A(s,a;\theta,\alpha) - \max_{a' \in |A|} A(s,a';\theta,\alpha)\Big) \tag{2.18}$$

Now, for

$$a^* = argmax_{a' \in A} Q(s,a';\theta,\alpha,\beta) = argmax_{a' \in A} A(s,a';\theta,\alpha), \tag{2.19}$$

we obtain

$$Q(s,a^*;\theta,\alpha,\beta) = V(s;\theta,\beta) \tag{2.20}$$

Hence, the stream $V(s;\theta,\beta)$ provides an estimate of the value function, while the other stream produces an estimate of the advantage function. Meanwhile, the learning update is done as in DQN and it is only the structure of the neural network that is modified.

Usually, a slightly different approach is preferred in practice because it tends to increase the stability of the learning process.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \Big(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a' \in A} A(s,a';\theta,\alpha)\Big) \tag{2.21}$$

where in that case, the authors state that the advantages only need to change according to the average value as baseline, which appears to work better in practice.

2.5.6 Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient (DDPG) [34] introduces the direct representation of a policy in such a way that it can extend the DQN algorithms to overcome the restriction of discrete actions. It is an off-policy, actor-critic algorithm that utilizes the Deterministic Policy Gradient (DPG) [59] and the DQN architectures.

This method presents a few upgrades to the traditional actor critic to make it applicable with neural networks, i.e., it uses experience replay, just like in DQN and target networks to compute the target y in the temporal difference error. To be more specific, target networks in this context are two separate networks, which are copies of the actor and critic

Algorithm 5: DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for *episode* $= 1,\ M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for $t = 1,\ T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action at and observe reward rt and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
    $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a] Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i)\nabla_{\theta^\mu}\mu(s|\theta^\mu)|s_i$$
    Update the target networks:
    $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
    $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
  end
end

network. Target networks in DDPG incorporate the soft target update, meaning that the networks' weights are adjusted in a slower pace using a temperature parameter to improve the learning stability. It was found to provide improvement on the resulting policy at the cost of slower learning. This transform the procedure of learning the optimal Q-function into a supervised learning problem.

The update for the critic is given by the standard DQN update by taking targets y_t^{DDPG} to be:

$$y_t^{DDPG} = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \theta^{\mu-}); \theta^{Q-}) \tag{2.22}$$

where $Q(s, a; \theta^{Q-})$ and $\mu(s; \theta^{\mu-})$ refer to the target networks for the critic and actor respectively.
The loss function is the following

$$L(\theta_t) = E_{s\sim\rho\beta,\alpha\sim\beta}(y_t^{DDPG} - Q(s_t, a_t; \theta_t^Q))^2 \tag{2.23}$$

where $Q(s, a; \theta^Q)$ refer to the critic network with weights θQ and β is the behaviour policy. Moreover, the policy (actor network) in DDPG is updated using the sampled policy gra-

dient, given a minibatch of transitions [59]:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a; \theta^Q)|_{s=s_i, a=\mu(s_t;\theta^\mu)} \nabla_{\theta^\mu} \mu(s; \theta^\mu)|_{s=s_i} \tag{2.24}$$

where $\mu(s; \theta^\mu)$ refers to the the actor network with parameters θ^μ and N is the size of the sampled minibatch from the buffer. As mentioned, DDPG also introduces the notion of soft target updates to improve upon learning stability, meaning that the network parameters of the actor (θ^μ) and critic (θ^Q) are updated in every iteration through a temperature parameter τ as: $\theta^- = \tau\theta + (1 - \tau)\theta^-$, with $\tau \ll 1$.

Finally, to address the issue of exploration in a continuous domain, the authors in [34], designed an exploration policy μ' by adding noise sampled from a noise process $\mathcal{N}$ to the actor policy.

$$\mu' = \mu(s_t|\theta^\mu) + \mathcal{N} \tag{2.25}$$

where $\mathcal{N}$ should be chosen according to the environment, while as we already mentioned in 2.4.2, they propose the Ornstein-Uhlenbeck process, in order to generate temporally correlated exploration for exploration efficiency.

The complete DDPG procedure, as presented in the original paper [34], is provided in Algorithm 5.

Chapter 3

Our Approach

In this chapter, the design of both agent and environment is presented. The implementation of a variety of deep reinforcement learning is also covered, as well as the different setups of agents that were trained.

3.1 The Lane-Free Traffic Environment

As a training environment, we consider a ring-road traffic scenario populated with multiple automated vehicles applying the lane-free driving behavior, as outlined in [7]. Our agent is an additional vehicle that adopts the proposed MDP formulation, learning a policy through observation of the environment. The observational capabilities of our agent includes the position (x, y), speed (v_x, v_y) of nearby vehicles and its own. Both the position and speed are observed as 2-dimensional vectors, consisting of the associated longitudinal (x axis) and lateral (y axis) values. All the observable vehicles share the same dimensions and movement dynamics. Each vehicle selects randomly a desired speed v_d, within a specified range ($[v_{d,min}, v_{d,max}]$), and this information can also be monitored. Our agent controls 2 (continuous) variables, namely the longitudinal and lateral acceleration values (a_x, a_y), and determines the gas/break through a_x, and left/right steering through a_y. Fig. 3.1 illustrates the traffic environment. The examined ring-road scenario is emulated through a highway, by having vehicles reaching the end-point reenter the highway appropriately. Vehicles' observations are adjusted accordingly, so that they observe a ring-road, e.g., vehicles towards the end of the highway observe vehicles in front, located after the highway's starting point.

As mentioned, other vehicles follow the lane-free vehicle movement strategy in [7], which does not involve learning, i.e., other agents follow a deterministic behavior w.r.t. their own surroundings. In addition, we disable the notion of nudging for other vehicles, since when enabled, other vehicles move aside whenever we attempt an overtake maneuver, meaning our agent would learn a very aggressive driving policy.

Figure 3.1: Snapshot of the Lane-Free Traffic environment

3.2 Related Work

Under the lane-free traffic paradigm, multiple vehicle movement strategies [7, 8, 9, 10] have already been proposed, with approaches stemming from Control Theory, Optimal Control and Multi-agent decision making. In more detail, [7] introduces a lane-free vehicle movement strategy based on heuristic rules that involve the notion of "forces" being applied to vehicles, in the sense that vehicles "push" one another so as to overtake, or in general to react appropriately. Now, [9] introduces a policy for lane-free vehicles based on optimal control methods, and more specifically *model predictive control*, where each vehicle optimizes its behavior for a specified future horizon, considering the trajectories of nearby vehicles as well. Furthermore, [10] designs a two-dimensional cruise controller for lane-free traffic, with more emphasis on Control Theory. Finally, [8] tackles the problem with the use of the max-plus algorithm, and constructs a dynamic graph structure of the vehicles, considering communication among vehicles as well. Yet, none of them tackle the problem with RL techniques. In this work, we introduce an alternative movement strategy based on Deep RL, providing various configurations for the reward function.

3.3 State Space

The state space describes the environment to the agent and must contain sufficient information for choosing the appropriate action. Thus, our observation space contains information about both the state of the agent in the environment and the surrounding vehicles. More specifically, regarding the state of the agent, it was deemed necessary to store its lateral position y, as well as both its longitudinal and lateral speed v_x, v_y. On the other hand, as far as the environment and the surrounding vehicles are concerned, matters seem to be more complicated, due to the nature of our problem. In particular, in lane-based experiments, the state space can be defined in a more straightforward manner, as an agent can be trained by utilizing information about the front and back vehicles on its lane, and the position of its previous vehicle on the adjacent lanes, in case it handles lane-changing movement as well. On the contrary, in a lane-free environment, we need a more extensive set of information that depicts our environment in its entirety, as the number of surrounding vehicles in the two dimensional space we consider is varying.

However, our state needs to include information about a predefined number of vehicles, because the MDP formulation does not handle state vectors with varying size. Hence, a

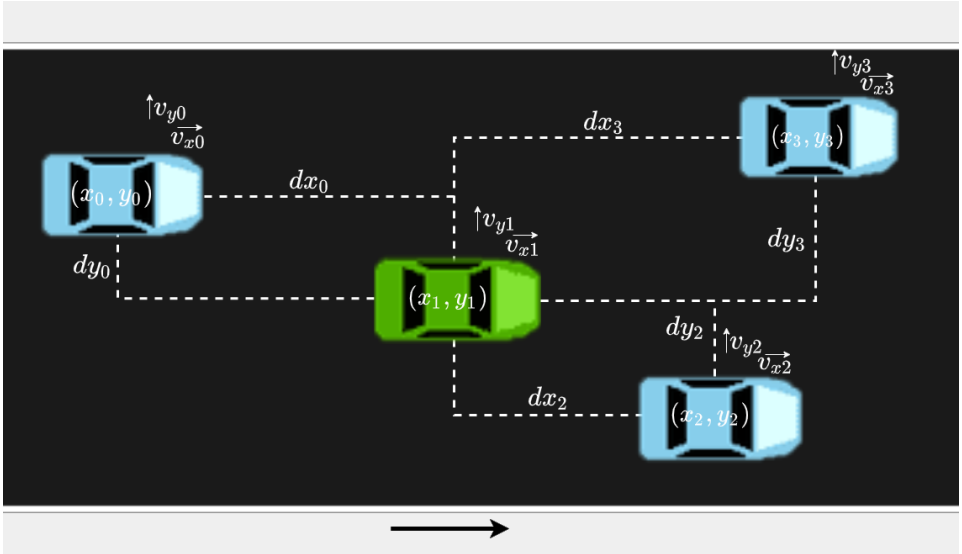


Figure 3.2: The estimation of state space

maximum of n surrounding vehicles are considered in a predefined longitudinal distance d. In case there are more vehicles in the scene, only the n closest vehicles are considered, while a neutral value is included in the state space, in the event that the number of vehicle is fewer than n.

We store information about the speed of the surrounding vehicles, both longitudinal and lateral. Additionally, intending to have a sufficient state representation, we include information regarding the distances of the agent from the cars within the aforementioned perimeter with a range of d meters. In this work, these above-mentioned distances are referred to as dx and dy and result from the distance of the agent's center and that of each neighboring vehicle, as shown in Fig. 3.2. Finally, the state space of our experiments contains our agent's desired speed v_d, alongside its desired lateral position y_d for each time-step, a notion relevant to a component of the reward function, that is discussed in Sec. 3.5.5.

3.4 Action Space

In this work, the primary objective is to find an optimal policy that generates the appropriate high-level driving behavior for the agent to move efficiently in a lane-free environment. Hence, our action space consists of two principal actions: one concerning the car's longitudinal movement by addressing braking and accelerating commands; and one relevant to the lateral movement through acceleration commands for acceleration towards the left or right direction.

Moreover, in the context of this work we investigated both a continuous and a discrete action space. Specifically, DQN and its extensions tend to generate state-action values for

each action, thus we had to use an appropriate set of discrete actions; a discrete action space consists of a finite set of distinct actions [18]. In detail, at each time-step t, the agent can perform one of 9 possible actions:

- a_0: zero acceleration in both axes
- a_1: longitudinal acceleration
- a_2: longitudinal deceleration
- a_3: lateral acceleration towards left
- a_4: lateral acceleration towards right
- a_5: combination of a_1 and a_3
- a_6: combination of a_2 and a_3
- a_7: combination of a_1 and a_4
- a_8: combination of a_2 and a_4

On the other hand, the DDPG method is developed on to environments with continuous action spaces. Therefore, we redesigned our action space into a single continuous real-valued space $\mathbf{a} \in \mathbb{R}^2$, coinciding with the two desired types of actions, a lateral acceleration and a longitudinal acceleration.

3.5 Reward Function Design

The design of the reward function is critical for the performance of (Deep) RL algorithms. Particularly, this is a very pivotal part of the course of this work, since the reward function should represent the desired driving behavior of our agent. It is worth noting that finding an appropriate reward function for this problem proved to be quite arduous due to the novel traffic environment. In particular, as stated, to the best of our knowledge, there is no similar work where we could draw ideas from, as most of the related work in the literature is typically based on the existence of driving lanes (in Sec. 3.2), which constitutes a different problem altogether. For this reason, a reward function was constructed specifically for lane-free environments.

Several components of reward functions were investigated to explore their mechanism of influence, as well as to find the most effective form. Before presenting the various components, we first determine the agent's objectives within the lane-free traffic environment.

The designed reward function should combine the two objectives of our problem, that is, maintaining the desired speed v_d and avoiding collisions with other vehicles. All of the presented reward components attempt to tackle these two objectives. Some are more targeted only towards the end goal, and do not provide the agent with information for

intermediate states, i.e., delayed rewards, while others are more elaborated and informative, and consequently tend to better guide the agent towards the aforementioned goals. Naturally, the more informative rewards aid in the learning process, and for the baseline algorithms examined, we also observe a strong influence in the results. Of course, such informative rewards arguably add bias to the optimization procedure, and provide more specific solutions, compared to delayed rewards, which allow for the agent to explore the whole solution space to obtain a (near) optimal solution to the problem at hand. Yet, this is not necessarily exploited and depends on the algorithm of choice, and its capabilities to harness the whole solution space.

3.5.1 Longitudinal Target

Regarding the desired speed objective, several alternative algorithms were created. One of the earliest was a purely linear algorithm that focuses on maintaining the required speed. In detail, the function is linear with respect to the agent's current longitudinal speed v_x and calculates a reward based on the the deviation from the desired speed v_d of the agent at that specific time-step.

To achieve this, the following mathematical formula is used:

$$r_x = \begin{cases} \frac{v_x}{v_d} & \text{if } v_x \leq v_d; \\ \frac{2 \cdot v_d - v_x}{v_d} & \text{otherwise} \end{cases} \tag{3.1}$$

where r_x refers to the reward component related to the longitudinal desired speed v_d. The reward is normalized with v_d, so as to be bounded within the range of $[0, 1]$.

After several experiments, we observed that this method did not have the expected results, as our agent showed some undesirable and extreme behaviors. As such, we examine an alternative version by adjusting Eq. 3.1, and adding either a positive or negative constant reward of magnitude k_r. In detail, we define a target area (for the longitudinal speed), and if the agent's current speed does not lie within, a positive reward is attributed whenever the agent executes an action that approaches the corresponding area, while in the opposite case, a negative reward is added.

We consider the targeted speed area $v_{range} = [v_d - \epsilon_v, v_d + \epsilon_v]$ to be a range of speeds that allows for a speed deviation from the desired speed up to ϵ_v, i.e., $|v_x - v_d| \leq \epsilon_v$. The corresponding equation is as follows:

$$r_{x,k_r} = \begin{cases} +k_r & \text{if } v_x \notin v_{range} \text{ \& selected action approaches } v_{range}; \\ -k_r & \text{if } v_x \notin v_{range} \text{ \& selected action distances } v_{range}; \\ r_x & \text{if } v_x \in v_{range} \end{cases} \tag{3.2}$$

While the linear form manages to provide the agent with information, we could not find a parameters' setting that results in a policy incorporating both goals sufficiently.

This led to the use of a reciprocal function form instead. We retain the same notion of trying to reach a longitudinal and lateral target (v_d and y_d). As such, for the longitudinal

component, we simply have a cost corresponding to the normalized deviation of the current longitudinal speed v_x from the desired speed v_d:

$$c_x = \frac{|v_x - v_d|}{v_d} \tag{3.3}$$

where v_d is a constant corresponding to the maximum desired speed, so that c_x is bounded within the range $[0, 1]$.

It is evident that this function tends to be minimized at 0 whenever we approach the respective goal. As such, the form of the total reward r_t is a reciprocal function that contains a weighted form of c_x in the denominator. That being the case, we determined that our evaluation function should reverse this, so we select a reciprocal form $(1/x)$, and we put c_x as a denominator.

$$r_t = \frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x} \tag{3.4}$$

where r_t is the total reward at any time-step t, while ϵ_r is a parameter that allows the reward to be maximized at 1 whenever c_x tends to 0. We choose a small value for ϵ_r, specifically $\epsilon_r = 0.1$, so as to make the minimum reward be close to 0 when c_x is maximized.

3.5.2 Overtake Motivation Term

In our preliminary experiments, we determined that our agent tends to get stuck behind slower vehicles, as it is deemed a "safer" action. However, this behavior is not ideal, as it usually leads to a greater deviation from the desired speed. To address this particular problem, we created a function that motivates the agent to overtake its surrounding vehicles.

In detail, a positive reward $c_{overtake}$ is attributed whenever our agent overtakes one of its neighboring vehicle. However, this reward is received only in cases that there are no collisions.

$$r_{t,o} = \begin{cases} r_t + c_{overtake} & \text{if agent does not collide \& overtakes a vehicle;} \\ r_t & \text{otherwise} \end{cases} \tag{3.5}$$

where r_t denotes the total reward function, as described in 3.4.

3.5.3 Collision Avoidance Term

Concerning the collision elimination objective, our first step was to incorporate the collisions into our reward function. In this light, we examined numerous components, with the first being a “simpler” reward, by incorporating the training objective directly into the reward, aiming to “punish” the agent whenever a collision occurs.

This is exclusively based on the collisions between our agent and its surrounding vehicles. Specifically, a negative reward $c_{collide}$ is received whenever a collision occurs. Essentially, provided with a reward r_t according to one of the aforementioned forms, the reward $r_{t,c}$ that the agent receives is calculated as:

$$r_{t,c} = \begin{cases} r_t + c_{collide} & \text{if agent is involved in a collision;} \\ r_t & \text{otherwise} \end{cases} \tag{3.6}$$

However, this imposes the issue of delayed rewards, as our agent only receives a negative reward due to a collision with another vehicle. Especially in our domain of interest, our agent can be in many situations where a collision is inevitable, even many time-steps before the collision actually occurs, depending on the speed of our agent, along with the speed deviation and distance from the colliding vehicle.

3.5.4 Lateral Targets

During the experimental evaluation of the aforementioned methods, we noticed that even though a significant number of collisions seemed to be avoided, there were still some occurring that our agent did not manage to avoid. Therefore, in order to drastically reduce the number of collisions, we followed methods similar to those described in Sec. 3.5.1

Initially, in a similar manner to Eq. 3.1 , we developed a linear reward concerning the lateral movement of the agent, so as to better inform it for overtaking maneuvers.

$$r_y = \begin{cases} \frac{y}{y_d} & \text{if } y \leq y_d; \\ \frac{2 \cdot y_d - y}{y_d} & \text{otherwise} \end{cases} \tag{3.7}$$

where r_y denotes the reward, regarding the lateral movement, y is our agent's lateral position at the time, and y_d describes the desired lateral position. As we later discuss in Sec. 3.5.5, to acquire an appropriate desired lateral position y_d, we created a method that searches for available zones to move towards to, evaluates them, and returns an appropriate lateral point to move to. As such, the total reward r_t concerning the aforementioned objectives is simply a linear combination of Eqs. 3.1, 3.7:

$$r_t = w_x \cdot r_x + w_y \cdot r_y \tag{3.8}$$

By setting the parameters w_x, w_y, we can control the level of influence of each component, and consequently, prioritize its associated goal.

While this linear reward function is quite informative, when the agent is quite afar from its longitudinal and lateral targets, the obtained reward values are negligible. Hence, likewise Eq. 3.2, we incorporate the same notion for the lateral position objective as well. We consider the corresponding target area to coincide with a calculated lateral region, that our agent utilizes to overtake its surrounding vehicles. More details regarding the calculation of this region, that we refer to as a zone, can be found in Sec. 3.5.5.

$$r_{y,k_r} = \begin{cases} +k_r & \text{if } y \notin zone \text{ \& selected action approaches } zone; \\ -k_r & \text{if } y \notin zone \text{ \& selected action distances } zone; \\ r_y & \text{if } y \in zone \end{cases} \tag{3.9}$$

Finally, when utilizing this alternative variant, as depicted in Eqs. 3.2, 3.9, the total reward r_t is again a linear combination aiming to balance the two associated objectives:

$$r_t = w_x \cdot r_{x,k_r} + w_y \cdot r_{y,k_r} \tag{3.10}$$

Moreover, we examine a reciprocal form, as well, for the lateral component, the associated cost is again the normalized deviation of the lateral position y from the desired lateral position y_d:

$$c_y = \frac{|y - y_d|}{w_r} \tag{3.11}$$

where w_r is the width of the road, so that the cost c_y is also bounded within $[0, 1]$. Considering that the vehicle always lies within the road boundaries, the deviation from the desired (lateral) position cannot exceed the width of the road.

In contrast to the linear formulation 3.10, the cost functions tend to be minimized at 0 whenever we approach the respective goal. As such, the form of the total reward r_{t_r} is a reciprocal function that contains a weighted sum of c_x, c_y in the denominator.

$$r_{t_r} = \frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x + w_y \cdot c_y} \tag{3.12}$$

where ϵ_r is a parameter that allows the reward to be maximized at 1 whenever the weighted sum of costs tends to 0. We choose a small value for ϵ_r, specifically $\epsilon_r = 0.1$, so as to make the minimum reward be close to 0 when c_x, c_y are maximized.

This last reward function we developed seems to be promising, as it leads to a faster learning process, where the results improve at a quicker pace. However, it is also a quite informative reward that adds bias to the optimization procedure.

3.5.5 Construction of Overtaking Zones

As mentioned earlier, we constructed an algorithm that estimates an appropriate desired lateral position y_d for our agent to occupy. This serves to provide information for our agent so as to avoid collisions with vehicles downstream, especially if an overtake maneuver is taking place (when vehicles downstream drive slower). For this computation, the space downstream the vehicle is partitioned (with respect to the y axis) into zones, as illustrated in Fig. 3.3. These zones reflect potential regions that the vehicle may choose to drive towards to. Naturally, the lateral space occupied by vehicles in front is discarded, as illustrated in Fig. 3.3. For the remainder zones, we also dismiss ones that our vehicle does not actually fit, e.g., the zone with red color in the figure. When more than one zones are available, we simply select the one closer to our agent, since it will result in a more gradual maneuver. The desired lateral position y_d will be the center point of the chosen zone (the potential choices for y_d are evident in the figure with dashed lines).

The observation range is dynamic for this process, adapting to the longitudinal speed of our agent, and the range is selected through a timegap value t_g. This dictates a longitudinal distance downstream our agent through its longitudinal speed v_x, i.e., the longitudinal

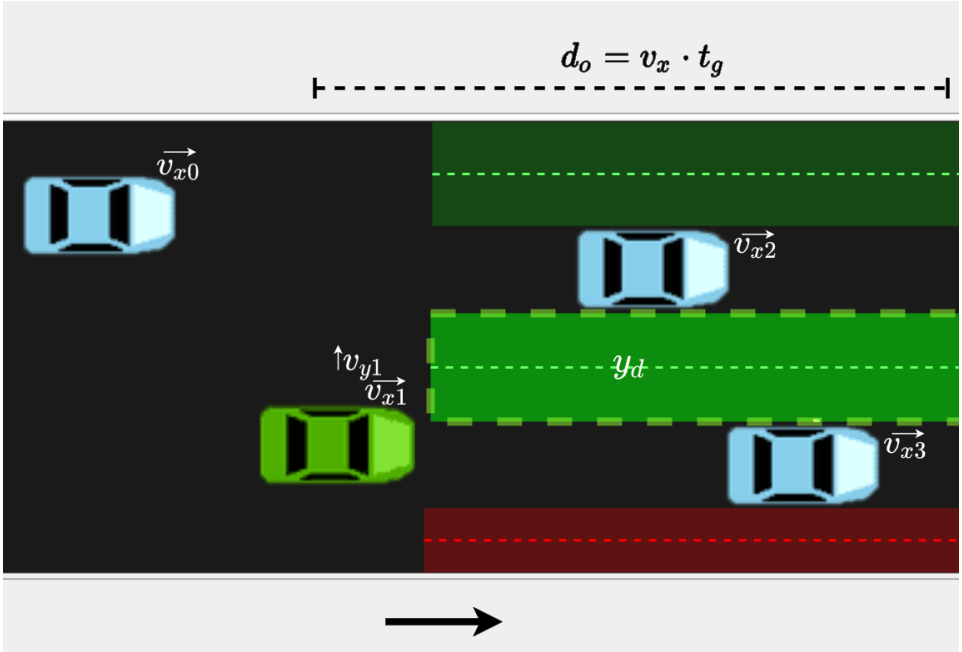


Figure 3.3: Zone Selection Process

distance d_o we scan for vehicles to determine y_d is calculated as $d_o = v_x \cdot t_g$. In case no vehicles are observed within the specified distance d_o, the entire road is considered as a single zone, therefore y_d is the center of the entire road width.

Note that we may also be unable to determine an available zone if vehicles downstream completely block any overtaking maneuver. In that case, it is evident that the methodology outlined above is not appropriate. When a zone cannot be determined due to heavy traffic, then our agent is in any case not able to overtake and forced to remain behind the vehicles in front. Therefore, the desired speed v_d of our agent is adjusted according to the slowest vehicle blocking our agent's way, and the desired lateral position y_d is set behind the fastest blocking vehicle in front. Consequently, our agent will be able to overtake sooner, and with the adjustment in the desired speed, it will not have any motivation to overtake unless it is actually feasible, i.e., without causing a collision.

However, as already discussed in 3.5.4, this particular method is considered too informative, as it imposes a large amount of bias within the learning process and does not prompt the agent to properly explore the environment. This fact is particularly noticeable in the experiments presented in 4.3.4, as the agent does not manage to handle adequately more demanding environments when utilizing a reward function based on this particular method. Thus the final reward function does not include components based on this methodology.

3.5.6 Potential Fields

To tackle the problem of delayed rewards, we also employ an alternative, more informative reward component, one that "quantifies" the danger of collision among two vehicles. The

use of ellipsoid fields has been already utilized for lane-free autonomous driving as a measurement of collision danger with other vehicles [8, 9]. Provided with a pair of vehicles, the form of the ellipsoid functions calculates a utility that evaluates the danger of collision, taking into account the longitudinal and lateral distances, along with the respective longitudinal and lateral speeds of the vehicles and their deviations.

Given our agent and a neighboring vehicle j, with longitudinal and lateral distance dx_j, dy_j, and longitudinal and lateral speed deviation $dv_{x,j}, dv_{y,j}$, the form of the ellipsoid functions is as follows:

$$u_j = E_c(dx_j, dy_j) + E_b(dx_j, dy_j, dv_{x,j}, dv_{y,j}). \tag{3.13}$$

Both $E_c(dx_j, dy_j)$ and $E_b(dx_j, dy_j, dv_{x,j}, dv_{y,j})$ have an ellipsoid function and capture a critical and broad region respectively.

The particular ellipsoid form adopted in this approach is the following [60]:

$$E(d_x, d_y) = \frac{m}{((\frac{|d_x|}{a})^{p_x} + (\frac{|d_y|}{b})^{p_y} + 1)^{p_t}} \tag{3.14}$$

where d_x and d_y are longitudinal and lateral distances, while a and b are regulation parameters for the range of the field for the two dimensions x and y respectively. The exponents marked as p_x,p_y and p_t shape the ellipse and lastly the parameter m defines the magnitude when the distances are close to 0.

Essentially, the critical region is based only on the distance of the two vehicles, while the broader region stretches appropriately according to the speed deviations, so as to properly inform on the danger of collision from a greater distance, and consequently the agent has more time to respond appropriately. The interested reader may refer to [8] for more information on these functions.

Moreover, we also need to accumulate the corresponding values for all neighboring agents, i.e., $u_t = \sum_j u_j$ for each neighboring vehicle j within our state observation at a given time-step t. Finally, to bound the associated reward, we have

$$r_{fields} = \min\{u_t, 1\} \tag{3.15}$$

We know that each ellipsoid function is bounded within $[0, m_u]$, where m_u is a tuning parameter. As such, each utility u_j is bounded within $[0, 2m_u]$. Therefore, m is set accordingly ($m_u = 0.5$), so as to normalize each u_j values to $[0, 1]$. Thus, provided with a reward r_t according to one of the aforementioned variants, the reward $r_{t,fields}$ that the agent receives is calculated as:

$$r_{t,fields} = \frac{\epsilon_r}{\epsilon_r + w_{x_f} \cdot c_x + w_f \cdot r_{fields}} \tag{3.16}$$

Notice that $r_{t,fields} = r_t$ whenever there is no captured danger with neighboring vehicles, i.e., the ellipsoid functions for each neighbor j returns $u_j = 0$.

3.5.7 Putting all together: aiming to find the most efficient and final reward function

To further improve our agents' performance, we form a set of various reward functions by combining some of the previous components.

Seeking a more efficient policy, we further study, at first, reward functions that involve the construction and utilization of the overtaking zones. To be specific, in order to reduce the number of collisions, we add the the Collision Avoidance Term (in Sec. 3.5.3) to the Reciprocal reward function described in Eq. 3.12. We refer to that reward function as "Reciprocal and Collision Avoidance RF".

$$r_{t_r,c} = \begin{cases} \frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x + w_y \cdot c_y} + c_{collision} & \text{if agent is involved in a collision;} \\ r_{t_r} & \text{otherwise} \end{cases} \tag{3.17}$$

On the other hand, in order to tackle the speed objective more effectively, we add the the Overtake Motivation Term (in Sec. 3.5.6) to the reciprocal reward function of Eq. 3.12. We refer to that reward function as "Reciprocal and Fields RF".

$$r_{t_r,f} = \begin{cases} \frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x + w_y \cdot c_y + w_f \cdot r_{fields}} & \text{if there is captured danger with neighboring vehicles;} \\ r_{t_r} & \text{otherwise} \end{cases} \tag{3.18}$$

To combine the above, we merge both previous reward functions in a single reward titled "Reciprocal, Collision Avoidance and fields RF":

$$r_{t_r,o,f} = \begin{cases} \frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x + w_y \cdot c_y + w_f \cdot r_{fields}} + c_{collision} & \text{if agent is involved in a collision} \\ \frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x + w_y \cdot c_y + w_f \cdot r_{fields}} & \text{otherwise} \end{cases} \tag{3.19}$$

However, as evident by the results in 4.3.1, these methods did not improve the agent's performance, and therefore we examine different reward functions that do not include the lateral target components.

Consequently, to tackle both our objectives more efficiently, we combine the reciprocal longitudinal target reward of Eq. 3.4, the Collision Avoidance Term (in Sec. 3.5.3) and the Overtake Motivation Term (in Sec. 3.5.2) components in a single reward function, noted as "Overtake and Collision Avoidance RF", that is calculated as:

$$r_{t,o,c} = \begin{cases} r_t + c_{overtake} & \text{if agent does not collide \& overtakes a vehicle;} \\ r_t + c_{collide} & \text{if agent is involved in a collision;} \\ r_t & \text{otherwise} \end{cases} \tag{3.20}$$

Furthermore, we refer to the assembly of the Potential Fields reward component (in Sec. 3.5.6) of Equation (in Sec. 3.5.6) and the Collision Avoidance reward component (in

Sec. 3.5.3), as "Fields and Collision Avoidance RF". This reward function is designed as demonstrated:

$$r_{t,f,c} = \begin{cases} r_{t,fields} + c_{collide} & \text{if agent is involved in a collision;} \\ r_{t,fields} & \text{otherwise} \end{cases} \tag{3.21}$$

Finally, we combine the Potential Fields (in Sec. 3.5.6), the Collision Avoidance Term (in Sec. 3.5.3) and the Overtake Motivation Term (in Sec. 3.5.2) components in a single reward function, noted as "Fields, Collision Avoidance and Overtake RF", that is calculated as:

$$r_{t,f,c,o} = \begin{cases} r_{t,fields} + c_{collide} & \text{if agent is involved in a collision;} \\ r_{t,fields} + c_{overtake} & \text{if agent does not collide \& overtakes a vehicle ;} \\ r_{t,fields} & \text{otherwise} \end{cases} \tag{3.22}$$

Notice that $r_{t,fields}$ denotes the function described in Eq. 3.16.

As evident from the experiments described in 4.3.2, of the last three functions, the "Fields, Collision Avoidance and Overtake RF" results in the best policy. In addition, this particular reward function proved to be the most effective overall, as the resulting policy manages to tackle both objectives the best, and its performance remains consistent even in environments with varying levels of difficulty, as discussed in 4.3.4.

Chapter 4

Experimental Evaluation

In this section we present: our experimental results through a comparative study of the different reward functions that we propose; various parameter settings that aim to showcase trade-offs between the two objectives; and a comparison between the examined Deep RL algorithms.

4.1 RL Algorithms Setup

First, we specify some technical aspects of our implementation. Regarding the DQN algorithm and its extensions, we employed the Adam [48] optimization method, to update the weight coefficients of the proposed network at each learning step. The epsilon-Greedy policy was employed for action selection, in order to balance exploration and exploitation, with ϵ decreasing linearly from 1 (100% exploration) to 0.1 (10% exploration) over the first 200 episodes, and fixed at 0.1 thereafter. We utilised DQN and its extension with a Deep Neural Network of 128 neurons in the first hidden layer, 64 in the second hidden layer, 9 in the output layer, while using Rectified Linear Unit (ReLU) and Linear activation.

In the setup for the DDPG algorithm, we also used Adam, to minimize the loss function of the actor and critic. Furthermore, we chose to use Ornstein-Uhlenbeck Process to add noise (as an exploration term) to the action output, as proposed in the original paper [34]. The actor neural network employed in the DDPG implementation, contains 256 neurons in the first hidden layer, 128 in the second hidden layer and 2 in the output layer. Again, ReLU activation function is used for all hidden layers, while the output uses the hyperbolic tangent (tanH) activation function, so as to provide a vector of continuous values within the range $[-1, 1]$. Similarly, the critic neural network utilises 256 neurons in the first hidden layer, 128 in the second hidden layer and 1 neuron in the output layer with ReLU activation function for all hidden layers, and linear activation at the output layer.

We trained for a total of 625 episodes across all experiments. The hyper-parameters used are provided in Table 4.1. We empirically examined different parameter tunings, and selected the ones that provide the best results for each RL algorithm.

Parameter	Value
Learning rate used by Adam α	0.001
Mini-Batch size	64
Discount factor (DQN)	0.98
Discount factor (DDPG)	0.98
Replay Memory size (DQN)	50000
Replay Memory size (DDPG)	100000
Soft update parameter (DDPG)	0.001
Number of episodes for training N	625

Table 4.1: Hyper-parameters for RL algorithms

4.2 Simulation Setup

Our implementation heavily relies on ANNs, since all DRL methods incorporate them for function approximation. As such, in the context of this work we utilise:

- ***TensorFlow*** [61]: is an open-source platform, used for numerical computation, machine learning and artificial intelligence applications, that was developed by Google. It supports commonly used languages, such as Python and R, and also makes developing neural networks faster and easier.

- ***Keras*** [62]: is a high level, open-source software library for implementing ANNs. In addition, Keras facilitates multiple backend neural network computations, while providing a Python frontend and being complementary to the TensorFlow library.

We train and evaluate our methods on a lane-free extension of the Flow [63] simulation tool, as described in [8]. Moreover, to facilitate our experiments, we utilized the Keras-RL library [64]. The Keras-RL library implements some of the most widely used deep reinforcement learning algorithms in Python and seamlessly integrates with Tensorflow and Keras. However, technical adjustments and modifications were necessary to make this library compatible with our problem and environment.

Furthermore, the proposed lane-free driving behavior decision-making model was tested in the highway scenario with the specified parameter choices of Table 4.2, whereas in Table 4.3, we provide the parameter settings related to the MDP formulation.

Parameter	Value
Highway length	500 m
Highway width	10.2 m
Vehicles' length	3.5 m
Vehicles' width	1.8 m
Types of vehicles	2
Num. of vehicles	35
Agent's length	3.2 m
Agent's width	1.6 m
Execution Time	200 s
Time-Interval	0.25 s

Table 4.2: Simulation Parameters

Parameter	Value
timegap t_g	0.7
agent's desired speed v_d	20 m/s
other vehicles' desired speed	18 m/s to 22 m/s
num. of vehicles in state n	5
longitudinal observation distance d	80 m
w_y	0.65
w_f	1
w_x (for RFs with lateral target)	0.35
w_x (for RFs without lateral target)	0.65
k_r	0.5
$c_{collide}$	-2.5
$c_{overtake}$	2
e_v	0.4

Table 4.3: Parameter choices related to MDP formulation

4.3 Results and Analysis

As discussed in Sec. 3.5, we proposed several reward components and their effectiveness is evaluated based on three metrics. These are: the average reward value, the speed deviation from the desired speed (for each step, we measure the deviation of the current longitudinal speed from the desired one $(v_x - v_d)$, in m/s), and of course, the average number of collisions. All results are averaged from 10 different runs.

Moreover, we have to clarify that we obtained the majority of the showcased results (Figs. 4.1–4.15) with DDPG algorithm since, as we discuss later (Figs. 4.16–4.21), it exhibits the best overall performance. We typically demonstrate in our figures our agent's average reward, speed deviation, and the average number of collisions for each episode respectively.

4.3.1 Longitudinal and Lateral Targets with Other Components

In Figs. 4.1, 4.2 and 4.3 we provide the results for a variety of reward functions that serve to evaluate both the Longitudinal and lateral reward components. In each figure, there are four curves that represent the performance of the proposed reward functions. Specifically, the Longitudinal and Lateral Desired Targets reward functions (Eq. 3.8 in Sec. 3.5.1 and 3.5.4) are depicted as 'Linear Reward Function', while the second variant of the total linear reward, i.e., Eq. 3.10, is labeled as 'Linear v2 Reward Function '. Moreover, the reward function described in Eq. 3.12 is labeled as 'Reciprocal Reward Function'.

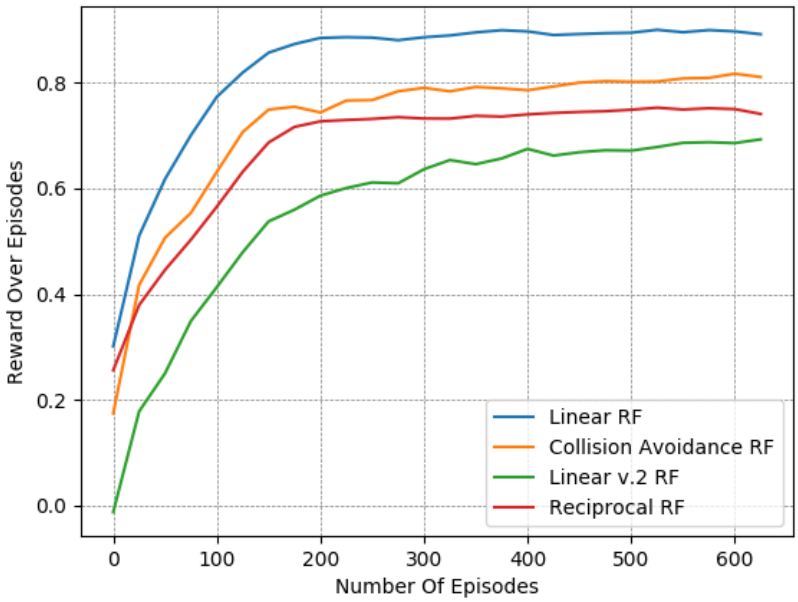


Figure 4.1: Reward over time for different reward functions using Lateral Target

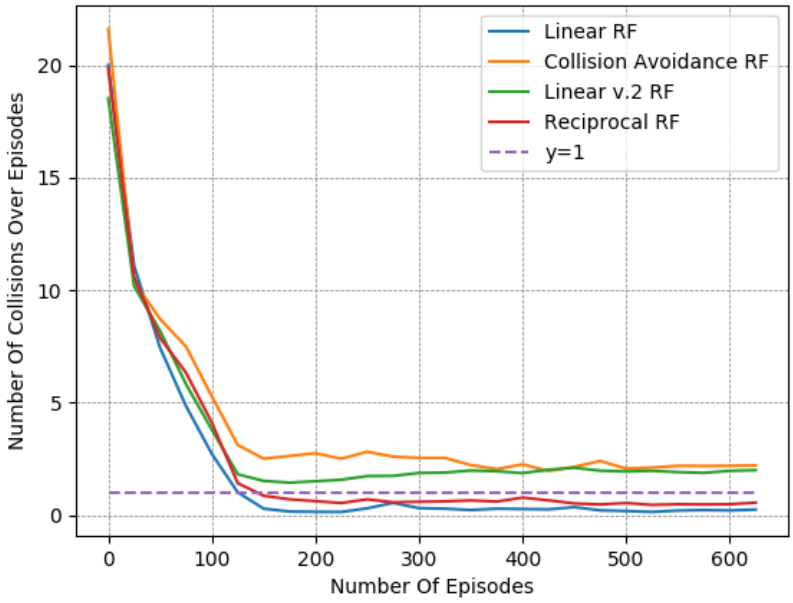


Figure 4.2: Collisions over time for different reward functions using Lateral Target

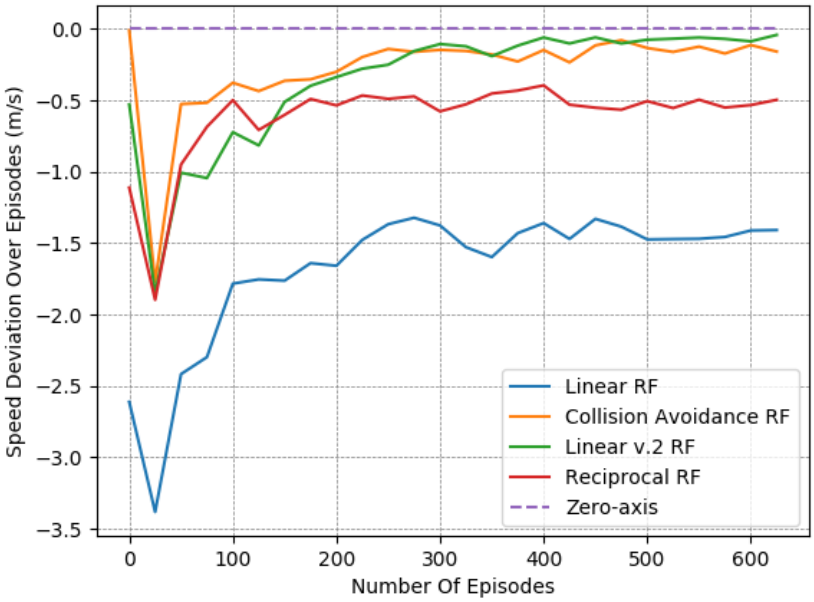


Figure 4.3: Speed Deviation over time for different reward functions using Lateral Target

Additionally, we wish to motivate the use of zones instead of a more simplistic approach

by including one more variant of the reward function for comparison. We refer to the assembly of just the Collision Avoidance Term (without the inclusion of fields) (in Sec. 3.5.3) and the Reciprocal function, but only with the longitudinal (c_x) cost (Eq. 3.3 in Sec. 3.5.1) as 'Collision Avoidance RF'. This particular reward function is mathematically described in Eq. 3.6 As such, the agent receives a positive reward regarding its desired speed, and only a negative reward of magnitude $c_{collide}$ whenever a collision occurs, without any other intermediate value for this objective. The weighting coefficients for each reward function are set according to Table 4.3. As we will later discuss, the reported configuration provided the best results. Note that collision avoidance is of higher priority compared to keeping the desired speed.

All of the aforementioned functions showcase an improvement of the agent's policy over time. However, each of them exhibits different behavior. First, the 'Linear Reward' manages to avoid most collisions, at the expense of maintaining noticeably lower longitudinal speed than the desired one, as indicated by the negative values in Fig. 4.3. On the contrary, the second variant of Linear rewards, 'Linear Reward v2', provides notably a better policy regarding the longitudinal desired speed. Yet, it does not perform sufficiently on collision avoidance. Moreover, while the 'Reciprocal Reward' has a similar performance with the 'Linear Reward' in terms of collision avoidance, it obtains significantly better results in terms of speed deviations, therefore balancing the two objectives much better, making it the prevalent choice for a more effective policy overall. Finally, the additional variant tested, 'Collision Avoidance Reward', is by far the worst in terms of collision avoidance, and it shows that the agent is only focused on the desired speed objective. As such, the use of the lateral target we introduced, i.e., the more informative reward, strongly benefits the resulting behaviour of our agent.

In Table 4.4 ,we present a detailed comparison between the above methods, of the average collision number and the average speed deviation from the desired speed , during the course of the last 50 episodes. This table verifies that the 'Reciprocal RF' is the most effective so far, as it is the only one that significantly reduces the number of collisions to around 0.5, while at the same time maintaining a speed that nears the objective. It is obvious, that the speed deviation has a significant impact on the number of collisions, a case that was noticeable during the most of our experimentation.

Evidently, higher rewards do not coincide with less collisions, meaning that the reward metric should not be taken at face value, especially since we compare different reward functions. This is particularly noticeable in the case of the reciprocal reward function, where there is a reduced reward over the episodes, but when observing each objective, it

Function	Linear v1	Linear v2	Collisions	Reciprocal
Collisions	0.233	1.98	2.206	0.526
Speed Dev. (m/s)	-1.409	-0.064	-0.135	-0.514

Table 4.4: Comparing different Reward Functions using Lateral Target

clearly showcases the best performance. This is expected, since reciprocal reward has a different form entirely. In the subsequent experiments, we omit commenting the report of the average reward, and only take into consideration the results regarding the two objectives of interest. Nevertheless, we adhere to demonstrating them, so as to prove the general learning improvement over episodes.

Furthermore, we must point out that a slight deviation from the desired speed in our experiments is to be expected. Maintaining the desired speed throughout an episode is not realistic. Our agent may occasionally get stuck behind slower vehicles, and then wait until the conditions allow for an overtake.

As mentioned, the most promising reward function form at this point is apparently the Reciprocal one. Therefore, we further investigate it, by examining two additional components, namely Collision avoidance term and Potential Fields term, as presented in 3.5.3 and 3.5.6 respectively. In particular, we present in Fig. 4.4, 4.5 and 4.6 a detailed comparison between: the Reciprocal function in Eq. 3.12; the Reciprocal function with the Collision avoidance term in Eq. 3.17; the Reciprocal function with the potential fields term in Eq. 3.16; and the Reciprocal Function with both components in Eq. 3.18.

Moreover in Table 4.5 we provide a closer look to the comparison between these 4 reward functions. The reported results are averaged from the last 50 episodes of each variant, where the learned policy has converged in all cases. In these results, we observe that just the addition of the Collision avoidance term ('&Coll.') does not improve performance, as it manages to moderately mitigate collision occurrences, at the expense of the desired

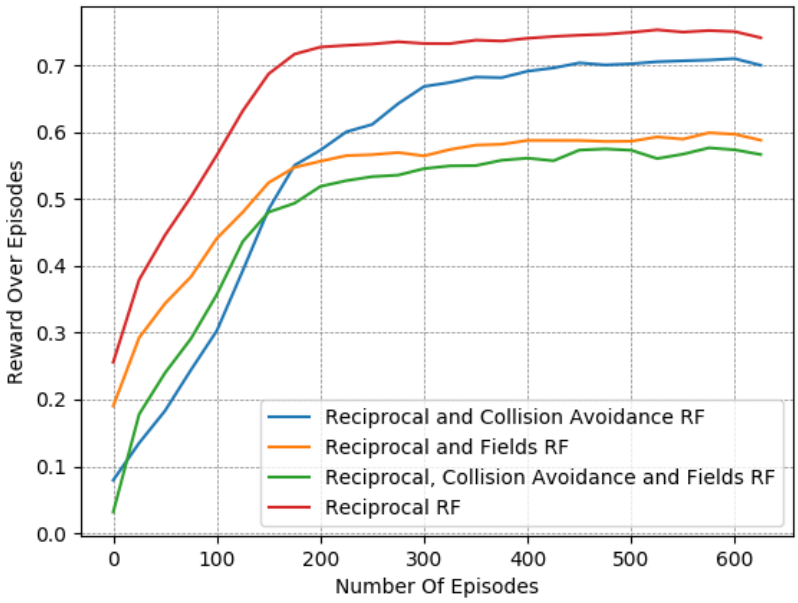


Figure 4.4: Reward over time for different forms of Reciprocal RF

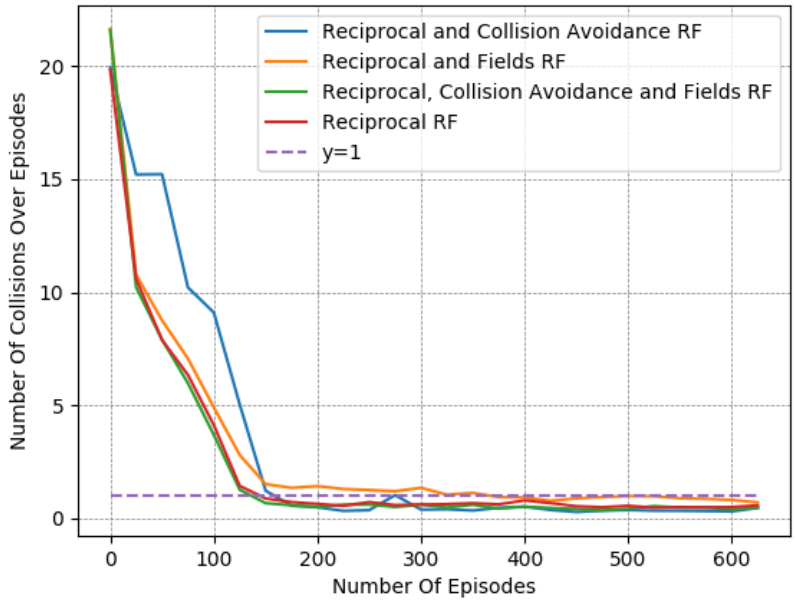


Figure 4.5: Collisions over time for different forms of Reciprocal RF

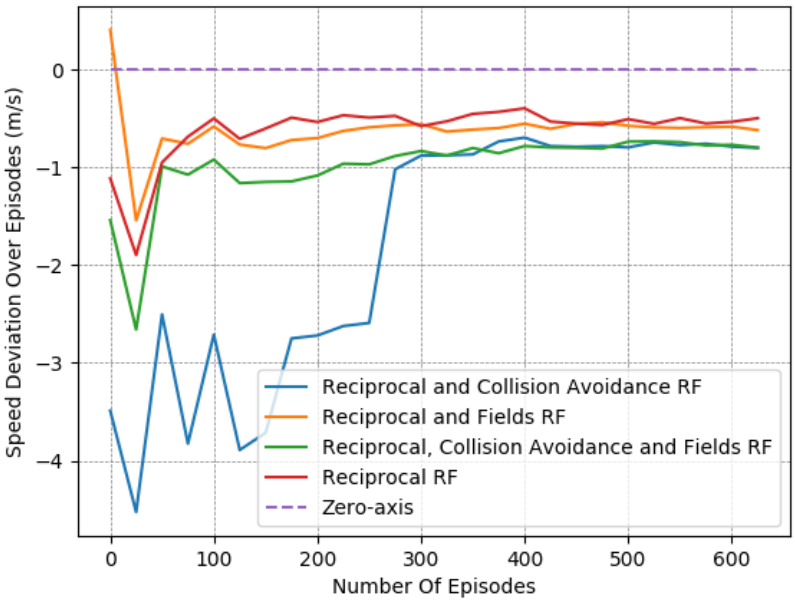


Figure 4.6: Speed Deviation over time for different forms of Reciprocal RF

Function	Rec.	&Coll.	&Fields	&Coll.&Fields
Collisions	0.526	0.343	0.928	0.646
Speed Dev. (m/s)	-0.514	-0.819	-0.279	-0.438

Table 4.5: Comparing the different forms of Reciprocal RF

speed objective. Simultaneously, when combined with the fields reward ('&Coll. &Fields'), it actually achieves maintaining the desired speed, but fails to decrease the number of collisions. Nonetheless, the agent's behavior deviates from the goal, when just integrating the field component ('&Fields').

Throughout our experiments, it is obvious that the two objectives are countering each other since a vehicle operating with slower speed is more conservative, while a vehicle wishing to maintain higher speed than its neighbors needs to overtake in a safe manner, and consequently learn a more complex policy that performs such elaborate maneuvering.

This remark is also visible in Figs. 4.7, 4.8 and 4.9, as well as, in Table 4.6, which contain information about running the 'Reciprocal Reward Function' with different weights of the collision and speed cost components, attempting to find a more balanced solution.

The expected trade-off between the two objectives is better highlighted here, as the weights clearly dictate a preference towards one goal or another, and intermediate values attempt to balance both. The 'safe' policy (with $(w_x = 0.05, w_y = 0.95)$), almost com-

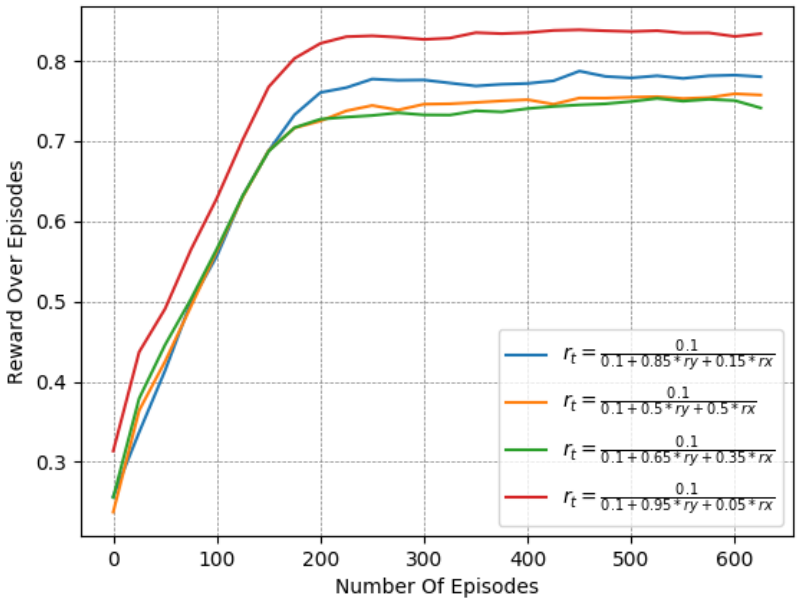


Figure 4.7: Reward over time for the Reciprocal RF with different weights

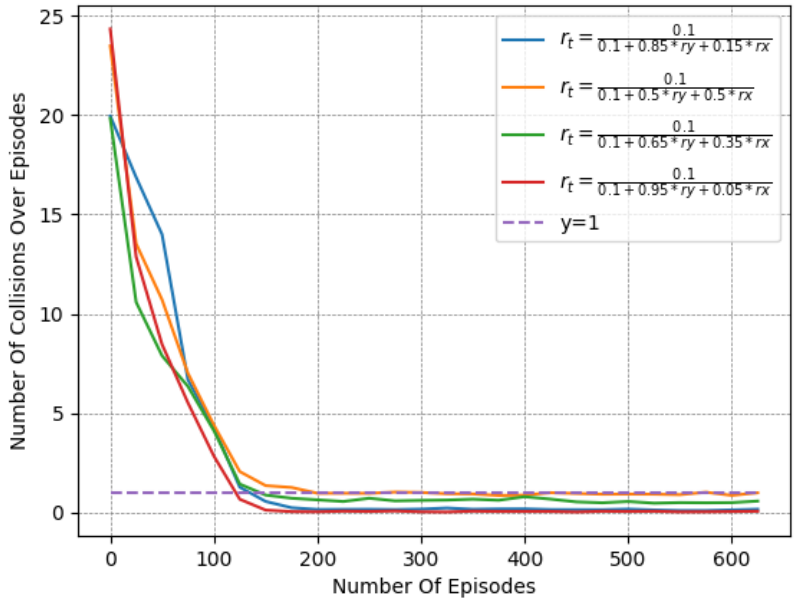


Figure 4.8: Collisions over time for the Reciprocal RF with different weights

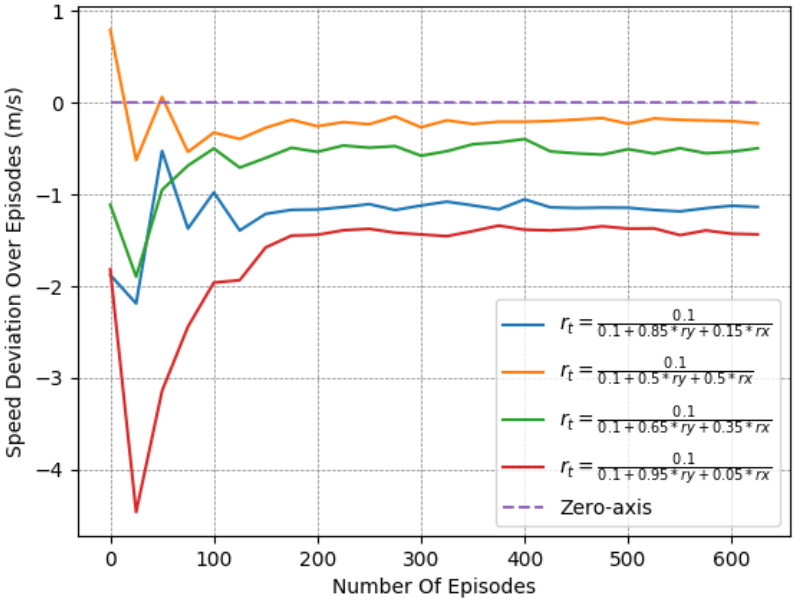


Figure 4.9: Speed Deviation over time for the Reciprocal RF with different weights

$\frac{\epsilon_r}{\epsilon_r + w_x \cdot c_x + w_y \cdot c_y}$	$w_y = 0.5$ $w_x = 0.5$	$w_y = 0.65$ $w_x = 0.35$	$w_y = 0.85$ $w_x = 0.15$	$w_y = 0.95$ $w_x = 0.05$
Collisions	0.917	**0.526**	0.137	0.046
Speed Dev. (m/s)	-0.211	$\mathbf{-0.514}$	-1.129	-1.431

Table 4.6: Comparing different weights of the collision and speed cost components for Reciprocal Reward Function.

pletely minimizes collisions, as evident by the result, but also completely disregards the desired speed goal. Essentially, the policy results in a simplistic behavior, as it guides the vehicle to never overtake, and follow the leading vehicles. As such, the vehicle with higher desired speed inevitably faces at some point slower vehicles, and simply adheres to their lower speed. By contrast, when giving more emphasis on speed, vehicle drives more aggressively, also causing collisions more often. Now, as mentioned already, we give higher priority to collision avoidance, but we require the policy to balance both objectives, therefore the one that gives a smaller preference towards safety is the most preferred ($w_x = 0.35, w_y = 0.65$).

4.3.2 Longitudinal Target with Fields and Overtake RFs

However, the lateral target imposes a lot of bias within the algorithm, as it provides an explicit reward signal that guides the agent to an appropriate zone. While this manages to provide overall good results, it is also both (i) a limitation, since it does not allow the agent to explore the environment appropriately, i.e., adds bias, and (ii) is arguably "too informative" in the sense that this reward component's information is too specific, more complex in its design, and requires certain computational effort (see Sec. 3.5.5). Moreover, the use of the lateral reward component does not generalize well under more intense traffic conditions, as we later showcase in Sec. 4.3.4. As such, we now examine our agent without the use of the lateral target, in order to have the notion of lateral placement as an implicit learning task and thus to improve the agent's performance.

Therefore, in Figs. 4.10, 4.11 and 4.12 we demonstrate our agent's performance, when utilising different reward functions. In each of these figures, there are four curves that depict the performance of the proposed reward functions. Specifically, in each examined reward function, the Longitudinal Target reward 3.4 is combined with an associated component to tackle the collision avoidance objective. We refer to the reward associated with the Collision Avoidance Term (Eq. 3.6) and the addition of the overtaking motivation (Eq. 3.5), as 'Overtake and Avoid Collision Reward Function'. Furthermore, the use of the fields (Eq. 3.16) for that objective is labeled as 'Fields Reward Function' and 'Fields and Avoid Collision Reward Function' when combined with the Collision Avoidance Term. Finally, the assembly of all components in a single reward function (Eq. 3.22) is referred to as 'Fields, Overtake and Avoid Collision Reward Function'. All of the aforementioned methods showcase an improvement of the agent's policy over time, nonetheless, they do not exhibit

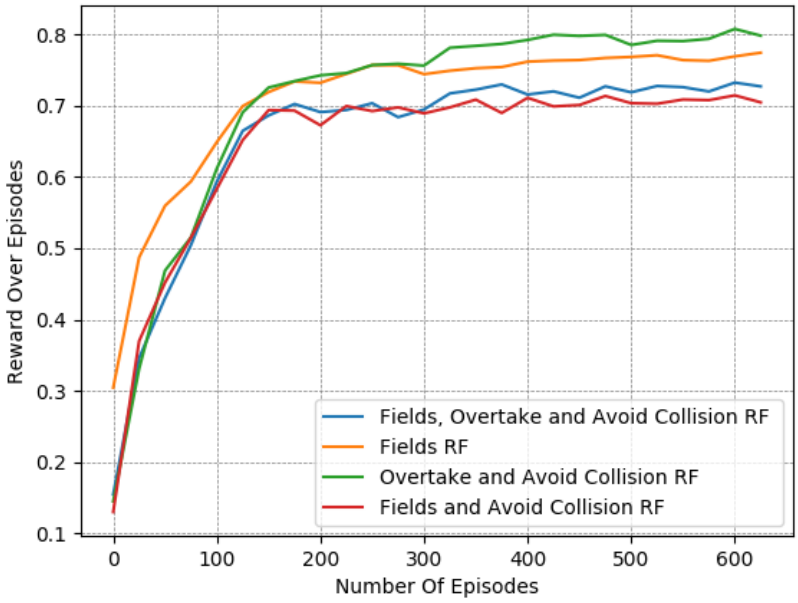


Figure 4.10: Reward over time for different reward functions

similar learning behavior.

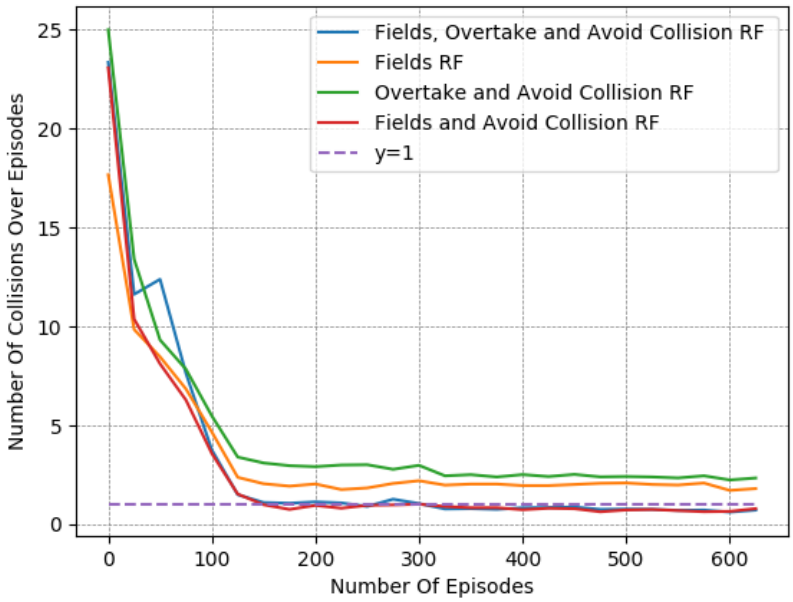


Figure 4.11: Collisions over time for different functions

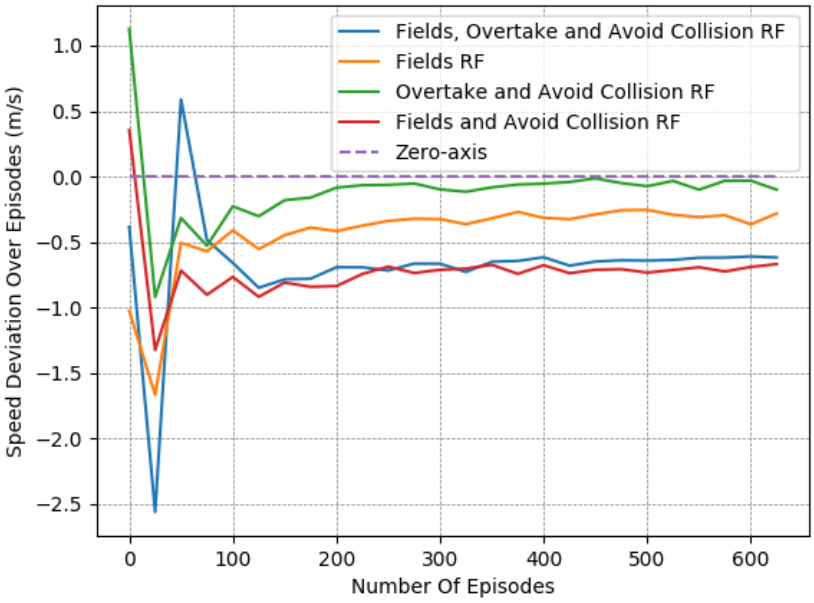


Figure 4.12: Speed Deviation over time for different functions

As stated previously, the 'Collision Avoidance Reward Function' manages to maintain a longitudinal speed close to the desired one. Yet, it does not manage to decrease the number of collisions sufficiently. Moreover, as evident in Figs. 4.10, 4.11 and 4.12 the addition of the overtaking component, in 'Overtake and Avoid Collision Reward Function', achieves a longitudinal speed slightly closer to the desired one, while the collision number is still relatively high. On the contrary according to the same figures, the 'Fields Reward Function' exhibits a similar behavior to the previous mentioned reward functions, with a slight improvement on collision occurrences. Finally, both the 'Fields and Avoid Collisions Reward Function' and the 'Fields, Overtake and Avoid Collision Reward Function' perform slightly worse in terms of speed deviations. However, they obtain significantly better results in terms of collision avoidance, therefore balancing the two objectives much better. On closer inspection though, the 'Fields, Overtake and Avoid Collision Reward Function' manages to maintain a smaller speed deviation and number of collisions, thus making it the prevalent choice for a more effective policy overall.

To further demonstrate this point, we present in Table 4.7 a detailed comparison between these 4 reward functions. The reported results are averaged from the last 50 episodes of each variant, where the learned policy has converged in all cases, as noticeable in Figs. 4.10, 4.11 and 4.12.

In addition, we must point out that during our experimentation process, we tested a variety of weight coefficients to culminate in the most efficient reward function. In Figs. 4.13, 4.14, and 4.15 we demonstrate this experimental process. From top to bottom, we showcase and compare the performance of alternative versions of the "Fields, Overtake

Function	&Fields	&Coll.&Fields	&Coll.&Overtake	&Fields &Coll.&Overtake
Collisions	1.761	0.726	2.293	0.649
Speed Dev. (m/s)	-0.323	-0.698	-0.058	-0.613

Table 4.7: Comparing the different rewards

and Avoid Collision RF".

According to the showcased curves, it is evident that it is quite an arduous task to find the perfect balance between all those components to tackle the problem at hand efficiently. Once more, the fact that the two objectives are countering each other is noticeable. Specifically, we do notice that the experiments with the lowest speed deviation are those with the highest number of collisions, while on the other hand, those that showcase a low number of collisions deviate the most from the desired speed.

To demonstrate this concern, in Table 4.8 we provide a closer look on the averaged outcomes of the last 50 runs for each of these curves. According to the showcased results, it is apparent that the selected configuration ($w_f = 1$, $w_x = 0.65$, $c_{collisions} = 2.5$, $c_{overtake} = 2$) tackles the problem at hand most efficiently.

The following results also support the claim that policies, which minimize collision avoidance, exhibit a very simplistic behavior where the learned agent just follows the

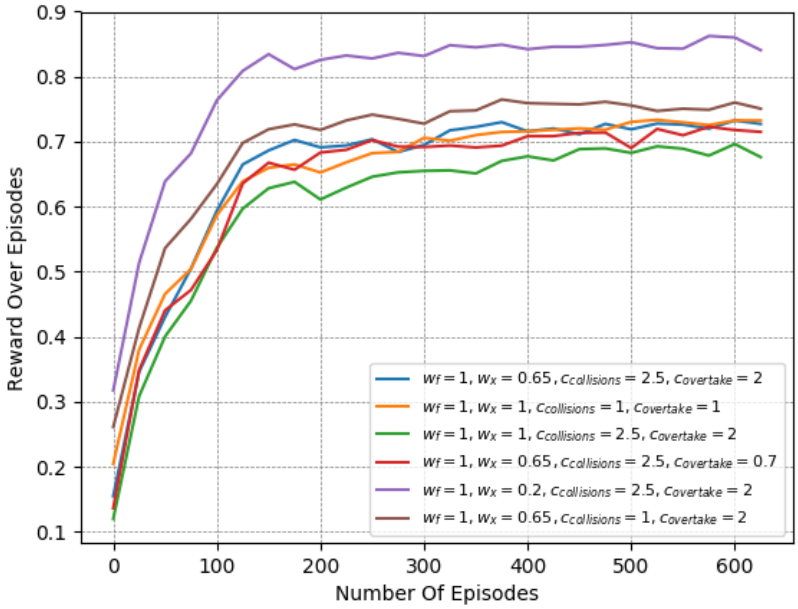


Figure 4.13: Reward over time for the Fields, Overtake and Avoid Collision RF with different weights

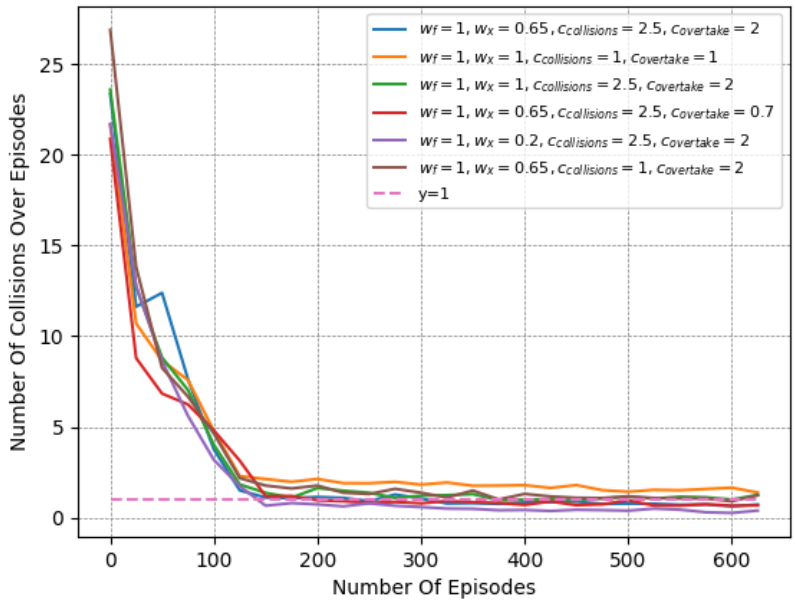


Figure 4.14: Collisions over time for the Fields, Overtake and Avoid Collision RF with different weights

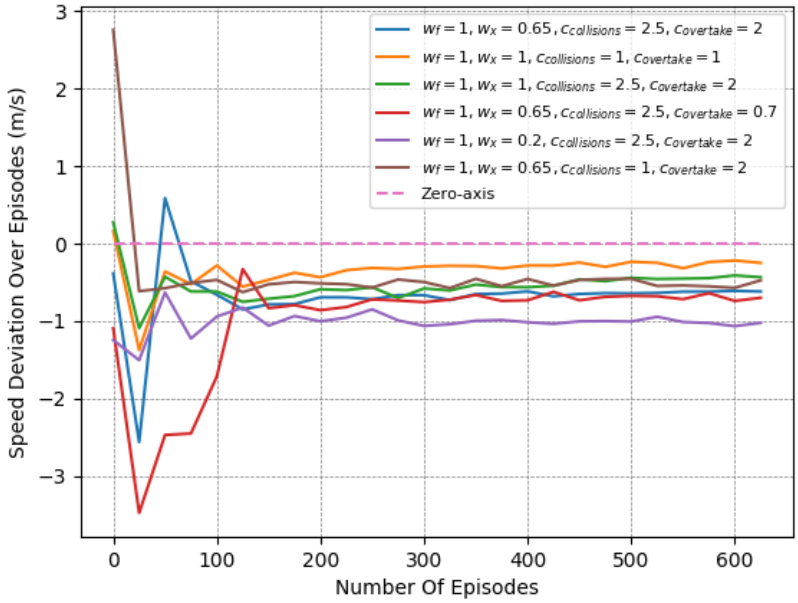


Figure 4.15: Speed Deviation over time for the Fields, Overtake and Avoid Collision RF with different weights

w_f	1	1	1	1	1	1
w_x	0.65	1	1	0.65	0.2	0.65
$c_{collisions}$	2.5	1	2.5	2.5	2.5	1
$c_{overtake}$	2	1	2	0.7	2	2
Collisions	0.669	1.51	1.12	0.664	0.322	1.0625
Speed Dev. (m/s)	-0.613	-0.232	-0.421	-0.718	-1.042	-0.521

Table 4.8: Comparing different weights of the collision and speed cost components for Fields, Overtake and Avoid Collision Reward Function.

speed of a slower moving vehicle in front, i.e., is too defensive and never attempts overtake. Therefore, the associated reward components are not well-fitted for our training objectives even if they showcase better results on collision avoidance only.

4.3.3 Comparison of Different DRL algorithms

The following set of experiment compares different Deep RL algorithms, in Figs. 4.16, 4.17, 4.18 we employ our Reciprocal Reward, using DQN, Double DQN, DQN with Dueling Architectures and DDQN with Prioritized Experience Replay and compare their performance to that of DDPG. It is evident that all five learning based methods make the agent learn the expected behaviour to an extend. However, DDPG seems to have the best performance, as it is the only method that decreased the number of collisions to under 1 on average, while managing to preserve a speed that is near the desired one.

Upon closer examination, by observing the averaged results extracted from the last episodes of each variant, as presented in Table 4.9, DQN, DDQN and DNA result to a smaller speed deviation, yet they exhibit a higher number of collisions.

In detail, although the discrete methods, and especially the use of PER, graphically show a significant reduction in the number of collisions, they worsen the overall result, as this reduction is due to the excessive drop in our agent's speed.

To further prove our point, in Figs. 4.19, 4.20 and 4.21, we compare the performance of DQN, Double DQN, DQN with Dueling Architectures, and DDQN with Prioritized Experience Replay to that of DDPG, while utilizing the 'All Components RF'.

On closer inspection, as shown in Table 4.10 the compared DRL algorithms result to a slighter speed deviation, with DQN touching -0.5, DDQN -0.27 , DNA -0.33 and PER

DRL Algorithms	DDPG	DQN.	DDQN	DNA	PER
Collisions	0.526	2.017	2.081	1.822	0.696
Speed Dev. (m/s)	-0.514	-0.406	-0.382	-0.424	-0.975

Table 4.9: Comparing different Deep Reinforcement Learning Algorithms, using Reciprocal Reward Function.

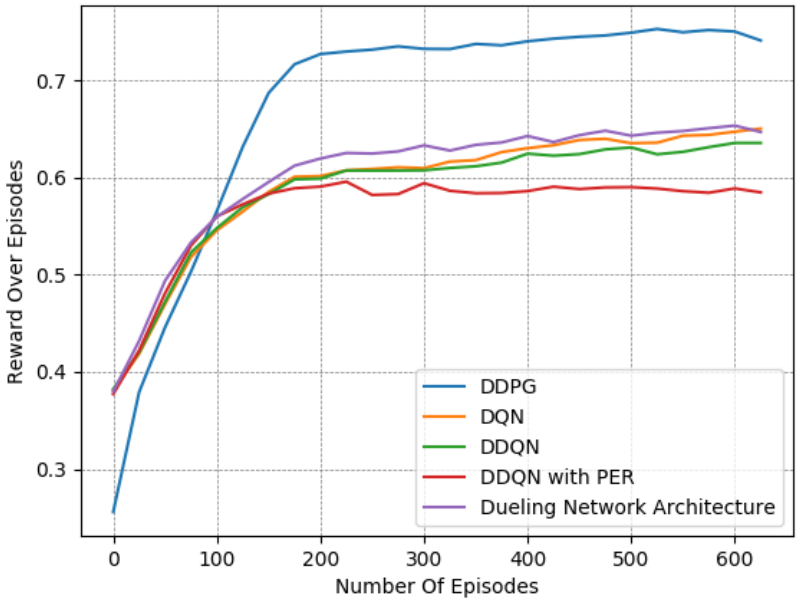


Figure 4.16: Reward over time for the Reciprocal RF with different DRL algorithms

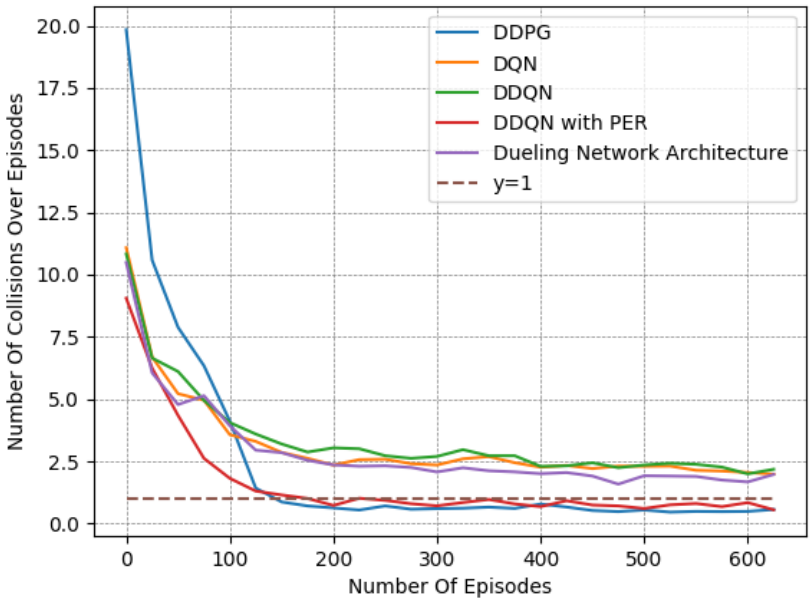


Figure 4.17: Collisions over time for the Reciprocal RF with different DRL algorithms

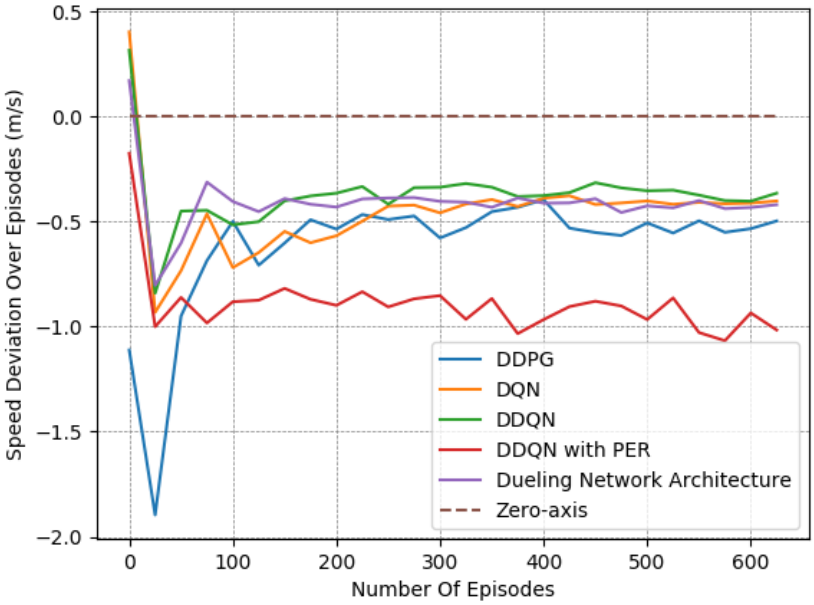


Figure 4.18: Speed Deviation over time for the Reciprocal RF with different DRL algorithms

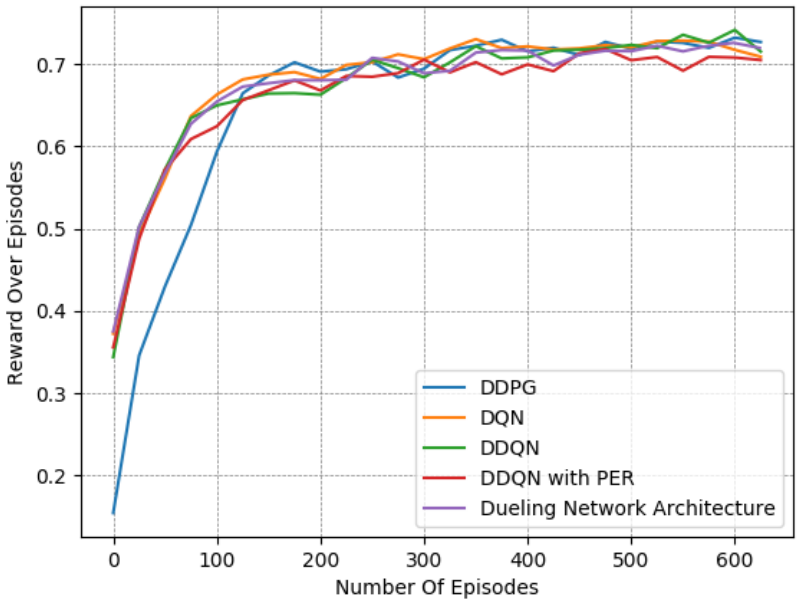


Figure 4.19: Reward over time for the Fields, Overtake and Avoid Collision RF with different DRL algorithms

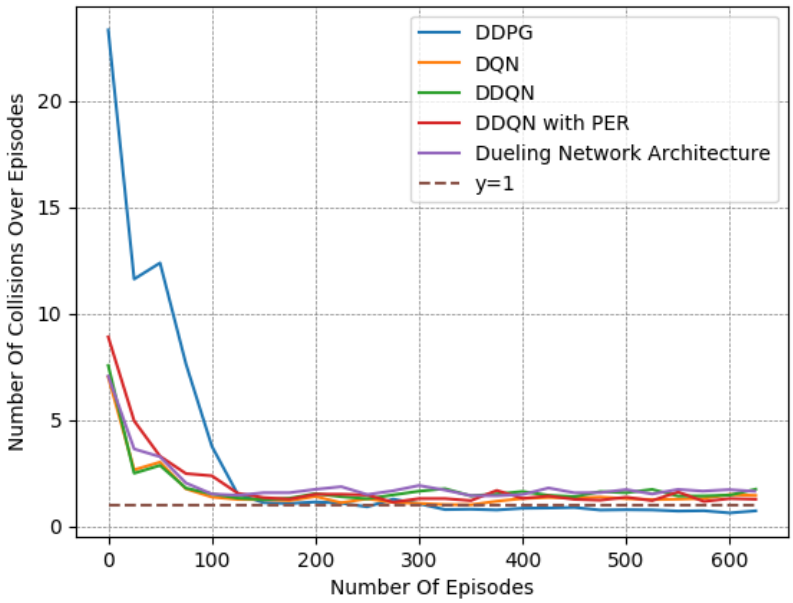


Figure 4.20: Collisions over time for the Fields, Overtake and Avoid Collision RF with different DRL algorithms

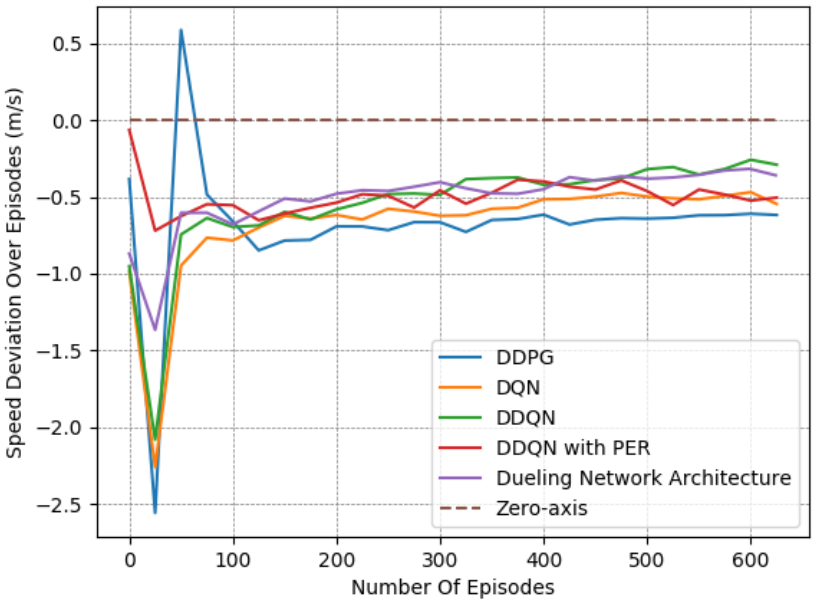


Figure 4.21: Speed Deviation over time for the Fields, Overtake and Avoid Collision RF with different DRL algorithms

DRL Algorithms	DDPG	DQN.	DDQN	DNA	PER
Collisions	0.669	1.462	1.598	1.678	1.280
Speed Dev. (m/s)	-0.613	-0.507	-0.273	-0.337	-0.514

Table 4.10: Comparing different Deep Reinforcement Learning Algorithms, using Fields, Overtake and Avoid Collision Reward Function.

-0.51. However, they also exhibit a higher number of collisions, in detail during the last 50 episodes DQN exhibits an average of 1.46 collisions per episode, DDQN an average of 1.59 collisions, DNA an average of 1.67 collisions per episode, PER an an average of 1.26 collisions, while as we already mentioned DDPG outperforms them and displays 0.66 collisions per episode.

It is apparent that DDPG tackles our problem more efficiently, as policy gradients can be easily applied to model continuous action spaces, since their policy network is designed to model probability distribution. Meanwhile, DQN requires discretizing the action space, which rarely leads to the ideal solution. Moreover, we attribute its improved performance to the complexity of our reward function and training environment, as DDPG tends to outperform DQN in such cases.

4.3.4 Evaluation for different Traffic Densities

Finally, to obtain more comprehensive and thorough results, we decided to test the 2 most promising reward functions in more complex and demanding lane-free environments. We chose to run both the 'Reciprocal RF' and the 'Fields, Overtake and Avoid Collision RF', using a set of different traffic densities. Specifically, in Figs. 4.22, 4.23 and 4.24 we illustrate the results of running the Reciprocal RF, using densities equal to 70, 90, and even 120. Meanwhile, in figures s 4.25, 4.26 and 4.27 we demonstrate the corresponding outcomes, when running the Fields, Overtake and Avoid Collision RF for the same set of traffic densities.

The above figures prove our claim for the unsuitability of the Reciprocal Reward Function. We observe that when the density value is set to 70, then the agent showcases the best overall policy. However, that is not the case when the difficulty of the environment increases. In particular, in environments with higher densities, the agent chooses to focus on maintaining the desired speed and disregards completely the task of avoiding collisions. This fact is mainly confirmed by the experiment that density is set to 120, as by the end the number of collisions is close to 6.

On the contrary, we observe that when using the 'Fields, Overtake and Avoid Collision Reward Function', the agent tends to handle the surrounding traffic quite well. In detail, it is noticed that the number of collisions decreases dramatically with the passage of the episodes, while, in all cases, at the end of the training it approaches or falls below 1. At the same time, the collisions and the difference in the agent's speed from the desired one are proportional to the density of the surrounding vehicles. However, this behavior is expected

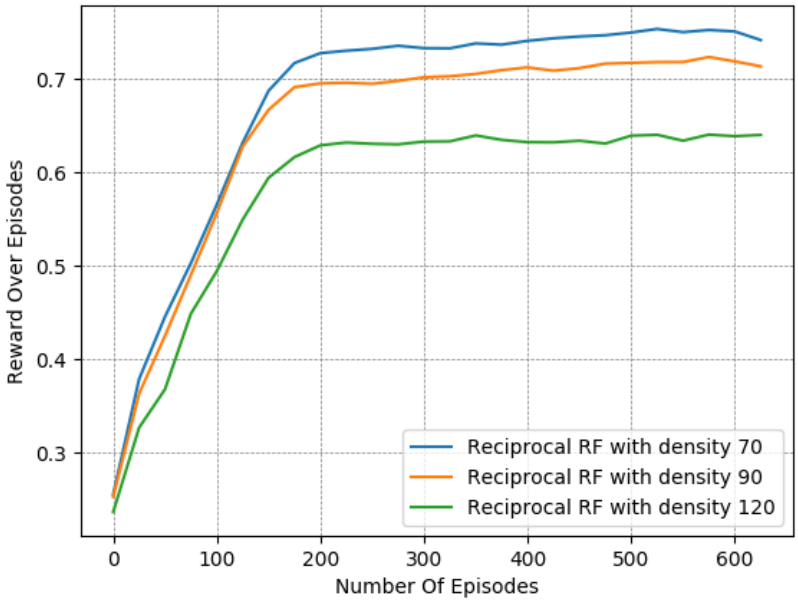


Figure 4.22: Reward over time for the Reciprocal Reward Function for different densities

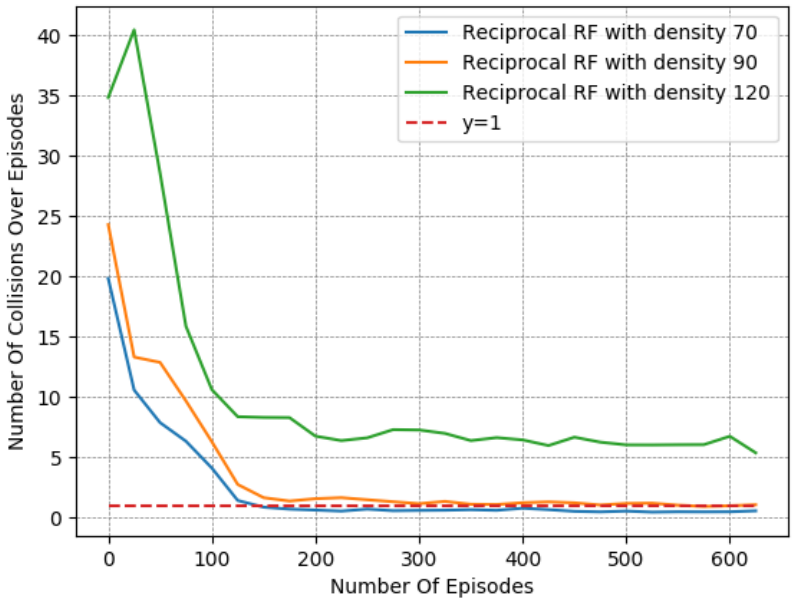


Figure 4.23: Collisions over time for the Reciprocal Reward Function for different densities

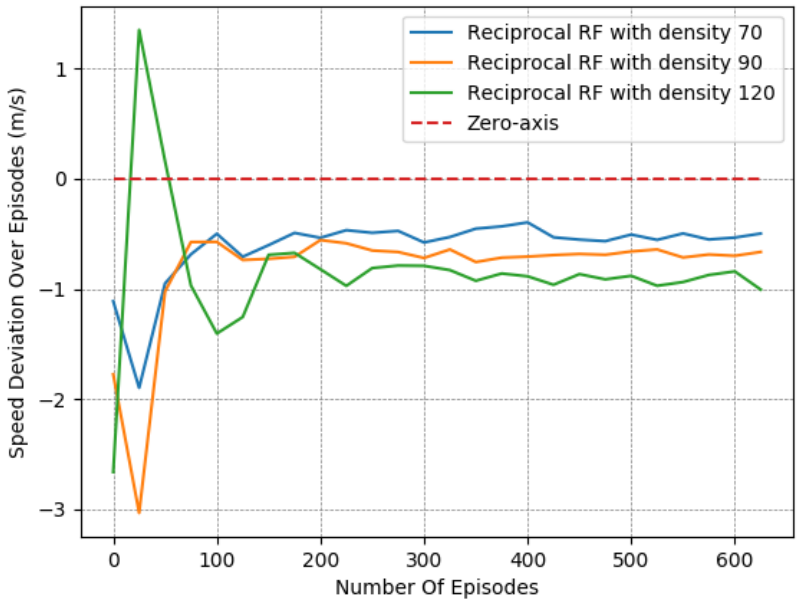


Figure 4.24: Speed Deviation over time for the Reciprocal Reward Function for different densities

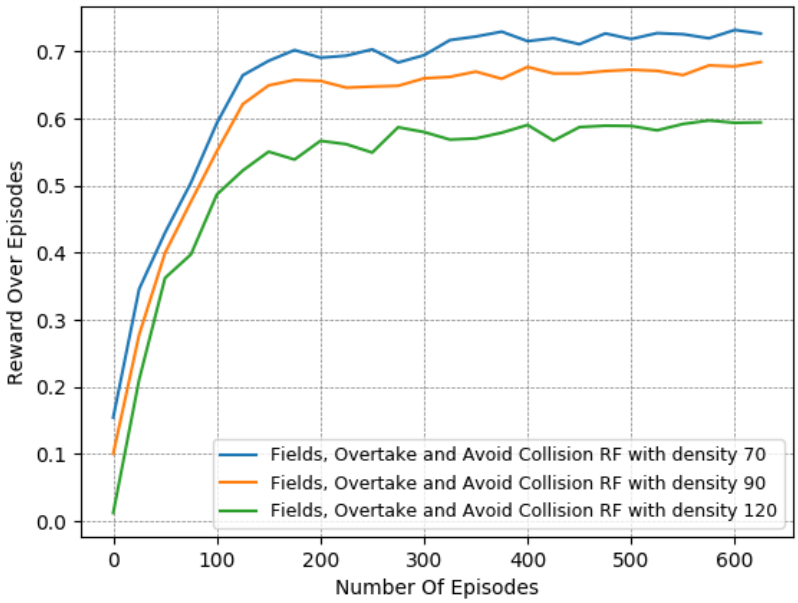


Figure 4.25: Reward over time for the Fields, Overtake and Avoid Collision RF for different densities

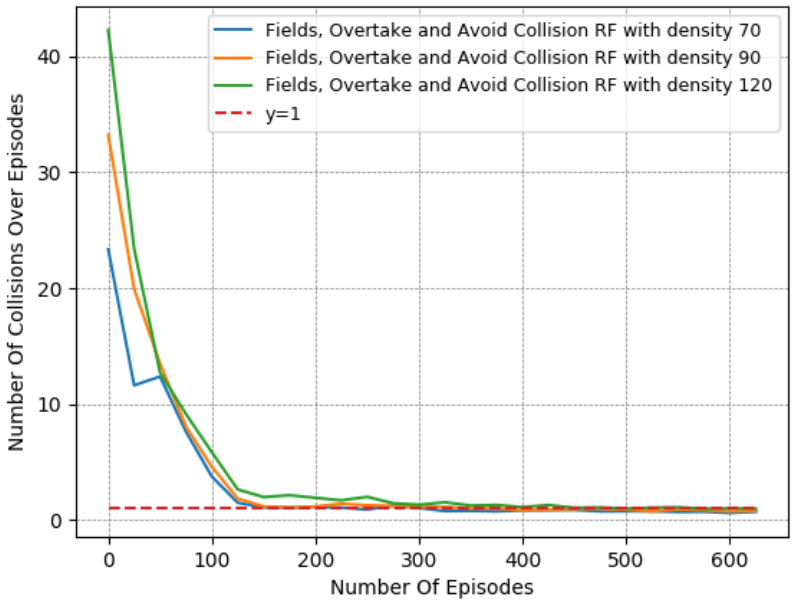


Figure 4.26: Collisions over time for the Fields, Overtake and Avoid Collision RF for different densities

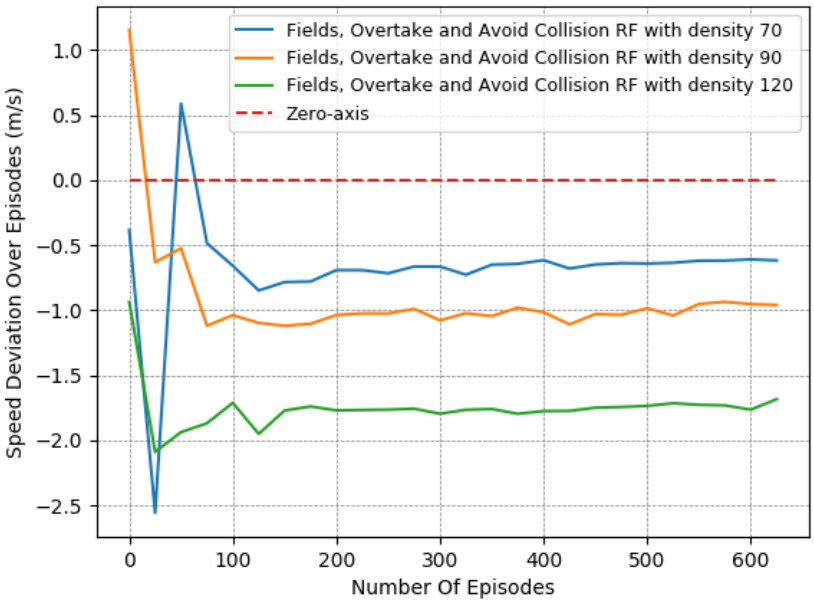


Figure 4.27: Speed Deviation over time for the Fields, Overtake and Avoid Collision RF for different densities

Reward Function	Reciprocal RF			Fields, Overtake and Avoid Collision RF		
Density	70	90	120	70	90	120
Collisions	0.526	1.021	6.062	0.669	0.771	1.066
Speed Dev. (m/s)	-0.514	-0.680	-0.922	-0.613	-0.956	-1.715

Table 4.11: Comparing different Deep Reinforcement Learning Algorithms, using Fields, Overtake and Avoid Collision Reward Function.

since in busier highways, vehicles operate with lower speeds and danger of collision is more present.

In addition, a more direct comparison of the behavior of the two reward functions is found in Table 4.11. The numerical results presented confirm the superiority of the 'Fields, Overtake and Avoid Collision Reward Function' as its performance remains consistent, regardless of the difficulty of the environment. Therefore we conclude that this particular function is the prevalent choice among the rewards examined in Deep Reinforcement Learning for lane-free autonomous driving, since it manages to tackle both objectives efficiently in every configuration that was tested. [1]

Eventually, we must point out that to the best of our knowledge this is one of the earliest endeavors, if not the first, that introduces the concept of deep reinforcement learning to the lane-free environment. Thus the main focus here is not to deploy a "perfect" policy that eliminates collisions, but to examine the limitations and potentials of DRL in a novel and evidently quite challenging domain. Of course, with a (decentralized) coordination approach, using, for instance message-passing algorithms such as max-sum [8, 65], one could eventually guarantee zero collisions (at least experimentally). However, such approaches assume that all agents are interconnected with "fail-safe" communication capabilities, and that they accept to be part of the coordination protocol. By contrast, RL makes no such assumptions. Finally, our approach is at an early stage, and therefore there is no discussion regarding realistic deployment, even if we managed to have 0 collisions on average. The absence of hard constraints and fallback mechanisms in any type of safety-critical system cannot lead to a real-world deployment, and that should be addressed in future work. This absence is even more crucial for algorithms that rely on ANNs (and Machine Learning in general), where explainability endeavours are still not mature enough [66].

[1] Videos showcasing a trained agent with 'Fields, Overtake and Avoid Collision Reward Function' marked as 'All-Components Reward Function' can be found at: `https://bit.ly/3OOLjJW`.

Chapter 5

Conclusions

5.1 Summary

This thesis focused on introducing the concept of deep reinforcement learning to the lane-free traffic domain. Specifically, the aim of this work was to train a single autonomous agent to operate in a lane-free traffic environment, using DRL techniques. In particular, due to the novelty of our training environment, we formulated the above problem as a Markov Decision Process. For that purpose, we introduced a set of reward components at various levels of information that we combined in different reward functions in order to tackle the two objectives, namely collision avoidance and targeting a specific speed of interest. In addition, we thoroughly tested those reward functions for environments with varying difficulties, using a set of both discrete and continuous deep reinforcement learning algorithms.

Throughout our experimental evaluation, we perform a quantitative comparison of all the proposed reward variants and different DRL methods, in order to exhibit their respective performance, and in general provide insight for an RL application on the lane-free traffic domain. We conclude that DRL could be a valuable asset to tackling problems in a lane-free traffic environment and discuss some possible scenarios for extension of our approach.

5.2 Future Work

We encourage the utilization of this work in similar and more complex lane-free experiments, as well as its further improvement. In detail, since our work is focused on a novel traffic model, many future work endeavors can be considered. Different RL algorithms can be employed as training methods for the problem at hand, such as PPO [67], which is a continuous, on-policy algorithm, that has been proved to provide a better convergence and performance rate than most deep reinforcement learning algorithms. An additional potential deep RL technique to be examined is that of NAF [68], which can be described as DQN for Continuous Control Tasks and according to the authors outperforms DDPG

on the majority of tasks [68].

Furthermore, we could consider adopting other noteworthy advancements from the Deep RL literature [3], such as the utilisation of a different parameterization of the state-action value function, similar to the one suggested by the authors. Another interesting endeavour would be to utilize Deep RL techniques that explicitly tackle multi-objective problems. Specifically in [69], the authors propose a method that could be effective for challenging problems, like lane-free traffic, whose objectives can not be easily expressed using a scalar reward function, due to the complexity of the environment.

We also intend to utilise other methods that do not necessarily involve learning, such as Monte-Carlo tree search (MCTS) [70]. The MCTS technique could potentially be a great solution to the problem of autonomous driving in a lane-free traffic environment, using the proposed reward functions, as we expect that delayed rewards will provide better policies in terms of the overall performance. An alternative future consideration is to address the multi-agent aspect of this problem, considering other vehicles that learn using the proposed learning behaviors [71].

Finally, we believe that the incorporation of safety rules that regulate the agent's behavior will be beneficial for forthcoming endeavors and will result in better balance of the two objectives. Regarding this goal, there are two potential methodologies, with the former being the incorporation of novel safe RL techniques that consider a set of 'safe' states on the MDP formulation that the agent is allowed to be located, and utilizing optimization techniques to guarantee a safe policy [72]. Alternatively, we can view this problem without the lens of RL formulation, and consider the Responsibility-Sensitive Safety [73], that proposes a specific set of rules for Autonomous Vehicles that ensures safety.

Bibliography

[1] Szilárd Aradi. "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (2022), pp. 740–759.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[3] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. "Agent57: Outperforming the Atari Human Benchmark". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 507–517.

[4] Xuan Di and Rongye Shi. "A survey on autonomous vehicle control in the era of mixed-autonomy: From physics-based to AI-guided driving policy learning". In: *Transportation Research Part C: Emerging Technologies* 125 (2021).

[5] 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". In: *IEEE Transactions on Intelligent Transportation Systems* (2021), pp. 1–18.

[6] Alex Kendall et al. "Learning to Drive in a Day". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8248–8254.

[7] Markos Papageorgiou, Kyriakos-Simon Mountakis, Iasson Karafyllis, Ioannis Papamichail, and Yibing Wang. "Lane-Free Artificial-Fluid Concept for Vehicular Traffic". In: *Proceedings of the IEEE* 109.2 (2021), pp. 114–121. DOI: 10.1109/JPROC.2020.3042681.

[8] Dimitrios Troullinos, Georgios Chalkiadakis, Ioannis Papamichail, and Markos Papageorgiou. "Collaborative Multiagent Decision Making for Lane-Free Autonomous Driving". In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '21. Virtual Event, United Kingdom: International Foundation for Autonomous Agents and Multiagent Systems, 2021, pp. 1335–1343. ISBN: 9781450383073.

[9] Venkata Karteek Yanumula, Panagiotis Typaldos, Dimitrios Troullinos, Milad Malekzadeh, Ioannis Papamichail, and Markos Papageorgiou. "Optimal Path Planning for Connected and Automated Vehicles in Lane-free Traffic". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. 2021, pp. 3545–3552.

[10] Iasson Karafyllis, Dionysis Theodosis, and Markos Papageorgiou. "Two-dimensional cruise control of autonomous vehicles on lane-free roads". In: *60th IEEE conference on Decision and Control (CDC2021)*. 2021, pp. 2683–2689.

[11] Athanasia Karalakou, Dimitrios Troullinos, Georgios Chalkiadakis, and Markos Papageorgiou. "Deep RL reward function design for lane-free autonomous driving". In: *20th International Conference on Practical Applications of Agents and Multi-Agent Systems*. L'Aquila, Italy, July 2022.

[12] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[13] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 2016.

[14] Qiong Liu and Ying Wu. "Supervised Learning". In: *Encyclopedia of the Sciences of Learning*. Ed. by Norbert M. Seel. Boston, MA: Springer US, 2012, pp. 3243–3245. ISBN: 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_451. URL: https://doi.org/10.1007/978-1-4419-1428-6_451.

[15] Geoffrey Hinton and Terrence J. Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*. The MIT Press, May 1999. ISBN: 9780262288033. DOI: 10.7551/mitpress/7011.001.0001. URL: https://doi.org/10.7551/mitpress/7011.001.0001.

[16] Xiaojin Zhu and Andrew B. Goldberg. "Introduction to Semi-Supervised Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3.1 (2009), pp. 1–130. DOI: 10.2200/S00196ED1V01Y200906AIM006. URL: https://doi.org/10.2200/S00196ED1V01Y200906AIM006.

[17] A. W. Moore L. P. Kaelbling M. L. Littman. "Reinforcement Learning: A Survey." In: *Journal of Artificial Intelligence Research* 4.1 (1996), pp. 237–285. DOI: https://doi.org/10.1613/jair.301.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[19] Aske Plaat. "Deep Value-Based Reinforcement Learning". In: *Deep Reinforcement Learning*. Springer Nature Singapore, 2022, pp. 69–100. DOI: 10.1007/978-981-19-0638-1_3. URL: https://doi.org/10.1007%2F978-981-19-0638-1_3.

[20] Aske Plaat. "Policy-Based Reinforcement Learning". In: *Deep Reinforcement Learning*. Springer Nature Singapore, 2022, pp. 101–133. DOI: 10.1007/978-981-19-0638-1_4. URL: https://doi.org/10.1007%2F978-981-19-0638-1_4.

[21] Mohit Sewak. "Policy-Based Reinforcement Learning Approaches". In: *Deep Reinforcement Learning*. Springer Singapore, 2019, pp. 127–140. DOI: 10.1007/978-981-13-8285-7_10. URL: https://doi.org/10.1007%2F978-981-13-8285-7_10.

[22] Paweł Wawrzyński. "Real-time reinforcement learning by sequential Actor–Critics and experience replay". In: *Neural Networks* 22.10 (2009), pp. 1484–1497. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2009.05.011. URL: https://www.sciencedirect.com/science/article/pii/S0893608009001026.

[23] Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. "Model-based Reinforcement Learning: A Survey". In: *CoRR* abs/2006.16712 (2020). arXiv: 2006.16712. URL: https://arxiv.org/abs/2006.16712.

[24] Lukasz Kaiser et al. "Model-Based Reinforcement Learning for Atari". In: *CoRR* abs/1903.00374 (2019). arXiv: 1903.00374. URL: http://arxiv.org/abs/1903.00374.

[25] Sinan Çalışır and Meltem Kurt Pehlivanoğlu. "Model-Free Reinforcement Learning Algorithms: A Survey". In: *2019 27th Signal Processing and Communications Applications Conference (SIU)*. 2019, pp. 1–4. DOI: 10.1109/SIU.2019.8806389.

[26] Richard Bellman. "A MARKOVIAN DECISION PROCESS." In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684. DOI: http://www.jstor.org/stable/24900506..

[27] Matthijs T. J. Spaan. "Partially Observable Markov Decision Processes". In: *Reinforcement Learning: State-of-the-Art*. Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 387–414. ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_12. URL: https://doi.org/10.1007/978-3-642-27645-3_12.

[28] Claude Sammut and Geoffrey I. Webb. *Encyclopedia of Machine Learning*. Springer US, 2010, pp. 97–97. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_71. URL: https://doi.org/10.1007/978-0-387-30164-8_71.

[29] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292.

[30] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[31] Oded Berger-Tal, Jonathan Nathan, Ehud Meron, and David Saltz. "The Exploration-Exploitation Dilemma: A Multidisciplinary Framework". In: *PLOS ONE* 9.4 (Apr. 2014), pp. 1–8. DOI: 10.1371/journal.pone.0095693. URL: https://doi.org/10.1371/journal.pone.0095693.

[32] G. E. Uhlenbeck and L. S. Ornstein. "On the Theory of the Brownian Motion". In: *Phys. Rev.* 36 (5 Sept. 1930), pp. 823–841.

[33] S. D. Chatterji. "The mathematical work of N. Wiener (1894–1964)". In: *Current Science* 67.12 (1994), pp. 930–936. ISSN: 00113891. URL: http://www.jstor.org/stable/24096776.

[34] 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". In: (2016). Ed. by Yoshua Bengio and Yann LeCun.

[35] Yuxi Li. "Deep Reinforcement Learning: An Overview". In: *CoRR* abs/1701.07274 (2017). arXiv: 1701.07274. URL: http://arxiv.org/abs/1701.07274.

[36] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. "An Introduction to Deep Reinforcement Learning". In: *CoRR* abs/1811.12560 (2018). arXiv: 1811.12560. URL: http://arxiv.org/abs/1811.12560.

[37] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[38] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: 10.1109/MSP.2017.2743240.

[39] Warren Mcculloch and Walter Pitts. "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[40] Stephen Cole Kleene. "Representation of Events in Nerve Nets and Finite Automata". In: 1951.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[42] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *International Journal of Engineering Applied Sciences and Technology* 04 (May 2020), pp. 310–316. DOI: 10.33564/IJEAST.2020.v04i12.054.

[43] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375.

[44] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

[45] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[46] Sergios Theodoridis. "Stochastic Gradient Descent". In: *Machine Learning*. Elsevier, 2015, pp. 161–231. DOI: 10.1016/b978-0-12-801522-3.00005-7. URL: https://doi.org/10.1016%2Fb978-0-12-801522-3.00005-7.

[47] Andrew Murphy and David Wang. *Stochastic gradient descent*. July 2018. DOI: 10.53347/rid-61715. URL: https://doi.org/10.53347%2Frid-61715.

[48] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[49] George Bebis and Michael Georgiopoulos. "Feed-forward neural networks". In: *IEEE Potentials* 13.4 (1994), pp. 27–31. DOI: 10.1109/45.329294.

[50] Lakhmi C. Jain and Larry R. Medsker. "Recurrent Neural Networks: Design and Applications". In: 1999.

[51] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *CoRR* abs/1511.08458 (2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[52] Jiuxiang Gu et al. "Recent Advances in Convolutional Neural Networks". In: *CoRR* abs/1512.07108 (2015). arXiv: 1512.07108. URL: http://arxiv.org/abs/1512.07108.

[53] Gasser Auda and Mohamed Kamel. "Modular Neural Networks: A Survey". In: *International Journal of Neural Systems* 09.02 (1999), pp. 129–151. DOI: 10.1142/S0129065799000125. eprint: https://doi.org/10.1142/S0129065799000125. URL: https://doi.org/10.1142/S0129065799000125.

[54] Christopher John Cornish Hellaby Watkins. "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College, May 1989.

[55] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: vol. 30. Mar. 2016.

[56] Hado Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[57] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. *Prioritized Experience Replay*. 2016. eprint: 1511.05952.

[58] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. "Dueling Network Architectures for Deep Reinforcement Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 1995–2003.

[59] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, June 2014, pp. 387–395.

[60] Panagiotis Typaldos, Markos Papageorgiou, and Ioannis Papamichail. "Optimization-based path-planning for connected and non-connected automated vehicles". In: *Transportation Research Part C: Emerging Technologies* 134 (2022), p. 103487. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.103487. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21004733.

[61] Martin Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[62] François Chollet et al. *Keras*. https://keras.io. 2015.

[63] Cathy Wu, Abdul Rahman Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M. Bayen. "Flow: A Modular Learning Framework for Mixed Autonomy Traffic". In: *IEEE Transactions on Robotics* (2021), pp. 1–17. ISSN: 1941-0468.

[64] Matthias Plappert. *keras-rl*. https://github.com/keras-rl/keras-rl. 2016.

[65] Dimitrios Troullinos, Georgios Chalkiadakis, Vasilis Samoladas, and Markos Papageorgiou. "Max-sum with Quadtrees for Decentralized Coordination in Continuous Domains". In: *31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence*. Vienna, Austria, July 2022.

[66] Nadia Burkart and Marco F. Huber. "A Survey on the Explainability of Supervised Machine Learning". In: *J. Artif. Int. Res.* 70 (May 2021), pp. 245–317. ISSN: 1076-9757. DOI: 10.1613/jair.1.12228. URL: https://doi.org/10.1613/jair.1.12228.

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

[68] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous Deep Q-Learning with Model-based Acceleration". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 2829–2838.

[69] Changjian Li and Krzysztof Czarnecki. "Urban Driving with Multi-Objective Deep Reinforcement Learning". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '19. Montreal QC, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 359–367.

[70] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games*. 2007, pp. 72–83.

[71] Xuan Di and Rongye Shi. "A survey on autonomous vehicle control in the era of mixed-autonomy: From physics-based to AI-guided driving policy learning". In: *Transportation Research Part C: Emerging Technologies* 125 (2021), p. 103008. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.103008. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21000401.

[72] Ali Baheri, Subramanya Nageshrao, Ilya Kolmanovsky, Anouck Girard, H. Eric Tseng, and Dimitar Filev. *Deep Q-Learning with Dynamically-Learned Safety Module: A Case Study in Autonomous Driving*. Oct. 2019.

[73] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. *On a Formal Model of Safe and Scalable Self-driving Cars*. 2017. DOI: 10.48550/ARXIV.1708.06374. URL: https://arxiv.org/abs/1708.06374.