POLITECNICO DI TORINO

Master's Degree

in Mechatronic Engineering

Master's Thesis

Artificial Intelligence Controls for Vehicle Emergency Maneuvering

Supervisor
prof. Stefano Alberto Malan

Candidate
Giovanni Castania

Addfor S.p.A. Supervisor
Dott. Stefano Ballesio

Academic year 2019-2020

Abstract

Autonomous driving is the future of automotive. The most important challenge in autonomous driving applications is represented by safety. An autonomous driving system to be perceived safe has to be trustful in emergency situations. To recognize an emergency situation and act immediately to avoid collisions is one of the most important feature of an autonomous driving system. This thesis work investigated the topic of collision avoidance in emergency situation in autonomous driving using deep reinforcement learning. According to the literature related to this problem, two kinds of ADAS controls were studied. The studied controls were: Autonomous Emergency Steering and Autonomous Emergency Braking. The use of reinforcement learning based methods in autonomous driving field is increasing, because of the versatility of these algorithms. The deep reinforcement learning algorithm used in this thesis work was a Double Deep Q-Network with Dueling Architecture. This model-free deep reinforcement learning algorithm allowed the reinforcement learning agent to learn complex controls in realistic simulated environment. To provide the most realistic environment possible, a simulator with state of the art rendering quality was used. The problem was contextualized in an urban scenario where a pedestrian crossed the street at an unexpected timing. The information were provided to the agent through sensors. Cameras, collision and lane invasion sensors were used. The information used by the agent to learn the policy were provided by the camera sensor. The output frames of the camera were processed and stacked together to give the agent the sense of motion. The outputs of the collision and the lane invasion sensors were used to determine the terminal state of the learning episodes. The settings of the tests and their main parameters were chosen taking as reference the Euro NCAP AEB test protocol. The agent successfully learned from scratch how to perform an emergency brake action and an emergency overtake action. In the Autonomous Emergency Steering test the agent also learned from scratch how to keep the lane. This thesis work makes contribution on the topic of investigating collision avoidance in emergency situation using deep reinforcement learning. Since the input was given to the agent through a low cost and established sensor, the trained agent can be used in real vehicles for further studies.

Contents

1. Introduction ……...…1

2. Reinforcement Learning ……....…….4

2.1 Reinforcement Learning Overview ……..…...6

2.1.2 Reinforcement Learning Formalism ……10

2.2 Temporal Difference Learning ……...….14

2.2.1 Q-learning ……14

2.3 Deep Reinforcement Learning …….…….15

2.3.1 Deep Q Network ……...…16

2.4 Improvements in Deep Q-Network……...18

2.4.1 Double Deep Q Network ……..…18

2.4.2 Duel Deep Q Network ……19

3.Autonomous Driving ……20

3.1 Components of Autonomous Driving System …….23

3.2 Deep Reinforcement Learning for Autonomous Driving ……..26

3.2.1 Deep Reinforcement Learning for Autonomous Driving Tasks ……...27

3.3 CARLA Simulator ……29

3.4 Euro NCAP Test Protocol – AEB VRU systems ……..….32

4. Algorithm Tests ……...……33

4.1 Experiment Settings ……..34

4.1.1 Time to Collision and Target Velocity …….35

4.1.2 Map ……36

4.1.3 Weather ……..37

4.1.4 Ego-vehicle Controls …….38

4.1.5 Ego-vehicle Sensors ……..39

4.1.6 Pedestrian Target ……..40

4.2 Autonomous Emergency Braking ……41

4.2.1 AEB – Test description ……42

4.2.2 AEB – Reward and Terminal States ……..……43

4.2.3 AEB – Hyper-parameter Values and Neural Network…….45

4.3 Autonomous Emergency Steering ……47

4.3.1 AES – Test Description ……...……48

4.3.2 AES – Reward and Terminal States ……..……49

4.3.3 AES – Hyper-parameter Values and Neural Network ……..……52

5. Algorithm Results ……54

5.1 Autonomous Emergency Braking ……55

5.1.1 Training Results ……..55

5.1.2 Test Results ……..57

5.2 Autonomous Emergency Steering ……62

5.2.1 Training Results ……..…62

5.2.2 Test Results ……..64

6.Conclusions ……..…….69

6.1 Future Work ……...…70

6.1.1 Improvements in the Agent ……...……70

6.1.2 Benchmark among RL Algorithms ……...…71

6.1.3 Implementation of Rule-Based Constraints ……...…..71

6.1.4 Use of a Regret-Based Human Model ……...…71

Bibliography ……*72*

List of Figures

Figure 2.1 Machine Learning categories [2] ……...…4

Figure 2.2 Scheme of a Reinforcement Learning problem [2] ……..…....…….7

Figure 2.3 Atari games Reinforcement Learning example [2]…….…...9

Figure 2.4 Reinforcement Learning categories [2]……..…….9

Figure 2.5 The Agent-Environment interaction in a Markov Decision Process [1]……..10

Figure 2.6 Q-learning Algorithm Scheme…….……14

Figure 2.7 Deep Q-learning Algorithm Scheme ……..…….…….16

Figure 2.8 Standard Network (above) Dueling Network (below) [10] ……...…..…19

Figure 3.1 A modern Autonomous Driving System pipeline [23]…….....23

Figure 3.2 Example of CARLA environment in four different weather conditions [32].……..29

Figure 3.3 Vehicle representation as a rigid body actor [33] …….30

Figure 3.4 Vehicle representation as a collection of sprung masses of mass M1 and M2 [33].……30

Figure 4.1 Town 02 Map [35] ……...….36

Figure 4.2 Frame of the Experiment during CPNC test……..……..37

Figure 4.3 Image Processing Steps – From the left: input image, black and white, resized image ..39

Figure 4.4 Input given to the Reinforcement Learning agent consisting in four stacked frames …..39

Figure 4.5 Comparison of pedestrian targets – On the left there are the CARLA models [36] on the right there are the EuroNCAP targets[34] …….……....40

Figure 4.6 Autonomous Emergency Braking test Spawning Positions……..…42

Figure 4.7 Autonomous Emergency Braking Neural Network……..46

Figure 4.8 Autonomous Emergency Steering test Spawning and Goal Positions …….48

Figure 4.9 Autonomous Emergency Steering Neural Network ……..…53

Figure 5.1 Autonomous Emergency Braking Training ……..55

Figure 5.2 Autonomous Emergency Braking Exploration Exploitation Chart ……..….56

Figure 5.3 Autonomous Emergency Braking Testing ……..…..57

Figure 5.4 Results of Autonomous Emergency Braking Test – Test Type ……..…..58

Figure 5.5 Results of Autonomous Emergency Braking Test – Pedestrian Behavior ……...…59

Figure 5.6 Results of Autonomous Emergency Braking Test – Target Velocity ……...…60

Figure 5.7 Results of Autonomous Emergency Braking Test – Time To Collision …..……..…61

Figure 5.8 Autonomous Emergency Steering Training …….62

Figure 5.9 Autonomous Emergency Steering Exploration Exploitation Chart ……63

Figure 5.10 Autonomous Emergency Steering Testing …….64

Figure 5.11 Results of Autonomous Emergency Steering Test – Test Type ……...…...65

Figure 5.12 Results of Autonomous Emergency Steering Test – Pedestrian Behavior ……66

Figure 5.13 Results of Autonomous Emergency Steering Test – Target Velocity ……..…67

Figure 5.14 Results of Autonomous Emergency Steering Test – Time To Collision ……68

List of Tables

Table 2.1 Q-learning ……...……...…15

Table 2.2 Deep Q-learning with Experience Replay…….....…….17

Table 3.1 SAE levels of Driving Automation [12] …….…...21

Table 3.2 Sensor Characteristics [22] ……...….24

Table 4.1 Time To Collision – Ego-vehicle Target Velocity …….….35

Table 4.2 Ego-vehicle Controls …….……38

Table 4.3 Pedestrian Target Parameters ……...…….…….40

Table 4.4 Terminal States and Rewards – Autonomous Emergency Braking …….……...…44

Table 4.5 Hyper-Parameter Values – Autonomous Emergency Braking ……...45

Table 4.6 Neural Network Architecture – Autonomous Emergency Braking ……...…45

Table 4.7 Terminal States and Rewards – Autonomous Emergency Steering ……..….51

Table 4.8 Hyper-Parameter Values – Autonomous Emergency Steering …….....52

Table 4.9 Neural Network Architecture – Autonomous Emergency Steering …….……..52

Table 5.1 Autonomous Emergency Braking Training……...….56

Table 5.2 Autonomous Emergency Braking Testing – Overall ……...…….…….58

Table 5.3 Autonomous Emergency Braking Testing – Test Type …….…….58

Table 5.4 Autonomous Emergency Braking Testing – Pedestrian Behavior …….59

Table 5.5 Autonomous Emergency Braking Testing – Target Velocity …….60

Table 5.6 Autonomous Emergency Braking Testing – Time To Collision ……..…….61

Table 5.7 Autonomous Emergency Steering Training ...……...….63

Table 5.8 Autonomous Emergency Steering Testing – Overall ……...……...…….…….65

Table 5.9 Autonomous Emergency Steering Testing – Test Type ……...…….65

Table 5.10 Autonomous Emergency Steering Testing – Pedestrian Behavior ……..……66

Table 5.11 Autonomous Emergency Steering Testing – Target Velocity ……..…….67

Table 5.12 Autonomous Emergency Steering Testing – Time To Collision ……68

1. Introduction

Autonomous driving is the future of automotive. Experiments regarding driver-less systems were conducted since the early years of the past century. The evolution in these systems were many and the last trend is to invest in this technology for further innovation. The most important challenge in autonomous driving applications is represented by safety. Both for the vehicle occupants and the other traffic participants. An autonomous driving system to be perceived safe has to be trustful in emergency situations. Emergency situations caused by an unexpected behavior of a traffic participant can be particularly difficult to handle for a driver. According to a study by National Highway Traffic Safety Administration (NHTSA) [43], 94% of the critical pre-crash events can be attributed to drivers. Recognition error and decision error account for 41% and 31% of driver-related reasons. So, can be said that to recognize an emergency situation and act immediately to avoid the collision has to be one of the most important feature of an autonomous driving system. This was tragically evident in the first reported crash between a pedestrian and an autonomous car [20]. That accident of march 2018 highlighted that there is still too much work to do before implementing safely autonomous driving technology. For this reason the emergency situation were the main topic of this thesis work.

There are two approaches to the problem of the emergency situations. These two approaches can be grouped in two categories according to the controls taken into account: systems that can only brake and systems that can both brake and steer. Emergency systems that can brake, without possibility to steer were investigated, for example, in paper [40]. In paper [40] a method to compute the trigger time to activate the emergency braking system was presented. The main inconvenient of systems like the one in [40] is that, by only braking, it is impossible to avoid collisions for distances smaller than the braking space needed by the vehicle. This issue can be mitigated with systems that can both brake and steer. Emergency systems that can brake, with possibility to steer were investigated, for example, in paper [41] and [42]. In paper [41] how to avoid collisions using emergency maneuvers was studied. The minimum distance to avoid an obstacle given a certain speed was identified. The minimum distance was used to identify a clearance curve, useful to plan emergency maneuvers. In paper [42] a nonlinear model predictive control (NMPC) was proposed to perform emergency collision avoidance maneuvers. The papers presented [40], [41], [42], used heuristic methods to study the problem of reducing the harm of a collision scenario. A new approach in this field is to

introduce an artificial intelligence to learn how to perform an emergency maneuver. The development of deep reinforcement learning proved that this technology is suitable for complex problem and can handle unexpected situations. Deep reinforcement learning was used in paper [30] and [31]. In paper [30] the system belonged to the category that can only brake, in [31] the system belonged instead to the category that can both brake and steer. In [30] the agent learned, through a Deep Q-Network algorithm, to handle an emergency braking action and to avoid to hit a crossing pedestrian. In [31] the agent learned, through a Deep Deterministic Policy Gradient algorithm, how to perform an evasive maneuver to avoid any crash, both with pedestrian and vehicles.

This thesis work explored the use of reinforcement learning in autonomous driving application. The aim of this thesis work was to use reinforcement learning to study the problem related to avoiding collisions in emergency situation in autonomous driving. The objective was to model a valid and reproducible test and to design a suitable reward function to let the agent learn in the best possible way. According to the literature related to this problem, two kind of ADAS controls were studied. The studied controls were: Autonomous Emergency Steering and Autonomous Emergency Braking. The role of reinforcement learning in avoiding collisions by using braking and braking and steering control systems was investigated. The results of the trained agent were expected to show an improvement in performance by adding the steering ability to the agent. This because of the limitations of the needed baking space in only-braking systems.

The problem was contextualized in an urban scenario where a pedestrian crosses the street at an unexpected timing. To provide the most realistic environment possible, a simulator which uses state of the art rendering quality was used. The information were provided to the agent through sensors. Cameras, collision and lane invasion sensors were used. The information used by the agent to learn the policy were provided by the camera sensor. The camera output frames were processed and stacked together to give the agent the sense of motion. The outputs of the collision and the lane invasion sensors were used to determine the terminal state of the learning episodes. The settings of the tests and their main parameters were chosen taking as reference the Euro NCAP AEB test protocol [34]. This test was taken as reference also by the paper [30]. To learn the policy the agent used a Double Deep Q-Network with Dueling Architecture.

The main contributions of this thesis work are the following: using the deep reinforcement learning with camera observations, an artificial intelligence, able to avoid unexpected collisions in an urban scenario, was trained. Moreover, since the input was given to the agent through a low cost and

established sensor, the trained agent can be used in real vehicles for further studies. Moreover, the use of reinforcement learning based methods in autonomous driving field is increasing, because of the versatility of these algorithms. To study the applicability and the performances of reinforcement learning in autonomous driving is of great importance especially for safety situations.

This thesis work is organized as follows. Chapter 2 describes the theory of reinforcement learning. First, an overview on reinforcement learning is given, then the formalism is specified. The theory behind the algorithm used in this thesis work is explained. Chapter 3 describes the background of autonomous driving. The use of deep reinforcement learning on autonomous driving is described. The simulator used for the experiment and the test used as reference are presented. Chapter 4 describes the experiment done in this thesis work. The settings of the experiment are described. The two test for emergency braking and braking and steering are described. Chapter 5 describes the results obtained by the reinforcement learning agent in both training and testing. Finally, chapter 6 presents the conclusions of this thesis work.

2. Reinforcement Learning

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task T, as measured by P, improves with experience E. (Mitchell, T. (1997) Machine learning, McGraw Hill, p. 2)

This operational definition provided by Tom Mitchell, focuses on the tasks where Machine Learning is involved. To define Machine Learning in cognitive terms can be stated that: Machine Learning is the study of computer algorithms that can learn without being explicitly programmed.

Machine Learning, as shown in Figure 2.1, can be divided in three main categories: Supervised Learning, Unsupervised Learning, Reinforcement Learning.

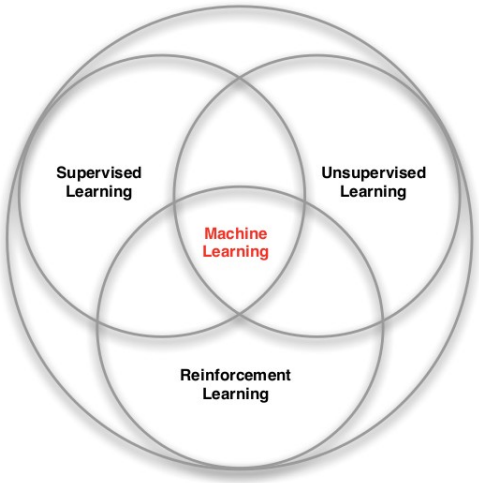

Figure 2.1 Machine Learning categories [2]

Supervised Learning is the machine learning paradigm where, given a set of examples, the algorithm has to derive a function to map new examples. In Supervised Learning each example is a pair consisting in an input object and a desired output value. The set of examples given to the algorithm is provided by the knowledge of an external supervisor.

Unsupervised Learning is the Machine Learning paradigm that self-organizes input data in order to return hidden structures that may be present in the data itself. The use of Unsupervised Learning

allows to derive structures from data without knowing the effects of the variables. Unsupervised Learning gives no feedback based on the prediction of the results.

Reinforcement Learning is the Machine Learning paradigm where a software agent learns how to maximize the reward interacting with the environment.

In this chapter a theoretical background on Reinforcement Learning is presented. Key concepts are introduced and described. First, the general theory of Reinforcement Learning is presented. Then, the theoretical background of the algorithm used in the thesis work is discussed. The Q-learning algorithm is presented, then the Deep Q-learning with experienced replay is discussed. Next, improvements to the Deep Q-learning are discussed. These improvements are the Double Q-learning algorithm and the Duel architecture.

2.1 Reinforcement Learning Overview

Reinforcement Learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.
(Sutton, R. & Barto, A. (1998) Reinforcement Learning: An Introduction, MIT Press, p. 1)

This definition by Sutton and Barto [1] gives all the main concepts behind the idea of reinforcement learning. The key concepts are: there is a learner, called agent, and the environment the agent interacts with. After each interaction with the environment the agent sees a partial observation of the state of the environment. From this observation the agent chooses the next interaction. For each action the agent perform it receives a reward. Reward is one of the main differences of reinforcement learning from other machine learning paradigms. In fact there is no supervisor for the agent's actions, only a reward signal. Another important difference is that the feedback is delayed, instead of being instantaneous. Moreover the agent action influences the subsequent environment state and reward it receives.
One of the challenges in reinforcement learning is the trade-off between *exploration* and *exploitation*. To discover what gives more reward the agent has to '*explore*' the environment to find out which actions return the highest profit. To maximize the reward the agent has to '*exploit*' the environment, which means to reiterate the action with highest reward. Although this problem has been studied for decades, it has not been solved yet.

Some of the concepts introduced need a proper definition. A reinforcement learning problem consists in these elements: the agent, the environment, states, actions, a reward, a policy, a value function and an optional model of the environment. Figure 2.2 schematize a typical reinforcement learning problem.

The *agent* is the main actor of a reinforcement learning problem. It can take actions and receive a reward for that actions. It is represented by the brain in Figure 2.2.

The *environment* is the world the agent interacts with. It is represented by the earth in Figure 2.2 The environment can change either according to its internal laws or because of the agent's actions.

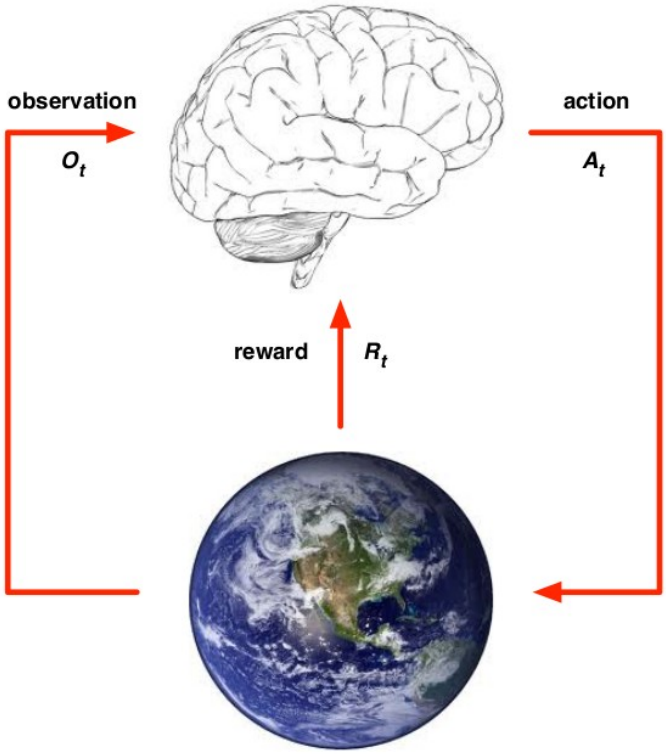

Figure 2.2 Scheme of a Reinforcement Learning problem [2]

The *state* is the information used by the agent to determine what happens next. It can be divided in environment state and agent state. The set of possible states is represented by $\mathcal{S}$.

The environment state is the exhaustive description of all the information of the environment the agent interacts with. It may contain irrelevant information for the agent.

The agent state, *observation*, is an incomplete description of the state, which may have less information. It is the information used by the agent to make decisions.

If the agent can observe the complete state of the environment, the environment is fully observed. If the agent can observe only a portion of the environment, the environment is partially observed.

The *actions* are the methods the agent has to interact with the environment. Each environment has a different action set, called action space. The set of possible actions is represented by $\mathcal{A}$.

$\mathcal{A}$ represents the actions the agent can make in the specific environment. Some environments has a finite number of action that can be taken, that kind of environments has discrete action space. Some

environments, instead, have continuous action space. In continuous space action are real-value vectors.

The *reward* is a numeric value that gives the agent the information on the quality of the actions taken. The set of possible rewards is represented by $\mathcal{R}$.
Higher is the reward, better is the chosen action. The reward hypothesis is so defined: the goal is always to maximize the cumulative reward. The agent task in a reinforcement learning problem is to fulfill the reward hypothesis.

The *policy* is the rule used by the agent to decide which action to take. It represents the agent's behavior. The policy is a mapping from states to action. The policy can be deterministic or stochastic.

The *value* function is the measure the agent has to determine how good is an action in the long term. It can be defined as the amount of reward the agent predicts to accumulate starting from a certain state. The aim of the agent is to do actions that lead to states with high value. The main difference the value has from the reward is that the reward is given by the environment, while the value is estimated by the agent from the sequence of observations.

The *model* is the way the agent represents the environment. It is used to make predictions about the next state and the next reward. The model is used for planning, it considers the consequences of the actions before experiencing them.

A way to better understand reinforcement learning can be through examples that have guided its development. In Figure 2.3 is shown a typical example of a reinforcement learning problem.
The example consists of an agent that learns how to play Atari video-games. The rules of the game are not given to the agent, in fact it has to discover them by itself. The learning is related to the interactions between the agent and the environment. The observation of the environment are the game's pixels, the actions are the controls, the reward is the score over the game. Seeing the pixel of the game the agent takes actions, that actions lead to a score, which influences the next observations and actions.
The methods for solving reinforcement learning problems, as shown in Figure 2.4, can be divided in four main categories: *model-based*, *model-free*, *on-policy*, *off-policy*.

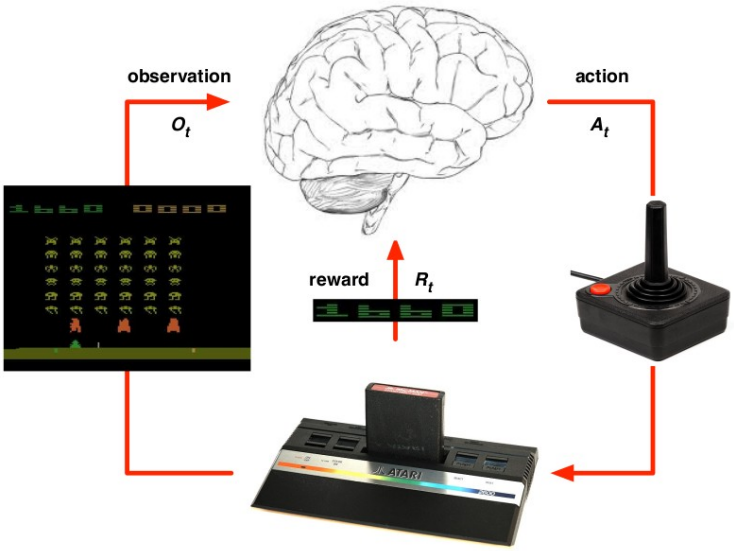

Figure 2.3 Atari games Reinforcement Learning example [2]

The methods that make use of models, known from the beginning or learned, and planning are called *model-based* methods.

The methods that use a trial and error strategy are called *model-free* methods.

The methods that use deterministic outcome or samples from the target policy to train the algorithm are called *on-policy* methods.

The methods that train the algorithm on a distribution of episode by different policy are called *off-policy* methods.

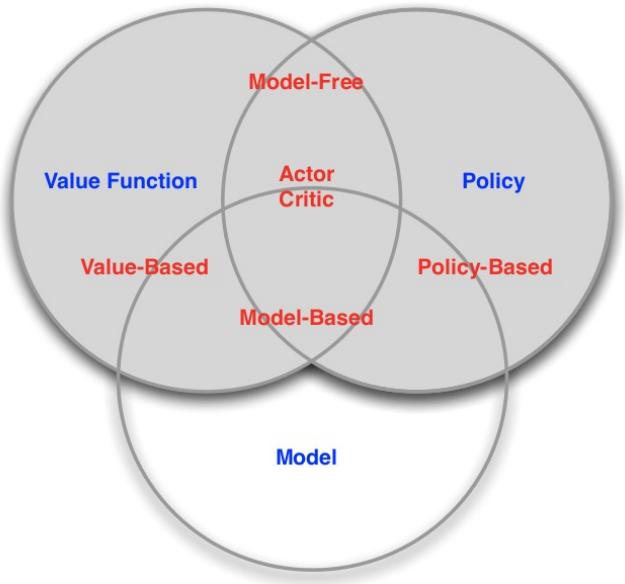

Figure 2.4 Reinforcement Learning categories [2]

2.1.1 Reinforcement Learning Formalism

Markov decision processes (MDPs) are an idealization of the reinforcement learning problem, so almost all reinforcement learning problems can be framed as MDPs.
In Figure 2.5 it is shown the agent-environment interaction in a Markov decision process.

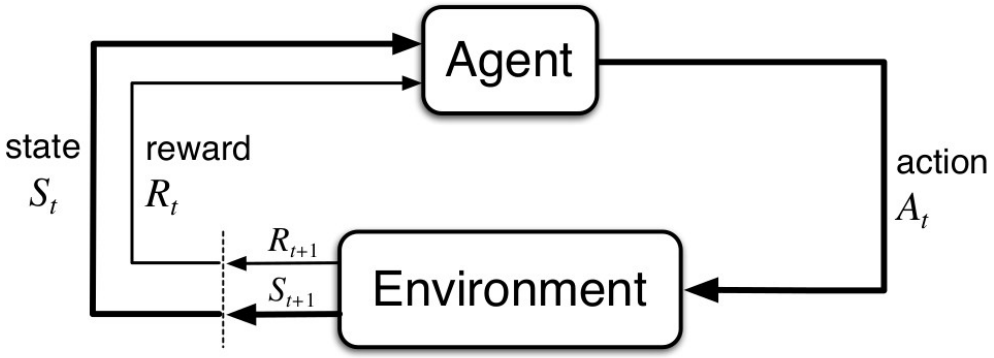

Figure 2.5 The Agent-Environment interaction in a Markov Decision Process [1]

Markov decision processes are a classical formalization of sequential decision making, the actions affect the immediate reward, the next situations and through the next situations the next rewards. In a MDP agent and environment interact at each sequence of discrete time steps. At each time step t the agent sees an observation of the environment state S_t and makes an action A_t . The next step the agent receives a reward $R_{(t+1)}$ and the next observation $S_{(t+1)}$.
The History H_t (2.1) is a sequence of observations, rewards and actions in the environment.

$$H_t = (O_0, R_0, A_0, \ldots A_{(t-1)}, O_t, R_t) \quad (2.1)$$

Formally the state is a function of the history (2.2).

$$S_t = f(H_t) \quad (2.2)$$

However the agent state, the observation, may be not function of the history.
The fundamental property of MDPs called 'Markov' property is that the next situations depends only on the current situation, not the history. In this way future and past are independent from the present. The information state, also called Markov state, contains all useful information from the history. So if the state is known it is all that is needed to predict the future. The history and the environment state are Markov.

The model is the agent's representation of the environment. It consists on two main parts. The transition probability function and the reward function.

The transition probability function $P(s',r|s,a)$ (2.3) gives the probability to pass from state s to s' after taking action a obtaining reward r . The symbol $\mathbb{P}$ is used for probability.

$$P(s',r|s,a)=\mathbb{P}[S_{(t+1)}=s',R_{(t+1)}=r|S_t=s,A_t=a] \quad (2.3)$$

The reward function $R(s,a)$ (2.4) relies upon the current state, the action took and the next state. It represents the expected reward consequent to an action.

$$R(s,a)=\mathbb{E}[R_t|S_{(t-1)}=s,A_{(t-q)}=a] \quad (2.4)$$

The agent's behavior is represented by the policy. The policy can be deterministic or stochastic. Deterministic policy (2.5) is a policy that at a given state will always return the same action.

$$\pi(s)=a \quad (2.5)$$

Stochastic policy (2.6) is a policy that gives a probability distribution over different actions.

$$\pi(a|s)=\mathbb{P}_\pi[A_t=a|S_t=s] \quad (2.6)$$

The value function measures how good is to be in a particular state or how good is a state-action pair by a prediction of the return. They are a prediction of the future reward.

The future reward is represented by the return. The return G_t is the cumulative reward over history or episodes, the aim of the agent is to maximize the return. There are two kinds of returns.

The sum of rewards in a fixed window step is defined as finite-horizon un-discounted return (2.7).

$$G_t=\sum_{k=0}^{T} R_t \quad (2.7)$$

The sum of all the rewards ever obtained by the agent, discounted by a factor which represents how far in the future this rewards are obtained is defined as infinite-horizon discounted return (2.8).

$$G_t=\sum_{k=0}^{\infty} \gamma^k R_{(t+k+1)} \quad (2.8)$$

The discount rate is $\gamma\in[0,1]$. It represents the value given to future rewards. A value of γ close to 0 means that the agent is mainly focused on immediate reward. A value of γ close to 1 means that the agent is mainly focused on future reward, so the agent is more farsighted. The discount rate provides mathematical convenience, in fact if $\gamma<1$ the return can converge if the reward sequence R_t is bounded.

There are two main value functions: on-policy value function, on-policy action-value function.

On-policy value function $V_\pi(s)$ (2.9) gives the expected return if the agents starts in state s and always acts according to policy $\pi(s)$. The return considered in value functions is always the infinite-horizon discounted return.

$$V_\pi(s)=\mathbb{E}_\pi[G_t|s_t=s] \quad (2.9)$$

On-policy action-value functions $Q_\pi(s,a)$ (2.10) gives the expected return if the agent starts in state s takes an action a and after that acts always according to policy $\pi(s)$.

$$Q_\pi(s,a)=\mathbb{E}_\pi[G_t|s_t=s,a_t=a] \quad (2.10)$$

The difference between action-value function and state-value function is defined as the action advantage function $A_\pi(s,a)$ (2.11).

$$A_\pi(s,a)=Q_\pi(s,a)-V_\pi(s) \quad (2.11)$$

The goal in Reinforcement Learning is to find the policy that maximizes the expected return. A policy that optimizes the expected return is defined optimal policy (2.12).

$$\pi_{opt}(s)=\underset{\pi}{argmax}\,\mathbb{E}_\pi[G_t] \quad (2.12)$$

Optimal value function $V_{opt}(s)$ (2.13) gives the maximum expected return if the agent starts in state s and always acts according to the optimal policy $\pi_{opt}(s)$.

$$V_{opt}(s)=\underset{\pi}{max}\,V_\pi(s) \quad (2.13)$$

Optimal action-value function $Q_{opt}(s,a)$ (2.14) gives the maximum expected return if the agent starts in state s , takes an action a and after that always acts according to the optimal policy $\pi_{opt}(s)$.

$$Q_{opt}(s,a)=\underset{\pi}{max}\,Q_\pi(s,a) \quad (2.14)$$

Optimal policy achieves optimal value function (2.15) and optimal action-value function (2.16).

$$\pi_{opt}(s)=\underset{\pi}{argmax}\,V_\pi(s) \quad (2.15)$$

$$\pi_{opt}(s)=\underset{\pi}{argmax}\,Q_\pi(s,a) \quad (2.16)$$

Bellman equations break down the value function into the immediate reward plus the discounted future value. In other words: the reward of the starting state is the expected reward of that state plus the discounted reward of the next state. The Bellman equations can be applied both to on-policy value functions and optimal value functions. The expected return is referred to the next state s' sampled from the transition probability function P . The actions a are taken according the

policy $\pi(s)$. The State-value Bellman equation is (2.17) and the action-value Bellman equation is (2.19).

$$V_\pi(s)=\mathbb{E}_\pi[R_{(t+1)}+\gamma V_\pi(S_{(t+1)})|S_t=s] \quad (2.17)$$

$$Q_\pi(s,a)=\mathbb{E}_\pi[R_{(t+1)}+\gamma Q_\pi(S_{(t+1)},A_{(t+1)})|S_t=s,A_t=a] \quad (2.18)$$

The recursive update process can be further decomposed using (2.19) and (2.20) . In this way state-value Bellman expectation equation (2.21) and state-value Bellman expectation equation (2.22) are obtained.

$$V_\pi(s)=\sum_{(a\in A)}\pi(a|s)Q_\pi(s,a) \quad (2.19)$$

$$Q_\pi(s,a)=R(s,a)+\gamma\sum_{(s'\in S)}P(s'|s,a)V_\pi(s') \quad (2.20)$$

$$V_\pi(s)=\sum_{(a\in A)}\pi(a|s)(R(s,a)+\gamma\sum_{(s'\in S)}P(s'|s',a)V_\pi(s')) \quad (2.21)$$

$$Q_\pi(s,a)=R(s,a)+\gamma\sum_{(s'\in S)}P(s'|s,a)\sum_{(a'\in A)}\pi(a'|s')Q_\pi(s',a') \quad (2.22)$$

Similarly on what done for Bellman expectation equations, (2.21) and (2.22), the recursive update process can be further decomposed using (2.23) and (2.24) . Instead of computing the expectation policy we can use the maximum return during the updates. In this way state-value Bellman optimality equation (2.25) and state-value Bellman optimality equation (2.26) are obtained.

$$V_{opt}(s)=\max_{a\in A}Q_{opt}(s,a) \quad (2.23)$$

$$Q_{opt}(s,a)=R(s,a)+\gamma\sum_{(s'\in S)}P(s'|s,a)V_{opt}(s') \quad (2.24)$$

$$V_{opt}(s)=\max_{(a\in A)}(R(s,a)+\gamma\sum_{(s'\in S)}P(s'|s',a)V_{opt}(s')) \quad (2.25)$$

$$Q_{opt}(s,a)=R(s,a)+\gamma\sum_{(s'\in S)}P(s'|s,a)\max_{(a'\in A)}Q_{opt}(s',a') \quad (2.26)$$

A MPD (2.27) formally consists then on a tuple of five elements : a finite set of states S , a finite set of actions A , a state-transition probability function P , a reward function R , a discount factor γ .

$$M=\langle S,A,P,R,\gamma\rangle \quad (2.27)$$

2.2 Temporal Difference Learning

Temporal Difference (TD) learning is a model-free algorithm of reinforcement learning and learns directly from episodes of experience. Since TD learning is a model-free algorithm it does not need the knowledge of MDP transitions or rewards. The main characteristic of TD algorithms is that they can learn from incomplete episodes of experiences. The main concept behind TD learning is to update the value function $V_{\pi}(s)$ towards the so called TD target $R_{(t+1)}+\gamma V(S_{(t+1)})$. How much to update the value function is defined by the learning rate hyper-parameter.

The learning rate α determines how much newly acquired information overrides old information. If $\alpha=0$ the agent can learn nothing, it can only exploit what it already knows. If $\alpha=1$ the agent overwrites all the old information.

The simplest TD method makes the update to the value function (2.9) and action-value function (2.10) as shown in (2.29) and (2.30).

$$V(S_t)\leftarrow V(S_t)+\alpha[R_{(t+1)}+\gamma V(S_{(t+1)})-V(S_t)] \quad (2.29)$$

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha[R_{(t+1)}+\gamma Q(S_{(t+1)},A_{(t+1)})-Q(S_t,A_t)] \quad (2.30)$$

2.2.1 Q-learning

Q-learning algorithm is an off-policy TD method. It was introduced by Chris Watkins in 1989 [3], and a convergence proof was presented by Watkins and Dayan in 1992 [4].

Q-learning method does not use the current policy to choose the next action, it estimates $Q_{opt}(s,a)$ out of the best $Q(s,a)$ values, independently of the current policy $\pi(s)$.

Each Q-value referred to the state-action pair is stored in a table called Q-table, which is updated every iteration. The scheme of the Q-learning algorithm is shown in Figure 2.6.

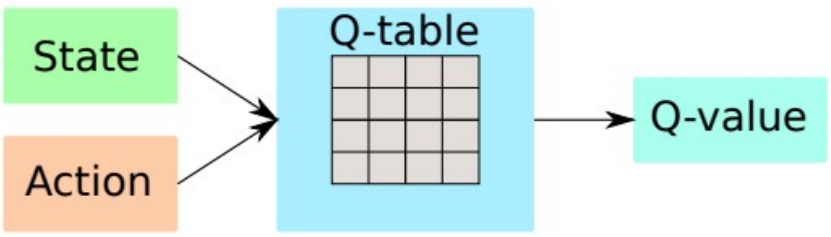

Figure 2.6 Q-learning Algorithm Scheme

One of the most chosen policy used to determine the action the agent takes is the $\epsilon-greedy$ algorithm. The $\epsilon-greedy$ algorithm makes use of the exploration-exploitation trade-off. The parameter $\epsilon\in[0,1]$ is used to decide how to choose an action. If ϵ is equal to 1 the actions are totally random, to allow exploration. If ϵ is equal to 0 the actions are always chosen to maximize the expected return. The $\epsilon-greedy$ algorithm changes the ϵ over the episodes. Normally, ϵ is equal to 1 at the beginning, when it is more important to explore. To have $\epsilon=1$ it means to choose a random action at 100% probability. At each episode ϵ is lowered using a function which can be for instance linear or quadratic. When ϵ is lower than 1 the probability to choose a random action is $(1-\epsilon)$ in percentage. When ϵ reaches value 0 there is 0% probability to choose a random action, so the actions are chosen to exploit the environment, which means to maximize the future reward. The Q-learning algorithm is shown in Table 2.1.

Q-learning *[Table 2.1]*

Initialize $Q(s,a)$, for all $s\in S$, $a\in A(s)$ arbitrarily,

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using $\epsilon-greedy$ policy

Take action A observe R, S'

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha[R_{(t+1)}+\gamma \max_{(a\in A)} Q(S_{(t+1)},A_{(t+1)})-Q(S_t,A_t)]$$

$S\leftarrow S'$

Until S is terminal

2.3 Deep Reinforcement Learning

The use of tables is the simplest method to store learned estimates, like values, policies and models. In simple environments the use of tabular representation is not an issue, but increasing the complexity of the environment there are memory and computational constraints. A function approximator can be used to store and retrieve estimates. Neural networks are frequently used as function approximators. Neural networks are computing systems inspired by biological neural networks. A deep neural network is defined as a neural network with multiple layers between input and output. The use of deep neural network in Reinforcement Learning is identified as a separate paradigm defined as Deep Reinforcement Learning.

2.3.1 Deep Q Network

In 2013 a system that combined a deep convolutional neural network with a variant of the Q-learning was introduced [5].

The variant of the Q-learning used in the paper made use of stochastic gradient descent to update the weights. The algorithm in [5] made use of a non linear function approximator, the neural network, with weights θ as a Q-network. To use a Q-table would have been impractical, since the state-action pairs would have been too memory expensive, for a computer, to fit all of them on a table. The policy used in [5] to choose the actions was the $\epsilon-greedy$ policy. Furthermore, it was used an experience replay mechanism [6] to mitigate the problems related to correlated data and non-stationary distribution. In fact, with experience replay, each step of experience $(S_t, A_t, R_t, S_{(t+1)})$ is potentially used in more than one weight update, this means more data efficiency. Moreover, learning from consecutive samples is not efficient because of the correlations between the samples. Randomizing the samples breaks the correlations and reduces the variance of the updates. The experience replay mechanism randomly samples previous transitions and twinkles the training distribution over the past behaviors. The Deep Q-Network algorithm stores the last N experience tuples in the replay memory D and samples tuples uniformly from D to perform the updates. In the paper [5] the agent learned how to play Atari games using the pixels of the screen as observation. The reward was represented by the change in the game score.

In Figure 2.7 the scheme of the Deep Q-learning algorithm is shown . Needs to be highlighted that in the Deep Q-learning algorithm, the number of the outputs of the neural network is equal to the number of the actions of the action set.

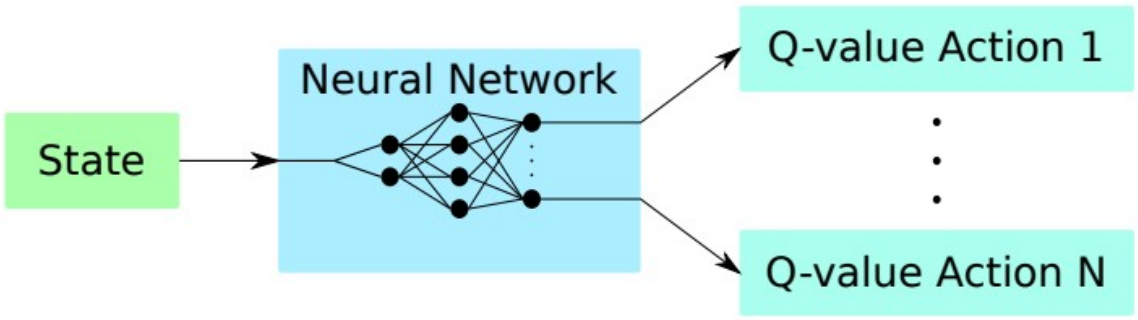

Figure 2.7 Deep Q-learning Algorithm Scheme

The Q-network was trained minimizing the sequence of loss functions $L_i(\theta_i)$ (2.31) updated every iteration i .

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a\ \rho(s,a))}[(y_i - Q(s,a;\theta))^2] \quad (2.31)$$

In equation (2.31), y_i (2.32) is the target for iteration i and $\rho(s,a)$ in (2.31) is a probability distribution over states and actions.

$$y_i = \mathop{\mathbb{E}}_{(s'\ \epsilon)}[r + \gamma \max_{(a')} Q(s',a';\theta_{(i-1)})|s,a] \quad (2.32)$$

Differentiating the loss function (2.31) with respect to the weights the gradient $\nabla_{(\theta_i)} L_i(\theta_i)$ (2.33) is obtained.

$$\nabla_{(\theta_i)} L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a\ \rho(s,a);s'\ \epsilon)}[((r + \gamma \max_{(a')} Q(s',a';\theta_{(i-1)}) - Q(s,a;\theta_i))\nabla_{(\theta_i)} Q(s,a;\theta))] \quad (2.33)$$

Instead of computing the full expectations of $\nabla_{(\theta_i)} L_i(\theta_i)$, stochastic gradient descent may be used to optimize the loss function. The Deep Q-learning algorithm is shown in Table 2.2.

Deep Q-learning with Experience Replay *[Table 2.2]*

Initialize Replay memory D to capacity N
Initialize action-value function Q with random weights
    Repeat (for each episode):
        Initialize s_1 and $\phi_1 = \phi(s_1)$
        Repeat (for each step of episode):
            Choose a_t from $\phi(s_t)$ using $\epsilon - greedy$ policy
            Take action a_t observe r_t , $s_{(t+1)}$
        Store transition $(\phi_t, a_t, r_t, \phi_{(t+1)})$ in D
        Sample random mini-batch of transition $(\phi_t, a_t, r_t, \phi_{(t+1)})$ from D
        Set $y_i = r_j + \gamma \max_{(a)} Q(\phi_{(j+1)}, a_j; \theta_j)$ for non-terminal $\phi_{(j+1)}$
            Set $y_i = r_j$ for terminal $\phi_{(j+1)}$
        Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta_j))^2$

The performances of the Deep Q-learning algorithm are comparable and in some cases superior to human performances on Atari games [7].

2.4 Improvements in Deep Q Network

Since Deep Q-Network was introduced, many improvements were achieved concerning the algorithm. Two improvements are described: the use of Double Q-learning algorithm and the Duel Architecture.

2.4.1 Double Deep Q Network

An improvement to the Deep Q-learning is represented by [8]. In this paper the main idea is to generalize the concepts behind Double Q-learning algorithm [9] using function approximation.

The double Q-learning algorithms main aim is to prevent the selection of overestimated values. In fact, since in both standard Q-learning and Deep Q-learning the same values are used to select and evaluate the actions the risk is to estimate overoptimistic values.In the Double Q-learning algorithm [9] two value functions are used, in order to decompose the evaluation from the selection. In Double Q-learning algorithm assigning each experience randomly to update one of the two value function, two value functions are learned. Using this method there are two sets of weights θ and θ' . A comparison between the target in Q-learning and Double Q-learning is done using (2.34) and (2.35). A decomposition between the evaluation and the selection in Q-learning is done in (2.34).

$$Y_t^Q = R_{(t+1)} + \gamma Q\left(S_{(t+1)}, \underset{(a)}{argmax}\, Q\left(S_{(t+1)}, a; \theta_t\right); \theta_t\right) \quad (2.34)$$

In the in Double Q-learning, (2.34) can be written as in (2.35).

$$Y_t^{DoubleQ} = R_{(t+1)} + \gamma Q\left(S_{(t+1)}, \underset{(a)}{argmax}\, Q\left(S_{(t+1)}, a; \theta_t\right); \theta_t'\right) \quad (2.35)$$

In (2.35) the value of the greedy policy is estimated according to the first set of weights θ . To evaluate the value of this policy the second set of weights θ' is used. However the role of the two sets of weights can be switched. In the Double Deep Q-learning the weights of the second network θ' are substituted by the weights of the target network θ^T . The update of the Double Deep Q-learning algorithm (2.36) is formally equal to the Double Q-learning (2.35) with the difference of the weights of the target network.

$$Y_t^{DoubleQ} = R_{(t+1)} + \gamma Q\left(S_{(t+1)}, \underset{(a)}{argmax}\, Q\left(S_{(t+1)}, a; \theta_t\right); \theta_t^T\right) \quad (2.36)$$

The changes in the algorithm to adapt Double Q-learning in Deep Q-learning are minimal.

It is sufficient to substitute (2.37) to the y_i used for non-terminal state in the Deep Q-learning algorithm (Table 2.2).

$$y_i = r_j + \gamma Q\left(\phi_{(t+1)} \underset{(a)}{argmax}\, Q\left(\phi_{(j+1)}, a_j; \theta_j\right); \theta_j^T\right) \quad (2.37)$$

The Double Deep Q-learning algorithm outperform the Deep Q-learning algorithm, achieving new state of the art results in Atari games [8]

2.4.2 Duel Deep Q Network

A different proposal for the network architecture used in Deep Q-learning was to introduce the Dueling architecture [10] to make an explicit separation between the action advantaged and the state values. The Dueling architecture consists in two streams, one stream represents the value of a state $V(s)$ the other one represents the advantage of taking an action $A(s,a)$. This two streams are combined to produce a single output. In Figure 2.8 the architecture of a Standard Network and a Dueling Network is compared.

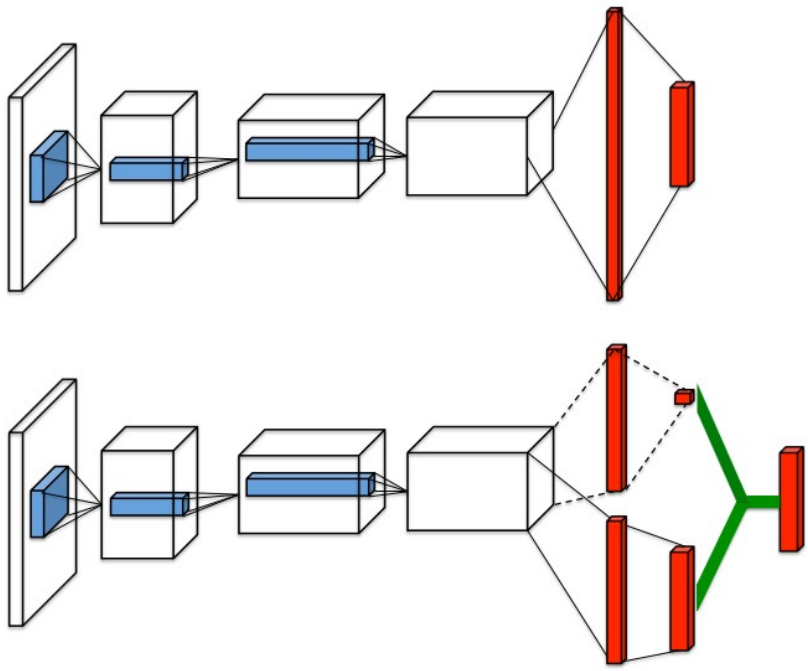

Figure 2.8 Standard Network (above) Dueling Network (below) [10]

The output of the Dueling network is a set of $Q(s,a)$. This output can be trained using an existing Deep Q-learning algorithm, without any limitation.

The aggregation layer, the last layer of the Dueling architecture network is computed by mean of the action advantage function $A_\pi(s,a)$ (2.11) the action-value function $Q_\pi(s,a)$ (2.10) and the value function $V_\pi(s)$ (2.9), with the needed adaptations due to the neural network. Two parameters that consider the streams divisions are used: α and β . Furthermore a parameter related to the convolutional neural network is used: ζ .

The equation used to determine the set of Q values is (2.38).

$$Q_\pi(s,a;\zeta,\alpha,\beta)=V_\pi(s;\zeta,\beta)+\left(A_\pi(s,a;\zeta,\alpha)-\max_{(a)} A_\pi(s,a;\zeta,\alpha)\right) \quad (2.38)$$

The Dueling architecture, implemented together with other Deep Q-learning algorithm's improvements, leads to better performances in the Atari games [10].

3. Autonomous Driving

An autonomous car can drive itself from Point A to Point B with no manual input from the driver. The vehicle uses a combination of cameras, radar systems, sensors, and global positioning system (GPS) receivers to determine its surroundings and uses artificial intelligence to determine the quickest and safest path to its destination. Mechatronic units and actuators allow the "brain" of the car to accelerate, brake, and steer as necessary.
(Morgan Stanley Research Global (2013). Autonomous Cars: Self-Driving the New Auto Industry Paradigm, p.14)

A classification system for autonomous driving applications was done by SAE international [12]. The first document that classified the six levels of autonomous driving was released in 2014. Since then the document was revised twice and the last version was done in 2018.
The classification defines six levels of driving automation, from 'no automation' to 'full automation'. The levels are organized on how much a human driver or an autonomous driving system are monitoring the environment. A full manual system corresponds to a total human responsibility for driving actions. In full automated driving system the actions are totally delegated to the the autonomous car. The levels are described in detail:

- *Level 0*: autonomous driving features can provide warnings or limited assistance to the driver. When the features are engaged, the driving actions are responsibility of the human driver. Example: lane departure warning.
- *Level 1*: autonomous driving features control steering or brake/acceleration actions to support the driver. When the features are engaged, the driving actions are responsibility of the human driver. Examples: lane centering or adaptive cruise control.
- *Level 2*: autonomous driving features control steering and brake/acceleration actions to support the driver. When the features are engaged, the driving actions are responsibility of the human driver. Examples: lane centering and adaptive cruise control.
- *Level 3*: autonomous driving features can control the vehicle under limited condition, when requested the human driver has to take control. When the features are engaged, the driving actions are taken by the autonomous system. Example: traffic jam chauffeur.

- *Level 4*: autonomous driving features can control the vehicle under limited condition. When the features are engaged, the driving actions are taken by the autonomous system. Example: local driver-less taxi.
- *Level 5*: autonomous driving features can control the vehicle in all conditions. When the features are engaged, the driving actions are taken totally by the autonomous system. Example: drive without a human driver or occupants.

Table 3.1 resumes the SAE classification of driving automation levels.

SAE levels of Driving Automation [12] *Table 3.1*

<table>
<tr><td>LEVEL 0</td><td>LEVEL 1</td><td>LEVEL 2</td><td>LEVEL 3</td><td>LEVEL 4</td><td>LEVEL 5</td></tr>
<tr><td colspan="3">Human is driving when Features are engaged</td><td colspan="3">Human not driving when Features are Engaged</td></tr>
<tr><td colspan="3">Human Supervision constantly Requested</td><td>Human Supervision may be Requested</td><td colspan="2">Human Supervision never Requested</td></tr>
<tr><td>Warning Support</td><td>Steering or Acceleration/ Brake Support</td><td>Steering and Acceleration/ Brake Support</td><td colspan="2">Drive the vehicle under Limited Conditions</td><td>Drive the vehicle under All Conditions</td></tr>
<tr><td>Lane Departure Warning</td><td>Lane Centering or Adaptive Cruise Control</td><td>Lane Centering and Adaptive Cruise Control</td><td>Traffic Jam Chauffeur</td><td>Local Driver-less Taxi</td><td>Absence of Human Driver</td></tr>
</table>

Experiments regarding autonomous driving have been conducted since the beginning of the last century.

In 1925, Houdina Radio Control modified a 1926 Chandler implementing an antenna to get radio impulses. The impulses actuated electric motors responsible for the car movements. The radio-controlled car was called the ‘American Wonder’. This is the first example of a driver-less application. In 1936 a demonstration of the ‘American Wonder’, renamed for the occasion ‘Phantom Auto’, took place in the streets of Fredericksburg [13].

In 1960s the United Kingdom’s Transport and Road Research Laboratory made tests for a driver-less Citroen DS . The car movements were influenced by magnetic cables integrated in the road. Studies continued until the 1970s with cruise control devices, but stopped when the funds for the experiments were withdraw [14].

In 1977 Sudayuki Tsugawa and his colleagues at Tsukuba Mecahanical Engineering Laboratory realized the first truly autonomous car [11]. This car used two cameras as sensors and a computer for signal processing. It achieved to drive at a speed of 30 km/h tracking white street markers on the road.

In 1996, professor Alberto Broggi of University of Parma started the ARGO project. The project used a modified version of a Lancia Thema to follow standard lane marks on an unmodified highway. The project achieved 94% of fully autonomous driving on a 1200 miles journey across northern Italy [11] [15].

In 2004 the US Department's Defense Advanced Research Project Agency (DARPA) started to organize long distance competitions for autonomous vehicle. This competition was held from 2004, on a open desert, to 2007, on a urban course. The competitions meant to develop technologies for military use, but considered also civilian applications [11].

In 2009 Google started to develop a driver-less car program using a mix of Google Maps data, radars and lidar [16] [17]. In 2012 a Toyota Prius tuned with Google driver-less technology, was the first self-driven car to get a driving license. The license was released by the Nevada Department of Motor Vehicles [18]. To get the license the Google car had to drive 22 km in a test route in Las Vegas, Nevada [17].

In 2017 Audi A8 was the first production car to achieve level 3 of autonomous driving [19]. Audi A8 used a 3D lidar system, cameras and ultrasonic sensors as inputs for the AI.

In march 2018 was reported the first fatal crash between a pedestrian and an autonomous car [20]. This accident highlighted that there is still much work to do before implementing safely the autonomous driving technology.

Nowadays, the trend is to invest in autonomous driving technology and all the most important car-makers are working in this direction. The most important challenge is related to safety and also security from hackers attack. However, the safety standards currently used in automotive industries, such as ISO 26262, may be immature for autonomous driving applications [21]. The latest trend involves the use of machine learning techniques to support the AI of the autonomous car [21] [23].

3.1 Components of Autonomous Driving System

A modern autonomous driving system pipeline consists in various tasks. Two main modules can be identified: scene understanding and decision making. The first main module aims to provide information from heterogeneous sources and to organize them for being processed. The second module aims to process the information taken from the first module and to provide a driving policy feasible for the vehicle controls. Five blocks can be defined: sensors, localization, mapping, planning, control. Figure 3.1 shows the components of an autonomous driving system pipeline.

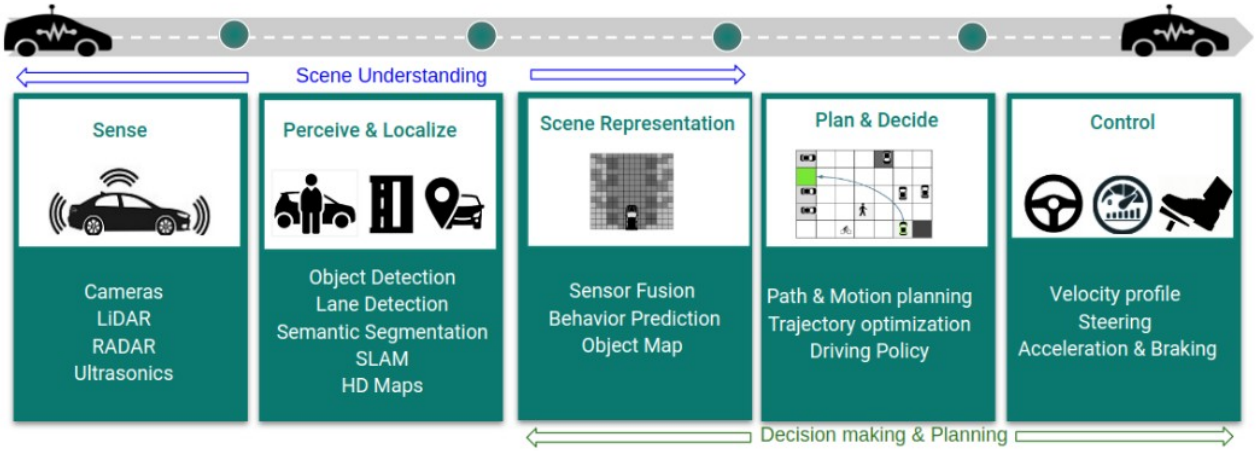

Figure 3.1 A modern Autonomous Driving System pipeline [23]

The first block of the pipeline in Figure 3.1 regards sensors. Autonomous driving systems make use of large number of sensors to get information from the environment. The most common used sensors are cameras, radar, lidar, ultrasonic.

Camera is a passive sensor since it not emits signals for measurements. It returns a video of the environment. It can be monocular or omni-directional, used for 360° vision. Cameras can sense colors which are important for traffic light recognition. Cameras performances are affected by illumination conditions and it is difficult to extract from videos depth information. Nonetheless, cameras are used by 2D computer vision algorithms, an established technology with a wide field of applications.

Radar is a sensor that emits radio waves to the environment and measures the time these waves take to bounce back. In this way radars can get information on the depth of the environment. It is a

simple and established technology, cheap and lightweight. Radars are not affected by illumination conditions, but they can interfere with other systems since they are active sensors.

Lidar is a sensor that emits light waves to the environment and measures the time these waves take to bounce back. The working principle is similar to radar. The main difference between radar and lidar is the accuracy over the distance: lidar is much more accurate in short distance and can recognize even small objects. Illumination condition do not interfere with lidar, but weather adverse conditions can affect the measurements.

Ultrasonic sensors emit ultrasonic waves to the environment and measures the time these waves take to bounce back. The working principle is similar to radar. Ultrasonic sensors can return information on the depth of the environment. They are used mostly as parking sensors.
Table 3.2 resumes the main sensors used in autonomous driving and their characteristics.

Sensor Characteristics [22] *Table 3.2*

Sensor	Affected by illumination	Affected by weather	Can sense colors	Application Range	Accuracy based on Distance	Size	Cost
Camera	Yes	Yes	Yes	-	-	Small	Low
Radar	No	No	No	High	Medium	Small	Medium
Lidar	No	Yes	No	Medium	High	Large	High
Ultrasonic	No	No	No	Short	Low	Small	Low

The second block of the pipeline in Figure 3.1 focuses on the perception of the surroundings. To perceive the environment is a very important task for autonomous driving systems. There are different applications that have as objective to extract information from the environment, the most important ones are object detection and semantic segmentation.

Object detection focuses on identifying and localizing objects. In general, the objects can be static or dynamic. In autonomous driving applications the features of high accuracy and low computational cost are of primary importance. The detection needs to operate faster than real time to allow as much time as possible for planning and control [21]. The state of the art methods for object detection use deep convolutional neural networks. Two main methods can be identified: one uses a single network to detect object location and make predictions simultaneously; the other main

method has two distinct stages: first regions of interest are determined in an image, then these regions are categorized by classifier networks [21].

Semantic segmentation classifies each pixel of an image with a class label. Semantic segmentation is beyond the object detection and it is of great importance in autonomous driving applications. The identified objects can have different trajectories and behaviors and they must be differentiated. To differentiate instances of the same class *instance segmentation* is used [21].

To merge the information collected from different sensors is crucial to create a representation of the environment, useful to make predictions. An example of application that uses more than one sensor is *road detection*. To identify and understand lane merges and intersections is one of the challenges of the road detection. Information from camera and radar/lidar are necessary to understand and represent the roads. information about colors are taken from cameras, these information are useful to understand lane marks and signals. Information about the depth of the road are taken from radar/lidar.

To have a good representation of the environment is the first step of the *decision making* module. *Mapping* process is useful to localize the position of other vehicles and objects in the area surrounding the Ego-vehicle. To better map the environment, sensor information can be used and integrated with data from high definition maps [23].

Planning purpose is to generate commands for the autonomous car. In autonomous driving applications the most used algorithms for planning are the rapidly-exploring random trees (RRT) [23]. RRT is an algorithm used to find non-convex, high dimensional spaces by randomly building a space-filling tree. The tree is built starting from the current configuration and it is enriched with random samples from the search space.

Control is the last part of the autonomous driving pipeline. A controller determines how much to brake, to steer and to accelerate to obtain the planned trajectory. Control methods used in autonomous drive applications are based on classical control theory. The controllers mainly used are PID, Proportional-Integral-Derivative, and MPC, Model Predictive Control. PID is a control loop that computes the difference between a desired set-point and a measured variable. MPC is a process control that controls a process while satisfying a set of constraints. The main difference between PID and MPC is that MPC can ‘predict’ the future and then anticipate the actions.

3.2 Deep Reinforcement Learning for Autonomous Driving

The development of Deep Reinforcement Learning field demonstrated that is possible to use this technology to face complex and high-dimensional problems. Recently, these algorithms were applied to autonomous driving tasks such as: controller optimization, path planning, dynamic path planning, trajectory optimization, motion planning. Moreover, these algorithms can be applied to learn: high level driving policies for complex navigation tasks; scenario-based policy learning for highways; to drive using Inverse Reinforcement Learning from data of expert pilots; policies to achieve risk estimation and to assure safety [23].

To successfully use Deep Reinforcement Learning algorithms in applications related to autonomous driving, it is necessary to design conveniently State Space, Action Space and Reward Functions.

State Space frequently used for autonomous vehicle applications consists in: position and velocity of the Ego-vehicle, occupancy grid in the Ego-vehicle surroundings, path curvature and trajectory [23]. It is useful to integrate the information from sensors to contextualize the information given to Ego-vehicle.

Action Space for autonomous vehicle applications is different according to the algorithm used. In general continuous-valued actuators are used. Deep Reinforcement Learning algorithms which use continuous Action Spaces, for instance Deep Deterministic Policy Gradient, fit with this kind of controls. Deep Reinforcement Learning algorithms which use a discrete Action Space, for instance Deep Q-Network, need to discretise the Action Space dividing uniformly the range of continuous actuators. The main drawback of a discrete action space is that it may lead to unfeasible trajectories. A trade-off between the number of discrete steps that may be computed and the number of discrete steps needed for stable and smooth control must be considered.

Many *Reward Functions* can be used to train Deep Reinforcement Learning agents in autonomous vehicle applications. The choice of the reward function is related to the task taken into account. Commonly a combination of Reward Functions is used, since autonomous driving is a multi-objective problem [23].

3.2.1 Deep Reinforcement Learning for Autonomous Driving Tasks

There are some examples that can be predented to describe the application of Deep Reinforcement Learning in autonomous driving tasks. Deep Reinforcement Learning is applied in tasks such as: motion planning, overtaking, intersections, lane keeping, lane change, autonomous braking, collision mitigation.

Motion Planning task regards the estimation of a path between the ego-vehicle and the destination. To plan a path in a dynamic environment is a key problem in autonomous driving. In [24] the authors proposed a method that uses the A* path finding algorithm. To learn an heuristic function a Deep Q-Network was used on a set of image-based input.

Overtaking tasks were studied in [25]. The Authors proposed a method to learn an overtaking policy maintaining a steady speed and ensuring safety avoiding crashes. A Multi-goal Reinforcement Learning framework was used. The algorithms used were Q-learning and double-action Q-learning.

Intersection/Merging tasks require the ego-vehicle to merge into highways or ramps negotiating the intersections. The authors of [26] addressed the highway merging task using a Deep Q-Network algorithm. The authors of [27] addressed the ramp merging task using a long short-term memory to model the interactive environment and a Deep Q-Network algorithm.

Lane Change tasks were studied in [28]. In the paper, the Ego-vehicle had to coordinate its actions with the surroundings to perform a safe lane change maneuver. In [28] both state space and action space were treated as continuous. A Q-function approximator, with a closed-form greedy policy, was used to improve the computation efficiency of a Q-learning algorithm.

Lane Keep tasks were studied in [29]. In the paper, the environment was simulated using TORCS, a car racing simulator. The problem was studied by dividing it in two approaches. Discrete actions were considered and a Deep Q-network algorithm was used. Continuous actions were considered applying a Deep Deterministic Actor-Critic algorithm.

Autonomous Braking was studied in [30]. The proposed autonomous braking system had to decide when to brake using the information obtained from sensors. The action space was divided in four brake actions, from no braking to strong braking. The policy was learned using a Deep Q-network.

Imminent Collision Mitigation was investigated in [31]. This paper, taking inspiration from [30], studied the situations where the collision is most probable and difficult to avoid, corresponding to a small Time To Collision. Two policies were considered, both used controls which allowed the Ego-vehicle to steer and brake. The two trained policies were compared to a baseline where the only allowed action was to brake. The Reinforcement Learning algorithm used was a Deep Deterministic Policy Gradient.

3.3 CARLA Simulator

CARLA (CAR Learning to Act) is an open-source simulator for autonomous driving research. It is designed for development, training and validation of autonomous urban driving systems, including perception and control [32].

Figure 3.2 Example of CARLA environment in four different weather conditions [32]

CARLA simulates a dynamic world providing state of the art rendering quality, the graphic engine is built in Unreal Engine 4. In Figure 3.2 an example of the rendered environment is given. Realistic physics and vehicle dynamics is provided using PhysX Vehicle SDK.

The PhysX Vehicle SDK models vehicles as collections of sprung masses. Each sprung mass represents a suspension line with associated tire and wheel data. The representation of the vehicle as a rigid body is given in Figure 3.3.

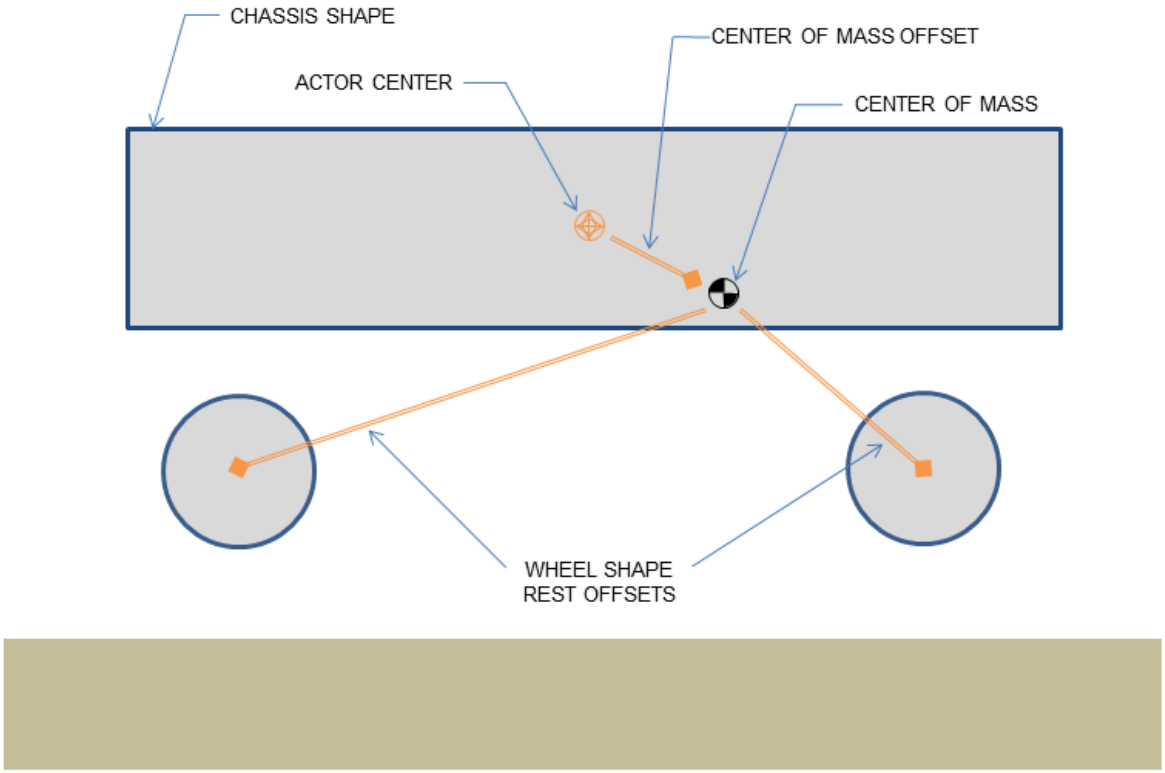

Figure 3.3 Vehicle representation as a rigid body actor [33]

These collections of sprung masses have a complementary representation as a rigid body actor. The mass, center of mass, and moment of inertia of the rigid body actor matches exactly the masses and coordinates of the sprung masses [33]. In Figure 3.4 a representation of the vehicle as a collection of sprung masses is given, the mass are M1 and M2.

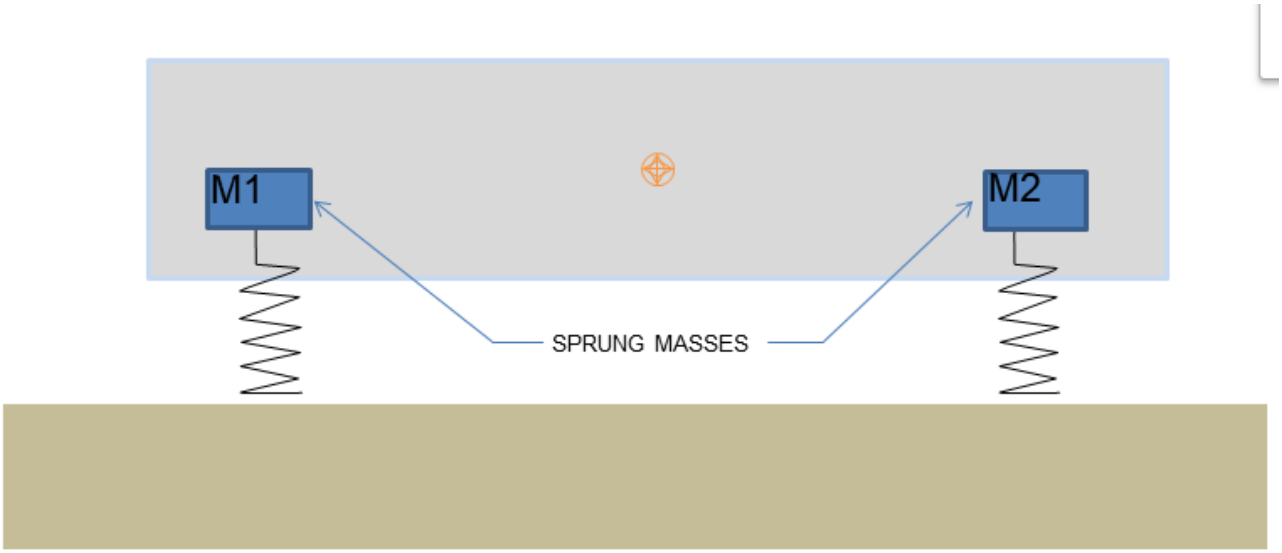

Figure 3.4 Vehicle representation as a collection of sprung masses of mass M1 and M2 [33]

CARLA, in order to implement a simple interface between the agent and the world, is designed using a server-client architecture. The server runs the simulation and renders the scene. It aims to achieve realistic results and it handles sensors, actors, physics, updates of the world-state. The client side consist in multiple client modules controlling the logic of the actors in the simulation. The client sends commands to the server and receives sensor readings. Commands related to steering, acceleration and braking in continuous range are used to control the vehicle. Meta-commands are used to control the server and to reset the simulation. Meta-commands also can change properties of the environment and sensor suite.

The environment simulated in CARLA consists in 3D models of static objects and dynamic objects. Static objects are: buildings, vegetation, traffic signs and infrastructure. Dynamic objects are vehicles and pedestrians. Different blueprints are available for every object.

CARLA includes also a vast choice of sensors and detectors. Their type, position and orientation is specified by the client.

3.4 Euro NCAP Test Protocol – AEB VRU systems

Euro NCAP, European New Car Assessment Programme, is a voluntary safety rating system. The standards and tests provided by Euro NCAP help to increase the safety of European automaker's car. Since car-to-pedestrian is one of the most occurring accident, Euro NCAP has designed a test protocol to check the performances of Autonomous Emergency Braking controls against Vulnerable Road Users. The last available version of the protocol to test Autonomous Emergency Braking systems is version 3.0.2, July 2019 [34]. The protocol specifies test procedure for pedestrian and bicyclist. The Euro NCAP AEB test protocol was used as test reference by [30].The purpose of this thesis work is to train and test a Reinforcement Learning agent in order to pass euro NCAP testing protocols. In particular, three test types are taken in consideration: CPFA, CPNA, CPNC.

Car-to-Pedestrian Farside Adult 50% (CPFA-50) – In this test a vehicle travels forwards towards an adult pedestrian crossing its path. The pedestrian target runs from the farside of the sidewalk. The frontal structure of the vehicle strikes the pedestrian at 50% of the vehicle width when no braking action is applied [34].

Car-to-Pedestrian Nearside Adult 25% (CPNA-25) – In this test a vehicle travels forwards towards an adult pedestrian crossing its path. The pedestrian target walks from the nearside of the sidewalk. The frontal structure of the vehicle strikes the pedestrian at 25% of the vehicle width when no braking action is applied [34].

Car-to-Pedestrian Nearside Child 50% (CPNC-50) – In this test a vehicle travels forwards towards a child pedestrian crossing its path. The pedestrian target is hid behind an obstruction and runs from in the nearside of the sidewalk. The frontal structure of the vehicle strikes the pedestrian at 50% of the vehicle width when no braking action is applied [34].

4. Algorithm Tests

This thesis work aims to investigate the implementation and the performances of a reinforcement learning agent in an autonomous driving application. The chosen application regards an emergency situation, where rapid choices can decide life or death of a vulnerable road user. Autonomous cars, to be safely implemented and to be perceived safe, have to react fast and precisely to avoid pedestrian collisions. Taking as reference the tragic accident of [20], where a pedestrian was killed by an autonomous car, it was chosen to recreate a situation where a pedestrian is in danger.

Starting from the Euro NCAP AEB-VRU testing protocol [34], a test was recreated in CARLA simulator to check the Autonomous Emergency Braking control and the Autonomous Emergency Steering control. It was investigated if the agent learns to recognize emergency situations and how the agent performs in avoiding them. During the tests the Ego-vehicle was running at constant velocity down a straight road. A pedestrian spawned on the right sidewalk and could cross the street depending on a certain Time To Collision. The Ego-vehicle should avoid the collision according to the controls under test.

If the emergency braking system was under test, and if the pedestrian was crossing the street, the Ego-vehicle should brake completely before the crash. If the pedestrian was not crossing the street, the Ego-vehicle should not brake at all and it should continue running.

During the test of the emergency steering control, if the pedestrian was crossing, the Ego-vehicle should avoid hitting the pedestrian by performing an emergency maneuver. If the pedestrian was just standing on the sidewalk, the Ego-vehicle should continue running without changing direction.

The reinforcement learning algorithm used in this thesis work was a Double Deep Q-Network with a Dueling Architecture, commonly called D3QN. It uses discrete actions, similarly to the Deep Q-Network described in paragraph 2.3.1. The two improvements in the Deep Q-Network implemented in this work, have been proved to lead to better results, as described in paragraph 2.4.1 and 2.4.2.

4.1 Experiment Settings

In this paragraph an overview is given regarding the settings of the experiment. The tests implemented in this thesis work, and their parameters are inspired by [34]. In particular, three tests types described in [34] were taken in consideration: CPFA, CPNA, CPNC.

The values of the Target Velocity of the Ego-vehicle and the Time To Collision were chosen according to the settings of the three tests described in paragraph 3.4. The Time To Collision and the Target Velocity influenced the choice of the Spawning Positions of the actors in the tests. The road where to set the tests and consequently the simulator town were chosen according to the Spawning Positions. The choice of the Ego-vehicle controls was influenced by the algorithm used for the reinforcement learning agent and the controls under test. Since cameras were used as main sensors, the input given to the reinforcement learning agent was a sequence of frames. It was necessary to get information from the environment to define the terminal states, so collision and lane invasion sensors were used. Since the main sensor used by the agent is the camera and according to the test settings described in [34], the choice of the weather conditions in the simulation was considered. The pedestrian parameters, like walking velocity and type of pedestrian, were inspired by the ones described in [34].

4.1.1 Time to Collision and Target Velocity

Time To collision (TTC) and the Ego-vehicle *Target Velocity* are important parameters of the test.
To give some variability to the experiment, a range of possible values for these parameters was defined. The probability distribution of the values for these parameters was uniform. The range of values chosen for these parameters influenced the choice of the Ego-vehicle blueprint and the test map. The range of values chosen for these variables was inspired by the test described in the Euro NCAP protocol for lateral crash with pedestrians [34].

The Ego-vehicle Target Velocity value represents the steady velocity the Ego-vehicle maintains during the test. The Euro NCAP testing protocol considers the Target Velocity in the range between 10 km/h and 60 km/h [34].

The TTC is the remaining time before the Ego-vehicle collides with the pedestrian target, assuming both are traveling at steady velocity. The Euro NCAP testing protocol considers a Time To Collision equal to 4 s [34].

Since the test was set in an urban scenario, a Target Velocity in the range between 20 km/h and 60 km/h, with interval steps of 5 km/h, was chosen. To choose the TTC range used in this thesis some considerations were made. TTC equal to 4 s was chosen as maximum value. To choose the minimum value for TTC it was decided to consider the braking capability of the Ego-vehicle. Therefore, the braking performances of the car models implemented in CARLA were considered. The Nissan Micra blueprint was chosen. The Nissan Micra model requires an average value of 12 m to completely brake at 60 km/h (16.2 m/s). The minimum TTC chosen for the test was equal to 1 s. The Nissan Micra braking performance gave margin to brake at the maximum Ego-vehicle Target Velocity chosen. Between the minimum and the maximum value of TTC an interval step of 0.25 s was chosen. Table 4.1 resumes the Time To Collision and Target Velocity values.

Time To Collision – Ego-vehicle Target Velocity *[Table 4.1]*

Parameter	Minimum	Maximum	Interval step	Probability of each value
Time To Collision	1 [s]	4 [s]	0.25 [s]	7.7 %
Ego-Vehicle Target Velocity	20 [km/h]	60 [km/h]	5 [km/h]	11.11 %

4.1.2 Map

The test, like the protocol described in [34], consists in a vehicle which travels forward towards a pedestrian target crossing its path. To choose the length of the road for the test, the range of values used for TTC and Ego-vehicle Target Velocity was considered. Since the vehicle should always have the needed space to accelerate and gain the Target Velocity, to set the pedestrian Start-Spawning Point at a minimum distance of 60 m from the Ego-vehicle Spawning Point was chosen.
The pedestrian Spawning Point was chosen according to (4.1):

$$x_p = x_s + d_t \quad (4.1)$$

Where x_s is the pedestrian Start-Spawning Point. It was chosen randomly in the interval from 60 m to 80 m with the step of 2 m. The trigger distance d_t was computed from the Ego-vehicle Target Velocity and TTC. According to the maximum value of all the parameters involved, the maximum pedestrian Spawning Point x_p is 144,8 m distant from the Ego-vehicle Spawning Point. The minimum pedestrian Spawning Point x_p is 65,4 m distant from the Ego-vehicle Spawning Point. To set the test on a straight plane road of almost 200 m was chosen. The chosen road had a slow lane, a fast lane to permit overtaking, and sidewalk where the pedestrian could spawn. From the cities provided by the simulator, to set the test in Town02 was chosen. The Ego-vehicle Spawning Point was set at coordinates $(x = 2\,m, y = 119.5\,m)$ of Town02.
In Figure 4.1 the map of the town chosen to set the test is shown. In Figure 4.1 the chosen road to set the test is highlighted in yellow.

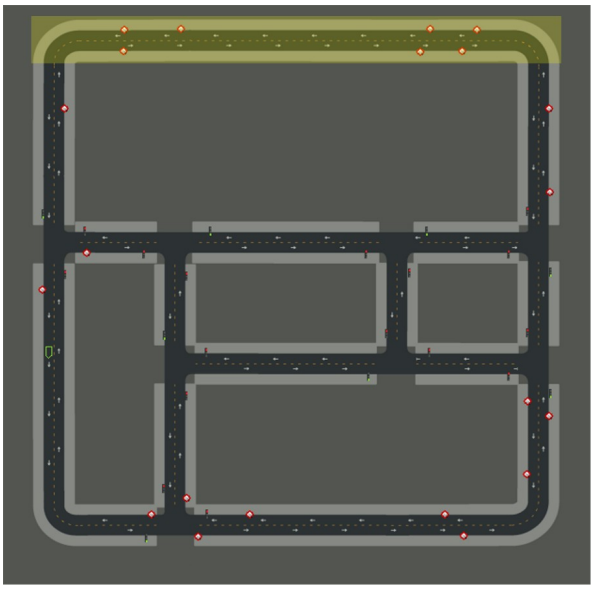

Figure 4.1 Town 02 Map [35]

4.1.3 Weather

Euro NCAP protocol [34] describes the requirements for the weather conditions needed for the test. The requirements are referred to the temperature of the ambient and the track, the speed and the direction of the wind, and the illumination of the ambient. The temperature requirements were not considered in the simulation, but wind speed and ambient illumination were taken into account. The test should be performed in dry condition to improve the horizontal visibility, so it was not simulated rain during the test. Wind speed should be at low intensity to minimize disturbances, so it was chosen to put this parameter equal to zero. Ambient illumination should be homogeneous and maximize the visibility of the environment. It should be avoided to perform the test facing direct sunlight or having it behind. It was chosen to recreate the conditions of a dry day at noon with almost no clouds to maximize the ambient illumination, while minimizing the shadows cast across the test area. Figure 4.2 shows a frame of the experiment during the CPNC test.

Figure 4.2 Frame of the Experiment during CPNC test

4.1.4 Ego-vehicle Controls

The vehicle controls in CARLA simulator deal with continuous values. The algorithm of reinforcement learning used in the experiment uses discrete actions. To make the action space compatible with the algorithm, discretization of the control values was needed.

Two kind of tests were run: Autonomous Emergency Braking test and Autonomous Emergency Steering test. Each test had its own controls. The Ego-vehicle could either brake or do nothing during the Autonomous Emergency Braking test. The Ego-vehicle could brake, turn left, turn right, or do nothing during the Autonomous Emergency Steering test. The agent could perform a discrete action every 0.07 s, and for this amount of time the last chosen action was repeated. The time to perform an action was chosen to correspond to 1 frame of the simulation, considering the simulation running at 15 Frames Per Second (FPS) in average.
Simulation running at 15 FPS, corresponds to lowest performance obtainable with the computer used for this work. The lowest performance was considered in order to make the results independent from the simulation speed.

To give an incremental value to the wheel steering angle, a steer amount parameter was introduced.
A value of 0.15 was chosen for the steering amount. This value allows to obtain a complete wheel steering in half a second of repetitive steering actions. This is a reasonable value to perform a realistic emergency steering action.
Table 4.2 resumes the Ego-vehicle controls for the two tests.

Ego-vehicle Controls *[Table 4.2]*

Autonomous Emergency Braking		**Autonomous Emergency Steering**	
0 = Do nothing	[ ]	0 = Do nothing	[ ]
1 = Brake	[1.0]	1 = Brake	[1.0]
-	-	2 = Turn left	[-1.0 * SteerAmount] SteerAmount = 0.15
-	-	3 = Turn right	[1.0 * SteerAmount] SteerAmount = 0.15

4.1.5 Ego-vehicle Sensors

Three different types of sensors were used to let the Ego-vehicle interact with the environment.
An RGB camera, a collision sensor and a lane invasion sensor were used. The collision sensor and the lane invasion sensor implemented in CARLA are event driven and they give details about the event in exam. The outputs of the collision sensor are the intensity of the collision and the actors involved in it. The lane invasion sensor returns the actor that crosses any road boundary and the type of boundary. For instance: crossing broken or solid line, invading the sidewalk. Although the output of these two sensors were not explicitly given as input to the agent, they helped to define the terminal states. The RGB camera implemented in CARLA returns a 32-bit BGRA image. The main information were given to the agent by a RGB camera attached in the front of the Ego-vehicle. The RGB camera used in the experiment had a resolution of 640x480 and a horizontal field of view of 110°. The camera output is processed before being given to the agent as input. The first operation of processing was to remove all the colors from the image, making it black and white. After that the image was resized to 50x50, making it easier to analyze. Figure 4.3 shows the processing steps. At last, to give the agent the sense of motion, a sequence of frames was stacked together, then fed to the agent as input. Four frames of the observation were stacked together. The reinforcement learning agent received the stacked and processed information to compute the best action according to the current situation. Figure 4.4 shows the input given to the agent.

Figure 4.3 Image Processing Steps – From the left: input image, black and white, resized image

Figure 4.4 Input given to the Reinforcement Learning agent consisting in four stacked frames

4.1.6 Pedestrian Target

The pedestrian parameters were inspired by the the Euro NCAP protocol for lateral crash with pedestrians [34]. The velocity of the pedestrian and his the distance from the center of the slow lane were chosen in order to recreate similar condition to the Euro NCAP tests CPNA (Car-to-Pedestrian Nearside Adult), CPFA (Car-to-Pedestrian Farside Adult), CPNC (Car-to-Pedestrian Nearside Child) [34]. The pedestrian target spawned in the right sidewalk according to (4.1), with probability of crossing the street of 50%. In Figure 4.5 the pedestrian models used in the simulation and the one used in the Euro NCAP tests [34] are compared. In CPFA and CPNA tests the model of the adult was used. In CPNC test the model of the child was used and a lane of parked cars spawned on both sides of the pedestrian target. The parked cars were meant to hide the pedestrian target. Each test (CPFA, CPNA, CPNC) had the same probability to occur. Table 4.3 resumes the parameters used for the pedestrian target.

Figure 4.5 Comparison of pedestrian targets –
On the left there are the CARLA models [36]
on the right there are the EuroNCAP targets[34]

Pedestrian Target Parameters *[Table 4.3]*

Test Type	Walking velocity	Pedestrian distance from slow-lane center	Type of pedestrian	Test probability	Walking probability
CPNC	5 [km/h]	3.5 [m]	Child	33.3%	50%
CPNA	5 [km/h]	3.5 [m]	Adult	33.3%	50%
CPFA	8 [km/h]	5 [m]	Adult	33.3%	50%

4.2 Autonomous Emergency Braking

The Autonomous Emergency Braking test implemented in this thesis work takes inspiration by [30]. In that paper it was proposed an autonomous braking system based on deep reinforcement learning. The autonomous braking system implemented in [30] had to decide if brake each time step considering the risk of collision. The autonomous braking system was designed for urban roads scenarios, where the vehicle faced a pedestrian that crossed the street at a random timing.

The observation given to the reinforcement learning agent was related to the position and the velocity of the vehicle and the position and velocity of the pedestrian. The action space used was discrete and consisted in four braking actions: no braking, weak, mid, strong. The reward function was chosen considering a trade-off between a behavior too conservative or too reckless. The reward achieved by the agent when the vehicle ran out the risk rapidly was balanced by the punishment in case of accident.

For the simulations the commercial software PreScan was used. The software could model vehicle dynamics in real time. The reinforcement learning algorithm used was a Deep Q-Network. The results of the paper showed that the agent could learn how to safely brake for Time To Collision values above 1.5 s. Moreover, additional tests were conducted on the trained agent. The test were performed according to [34]. The trained agent could pass the CPFA and CPNA tests without collisions.

4.2.1 AEB – Test description

The Autonomous Emergency Braking test started by spawning the Ego-vehicle at the position mentioned in paragraph 4.1.2. On the right sidewalk, the pedestrian target spawned according to (4.1). The initial phase of acceleration of the Ego-vehicle, needed to achieve the Target Velocity, was handled automatically. The agent took control when Ego-vehicle reached the position defined as trigger distance d_t . After that point the agent could either brake or do nothing.

There were two possibilities: if the pedestrian was crossing the street, the agent had to stop completely the Ego-vehicle before hitting the target. If the pedestrian was just standing on the sidewalk the Ego-vehicle should pass over for at least 20 m after the pedestrian spawning point. If the Ego-vehicle took too long to pass over, a time out ended the episode and it was considered failed. The Ego-vehicle could also fail by hitting the pedestrian or going off-road.

The off-road option should not be considered since the vehicle could not control the lateral motion. Since some bug occured, the off road condition was also taken into account. However, the incidence of this failure condition was less than 1% over 2000 episodes during training. The scheme in Figure 4.6 resumes the Autonomous Emergency Braking test Spawning Positions.

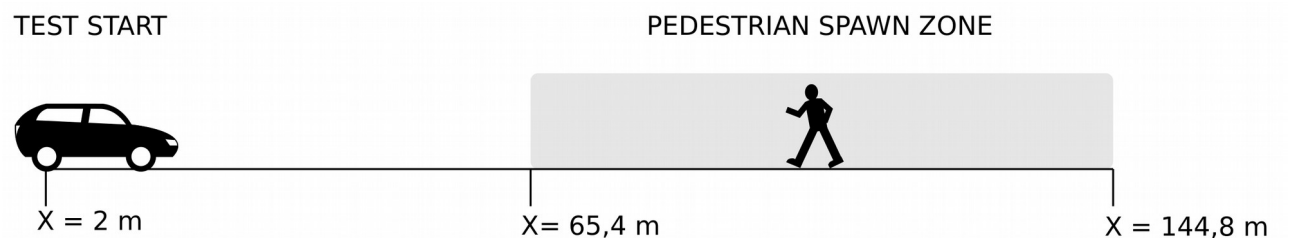

Figure 4.6 Autonomous Emergency Braking test Spawning Positions

4.2.2 AEB – Reward and Terminal States

The training was performed according to the scheme described in paragraph 4.2.1. In this paragraph the rewards and the terminal states are described in detail. A reward was given to the agent according to the terminal state, instead of accumulating rewards over the episode. In this way the agent was mainly focused on what was the end of the test. Moreover, the test was structured in such a way that the terminal state had much more importance than how the agent arrived to it.
The terminal states were:

• *Collision*

This terminal condition implied the Ego-vehicle to collide with anything in the simulation. There was also the possibility to record the "intensity" of the crash considering the velocity of the Ego-vehicle at the moment of the impact. This feature was not used and the same reward was given to the Ego-vehicle for any crash intensity.
The reward for this terminal state was set to $R=-100$.

• *Off road*

As mentioned in paragraph 4.2.1, this terminal condition should not happen. Even though, it occured anyway. For this reason, the situation in which the Ego-vehicle invades the sidewalk was considered. However, since this condition was not controllable by the Ego-vehicle, to not penalize this situation was chosen.
The reward for this terminal state was set to $R=0$.

• *Time up*

To discourage a too conservative behavior, in which the Ego-vehicle brakes to anything it detects as a threat, a limit in time for the episode was set. This time limit was chosen according to a trade off between the time to reach the most distant possible terminal point (that could be 164,8 m in agreement with the parameters of the test) even at the minimum Ego-vehicle Target Velocity and the possibility to perform some braking action. In fact, the agent needed to be able to avoid the time up condition even during the early phases of the training. Even though, the time up terminal condition should not be so easy to avoid. The time limit for each episode was chosen equal to 35 s.
The reward for this terminal state was set to $R=-100$.

• *Passed*

When the pedestrian was just standing on the sidewalk the Ego-vehicle should not brake at all and should pass on. It was of crucial importance for the Ego-vehicle to learn whether it was necessary to brake or not. Otherwise the Ego-vehicle could learn to brake whenever a pedestrian was detected indiscriminately. When the pedestrian was standing, which occurred 50% of the time, a complete brake action would not end the episode and leaded to a time up terminal state instead. To end correctly an episode when the pedestrian was standing, the Ego-vehicle should pass over 20 m the pedestrian position. The reward could be affected by a penalty P_v related to Ego-vehicle velocity. If the Ego-vehicle was slower than the Target Velocity of the episode by 10 km/h, the reward was halved. In this way the agent still received a positive reward for the episode, but should learn to avoid braking when it is not necessary.

The reward for this terminal state was set to $R = 100 - (50 * P_v)$.

• *Braked*

When the pedestrian was walking the Ego-vehicle should avoid the crash. Since the agent could not control the lateral direction of the Ego-vehicle, the only option was to stop completely. A complete brake action that stopped the Ego-vehicle could end the episode. The parameters of the test were designed to always give the Ego-vehicle the minimum braking distance needed to stop.

The reward for this terminal state was set to $R = 100$.

The Table 4.4 resumes the rewards and the terminal conditions.

Terminal States and Rewards – Autonomous Emergency Braking *[Table 4.4]*

Pedestrian Condition	Done Condition	Reward	Description	Notes
-	Collision	*-100*	*Collision with Anything*	-
-	Off Road	*0*	*Vehicle goes Out of the Road*	*Bug AEB*
-	Time Up	*-100*	*Episode timer over 35 s*	-
Standing Pedestrian	Passed	*100 – (50 * Pv)*	*Vehicle passes over Pedestrian*	-
Walking Pedestrian	Braked	*100*	*Vehicle Brakes Completely*	-

4.2.3 AEB – Hyper-parameter Values and Neural Network

In this paragraph all the hyper-parameter values and the neural network architecture are presented. A systematic grid search for the hyper-parameter values was not performed. It is conceivable that even better results could be obtained by systematically tuning the hyper-parameter values.

Table 4.5 resumes all the hyper-parameter values related to the Autonomous Emergency Braking test.

Hyper-Parameter Values – Autonomous Emergency Braking		*[Table 4.5]*
Hyper-Parameter	**Value**	**Description**
Buffer Size	1000000	*Replay Buffer Size*
Batch Size	64	*Mini Batch Size*
γ	0.99	*Discount Factor*
τ	0.05	*Soft Update of Target Parameters*
α	0.001	*Learning Rate*

Table 4.6 resumes the details about the neural network architecture related to the Autonomous Emergency Braking test.

Neural Network Architecture – Autonomous Emergency Braking				*[Table 4.6]*
Layer	**Kernel Size**	**Stride**	**Hidden Nodes**	**Activation Function**
CNN	*8*	*4*	*N/A*	ReLU
CNN	*4*	*2*	*N/A*	ReLU
CNN	*3*	*1*	*N/A*	ReLU
FC	*N/A*	*N/A*	*512*	ReLU
FC	*N/A*	*N/A*	*512*	-

In Figure 4.7 the neural network used for the Autonomous Emergency Braking test is shown. The stacked frames were fed to the neural network. First there were three convolutional layers, then two fully connected layers. The information passed through the neural network layers using a ReLU activation function. On the third convolutional layer, after the ReLU activation function, the information was reshaped. Then, the reshaped information was fed to the first fully connected layer. The output of this layer was split in two fully connected layers: one considered the Advantage, the other one the Value. The two fully connected layers for Advantage and Value had the same number

of nodes as the first fully connected layer. The Advantage and the Value were reunited in the aggregation layer. The output of the aggregation layer were the Q-values of the state action pairs.

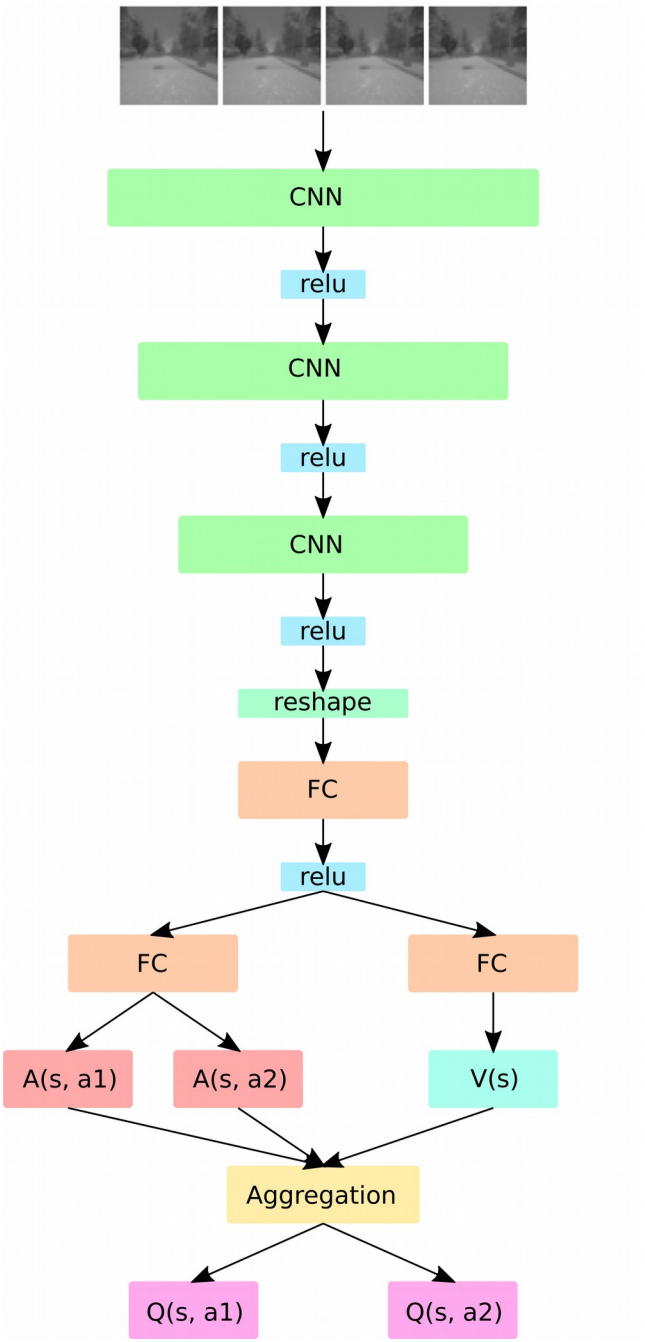

Figure 4.7 Autonomous Emergency Braking Neural Network

4.3 Autonomous Emergency Steering

The Autonomous Emergency Steering test implemented in this thesis work takes inspiration from work [31]. In that paper the role of reinforcement learning was investigated in reducing the severity of the collisions in situations when the collision is imminent. The autonomous driving system implemented in [31] could control the velocity and the steering of the vehicle. The autonomous steering test was designed for urban roads scenarios. The simulated environment consisted in vehicles and pedestrians that could both take part in the collisions.

The observation given to the Reinforcement Learning agent as input were images taken from a camera sensor. The action space used was continuous to allow more degrees of control. The actions allowed the agent to control the throttle and the steering of the vehicle. Two reward functions were used, they took into account the safety of both pedestrians and car occupants.

For the simulations the open source simulator CARLA was used . The reinforcement learning algorithm used was a Deep Deterministic Policy Gradient. The results of the paper showed that the agent could learn how to avoid any collision for Time To Collision values above 1.1 s.

4.3.1 AES – Test Description

The Autonomous Emergency Steering test started by spawning the Ego-vehicle at the position mentioned in paragraph 4.1.2. On the right sidewalk, the pedestrian target spawned according to (4.1). The initial phase of acceleration of the Ego-vehicle, needed to achieve the Target Velocity, was handled automatically. The agent took control when Ego-vehicle reached the position defined as trigger distance d_t . After that point the agent could either brake, do nothing, turn to left or right.

There were two possibilities: if the pedestrian was crossing the street, the Ego-vehicle had to perform an overtake action before hitting the pedestrian target. If the pedestrian was just standing on the sidewalk the Ego-vehicle should pass over to reach the Goal of 180 m. If the pedestrian was standing and the Ego-vehicle took too long to pass over, a time out ended the episode which was considered failed. The Ego-vehicle could also fail by hitting the pedestrian or going off road.

This task was certainly more challenging to learn than the one considered for the Autonomous Emergency Braking. This because the Ego-vehicle should learn from scratches how to keep the lane and how to perform a successfully overtake. The scheme in Figure 4.8 resumes the Autonomous Emergency Steering test Spawning and Goal Positions.

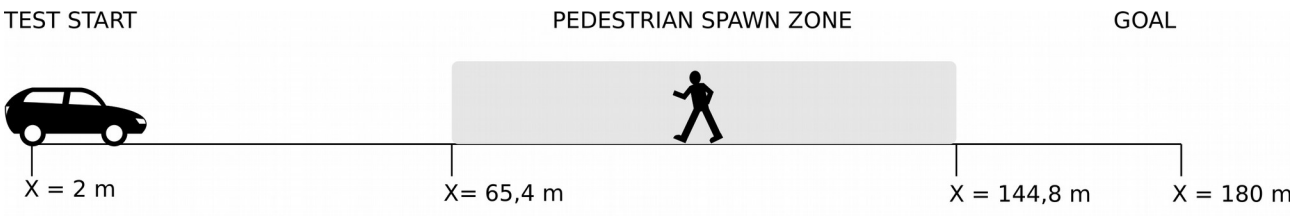

Figure 4.8 Autonomous Emergency Steering test Spawning and Goal Positions

4.3.2 AES – Reward and Terminal States

The training was performed according to the conditions described in paragraph 4.3.1. For this test, a reward accumulated throughout the episode was given to the agent. The reward was set proportional to the space covered in the longitudinal direction: T_x . This decision was made to give the agent the hint that the more space it covered in the longitudinal direction, the better. In addition to that, a penalty P_o was put if the Ego-vehicle was in the fast lane when it was not needed. To know if the Ego-vehicle had or not the need to stay in the fast lane, the pedestrian behavior was considered. If the pedestrian was just standing, the Ego-vehicle should not stay in the fast lane at all, otherwise the penalty was proportional to the meters spent in the fast lane. If the pedestrian was crossing the street, to overtake it the Ego-vehicle should stay in the fast lane. To prevent the Ego-vehicle from spending in the fast lane more time than needed a maximum distance spendable in the fast lane was set to perform the overtake: T_o . This distance T_o was proportional to the Target Velocity and the TTC. The distance to perform the overtake T_o was determined by (4.2).

$$T_o = (TTC + 2) * (Target\ Velocity) \quad (4.2)$$

The total reward was multiplied by a constant to make its maximum equal to 100. The terminal states were:

• *Collision*

This terminal condition implies the Ego-vehicle to collide with anything in the simulation. There was also the possibility to record the "intensity" of the crash considering the velocity of the Ego-vehicle at the moment of the impact. This feature was not used and the reward was given to the Ego-vehicle proportionally to the covered longitudinal distance and considering the overtaking penalty.

The reward for this terminal state was set to $R = (T_x - P_o) * 0.56$.

• *Off road*

This condition happened when the Ego-vehicle invaded the sidewalk. To discourage the Ego-vehicle to go off road, this state was set as terminal state. The reward was given to the Ego-vehicle proportionally to the covered longitudinal distance and considering the overtaking penalty.

The reward for this terminal state was set to $R = (T_x - P_o) * 0.56$.

• *Time up*

To discourage a too conservative behavior, in which the Ego-vehicle brakes to anything it detects as a threat, a limit in time for the episode was set. This time limit was chosen according to a trade off between the time to reach the most distant possible terminal point (that could be 180 m in agreement with the parameters of the test) even at the minimum Ego-vehicle Target Velocity and the possibility to perform some braking action. In fact, the agent needed to be able to avoid the time up condition even during the early phases of the training. Even though, the time up terminal condition should not be so easy to avoid. The time limit for each episode was chosen equal to 60 s. The reward was given to the Ego-vehicle proportionally to the covered longitudinal distance and considering the overtaking penalty.

The reward for this terminal state was set to $R=(T_x-P_o)*0.56$.

• *Passed*

When the pedestrian was just standing on the sidewalk the Ego-vehicle should not brake at all and it should pass on. The Ego-vehicle to end the episode and achieve this terminal state had to reach the position x = 180 m. The reward could be affected by a penalty P_v related to Ego-vehicle velocity. If the Ego-vehicle was slower than the Target Velocity of the episode by 10 km/h, the reward was reduced. In this way the agent still received a positive reward for the episode, but should learn to avoid braking when it was not necessary. Moreover the space covered in fast lane was penalized considering P_o .

The reward for this terminal state was set to $R=200-(50*P_v)-P_o$.

• *Overtook*

When the pedestrian was walking, the Ego-vehicle should avoid the crash. The only option to end the episode and achieve this terminal state was to reach the position x = 180 m. In this case, if the Ego-vehicle was slower than the Target Velocity of the episode by 10 km/h the reward was reduced. In this way the agent is still going to receive a positive reward for the episode, but should learn to avoid braking when it was not needed to. In addiction the space covered in fast lane was penalized considering P_o .

The reward for this terminal state was set to $R=200-(50*P_v)-P_o$.

The Table 4.7 resumes the rewards and the terminal conditions.

Terminal States and Rewards – Autonomous Emergency Steering			*[Table 4.7]*
Pedestrian Condition	**Done Condition**	**Reward**	**Description**
-	Collision	*(Tx – Po) * 0.56*	*Collision with Anything*
-	Off Road	*(Tx – Po) * 0.56*	*Vehicle goes Out of the Road*
-	Time Up	*(Tx – Po) * 0.56*	*Episode timer over 60 s*
Standing Pedestrian	Passed	*200 – (50*Pv) - Po*	*Vehicle reaches Goal Position*
Walking Pedestrian	Overtook	*200 – (50*Pv) - Po*	*Vehicle Overtakes Successfully*

4.3.3 AES – Hyper-Parameter Values and Neural Network

In this paragraph all the hyper-parameter values and the neural network architecture are presented. A systematic grid search for the hyper-parameter values was not performed. It is conceivable that even better results could be obtained by systematically tuning the hyper-parameter values.

Table 4.8 resumes all the hyper-parameter values related to the Autonomous Emergency Steering test. These hyper-parameter values were optimized with respect to the ones used for the Autonomous Emergency Braking test. In fact, because of the bigger action space an higher amount of memory was required.

Hyper-Parameter Values – Autonomous Emergency Steering		*[Table 4.8]*
Hyper-Parameter	**Value**	**Description**
Buffer Size	10000	*Replay Buffer Size*
Batch Size	32	*Mini Batch Size*
γ	0.99	*Discount Factor*
τ	0.001	*Soft Update of Target Parameters*
α	0.0001	*Learning Rate*

Table 4.9 resumes the details about the neural network architecture related to the Autonomous Emergency Steering test.

Neural Network Architecture – Autonomous Emergency Steering				*[Table 4.9]*
Layer	**Kernel Size**	**Stride**	**Hidden Nodes**	**Activation Function**
CNN	*8*	*4*	*N/A*	ReLU
CNN	*4*	*2*	*N/A*	ReLU
CNN	*3*	*1*	*N/A*	ReLU
FC	*N/A*	*N/A*	*512*	ReLU
FC	*N/A*	*N/A*	*512*	-

In Figure 4.9 the neural network used for the Autonomous Emergency Steering test is shown. The stacked frames were fed to the neural network. First there were three convolutional layers, then two fully connected layers. The information passed through the neural network layers using a ReLU activation function. On the third convolutional layer, after the ReLU activation function, the information was reshaped. Then, the reshaped information was fed to the first fully connected layer. The output of this layer was split in two fully connected layers: one considered the Advantage, the

other one the Value. The two fully connected layers for Advantage and Value had the same number of nodes as the first fully connected layer. The Advantage and the Value were reunited in the aggregation layer. The output of the aggregation layer were the Q-values of the state action pairs.

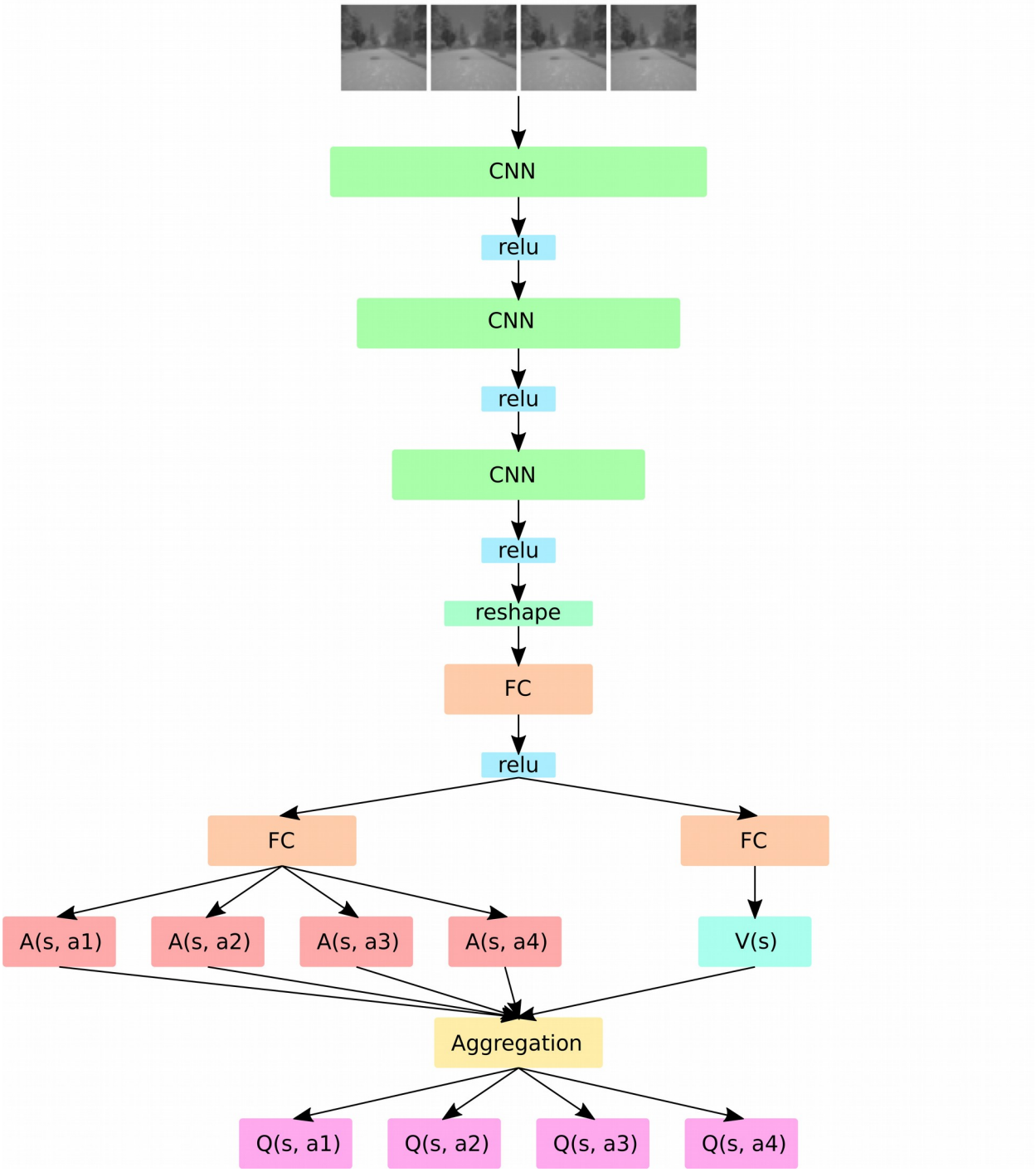

Figure 4.9 Autonomous Emergency Steering Neural Network

5. Algorithm Results

In this chapter the results of the Autonomous Emergency Braking test and Autonomous Emergency Steering test are presented. First, the results of the training are discussed, then the results of the testing. For each test a graph of the rewards over the episodes and a tabular resume are presented. For the testing results a bar chart is also presented. For the training, a graph of the epsilon parameter used is presented. The tables are organized to show the overall results and also the results with respect to every significant simulation parameter. The significant parameter chosen to investigate their relation with the results are: the test type, the pedestrian behavior, the Ego-vehicle Target Velocity, the Time To Collision. Moreover, the failure and the success reasons are considered in order to have a better idea of the simulation and also a statistic analysis. The test parameters are analyzed referring the total number of episodes for the training and to the

The agent could find a good policy for the Autonomous Emergency Braking control test. However, the agent final results were not good enough to pass the Euro NCAP AEB Test.

The agent could find a good policy for the Autonomous Emergency Steering control test. In fact, the agent final results were good enough to pass the Euro NCAP AEB Test.

5.1 Autonomous Emergency Braking

In this paragraph the results of the training and the testing of the Autonomous Emergency Braking test are presented and commented.

5.1.1 Training Results

The chart in Figure 5.1 shows the results of the training for the Autonomous Emergency Braking test. Figure 5.1 takes into account three scores: the score over all the episodes in gray, the average score over 10 episodes in blue and the average score over 100 episodes in orange. The maximum reward obtainable for an episode was equal to 100. Since it is more readable, it is commented the average score over 100 episodes.

The agent could achieve a reward above zero stably after almost 250 episodes and above 50 after 1000 episodes. After 1500 episodes the learning curve is stable on a reward between 60 and 75.

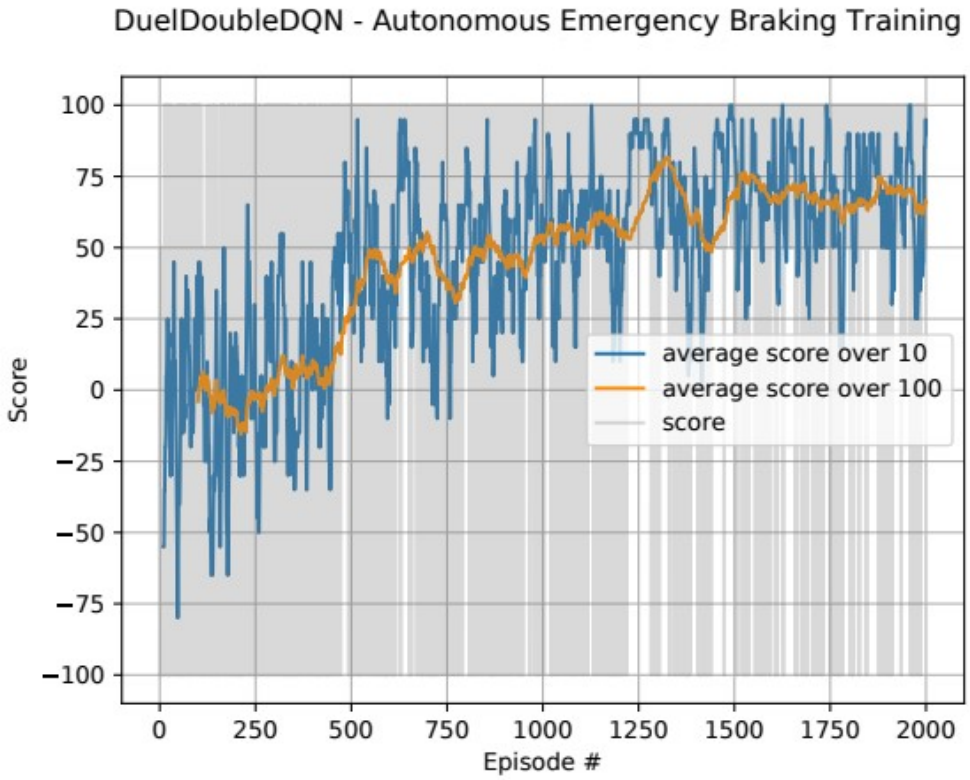

Figure 5.1 Autonomous Emergency Braking Training

The chart in Figure 5.2 shows the value of the epsilon parameter over the episodes of the Autonomous Emergency Braking training.

The trade off between exploration and exploitation was designed to have an exponential decrease of the epsilon parameter until the half of the total number of episodes. The lowest value of the epsilon parameter was set to 0.01 and it was reached in 1000 episodes.

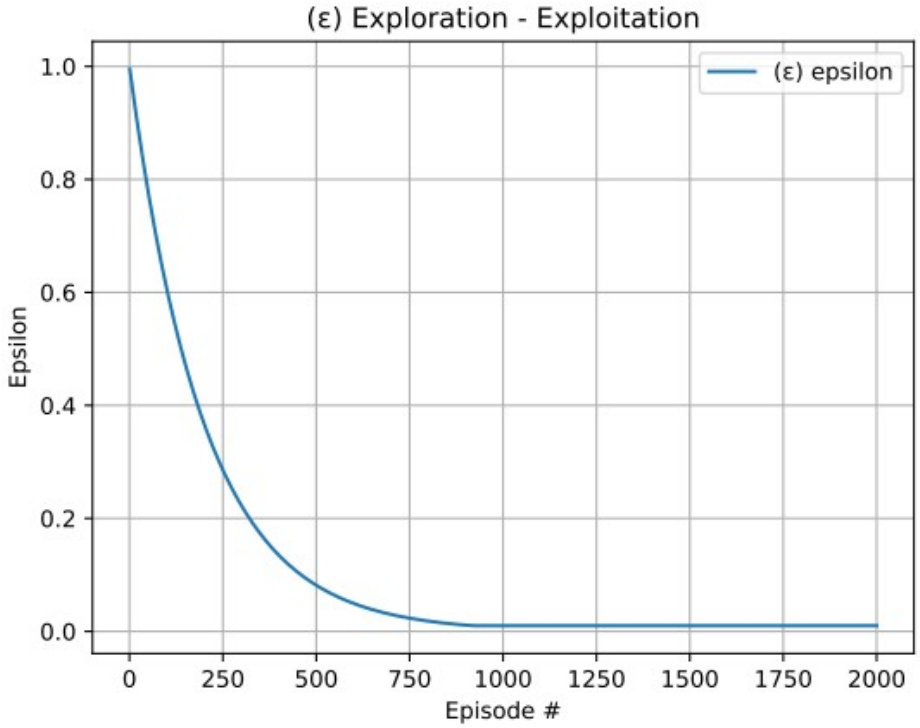

Figure 5.2 Autonomous Emergency Braking Exploration Exploitation Chart

The Table 5.1 gives a statistic analysis of the training. The incidence of *Off Road* terminal state, considerable a bug for this test, was below 1%. The success rate is significantly higher of the failure rate, this means that over the episodes a good policy was learned early in the training. This is confirmed by Figure 5.1. The success rate for reason *Passed* was higher than the one for reason *Braked*. An hypothesis to explain this is that the *Time Up* terminal state was easier to avoid than *Collision* terminal state. In fact, it is reasonable to think that it was easier to learn whether to brake than braking with a low TTC value.

Table 5.1			**Failure Reason**			**Success Reason**	
Episodes	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Braked**
2000	21.35%	78.65%	4.00%	17.05%	0.30%	44.95%	33.70%

5.1.2 Test Results

The chart in Figure 5.3 shows the results of the testing for the Autonomous Emergency Braking test. Figure 5.3 takes into account three scores: the score over all the episodes in gray, the average score over 10 episodes in blue and the average score over 100 episodes in orange. The maximum reward obtainable for an episode was equal to 100. Since it is more readable, the average score over 100 episodes is commented.

The agent obtains an average reward between 75 and 50, which is coherent with the last part of the training shown in Figure 5.1.

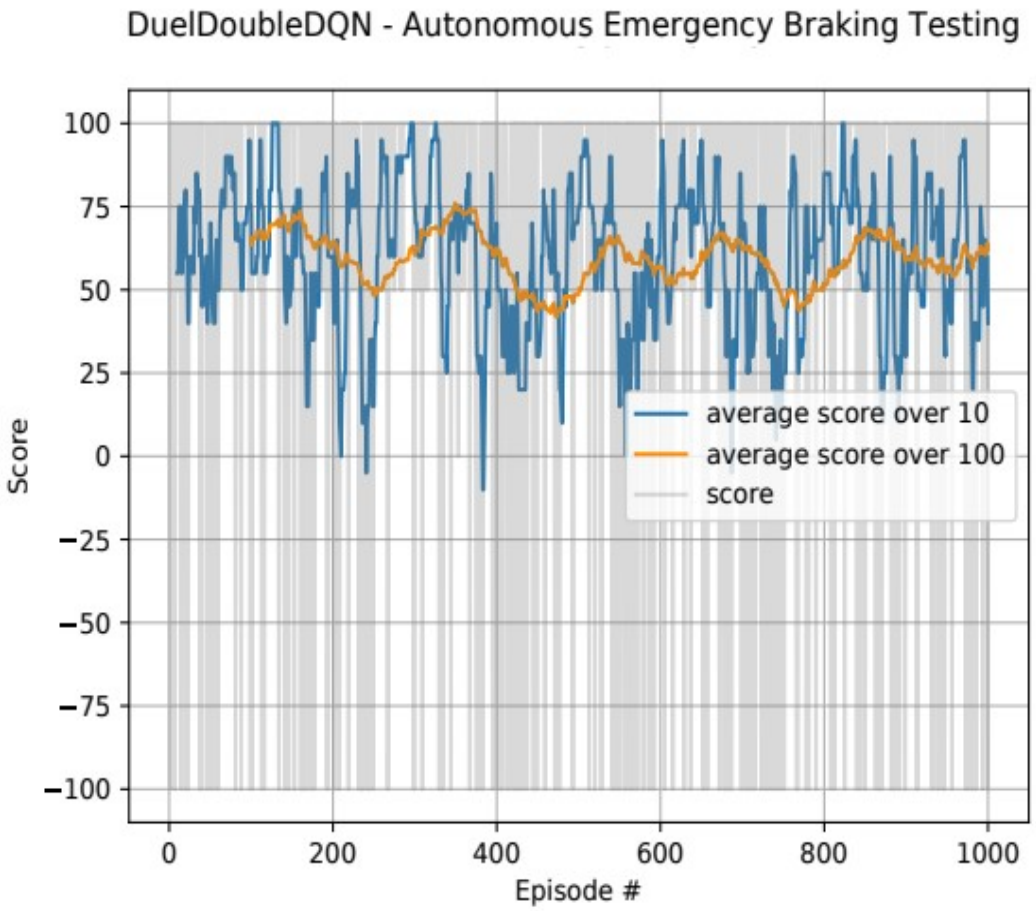

Figure 5.3 Autonomous Emergency Braking Testing

The Table 5.2 gives a statistic analysis of the testing. The success rate of the testing is higher than the success rate of the training, but not significantly. This can be related to the fact that the training episode were not enough to let the agent learn a better policy. Another reason can be found on the choice of the hyper-parameter values, their tuning can surely lead to better results. The *Time Up* terminal condition was always avoided. The *Off Road* terminal condition was possible, but its rate was lower than 1%.

Table 5.2			Failure Reason			Success Reason	
Episodes	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Braked**
1000	16.10%	83.90%	0.00%	15.60%	0.50%	48.30%	35.60%

The Table 5.3 and Figure 5.4, show the results of the agent in relation to the type of the test. There are no substantial differences between the three tests types.

Table 5.3			Failure Reason			Success Reason	
Test	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Braked**
CPFA	16.46%	83.54%	0.00%	15.84%	0.62%	50.00%	33.54%
CPNA	14.24%	85.76%	0.00%	13.95%	0.29%	48.55%	37.21%
CPNC	17.66%	82.34%	0.00%	17.07%	0.60%	46.41%	35.93%

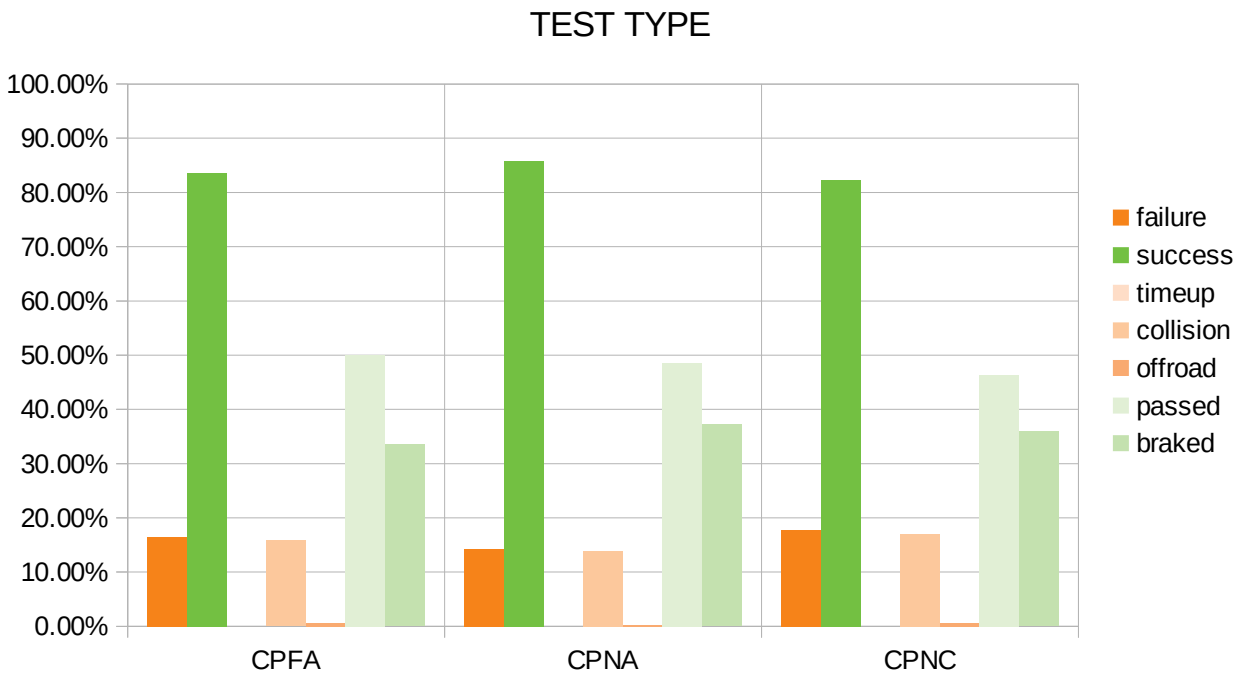

Figure 5.4 Results of Autonomous Emergency Braking Test – Test Type

The Table 5.4 and Figure 5.5, show the results of the agent in relation to the pedestrian behavior. The Standing behavior had 0% of failure rate. This means that the agent learned well when it was not needed to brake. Since the failure rate is only related to *Collision* terminal state, more training is surely needed to optimize braking performances.

Table 5.4			Failure Reason			Success Reason	
Pedestrian Behavior	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Braked**
Walking	31.14%	68.86%	0.00%	30.17%	0.97%	0.00%	68.86%
Standing	0.00%	100.00%	0.00%	0.00%	0.00%	100.00%	0.00%

Figure 5.5 Results of Autonomous Emergency Braking Test – Pedestrian Behavior

The Table 5.5 and Figure 5.6, show the results of the agent in relation to the Ego-vehicle Target Velocity. Tendentially, the *Failure* rate increases as the Target Velocity increases. The performance of the agent were not affected by the Target Velocity if the pedestrian behavior was Standing, as showed by the *Passed* rate.

Table 5.5			Failure Reason			Success Reason	
Target Velocity	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Braked**
20 Km/h	6.87%	93.13%	0.00%	6.87%	0.00%	48.85%	44.27%
25 Km/h	14.12%	85.88%	0.00%	14.12%	0.00%	42.35%	43.53%
30 Km/h	9.48%	90.52%	0.00%	9.48%	0.00%	48.28%	42.24%
35 Km/h	12.50%	87.50%	0.00%	10.42%	2.08%	42.71%	44.79%
40 Km/h	14.05%	85.95%	0.00%	14.05%	0.00%	47.11%	38.84%
45 Km/h	23.42%	76.58%	0.00%	23.42%	0.00%	45.05%	31.53%
50 Km/h	13.21%	86.79%	0.00%	10.38%	2.83%	55.66%	31.13%
55 Km/h	22.03%	77.97%	0.00%	22.03%	0.00%	50.85%	27.12%
60 Km/h	29.31%	70.69%	0.00%	29.31%	0.00%	51.72%	18.97%

Figure 5.6 Results of Autonomous Emergency Braking Test – Target Velocity

The Table 5.6 and Figure 5.7, show the results of the agent in relation to the Time To Collision. The *Collision* rate was below 20% for TTC higher than 1.75 s. Since that not even the maximum TTC leads to 0% *Failure* rate, the trained agent could not pass the Euro NCAP AEB Pedestrian Test.

Table 5.6			Failure Reason			Success Reason	
Time To Collision	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Braked**
1.0 s	46.67%	53.33%	0.00%	46.67%	0.00%	50.67%	2.67%
1.25 s	44.12%	55.88%	0.00%	44.12%	0.00%	51.47%	4.41%
1.5 s	29.00%	71.00%	0.00%	29.00%	0.00%	55.00%	16.00%
1.75 s	22.83%	77.17%	0.00%	20.65%	2.17%	47.83%	29.35%
2.0 s	16.13%	83.87%	0.00%	14.52%	1.61%	45.16%	38.71%
2.25 s	12.12%	87.88%	0.00%	12.12%	0.00%	50.00%	37.88%
2.5 s	4.60%	95.40%	0.00%	4.60%	0.00%	44.83%	50.57%
2.75 s	4.76%	95.24%	0.00%	4.76%	0.00%	38.10%	57.14%
3.0 s	2.53%	97.47%	0.00%	2.53%	0.00%	37.97%	59.49%
3.25 s	5.00%	95.00%	0.00%	5.00%	0.00%	51.25%	43.57%
3.5 s	2.78%	97.22%	0.00%	1.39%	1.39%	59.72%	37.50%
3.75 s	5.06%	94.94%	0.00%	3.80%	1.27%	50.63%	44.30%
4.0 s	11.69%	88.31%	0.00%	11.69%	0.00%	42.86%	45.45%

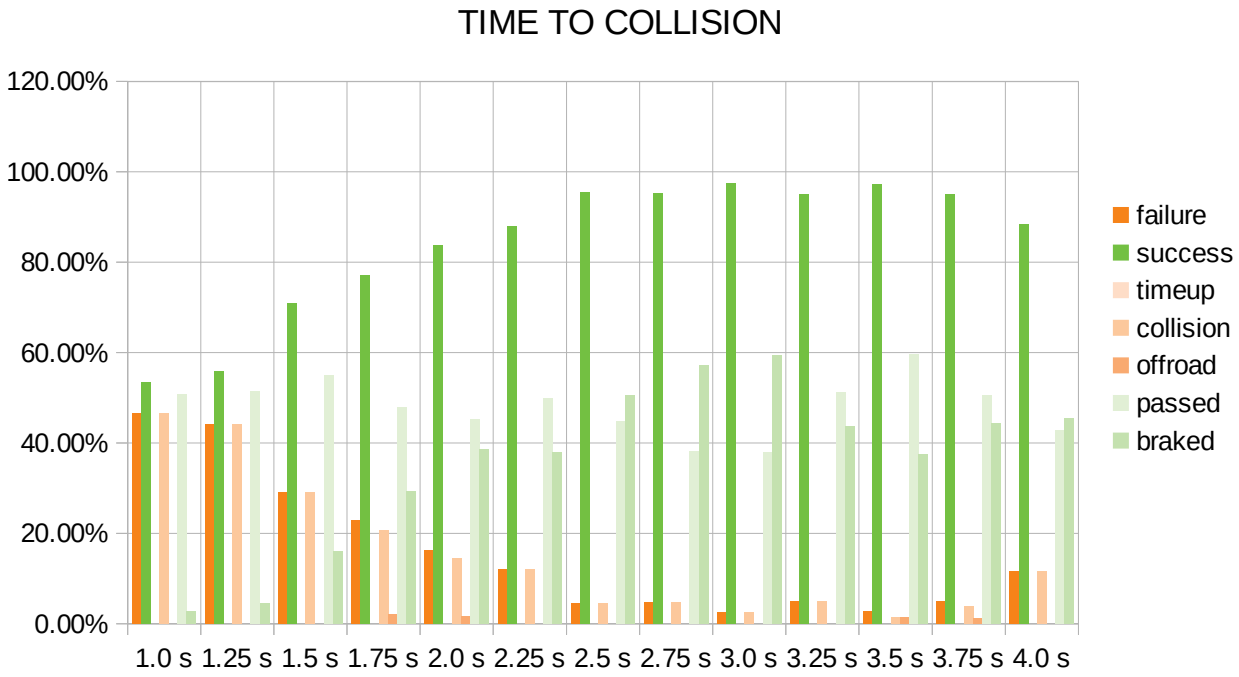

Figure 5.7 Results of Autonomous Emergency Braking Test – Time To Collision

5.2 Autonomous Emergency Steering

In this paragraph the results of the training and the testing of the Autonomous Emergency Steering test are presented and commented.

5.2.1 Training Results

The chart in Figure 5.8 shows the results of the training for the Autonomous Emergency Steering test. Figure 5.8 takes into account three scores: the score over all the episodes in gray, the average score over 10 episodes in blue and the average score over 100 episodes in orange. The maximum reward obtainable for an episode was equal to 200. Since it is more readable, the average score over 100 episodes is commented.

The agent could achieve a reward above 150 stably after 1500 episodes and above 100 after 1000 episodes. The average reward after 2000 episodes is 175. The agent between 750 and 1250 episodes had a rapid trend of learning.

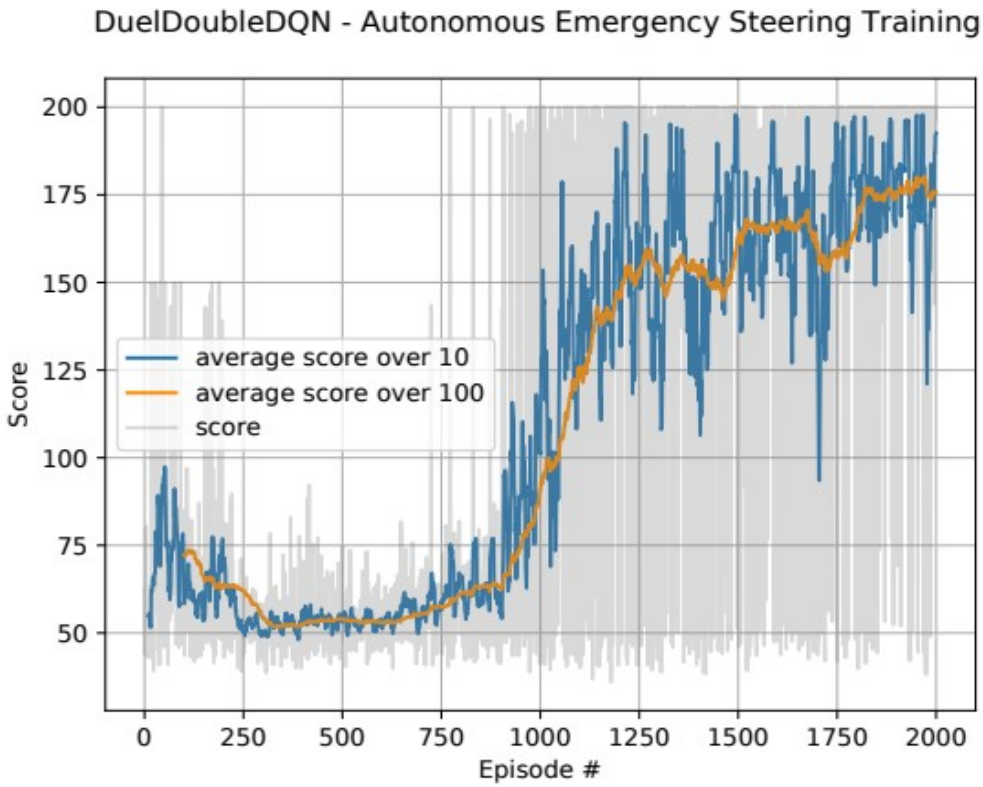

Figure 5.8 Autonomous Emergency Steering Training

The chart in Figure 5.9 shows the value of epsilon parameter over the episodes of the Autonomous Emergency Steering training.

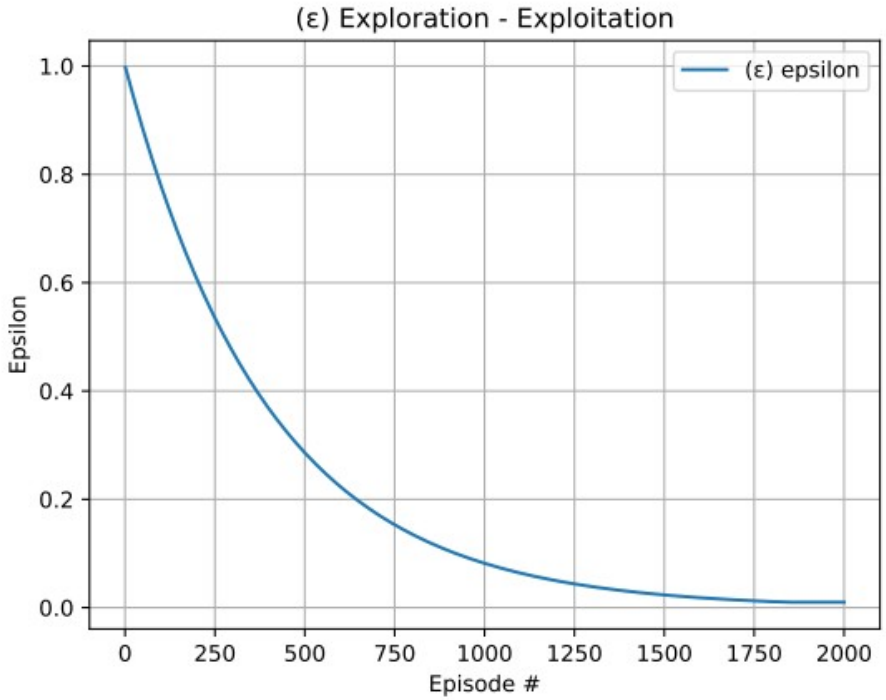

Figure 5.9 Autonomous Emergency Steering Exploration Exploitation Chart

The trade off between exploration and exploitation was designed to have an exponential decrease of the epsilon parameter for the total number of episodes. The lowest value of the epsilon parameter was set to 0.01 and it was reached in 2000 episodes.

The Table 5.7 shows a statistic analysis of the training. The failure rate is higher than the success rate. This can be confirmed by the chart of Figure 5.4 where the reward is below 200 for almost half of the training. The incidence of *Off Road* rate, was the highest among the terminal states. This implies that the most challenging task of the training was to keep the lane. This is confirmed by the *Time Up* rate which was below 1%. This implies that the agent went off road or crashed before reaching the time limit of the episode. The second highest rate is the *Collision* rate, then *Passed* then *Overtook*. Since the *Passed* rate is higher than *Overtook* rate is reasonable to think that learning to perform the overtook action was more difficult for the agent.

Table 5.7			**Failure Reason**			**Success Reason**	
Episodes	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Overtook**
2000	60.55%	39.45%	0.85%	25.15%	34.55%	24.85%	14.60%

5.2.2 Test Results

The chart in Figure 5.10 shows the results of the testing for the Autonomous Emergency Steering test. Figure 5.10 takes into account three scores: the score over all the episodes in gray, the average score over 10 episodes in blue and the average score over 100 episodes in orange. The maximum reward obtainable for an episode was equal to 200. Since it is more readable, is commented the average score over 100 episodes.

The agent obtained an average reward above 175 for all the testing. This is coherent with what seen in the latest part of the training chart of Figure 5.4.

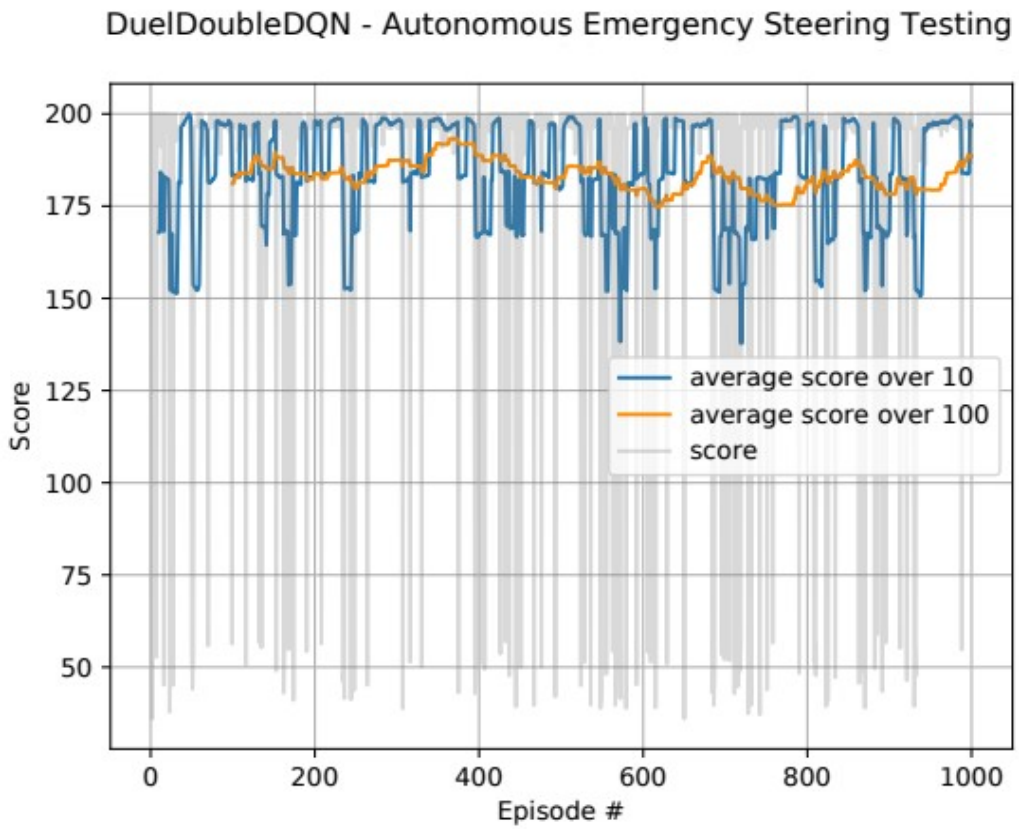

Figure 5.10 Autonomous Emergency Steering Testing

The Table 5.8 gives a statistic analysis of the training. The success rate is significantly higher than the failure rate. This means that the agent learned a good policy for the emergency steering. The *Off Road* rate was almost 0%, this means that the agent successfully learned how to keep a straight lane during the test. The *Time Up* rate is higher in the testing than the training, this means that the agent had more occasions to run out the time of the episode.

Table 5.8			Failure Reason			Success Reason	
Episodes	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Overtook**
1000	9.50%	90.50%	1.70%	7.70%	0.10%	48.10%	42.40%

The Table 5.9 and Figure 5.11, show the results of the agent in relation to the pedestrian type. The *Collision* rate of CPNA test is substantially less than other two tests.

Table 5.9			Failure Reason			Success Reason	
Test	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Overtook**
CPFA	12.42%	87.58%	1.24%	11.18%	0.00%	50.00%	37.58%
CPNA	6.10%	93.90%	1.16%	4.94%	0.00%	48.26%	45.64%
CPNC	10.18%	89.82%	2.69%	7.19%	0.30%	46.11%	43.71%

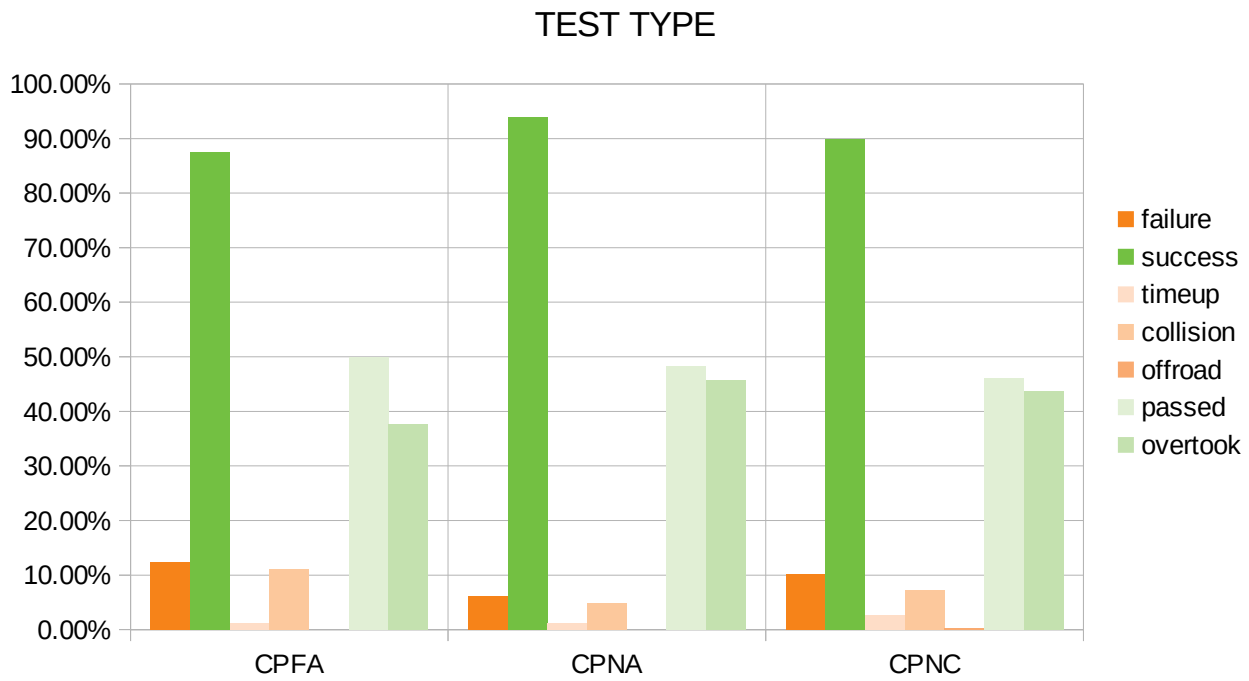

Figure 5.11 Results of Autonomous Emergency Steering Test – Test Type

The Table 5.10 and Figure 5.12, show the results of the agent in relation to the pedestrian behavior. The Walking behavior has less success rate compared to the Standing one. The Standing behavior has almost 0% of failure. The agent learned well when it was needed to pass away. The Waking behavior leads to more *Collision* and *Time Up* rates than Standing, this implies that the agent still needs training to optimize the overtaking performances.

Table 5.10			**Failure Reason**			**Success Reason**	
Pedestrian Behavior	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Overtook**
Walking	17.99%	82.01%	2.90%	14.89%	0.19%	0.00%	82.01%
Standing	0.41%	99.59%	0.41%	0.00%	0.00%	99.59%	0.00%

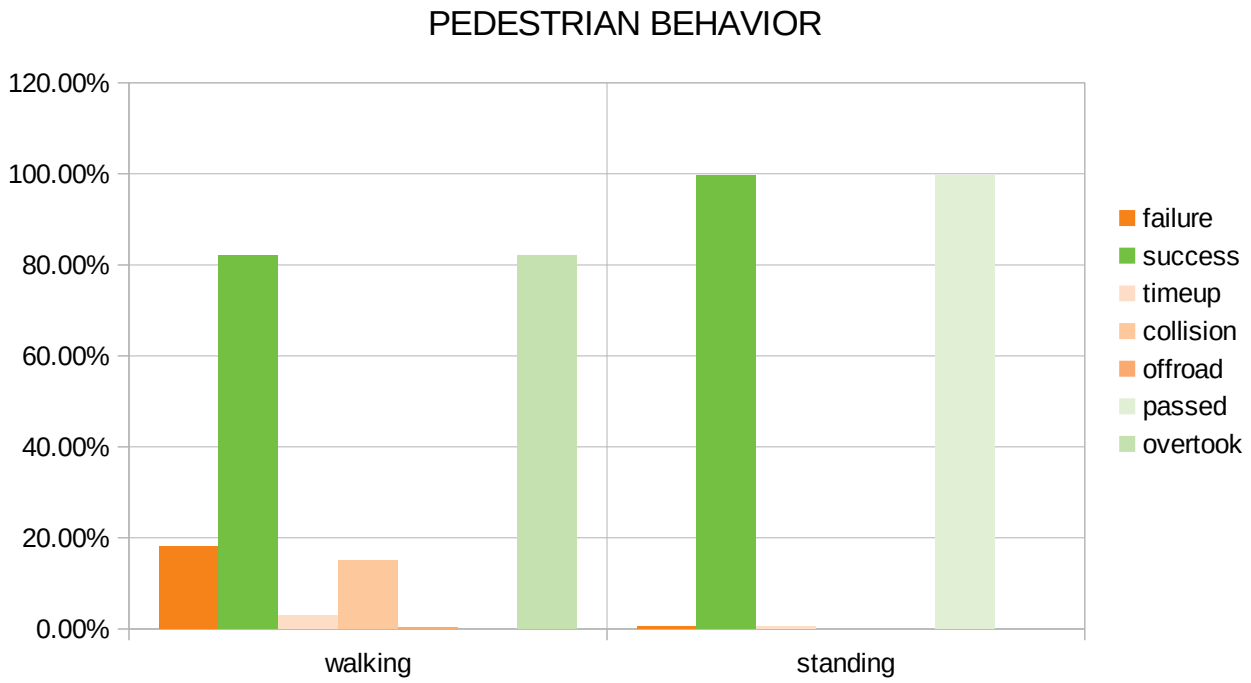

Figure 5.12 Results of Autonomous Emergency Steering Test – Pedestrian Behavior

The Table 5.11 and Figure 5.13, show the results of the agent in relation to the Ego-vehicle Target Velocity. The failure rate do not increase linearly as the Target Velocity increases. *Time Up* terminal state was more likely to happen for low values of Target Velocity. *Collision* Terminal state was more likely to happen for high values of Target Velocity.

Table 5.11			Failure Reason			Success Reason	
Target Velocity	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Overtook**
20 Km/h	12.21%	87.79%	4.58%	6.87%	0.76%	48.09%	39.69%
25 Km/h	12.94%	87.06%	8.24%	4.71%	0.00%	41.18%	45.88%
30 Km/h	9.48%	90.52%	2.59%	6.90%	0.00%	48.28%	42.24%
35 Km/h	6.25%	93.75%	0.00%	6.25%	0.00%	42.71%	51.04%
40 Km/h	7.44%	92.56%	0.83%	6.61%	0.00%	47.11%	45.45%
45 Km/h	8.11%	91.89%	0.00%	8.11%	0.00%	45.05%	46.85%
50 Km/h	6.60%	93.40%	0.00%	6.60%	0.00%	55.66%	37.74%
55 Km/h	12.71%	87.29%	0.00%	12.71%	0.00%	50.85%	36.44%
60 Km/h	9.48%	90.52%	0.00%	9.48%	0.00%	51.72%	38.79%

Figure 5.13 Results of Autonomous Emergency Steering Test – Target Velocity

The Table 5.12 and Figure 5.14, show the results of the agent in relation to the Time To Collision. The *Collision* rate tendentially increases as the Time To Collision decreases. For Time To Collision equal to 3.0 s and above 3.5 s the agent achieved 0% of *Collision* rate.

Table 5.12			Failure Reason			Success Reason	
Time To Collision	**Failure**	**Success**	**Time Up**	**Collision**	**Off Road**	**Passed**	**Overtook**
1.0 s	32.00%	68.00%	1.33%	30.67%	0.00%	50.67%	17.33%
1.25 s	16.18%	83.82%	0.00%	16.18%	0.00%	51.47%	32.35%
1.5 s	17.00%	83.00%	3.00%	14.00%	0.00%	54.00%	29.00%
1.75 s	14.13%	85.87%	3.26%	10.87%	0.00%	47.83%	38.04%
2.0 s	14.52%	85.48%	0.00%	14.52%	0.00%	45.16%	40.32%
2.25 s	9.09%	90.91%	3.03%	6.06%	0.00%	50.00%	40.91%
2.5 s	8.05%	91.95%	3.45%	4.60%	0.00%	44.83%	47.13%
2.75 s	3.17%	96.83%	1.59%	1.59%	0.00%	38.10%	58.73%
3.0 s	1.27%	98.73%	0.00%	0.00%	1.27%	37.97%	60.76%
3.25 s	2.50%	97.50%	1.25%	1.25%	0.00%	51.25%	46.25%
3.5 s	2.78%	97.22%	2.78%	0.00%	0.00%	58.33%	38.89%
3.75 s	1.27%	98.73%	1.27%	0.00%	0.00%	50.63%	48.10%
4.0 s	0.00%	100.00%	0.00%	0.00%	0.00%	42.86%	57.14%

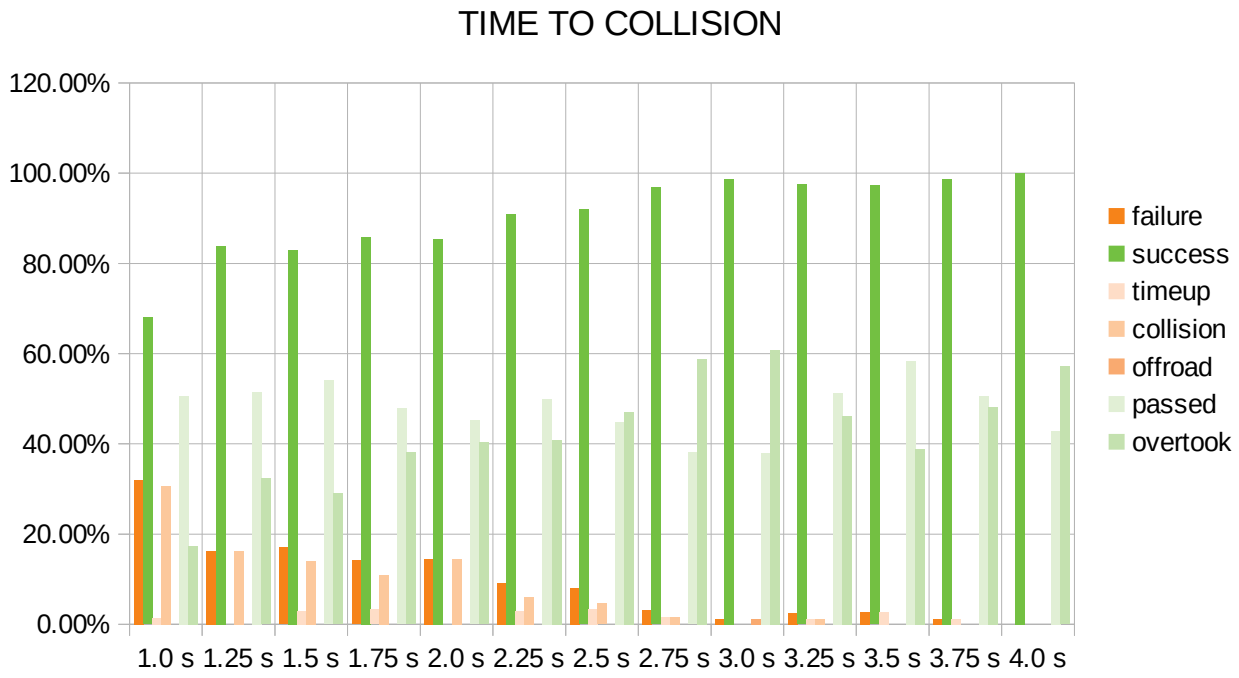

Figure 5.14 Results of Autonomous Emergency Steering Test – Time To Collision

6. Conclusions

This thesis work presented two autonomous control systems based on deep reinforcement learning. The two ADAS control were: Autonomous Emergency Braking and Autonomous Emergency Steering. These two ADAS control systems were studied in two different tests. Each test had its own parameters and its own reward functions. The reinforcement learning agent learned a policy for braking and overtaking in emergency situations from scratch, using vision as input and experiences under simulated environment. To provide the most realistic environment possible, a simulator with state of the art rendering quality was used.

The deep reinforcement learning algorithm used by the agent was a Double Deep Q-Network with Dueling Architecture. The information were provided to the agent through sensors. Cameras, collision and lane invasion sensors were used. The information, used by the agent to learn the policy, were provided by the camera sensor. The output frames of the camera were processed and stacked together to give the agent the sense of motion. The outputs of the collision and the lane invasion sensors were used to determine the terminal state of the learning episodes. The agent learned from experience through the proposed reward functions. The proposed reward functions were designed to let the agent learn in the most efficient way possible. The tests used to train the agent were designed to be valid and reproducible. The validity of the test was studied through the test parameters. They were designed to be significant and meaningful. The settings of the tests and their main parameters were chosen taking as reference the Euro NCAP AEB test protocol. The use of an open-source simulator and the vision as input ensured the reproducibility of the tests.

The agent was able to learn a fairly good policy after 20 hours of training, achieving a success rate in testing of 83.90% for Autonomous Emergency Braking and 90.50% in Autonomous Emergency Steering. Each important parameter of the simulation was investigated. The parameter considered for the statistical analysis were: test type, pedestrian behavior, target velocity, time to collision. The test type did not influenced the simulation outcome of the two tests in a significant way. The target velocity influenced the collision and the time-up rates as well as braked or passed rates. The time to collision influenced the success and the failure rate the most. In fact, for both tests the shorter was the time to collision the higher was the failure rate. The pedestrian behavior was of great

importance, it purpose was to check if the agent was able to understand the emergency situation without generalizing the braking or overtaking actions.

The tests results showed an improvement of the performances adding the steering ability to the agent. More training is surely needed to optimize the agent performances. The training results showed encouraging results for the learning of the agent. This learning performances can be optimized and investigated in further research. This thesis work makes contribution on the topic of investigating collision avoidance in emergency situation using deep reinforcement learning. The use of model-free deep reinforcement learning allowed the agent to learn complex controls in realistic simulated environment. Since the input was given to the agent through a low cost and established sensor, the trained agent can be used in real vehicles for further studies.

6.1 Future Work

Some ideas were considered for this thesis work, according to the literature related to the crash avoidance and reinforcement learning. They are considered for future work, since in this thesis the main aim was to create a reproducible test and a strong baseline of results to compare the further work.

6.1.1 Improvements in the Agent

Improvements in the algorithm can lead to better results. More than one improvement can be implemented in order to combine the advantages. Similarly to what was done in this thesis work, where on the Deep Q-Network the Double Deep Q-learning was implemented together with the Dueling Architecture. Other improvements can be integrated in a single algorithm. In paper [37] a single algorithm consisted in various improvements of the Deep Q-Network. The algorithm of [37] was called *Rainbow*, this algorithm achieved state of the art performance.

The extensions on the Deep Q-Network implemented in *Rainbow* were the Double Q-learning, the Prioritized Replay, the Dueling Architecture, the Multi-Step Learning, the Distributional Reinforcement Learning and the Noisy Nets.

6.1.2 Benchmark among RL Algorithms

The possibility to investigate the difference in performance of various reinforcement learning algorithms can be considered. A good alternative to the D3QN agent used in this thesis work can be the Deep Deterministic Policy Gradient, the Twin Delayed Deep Deterministic Policy Gradient, the Proximal Policy Optimization, the Soft Actor-Critic.

A collection of implementations for these algorithms and a benchmark can be found on the *Spinning Up* website [38].

6.1.3 Implementation of Rule-Based Constraints

Use of rule-based constraints can lead to better results. In paper [39] a combination of Deep Q-Network and rule-based constraints was applied for autonomous driving lane change decision making task. An efficient lane change behavior was achieved during the training in a real-world-like simulator.

The paper [39] claimed that the implementation of a rule-based Deep Q-Network method outperformed both the rule-based approach and the Deep Q-Network method.

6.1.4 Use of a Regret-Based Human Model

Use of regret-based human decision model can lead to better results. In paper [40] a method that uses the regret theory to implement a human decision-making model is proposed. In the paper regret theory was used to simulate the behavior of a human driver. The regret-based decision-making model was integrated into a safe reinforcement learning framework. The reinforcement learning agent used along with the regret model was a Double Deep Q-Network. In the paper the environment was simulated using CARLA.

The results of algorithm proposed in the paper were compared to the results of the conventional Double Deep Q-Network. The comparison showed that the proposed algorithm achieved in ensuring no collisions for the trained model.

Bibliography

1. *Sutton, R., Barto, A. (2017)* Reinforcement Learning: An Introduction, *MIT Press*

2. *Silver, D. (2015) UCL* Course on RL, *https://www.davidsilver.uk/teaching/*

3. *Watkins C.J. (1989)* Learning from Delayed Rewards *(Ph.D thesis), Cambridge University*

4. Watkins C.J, Dayan P. (1992) Q-learning, *Machine Learning* 8, 279–292

5. *Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.A. (2013).* Playing Atari with Deep Reinforcement Learning. *ArXiv, abs/1312.5602.*

6. *Long-Ji Lin. (1993)* Reinforcement learning for robots using neural networks. *Technical report, DTIC Document*

7. *Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2015)* Human-level control through deep reinforcement learning. *Nature 518, 529–533*

8. *Hasselt, H.V., Guez, A., & Silver, D. (2016).* Deep Reinforcement Learning with Double Q-Learning. *AAAI.*

9. *Van Hasselt, Hado. (2010).* Double Q-learning.. *2613-2621.*

10. *Wang, Z., Schaul, T., Hessel, M., Hasselt, H.V., Lanctot, M., & Freitas, N.D. (2016).* Dueling Network Architectures for Deep Reinforcement Learning. *ArXiv, abs/1511.06581.*

11. *Morgan Stanley Research Global (2013).* Autonomous Cars: Self-Driving the New Auto Industry Paradigm

12. *SAE International (2018).* Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. J3016_201806

13. "Phantom Auto" to Be Operated Here *(1932) The Free-Lance Star. Retrieved 14 September 2013. https://news.google.com/newspapers?id=PthNAAAAIBAJ&pg=6442,3879017*

14. *Waugh R. (2013)* How the first "driverless car" was invented in Britain in 1960 *Yahoo*! News. https://uk.news.yahoo.com/how-the-first--driverless-car--was-invented-in-britain-in-1960-093127757.html

15. *Albanesius C. (2010)* Google Car: Not the First Self-Driving Vehicle. *PC Magazine*

16. *Markoff J. (2010)* Google Cars Drive Themselves, in Traffic. *New York Times*

17. *Harris M. (2014)* How Google's Autonomous Car Passes the First U.S. State Self-Driving Test. *IEEE Spectrum: Institute of Electrical and Electronic Engineers*

18. *Slosson M. (2012)* Google gets the first self-driven car license in Nevada. *Reuters*

19. *Ross, Philip E. (2017).* The Audi A8: the World's First Production Car to Achieve Level 3 Autonomy. *IEEE Spectrum: Technology, Engineering, and Science News.*

20. *Levin S., Wong J., Carrie J. (2018)* Self-driving Uber kills Arizona woman in first fatal crash involving pedestrian. *The Guardian*

21. *Mallozzi P., Pelliccione P., Knauss A., Berger C., Mohammadiha N. (2019)* Autonomous Vehicles: State of the Art, *Future Trends, and Challenges. In: Dajsuren Y., van den Brand M. (eds) Automotive Systems and Software Engineering. Springer, Cham*

22. *Yurtsever, E., Lambert, J., Carballo, A., Takeda, K. (2020).* A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access, 8, 58443-58469.*

23. *Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A.A., Yogamani, S.K., Pérez, P. (2020).* Deep Reinforcement Learning for Autonomous Driving: A Survey. *ArXiv, abs/2002.00444.*

24. *A. Keselman, S. Ten, A. Ghazali, M. Jubeh (2018)* Reinforcement learning with a* and a deep heuristic, *arXiv preprint arXiv:1811.07745*

25. *D. C. K. Ngai, N. H. C. Yung (2011)* A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers *IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 2, pp. 509–522*

26. *D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, K. Fujimura (2018)* Navigating occluded intersections with autonomous vehicles using deep reinforcement learning, *IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 2034–2039*

27. *P. Wang, C.-Y. Chan (2017)* Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge, *in Intelligent Transportation Systems (ITSC), IEEE 20th International Conference on. IEEE, 2017, pp. 1–6*

28. *P. Wang, C.-Y. Chan, A. de La Fortelle (2018)* A reinforcement learning based approach for automated lane change maneuvers, *IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 1379–1384*

29. *A. E. Sallab, M. Abdou, E. Perot, S. Yogamani (2016)* End-to-end deep reinforcement learning for lane keeping assist, *arXiv preprint arXiv:1612.04340*

30. Chae, H., Kang, C.M., Kim, B., Kim, J., Chung, C.C., Choi, J.W. (2017). Autonomous braking system via deep reinforcement learning. *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 1-6

31. Porav, H., Newman, P. (2018). Imminent Collision Mitigation with Reinforcement Learning and Vision. *21st International Conference on Intelligent Transportation Systems (ITSC)*, 958-964.

32. *Dosovitskiy, A., Ros, G., Codevilla, F., López, A., Koltun, V. (2017)* CARLA: An Open Urban Driving Simulator. *CoRL*

33. *PhysX 4.1 SDK Guide (2019)* *https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Manual/Vehicles.html#references*

34. *European New Car Assessment Programme (2019) Test protocol – AEB VRU systems, Version3.0.2, https://www.euroncap.com/it/per-i-tecnici/protocols/vulnerable-road-user-vru-protection/*

35. *https://carla.readthedocs.io/en/latest/core_map/*

36. *http://carla.org/2019/07/12/release-0.9.6/*

37. Hessel, M., Modayil, J., Hasselt, H.V., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M.G., & Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. *ArXiv, abs/1710.02298*.

38. *https://spinningup.openai.com/en/latest/spinningup/bench.html*

39. *Wang, J., Zhang, Q., Zhao, D., & Chen, Y. (2019).* Lane Change Decision-making through Deep Reinforcement Learning with Rule-based Constraints. *2019 International Joint Conference on \Neural Networks (IJCNN), 1-6.*

40. *N. Kaempchen, B. Schiele and K. Dietmayer (2009)* Situation Assessment of an Autonomous Emergency Brake for Arbitrary Vehicle-to-Vehicle Collision Scenarios *in EEE Transactions on Intelligent Transportation Systems, vol. 10, no. 4, pp. 678-687, Dec. 2009, doi: 10.1109/TITS.2009.2026452.*

41. *Shiller, Zvi & Sundar, Satish. (1998).* Emergency Lane-Change Maneuvers of Autonomous Vehicles. *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR. 120. 10.1115/1.2801319.*

42. *Choi, Chulho & Kang, Yeonsik & Lee, Seangwock. (2012).* Emergency collision avoidance maneuver based on nonlinear model predictive control. *393-398. 10.1109/ICVES.2012.629428*

43. *S. Singh (2018).* Critical reasons for crashes investigated in the national motor vehicle crash causation survey *Nat. Highway Traffic Saf. Admin., Washington, DC, USA, Tech. Rep. DOT HS 812 506*