UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
"TECNOLOGIE DELL'INFORMAZIONE"

CICLO XXXV

# Deep Reinforcement Learning in Action: Innovative Approaches to Control a Real Self-Driving Vehicle

Coordinatore:
Chiar.mo Prof. Marco Locatelli

Tutore:
Chiar.ma Prof.ssa Monica Mordonini

Dottorando: Maramotti Paolo

Anni 2019/2022

*A Federica*

# Contents

# List of Figures

# Introduction

Since ancient times, humankind has always sought solutions to make travel faster and more comfortable. It started with the horse and chariot, and then moved through steam or coal powered means of transportation to the present day with vehicles powered by gas, petrol or electricity. Over time and especially in the last century, progress in mobility has been steady and exceptional and this is evidenced by the density of vehicles that fill populated areas on a daily basis. However this exaggerated amount of transportation has catastrophic consequences for the environment, because of the pollution they generate. For this reason and also because traffic accidents determined by human error are one of the primary causes of death, the autonomous driving has become worldwide topic of interest in recent years. Many major automotive companies and universities are investing large amounts of capital to achieve this goal as soon as possible.
This chapter introduces the concept of autonomous driving, focusing mainly on the benefits and difficulties involved in this technology. Then a brief overview of the research work done during these three years will be given.

## Benefits and Difficulties of Autonomous Driving

Autonomous Car, or also Driverless or Self-Driving Car are terms that identify vehicle that can move autonomously without the intervention of a human driver. In these cars, the driver is essentially replaced by a technological system that allows it to move in total autonomy. That of the self-driving car is the image which most comes to mind

when thinking about the future of four-wheelers. A future that in some respects is already materializing nowadays. In recent years, the investments of the major automakers in the field of autonomous driving have been enormous. Nowadays, particularly advanced driver assistance systems can be found on the cars driving in our cities, laying the groundwork for autonomous driving.
Autonomous vehicles rely on both traditional algorithms and advanced Artificial Intelligence (AI) systems that enable them to "understand" and "interpret" the environment and interact with it. In order to work, an autonomous driving system must be able to "see" what is happening around the car. To do this it takes advantage of the combined work of several tools, first and foremost, sensors. The ones mainly used are three: radar, ultrasound and lidar. In addition to the combination of some or all of these sensors, autonomous vehicles are usually equipped with cameras. If the cameras are stereoscopic, they behave exactly like human eyes; indeed, they are able to calculate the distance to objects due to the ability to analyze perspective. All the data obtained from these instruments, together with advanced computer vision functions are used to create a constantly updated map of their surroundings. From it, the vehicle is then able to autonomously identify obstacles and signs, define the appropriate route to follow, and drive along it correctly.
Autonomous vehicles, by now, are pure reality: it is no longer a question of wondering about the possibilities of creating them, but, simply, when they will begin to circulate on our streets. The fact that even today the spread of self-driving cars is extremely limited can be attributed to the fact that a good portion of the population has not yet accepted this technology. Indeed, at present, there are still too many critical issues related to autonomous driving. Mainly those related to ethics and responsibilities. Before dwelling on this fundamental point we will go over the benefits and difficulties involved in autonomous driving.

Certainly, the main benefit that would be achieved through autonomous driving would be increased safety in travel, greatly reducing the likelihood of accidents. A system like this is not subject to fatigue or distractions as a human being is, and, in addition, it is less subject to environmental conditions (low light, fog, rain, snow) because it may be able to adapt. In addition to these important factors, it must also

Figure 1: Critical traffic situation that mirrors what happens in most cities around the world.

be taken into account that with new technologies we have come to obtain vehicles that are able to make decisions with far greater responsiveness than a human driver. A person has limited senses and reflexes that vary depending on certain factors, such as age, level of fatigue, and others. In contrast, an autonomous vehicle has a sensor composition that determines these factors and makes it generally better than a human. In this regard, [1] , in a 2014 study, estimated that the number of traffic fatalities per day is a figure of around 3000 cases, with more than half of the people involved not being in a car. Besides, according to [1], unless effective action is taken, with the increase in stress and technological means such as smartphones, this number is set to rise to more than 2.4 million per year, becoming the fifth leading cause of death in the world.

In addition to this factor, which is certainly the most obvious one, there are also many other benefits if we came to have autonomous driving in our daily lives. First of all, there would be a reduction in congestion on the roads (Fig. 1) due to better traffic management skills than human drivers, especially if car sharing and improved public transportation services also take hold in parallel [2]. As a consequence, there would also be a drastic decrease in fuel consumption and thus pollution, which is a topic of very current interest. Also not insignificant is the amount of time that on average a person could save; indeed, the figure of the driver during the commute would no longer have the need to drive, but could occupy the time at his or her leisure like other passengers. A study conducted by CSA Research in collaboration with Citroen in 2016 estimates that on average in Europe each person spends 4 years and 1 month of his or her life driving.

An important benefit to note is also to give the possibility of being more autonomous to the elderly and the disabled, who, thanks to this technology, would be able to move freely with more safety. In addition to these direct consequences, there are also countless other indirect consequences, such as those related to the raising of speed limits (due to the increased active safety) to the abolition of age limits for driving (even a single underage passenger can travel, since he or she has no responsibility) or even to the reduction of signage (information is acquired directly from the on-board computer system). Given the rising cost of living in recent years, an innovation such as this could also lead to reducing a family's automobile costs. Less accidents would also decrease the cost of repairing damage, and even in the future might even lead to the abolition of insurance. Of course, with the introduction of automated vehicles into an environment where it is essential to have the readiness to react appropriately to external stimuli, it carries risks of considerable ethical significance as well. Special situations and contingencies are to date the main object of study and the main issue in autonomous driving. This is because the algorithm that would make it possible to avoid any kind of accident would have to depend on infinite factors and conditions, and for that reason it is very complex to develop. Lately there have been some fatal accidents caused by these vehicles, such as one where the autonomous car failed to avoid a woman who was jaywalking and in total darkness, causing her death [3]. In addition to algorithms, it is necessary also to take into account that any one component of a vehicle could malfunction, and this could compromise the entire system. For these reasons, there are still no fully automatic vehicles that can be purchased by individuals and used on the road.

In addition to this, it must also be taken into consideration that with the advent of these systems there would be a reduction in drivingrelated jobs (taxi drivers and truck drivers). Among the risks to be considered is the danger of hacking. Like any electronic device connected to a network, similarly the self-driving car is also attackable by hackers. Malicious third parties could "take over" the vehicle or modify normal driving functions, changing the temperature inside the cabin or governing the brake and accelerator. In an even worse scenario, terrorist and criminal activities could use these vehicles for attacks or to kill someone.

Ultimately, self-driving cars will definitely be part of our future, with their pros and cons, although it must be seen how much more time will be needed in order to improve and refine this new technology. Many lovers of "pure" driving are skeptical of such a change as they would like to retain the possibility of being able to drive themselves, thus maintaining those sensations that can only be experienced behind the wheel of one’s own car. Now all that remains is to find the perfect algorithm by which autonomous cars can finally avoid any kind of accident.

## Research Overview

The main theme of the research project is the development of Deep Reinforcement Learning and Imitation Learning algorithms in the field of autonomous driving in urban environments. Initially, we focused on solving the problem of crossing intersections. We implemented an algorithm that allows the agents involved to negotiate with each other and learn to respect the right of way rule. The system directly returns the acceleration and steering angle values that the vehicle must use to cross the intersection correctly and safely.
Then, we developed a Deep Reinforcement Learning planner, capable of driving comfortably in both a simulated and real obstacle-free environment. We showed that the system has good generalization capabilities and that it works even in areas on which it has not been directly trained. Moreover, in order to deploy the system on board of the real self-driving car and to reduce the gap between simulated and real-world performances, we also develop a module represented by a tiny neural network able to

reproduce the real vehicle dynamic behavior during the training in simulation.
In the last part of the project, we tested a different architecture than that used in previous work, attempting to make the system more robust by splitting the neural network used into several separately trained components. This architecture also has the capability of generating a model of the environment that it then exploits to predict the future. In this way, our system is able to evaluate what action to take now based on what it thinks might happen in the future.

# Chapter 1

# Autonomous Driving

In this chapter we will discuss in more detail the topic of autonomous driving, which we have already defined as a system that has the capabilities to replace the driver of a vehicle by performing driving tasks itself. Since such a system must operate in a real-world environment whose dynamics are unpredictable, it therefore needs to be extremely robust and well-structured. The creation of a vehicle that can drive autonomously involves, first of all, the selection of suitable hardware, especially sensors and cameras. In addition, it requires the development of software that can process the collected data and turn it into useful information to define the best actions to perform. In this chapter we will cover the history of autonomous driving, showing all the progress that has been achieved to date. Next, we will define the six canonical levels of automation, which have been identified to quantify how autonomous a given vehicle actually is. At the end of this chapter we will also discuss in more detail the different types of sensors that are generally used, and finally, we will also describe the typical architecture of an autonomous vehicle.

## 1.1 A Century of History

Contrary to what many people think, Google and Tesla are not the pioneers of autonomous driving. Autonomous vehicles have a century-long history, starting with

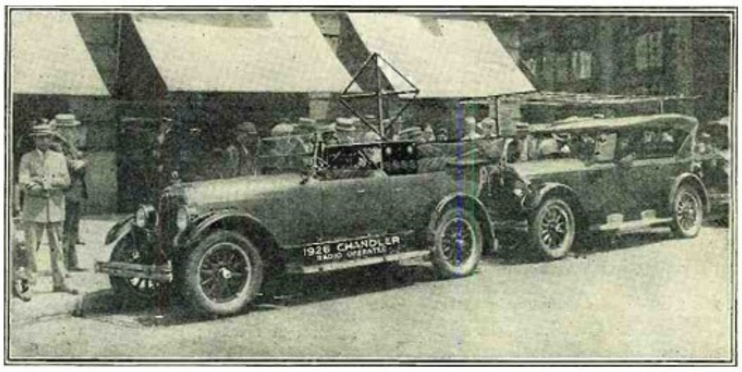


Figure 1.1: Linrrican Wonder, the radio-controlled vehicle that the Houndina Radio Control company developed in 1925 for the first concrete demonstration of autonomous driving.

the earliest radio-controlled models and continuing to the present with complex systems based on artificial intelligence and computer vision. For more than a hundred years, humans have dreamed of and designed a vehicle that can accelerate, brake, and safeguard the security of passengers and pedestrians autonomously. New technologies and progress have made feasible and less science fiction what was thought long ago. The first concrete demonstration of autonomous driving was in 1925, in New York, when the American radio equipment company Houdina Radio Control presented a radio-controlled vehicle named the Linrrican Wonder (Fig. 1.1). It was a Chandler accessorized with a radio antenna, which picked up impulses sent by an operator located on another vehicle and turned them into controls that allowed movement. The following year there was a similar experiment in Milwaukee, witnessed by newspapers of the time, which dubbed the vehicle the Phantom Auto. Thereafter for a decade or so there was no further relevant news about these technologies until prototypes of driverless cars and cabs began to make their way with the aim of reducing traffic congestion in American cities.

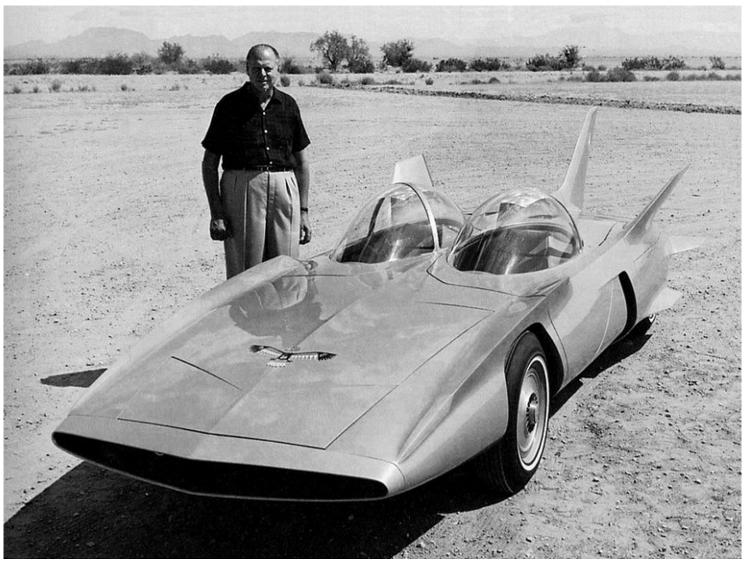


Figure 1.2: Firebird III, the model developed by General Motors in 1958 with the first cruise control system.

In 1939, at the New York World's Fair, General Motors presented its "Futurama" project, where an environment populated by radio-controlled cars driven by electromagnetic fields was showcased. The idea was simple: you would drive the car to the entrance of a highway, then engage the autopilot. The vehicle would stay in its own lane until the exit. Then again a halt in progress, probably caused by World War II. New designs are developed only in 1953 from the collaboration between RCA Labs of New York and General Motors. The prototype they made and unveiled in 1958 was capable of covering a 121-meter road. Its operation was based on a network of sensors placed in the asphalt that were capable of controlling the vehicle's actuation and determining the presence and speed of other obstacles along the way. Also in 1958 General Motors introduced a model called the Firebird III (Fig. 1.2) equipped with an early cruise control system, which allowed the car to travel on highways without the assistance of a driver. Despite these early achievements exclusively in the U.S., the first fully autonomous vehicle capable of moving without external auxiliary systems

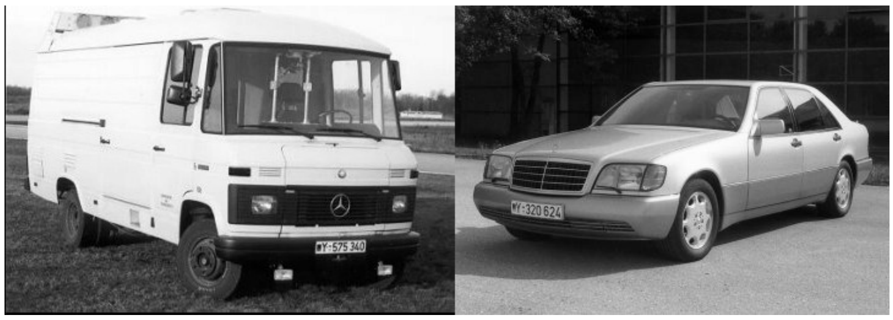


Figure 1.3: The vehicle on the left is VaMoRs, a Mercedes van set up by Dickmanns and his team. It dates back to 1986 and is considered the first autonomous vehicle capable of moving without external auxiliary systems. In contrast, the figure on the right shows Vamp, another autonomous car model developed in 1994 also by the collaboration of Mercedes and Dickmanns.

took shape in Germany. Ernst Dickmanns ([4]) with his team from the University of Munich presented "VaMoRs" in 1986 (Fig. 1.3), a Mercedes-Benz van adapted for this purpose. The van was able to move by processing data from its neighborhood, captured by the various cameras and sensors with which it was equipped.

Great strides were then made when the Eureka Prometheus Project was launched in 1987, a funding program that allocated about 750 million euros for projects in the field of autonomous driving. With these funds in 1994 Ernst Dickmanns ([5]) again in collaboration with Mercedes-Benz created twin vehicles, Vamp (Fig. 1.3) and Vita-2. Together they traveled many kilometers on highways in varying traffic conditions, reaching up to 175 km/h and experimenting with driving in traffic resulting in lane changes and overtaking. Sometimes with some man-made adjustments. In 1995 another major step was taken, the introduction and use of the first neural networks for autonomous driving. In the Navlab project ([6], [7]), Carnegie Mellon University created a prototype that could drive on the road in which the accelerator and brake were controlled by a human, but the steering wheel by a neural network.

Italy has also had its own role on the subject of autonomous driving. In 1998 the Department of Information Engineering of the University of Parma, with the work of

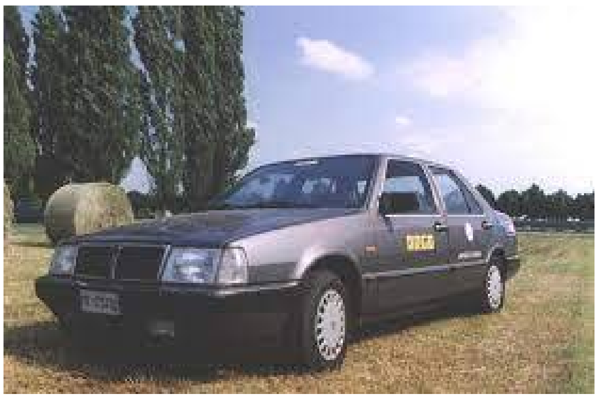


Figure 1.4: The picture represents ARGO, the autonomous vehicle developed in 1998 by Broggi and his team at the University of Parma. A Lancia Thema capable of understanding its surrounding thanks to stereoscopic vision algorithms.

Alberto Broggi ([8]) developed ARGO (Fig. 1.4), a Lancia Thema equipped with two low-cost video cameras. Thanks to stereoscopic vision algorithms, it is able to understand its surroundings; especially it is able to recognize road signs, lane markings, possible obstacles and other vehicles. The project's apotheosis was the MilleMiglia in Automatic, a 1900-kilometer journey that lasted 6 days on the roads of northern Italy. The car operated in fully autonomous mode for 94% of the route.

Big improvements in autonomous driving research comes with the Demo I, Demo II and Demo III competitions launched between 2004 and 2007 by the Defense Advanced Research Projects Agency (DARPA), the most prominent research organization of the United States Department of Defense. Large funds were put up for grabs for the winner. These kinds of competitions attracted many researchers and companies from all over the world. In the first two competitions (Demo I and Demo II) the goal was to create an autonomous vehicle capable of driving on difficult off-road terrain. The competition in both cases was held in the Mojave Desert in California and consisted of driving over 200 km in these rugged terrains. The Demo I ([9]) was a complete failure, none of the vehicles completed the race. The best one managed to cover only 12 km of the course before getting hung up on a rock after making a

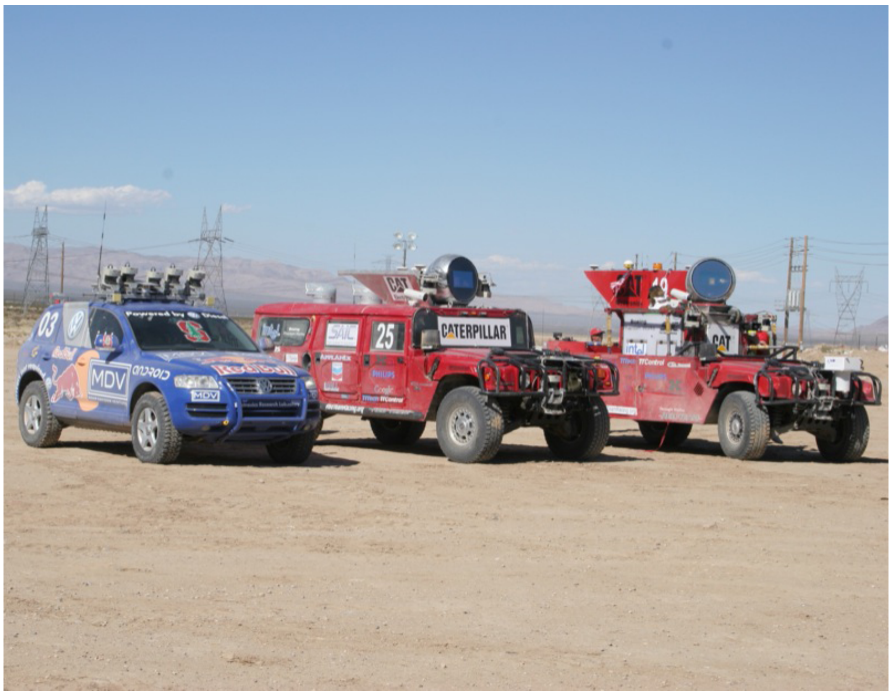


Figure 1.5: Three of the five vehicles that finished Demo II of the DARPA Grand Challenge. On the left the blue vehicle is Stanley, the overall winner from Stanford University's Thrun team. In the center and right, the two red vehicles are Sandstorm and H1ghlander, which finished second and third, respectively. These two vehicles were developed by Carnegie Mellon University.

switchback turn. Demo II went much better, five vehicles successfully completed the race. The winner was "Stanley" a modified Volkswagen Touareg (Fig. 1.5) proposed by Stanford University ([10]).
Demo III was held in 2007 and no longer consisted of driving on difficult off-road terrain, but was a challenge in an urban setting. Participating vehicles had to be able to comply with traffic regulations and drive through traffic while avoiding not only other vehicles, but also objects in the middle of the roadway. In this case the winner was "Boss" a Chevy Tahoe prototype from Carnegie Mellon University ([11]).

Figure 1.6: This image represent the fleet of four autonomous vehicles and the team that drove from Parma to Shanghai in the VisLab Intercontinental Autonomous Challenge (VIAC).

In 2008, the Netherlands introduced one of the first driverless public transportation systems, the ParkShuttle, while, also in the same year, Canadian mining company Rio Tinto Alcan began testing the Komatsu Autonomous Haulage System, the first fully automated construction vehicle.

Also Broggi (Italy) with his startup VisLab took on a challenge called VIAC (VisLab Intercontinental Autonomous Challenge) in which he and his team built a fleet of four autonomous vehicles that undertook a 3-month journey from Parma to Shanghai ([12], [13], [14]). Each vehicle was equipped with a total of seven cameras (5 forward and 2 backward looking), and four laserscanners with different characteristics. The complexity and magnitude of this challenge lay mainly in the plurality of scenarios, situations, environments, roads, and weather conditions they encountered along the way. Despite the countless difficulties, the four vehicles performed excellently and reached China almost completely on their own.

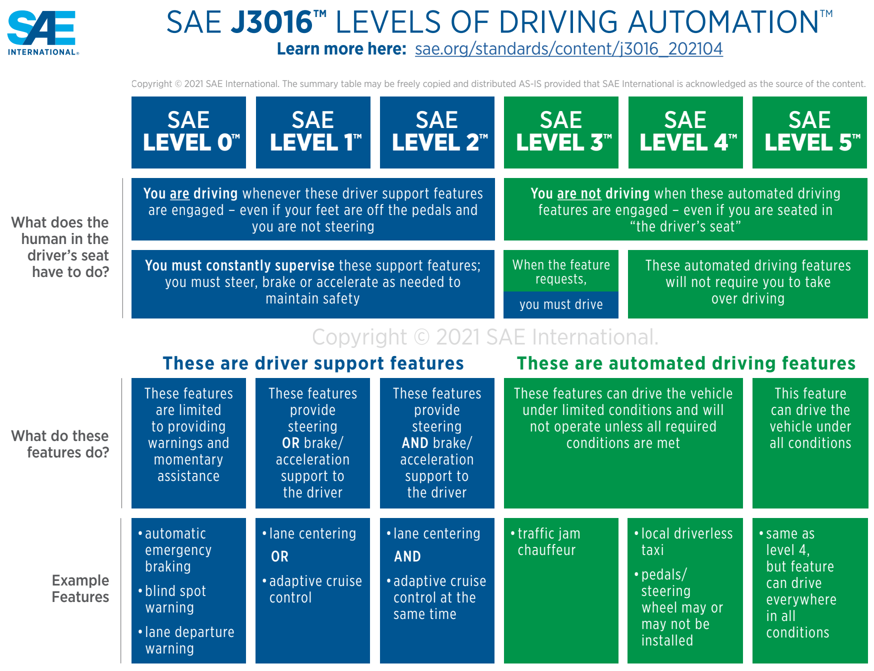
After this important Milestone the rest is recent history in which Google, Uber, Lyft, Tesla, and the major automakers are battling it out in a race toward the future. To date, thanks to artificial intelligence and technological progress, we already have very good systems with vehicles capable of autonomous driving in any traffic and environmental conditions. However, the next step that will allow us to commonly see these autonomous vehicles driving on the streets will only occur when the various nations organize themselves to have infrastructure and legislation that will allow it and, above all, when humankind is able to accept this technology.

## 1.2 Levels of Driving Automation

In the field of autonomous driving, there is an organization called SAE International, which is responsible for developing and defining the standards by which the level of automation of a given vehicle can be defined. It recently published a visual table (Fig. 1.7) with the purpose of clarifying and simplifying the standard J3016 "Levels of Driving Automation". This standard defines the levels of driving automation, from level 0 SAE, which refers to a vehicle with no automation, to level 5 SAE, which corresponds to complete vehicle autonomy. Let us now look in detail at the different levels:

- *Level 0*: Corresponding to a vehicle that has no automatics, every movement of the car is due to a command given by the driver. The only exception is automatic emergency braking, which stops the car's movement when sensors detect an obstacle in front of the vehicle (mandatory technology for vehicles registered from 2022). Also included in Level 0 are blind spot warning, which alerts the driver if another vehicle is in the rear-view mirror blind-spot, and lane departure warning, which warns of inadvertent occupation of the oncoming lane.

- *Level 1*: It contains those technologies that are supportive to the driver for steering or acceleration (but not simultaneously). Belonging to this category are the lane centering system, to keep the car in its lane, adaptive cruise control, to



SAE **J3016**™ LEVELS OF DRIVING AUTOMATION™

**Learn more here:** sae.org/standards/content/j3016_202104

| | SAE LEVEL 0™ | SAE LEVEL 1™ | SAE LEVEL 2™ | SAE LEVEL 3™ | SAE LEVEL 4™ | SAE LEVEL 5™ |
|---|---|---|---|---|---|---|
| What does the human in the driver's seat have to do? | **You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering | | | **You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat" | | |
| | **You must constantly supervise** these support features; you must steer, brake or accelerate as needed to maintain safety | | | When the feature requests, you must drive | These automated driving features will not require you to take over driving | |
| | These are driver support features | | | These are automated driving features | | |
| What do these features do? | These features are limited to providing warnings and momentary assistance | These features provide steering **OR** brake/ acceleration support to the driver | These features provide steering **AND** brake/ acceleration support to the driver | These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met | | This feature can drive the vehicle under all conditions |
| Example Features | • automatic emergency braking • blind spot warning • lane departure warning | • lane centering **OR** • adaptive cruise control | • lane centering **AND** • adaptive cruise control at the same time | • traffic jam chauffeur | • local driverless taxi • pedals/ steering wheel may or may not be installed | • same as level 4, but feature can drive everywhere in all conditions |

Figure 1.7: This visual table represents the SAE standard J3016 in which the 6 levels of driving automation are defined.

keep the vehicle at a certain speed, but without colliding the vehicle in front of it.

- *Level 2*: It is assigned to those vehicles that simultaneously have both lane keeping and autonomous acceleration capabilities. So if a vehicle has both lane centering and adaptive cruise control, it belongs in this category. To date, a large proportion of the cars that can be purchased have the option of integrating systems to make it level 2.

- *Level 3*: At this level, driving begins to become truly autonomous, although the driver is still in charge of the vehicle and obliged to intervene in case of problems or malfunction of an electronic system. It combines some advanced driving assistance systems (ADAS) with functionality achievable through neural networks. An example of Level 3 autonomous driving technology is what is known as traffic jam assist, or traffic jam chauffeur: when in the middle of a traffic jam, the driver activates the system and, until it deactivates, traffic jam assist takes control of the accelerator, brakes, and steering and moves the car forward to the end of the traffic jam. Some mid- to high-end cars already integrate this feature.

- *Level 4*: It is the first level in which the driver becomes a passenger and no longer has to take command of the car, such that the vehicle no longer needs to have pedals and steering. An example of Level 4 autonomous driving are Robotaxis, which are driverless cabs that Waymo is experimenting with in the United States. The only real limitation of Level 4 is that the automation systems cannot be activated under all conditions, but only when specific requirements are met.

- *Level 5*: With level five, the automation systems are always active in all conditions and without limits. As with four, there is no need for pedals and steering wheel for the driver because his or her intervention is not required.

To date, we are still a long way from being able to see a Level 5 system moving freely through the streets. Both because there are still limits at the technological level and on algorithms, and because there are still bureaucratic and legislative hurdles to overcome. Suffice it to say that in Italy, for example, only as of July 14, 2022, has the possibility of using Level 3 ADAS in vehicles been unlocked and thus achieving not quite autonomous driving, but advanced assisted driving. Despite this, some responsibilities may still fall on the driver, who must still intervene when needed. In fact, not only Italy is in this situation, but most European states because even at the legislative level, Europe has not yet come up with a common instrument to enable automated driving on public roads, delegating the individual member states to enact laws and regulations on the subject and assigning responsibility directly to the manufacturers. As also mentioned earlier, until society is able to fully accept this technology there

Figure 1.8: The Waymo Robotaxi, with a level 4 automation system.

is unlikely to be complete progress that would lead to seeing Level 5 systems on the road.

## 1.3 Software System Architecture

When we want to implement software for an autonomous driving vehicle we rely mainly on two types of architectures: a modular one, which is composed of specific systems for localization, mapping, planning and control, and one defined end-to-end in which the information obtained from sensors is directly turned into actuation values, with a single compact system. Before describing these two architectures in detail let us take a look at what provides the input data to both, namely the sensor component of the system.

### 1.3.1 Sensors

Capturing the environment, as we humans do with our senses, is essential for cars to be able to drive autonomously. Modern vehicles are therefore equipped with a wide variety of sensors that help them detect their surroundings and thus support the driver or even relieve him of some tasks. The most important sensors for sensing the environment are cameras, radar, sonar, and LiDAR sensors, let us now look at them in a little more detail:

- *Cameras*: this is the only one of those previously mentioned that does not rely on the time-of-flight principle and at the same time is the only one that allows a visual representation of the world in color. This also involves additional information related to texture and contrast on objects. In the field of autonomous driving, for example, it allows the identification and differentiation of two traffic signs with the same shape, or distinguish static objects from moving objects. In addition, it is a sensor that is generally inexpensive, with small and compact size and high resolution. However, it also has weaknesses; first, it is based on a passive measurement principle, meaning that objects are detected only if they are illuminated. Therefore, the reliability of the cameras is limited in harsh environmental conditions such as snow, ice or fog and in the dark. In addition, the cameras do not provide distance information. Obtaining 3D images requires at least two cameras, as in the case of stereo cameras, or image recognition software, which requires high computational performance ([15], [16], [17]).

- *Radar*: In recent decades they have also been installed in vehicles to measure distances in order to obtain reliable data for systems such as the spacer and the emergency brake assistant, regardless of weather conditions. Radar technology is based on the time-of-flight principle. Sensors emit short pulses in the form of electromagnetic waves (radio waves), which propagate almost at the speed of light. As soon as the waves hit an object, they are reflected and bounce back to the sensor. The shorter the time interval between transmission and reception, the closer the object. Based on the speed of wave propagation, the distance to the object can be calculated and determined with great accuracy. By putting together several measurements, vehicle sensors can also determine the speed. This technology enables driver assistance systems such as adaptive cruise control and collision avoidance. Radar sensors are robust, inexpensive, and usually provide reliable data even in adverse weather conditions. However, they have low resolution and lack color; they work very well for short distances, but with greater distances they struggle to differentiate different objects and classify them ([18], [19]).

- *Sonar*: Like radar, sonar is also based on the time-of-flight principle. In this case, sound waves, inaudible to the human ear, are emitted at a frequency of 20,000 Hz. Apart from parking assistance, sonar sensors are also used to monitor the blind spot and for emergency brake assistants. Ultrasonic sensors are robust and provide reliable distance data, both at night and in fog. They are also inexpensive and can detect objects, regardless of material or color. However, the range of these vehicle sensors is limited to less than 10 meters, which means that this technology can only be used at close range ([20]).

- *LiDAR*: This stands for "light detection and ranging", and they are sensors that can be used for both short- and long-range applications. Higher levels of autonomy are thought to be possible with the use of LiDAR technology ([21], [22], [23]). These sensors are also based on the principle of time-of-flight. Instead of radio or ultrasonic waves, however, they emit laser pulses that are reflected by an object and picked up again by a photodetector. LiDAR sensors emit up to one million laser pulses per second and summarize the results in a high-resolution 3D map of the environment. Because of the level of information in these so-called point clouds, objects cannot only be recognized but also classified. One can distinguish a pedestrian from a cyclist, for instance. Long-range, durable LiDAR sensors deliver accurate information that is mostly unbiased by environmental circumstances, allowing vehicles to make the best driving choices. In the past, these sensors were frequently highly pricey, mostly because of the complicated and labor-intensive design of the mechanically components. However, thanks to their solid-state design, which is becoming increasingly established, the cost of high-resolution 3D sensors is being reduced quite considerably.

All of these sensors clearly have strengths and weaknesses, and for that reason, gen-

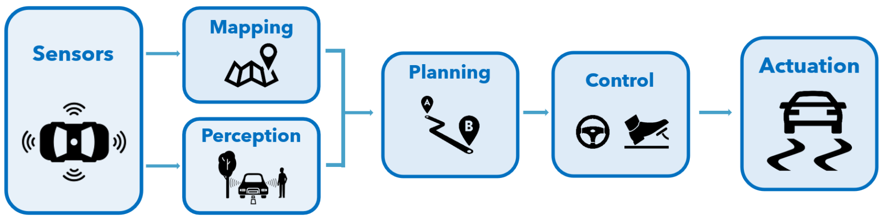


Figure 1.9: Typical modular architecture containing the main components of an autonomous driving stack.

erally in the structural composition of a vehicle, we try to include most of them. Each sensor will produce different data, and the vehicle's software must be able to interpret all these different data and apply what is known as "sensor fusion" ([24], [25]) to create a complete and as accurate as possible map of its surroundings.

### 1.3.2 Modular Architecture

Modular architecture, among the two main ones, is considered the most frequent. In this type there are different components as shown in Fig. 1.9, namely: mapping, perception, planning, control and actuation, which will be explained individually later. In the picture, however, only the main ones have been included, these then can be divided into many other sub-modules, or other modules can be added. The final architecture is highly dependent on the autonomous driving task for which it is used, the one presented is only a simplified and general form of architecture. Let us now look at the different modules of this typical autonomous driving pipeline:

- *Perception*: In self-driving cars, it is nothing more than how the car perceives and understands its surroundings. It is the most important and complex thing. For humans it is very simple to perceive our surroundings because we have eyes, ears, etc. and human intelligence, but for cars it is a very difficult and complex task of taking the huge amount of data from sensors and using computer intelligence to evaluate the data and derive meaning from it. The self driving cars have four core tasks to perceive the world:

1. *Object detection*: is a computer vision technique for identifying and locating objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise location. Generally, each object that is identified is framed by a 2D or 3D bounding box ([26]). In the field of autonomous driving it is used, for example, to recognize other agents or obstacles on the roadway ([27]), or to recognize traffic signs ([28]), traffic lights, etc.

2. *Classification*: is defined as the process of recognition, understanding, and grouping of objects. Classification algorithms used in machine learning utilize input training data for the purpose of predicting the likelihood or probability that the data that follows will fall into one of the predetermined categories. In the context of autonomous driving, for example, it is used to distinguish pedestrians from bicycles, cars or other means of transportation ([29], [30]). This enables the vehicle to understand what kind of other elements are present around it.

3. *Tracking*: It means defining the trajectory of other dynamic objects on the scene ([31], [32]). To do this, it is necessary to be able to recognize the same object on consecutive frames of the same sequence. In most tracking techniques, all objects detected in a frame are given an ID and an attempt is made to match the IDs in consecutive frames. Given that the monitored objects may enter and exit the frame at various timestamps, this is frequently a challenging process. They might also be occluded by their surroundings or even by one another. Noise, sampling or compression artifacts, aliasing, or acquisition mistakes are examples of faults in the obtained images that could lead to additional issues. Other difficulties arise when accounting for variations in motion, such as when objects are subject to rotation or scaling transformations or when their relative movement speeds are fast. This task is therefore very complicated, but at the same time also very important because it can aid in obstacle avoidance, motion estimation, the prediction of the intentions of pedestrians

and other vehicles, as well as path planning.

4. *Depth Estimation*: This process is necessary if the sensor you are using is the camera because the data you are getting is 2D and is obtained from reproducing the scene on the image plane. In this case the depth dimension is lost. This type of task deals precisely with recalculating the depth by, for example, comparing the same pixel on two consecutive frames obtained from the same camera or on two frames obtained from different cameras whose relative positions are known exactly ([33], [34]).

- *Mapping*: This module can also be redefined Localization ([35], [36]) and Mapping ([37]), and its main task is to estimate the state of the vehicle and the model of the environment around it. If these two tasks are performed simultaneously, because, for example, the pose of the sensors is unknown, then it is called SLAM (Simultaneous Localization and Mapping [38], [39]). Thanks to this module, the vehicle is then able to understand where it is in the environment, and thus, consequently, to stay in its lane of travel. Also by knowing the position of other objects relative to itself it can move without crashing. From this we deduce the importance of accurate estimation of both the surroundings and the state of the vehicle, because even a small error can have catastrophic consequences. The more accurate the estimates, the better the maneuvers that can be performed. A recent tool that certainly helps in achieving good results in localization and mapping tasks are HD maps (High Definition Maps [40]). These maps contain topological information that can be accurate down to the centimeter. Moreover, being created specifically for autonomous driving they generally contain many other map elements such as road shape, road marking, traffic signs, barriers and much more. The vehicle, driving based on these maps definitely has a lower error rate, not having to create the map itself, and in addition it is already aware of a lot of useful information about the road it is driving on, but especially about the roads it will have to drive on. So for example it will be able to brake because it knows there will be a dangerous or difficult curve. Or it will easily be able to know the speed limits of a particular road or how

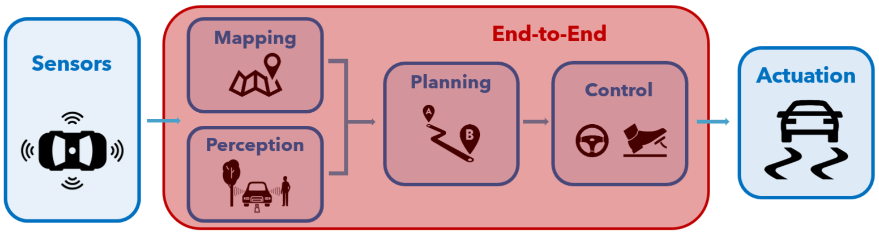


Figure 1.10: End-to-End architecture, allows actuation values to be obtained directly from sensor data.

many lanes there are and how wide they are.

- *Planning*: Motion planning or path planning ([41], [42]) means the task of defining a sequence of states and configurations that allow a system to move from a given starting position to a target point while avoiding any obstacles in the path. In the case of autonomous driving, motion planning must also take into account other constraints, such as speed limits. For this module to work properly, it is necessary that all the data obtained from the previous components be as accurate as possible. Because such a system must necessarily make high-level decisions in trajectory definition, such as entering or leaving a roundabout, changing lanes, or overtaking.

- *Control*: It is simply that component that transforms the motion planning output into actuation values for steering and acceleration or braking. It then allows specific low-level values to be obtained from a given high-level trajectory given to it by the previous module ([43]).

The last module of the architecture, which is called Actuation in Fig. 1.9, simply indicates the transformation of control values into actual vehicle actions that impact the environment.

### 1.3.3 End-to-End Architecture

Instead, this type of architecture encapsulates all the previous modules into a single module, as in the red area of Fig. 1.10. Indeed, it allows obtaining the actuation values to control the steering wheel, throttle, and brake directly from sensor inputs. Given its inherent complexity, this type of architecture can only be based on the use of Deep Learning and especially the approaches that will be described in the next chapter: Deep Reinforcement Learning and Imitation Learning ([44], [45], [46], [47]).

# Chapter 2

# Learning Approaches

Recently, thanks to the increase in calculus power of the computers that are used on autonomous vehicles, it has been possible to implement machine learning-based algorithms to perform increasingly onerous and complex tasks. In this chapter we will show what is meant by Machine Learning [48] and what are the main subcategories into which it is divided. In particular, we will focus on the two approaches mainly used in the context of this research project, namely Deep Reinforcement Learning and Imitation Learning.

## 2.1 Machine Learning

Machine learning is a data analysis method that automates the construction of analytical models. It is a branch of Artificial Intelligence (AI) [49] and is based on the idea that systems can learn from data, identify patterns independently and make decisions with minimal human intervention. Artificial intelligence is an umbrella term and refers to systems or machines that mimic human brain. The terms machine learning and AI are often used together and interchangeably, but they do not mean the same thing. An important distinction is that although everything about machine learning falls under artificial intelligence, AI does not include machine learning alone. Machine learning is a specific subset of AI that trains a machine how to learn. Machine learning stems from the theory that computers can learn to perform specific tasks without being programmed to do so, thanks to pattern recognition in data. It uses algorithms that learn from data in an iterative way. It allows computers, for example, to locate even unknown information without explicitly telling them where to look for it. The most important aspect of machine learning is repetitiveness, because the more patterns are exposed to the data, the more they are able to adapt autonomously. Computers learn from previous processing to produce results and make decisions that are reliable, replicable, and when possible generalizable. Every Machine learning project has main elements, which combine to build a system that learns relationships between data and stores them in a model:

- *Algorithm*: a set of rules, usually based on statistical methods, for extracting recurrent patterns from data. The difference between writing code and implementing logics with predetermined patterns is that the rules are not coded but are learned from the available example data.
- *Model*: a representation of a real context by an algorithm with appropriate parameters.
- *Training Dataset*: the set of data that given to the algorithm allows optimization of model parameters.
- *Testing Dataset*: the set of data that following the training phase is used to assess the quality of the model in terms of precision and accuracy.

The learning mechanism could happen in different ways based on the task we want to perform and on the available data; it could be supervised, unsupervised, semi-supervised or based on reinforcement learning approach.

### 2.1.1 Supervised Learning

In supervised machine learning, we have a known output value in the training dataset, and we use this information to train a model to be able to return a given output when a new given input not present in the training dataset occurs. There are two types of problem classes that can be solved with this type of approach:

- *Classification*: consists of providing as output a discrete value, a label or category. It classifies the required dataset into one of two (binary classifier) or multiple labels (multi-class). Some examples of classification algorithms are linear classifiers, decision trees and k-nearest neighbors, and in autonomous driving they are used, for example, in the context of perception to distinguish different road users: pedestrians, bicyclists, cars or other means of transportation; they can also be used to assess the status of a traffic light ([50], [51]).

- *Regression*: consists of predicting a continuous output. In this way a curve that represents the training set data to the best possible approximation is defined. By then mapping a new input, it is possible to obtain as output the prediction based on this curve ([52]).

In addition to the type of output and thus the context in which they are used, classification and regression also differ in the metrics defined to evaluate the quality of the model. In classification, correctly predicted labels, false positives and false negatives are counted and coefficients such as accuracy, precision, specificity, sensitivity, etc. are constructed; while in regression, an error function is evaluated, for example Mean Squared Error.
Another key aspect to take into account when talking about the supervised approach is that, compared to the others, it compulsorily requires a training dataset, which the more complete and extensive it is, the more accurate the model that can be obtained.

### 2.1.2 Unsupervised Learning

The unsupervised learning technique is based on the fact that you have a set of inputs but do not know the corresponding output. The model is built only with the input data, and the algorithm discovers hidden relationships in the data on its own. This type of approach is used when you have a data set with no output value or label associated with it. Thus, there are no training, validation and testing steps as in supervised; however, there are still methods to verify that the algorithm is optimized and that the resulting model produces meaningful outputs. These algorithms are suitable for discovering behaviors in the data that deviate from the usual, for discovering anomalies and exceptions. The two common uses of unsupervised learning are:

- *Clustering*: This technique is for grouping the unlabeled data based on their similarities or differences. A cluster is thus a set of data that have similarities with each other but, conversely, have dissimilarities with data in other clusters. The input of a clustering algorithm is a sample of elements, while the output is a number of clusters into which the elements of the sample are divided according to a measure of similarity. One of the popular clustering algorithm is K-Means where it sets aside the data points into different K groups ([53]).

- *Dimensionality Reduction*: This refers to representing the same data using lesser dimensions. This is usually used while removing noisy data from an image to improve its quality. A dataset with very large input characteristics complicates the predictive modeling task, putting performance and accuracy at risk. This condition is known as “the curse of dimensionality.” By reducing the number of input features, thereby reducing the number of dimensions in the feature space. Hence, dimensionality reduction simply means reducing the set of features that can represent the data.

### 2.1.3 Semi-Supervised Learning

Semi-supervised learning is a hybrid technique between the supervised and unsupervised methods. It is used when you have an input set consisting of a labeled part of the data and an unlabeled part. Semi-supervised learning determines correlations between data points, just like unsupervised learning, and then uses the labeled data to label those data points. Finally, the entire model is trained based on the newly applied labels. This type of approach has been shown to give accurate results and is applicable to many real-world problems where the small amount of labeled data would prevent supervised learning algorithms from working properly. This kind of approach is widely used in the field of autonomous driving ([54], [55], [56]).

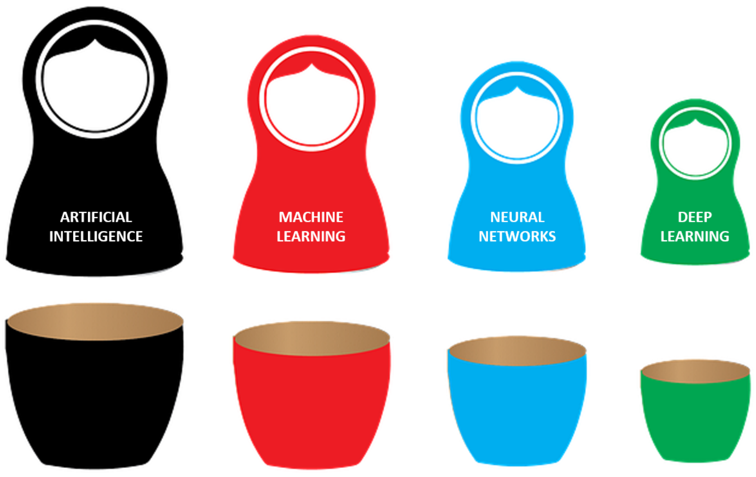


Figure 2.1: Russian nesting dolls of Artificial Intelligence

### 2.1.4 Reinforcement Learning

Reinforcement Learning is a learning technique that does not rely on labeled data, but on the experience made by an agent repeating given actions over and over again until it gets to do it to the best of its ability. A typical RL algorithm thus involves the interaction between an agent or agents and their environment. It is based on the trial-and-error mechanism and serves, for example, for an agent to learn to choose the best actions to perform at a given instant in a dynamic environment. To define which are the best there is a reward value that the agent receives as a consequence of its actions. The higher the reward the better the choice. This approach works very well when it has to perform a sequence of consecutive actions to achieve a given goal. We will elaborate more on the theory of this type of approach later in 2.3.

## 2.2 Neural Network

Generally the words artificial intelligence, machine learning, neural networks, and deep learning are mistakenly interchanged, each of them having a definite meaning and relationship to the others. They can be thought of as each one a component

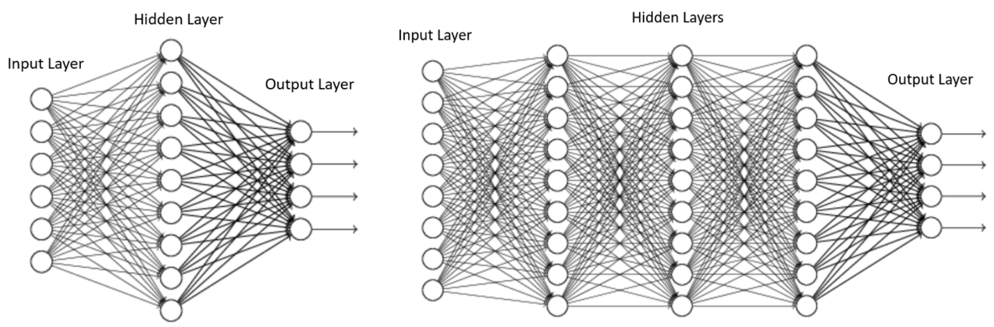


Figure 2.2: Schematic representation of two neural network structures, in which is shown the subdivision between input, hidden and output layers

contained in the previous one, like Russian nesting dolls (Fig. 2.1). As we defined before, machine learning is a subfield of artificial intelligence and in turn contains Deep Learning, the backbone of which is composed of neural networks. Let us now see what is meant by neural networks and then we will do a more specific overview of deep learning.

Neural network means an algorithm that can mimic the behavior of a human brain. Specifically, it is an algorithm that can predict an outcome from a given situation, and it can do this through accumulated experience. With a neural network, it is then possible to make a machine perform, completely autonomously, tasks that were previously intended to be done exclusively by humans; such as, for example, in the case of this thesis project: driving a vehicle. In the graphical and theoretical representation, a node in a neural network corresponds to a neuron in the human brain, and, just as with neurons in the human brain, a single node does not have great capabilities, but when combined together, they are able to perform complex tasks and achieve great results. A neural network can be represented with one of the two diagrams in Fig. 2.2. The nodes on the left indicate the input data, which can be the most varied and correspond to information procured by the senses in the case of the human brain. The nodes on the right, on the other hand, indicate the output, the prediction or reaction to stimuli. Their value can be continuous, discrete, binary or categorical. The nodes in the center correspond to the neurons in the brain and are used to "transform"

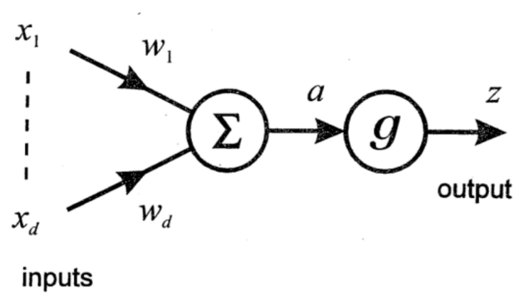


Figure 2.3: McCulloch-Pitts Model of a neural network with only a single neuron.

stimuli into reactions, that is, inputs into outputs. If the central part of the neural network, that is, the part between input and output, is composed of multiple layers, as in second network of Fig. 2.2, then we will speak of deep neural network and the learning technique used will be deep learning. These central layers are usually called hidden layers and make up the core of the neural network.

A neural network can be viewed as nonlinear mathematical functions that transform a set of independent variables $x = (x_1, ..., x_d)$, called inputs, into a set of dependent variables $z = (z_1, ..., z_c)$, called outputs. The precise form of these functions depends on the internal structure of the network and a set of values $w = (w_1, ..., w_d)$, called weights. We can then write the network function in the form $z = z(x; w)$ which denotes the fact that $z$ is a function of $x$ parameterized by $w$.

A simple mathematical model consisting of a single neuron was proposed by McCulloch and Pitts [57] at the origins of neural networks. It can be viewed as a nonlinear function that transforms the input variables $x_1, ..., x_d$ into the output variable $z$ (Fig. 2.3).

In this model, the weighted sum of the inputs is carried out, using $w_1, ..., w_d$ (which are analogous to the powers of synapses in the biological network) as weights, thus obtaining:

$$a = \sum_{i=1}^{d} w_i x_i + w_0 \tag{2.1}$$

where the parameter $w_0$ is called the bias.

If we define an additional input $x_0$, set constantly to 1.0, we can write 2.1 as:

$$a = \sum_{i=0}^{d} w_i x_i \tag{2.2}$$

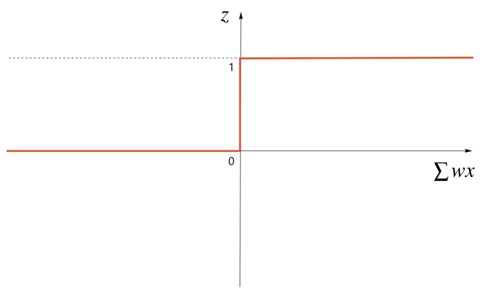
where $x_0 = 1.0$.
The output $z$ is obtained by applying a nonlinear transformation $g()$, called the activation function, to a weighted sum of the input values, yielding:

$$z = g(a) = g(\sum_{i=0}^{d} w_i x_i). \tag{2.3}$$

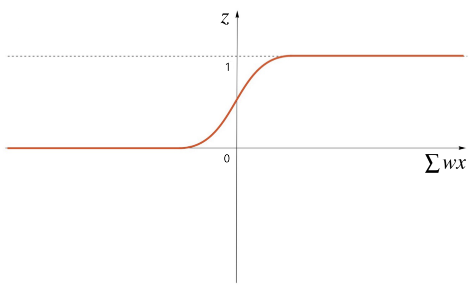
An activation function then is used to map input to output and thus helps a neural network learn complex relationships and patterns among data. These activation functions are all nonlinear because with a linear one it would not be enough to form a universal function approximator, but would reduce everything to a single neuron. The main activation functions are as follows:

- *Threshold Function*: the function returns 1.0 in the case where the weighted sum of the input signals is greater than or equal to zero, and 0.0 in the remaining cases. It is very useful if a binary output signal is needed (Fig. 2.4a).
- *Sigmund Function*: the codomain of the function, that is, the values that the neuron can return, ranges between 0.0 and 1.0 in a continuous interval (Fig. 2.4b).
- *Rectifier Function*: is the most commonly used activation function. It returns 0.0 if the weighted sum of input signals is less than or equal to zero, or $\sum wx$ in other cases. The codomain of the function ranges in this case from 0.0 to infinity (Fig. 2.4c).
- *Hyperbolic Tangent Function*: is similar to Sigmund Function from which it differs in the simple fact that its codomain ranges from -1.0 to +1.0 (Fig. 2.4d).

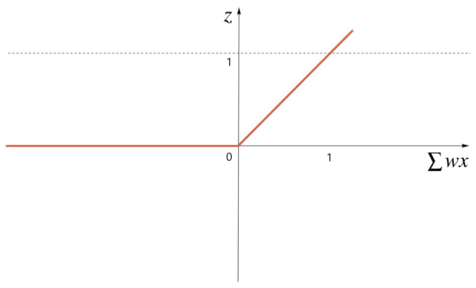
So far we have mentioned weights, without specifying their role; in the neural network a weight corresponds to a synapse, which tells the node (neuron) what information is useful and what is not. Training a model of a neural network corresponds



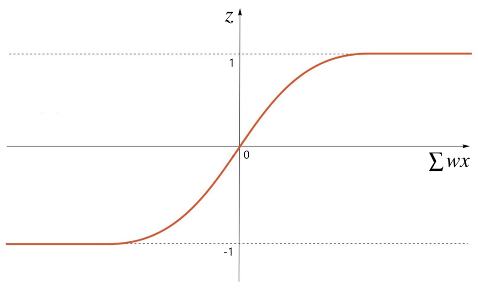
(a) Threshold Function



(b) Sigmund Function



(c) Rectifier Function



(d) Hyperbolic Tangent Function

Figure 2.4: The main activation functions: Threshold Function (Fig. 2.4a), Sigmund Function (Fig. 2.4b), Rectifier Function (Fig. 2.4c) and Hyperbolic Tangent Function (Fig. 2.4d)

precisely to the automatic optimization of these weights. Since a neural network mimics the functioning of a biological neural network, if we want it to learn something, we have to give it experience: we have to instruct it. Exactly as we do with a child, we have to show it a certain process several times, and automatically it will create useful preconceptions concerning that process. Assuming we have a dataset, that is, a collection of input-output pairs $(x, z)$ and giving an input to the network it will multiply it by the weights and return an output $\hat{z}$ that will be different from the output $z$ present in the dataset. Since the goal is to have increasingly accurate predictions, we will define a cost function that accounts for the deviation of our prediction $\hat{z}$ from the reality $z$. There are several types of cost functions, each useful in a different context. The most common is Mean Squared Error. Training a network means trying to minimize the cost function by acting on the values of the weights. That is, the weights are calibrated so that given input $x$, we get an output value of $\hat{z}$ as close to $z$ as possible. For this process, when you have a deep neural network you usually apply the technique of backward propagation: starting from the output layer, you try to define the values of the weights of the last layer in such a way that the expected result is as close as possible to the expected result. Then we proceed backward to the inner layers until we reach the input layer. For each neuron, the concept is always the same: find the values of the weights for which the value returned by the neuron is as close as possible to the value expected by the downstream neuron.

Let us now look at some types of layers and, subsequently, the main architectures used in this work:

- *Fully Connected Layer*: is a type of layer in which every node in the previous layer is connected to every node in the next layer. Since each connection between two nodes corresponds to a weight and thus to a model parameter, a layer of this type gets heavier and heavier as the number of nodes involved increases. It is therefore generally used only as the last layer to obtain the final outcome of the neural network.

- *Convolutional Layer*: is a layer that applies a convolution between an input matrix and a kernel matrix and whose result is a feature map extracted from the input image. The kernel matrix generally has a smaller height and width dimension than the input matrix, however, it has the same number of channels. Since the kernel corresponds to the set of learned parameters and a kernel is smaller than the input, it can be inferred that such a layer is less onerous than a fully connected layer.

- *Pooling Layer*: is a type of layer that is intended to increase the generalization capability of a feature map extracted from a convolutional filter. Indeed, without applying a pooling layer a network consisting only of convolutional layers will be able to find the features for which it is trained but only if they are in the same location. With the pooling layer a neural network assumes the ability to recognize features regardless of their position in the image. Besides that, it also aims to decrease the size of the convolutional feature map to reduce computational costs. The two main types of pooling layers are:

1. *Max Pooling*: Calculate the maximum value for each patch of the feature map.
2. *Average Pooling*: Calculate the average value for each patch on the feature map.

### 2.2.1 Convolutional Neural Network

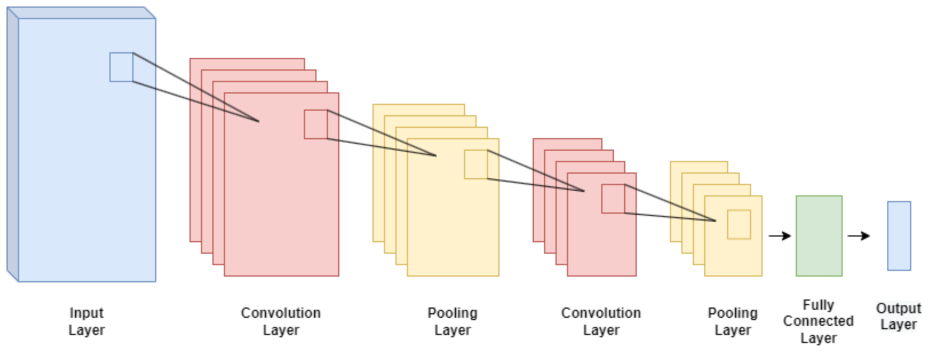


Figure 2.5: Typical Convolutional Neural Network Architecture. It is composed of alternating convolution layer and pooling layer and ends with one or more fully connected layers.

A convolutional neural network (CNN) [58], is a network architecture for deep learning commonly applied to analyze visual imagery. Unlike a traditional feed forward neural network that works on the "general information" of the image, a CNN works and classifies the image based on particular features of the image. In other words, depending on the type of filter used it is possible to identify patterns or objects on the reference image, e.g., figure outlines, vertical lines, horizontal lines, diagonals, etc. In Fig. 2.5 it is possible to see the structure of a typical CNN, which is composed of exactly the concatenation from the three types of layers described above. Specifically, by pairs of convolutional and pooling layers and a final part consisting of one or more fully connected layers.

### 2.2.2 Recurrent Neural Network

A Recurrent Neural Network (RNN) [59], is a type of architecture that differs from feed forward ones, such as CNNs, because it also admits loops and/or interconnections with neurons from a previous level. This feature makes this type of neural network very interesting, because the concept of recurrence intrinsically introduces the concept of a network's memory. In an RNN network, indeed, the output of a neuron can influence itself in a subsequent time step or can influence neurons in the previous chain, which in turn will interfere with the behavior of the neuron on which the loop closes.
There are more than one way to implement an RNN network, several different types of RNNs have been proposed and studied over time. The best known are those that are based on LSTMs (Long Short Term Memory) [60] and GRUs (Gated Recurrent Units) [61]. Thus, the data that an RNN network is capable of processing is a temporal sequence.
If with a CNN network it is possible to recognize objects or patterns in images, with an RNN it is possible to recognize behavior and predict future trends.

In the RNN cell at each instant $t$, the layer will receive not only its input $x_t$ but also its output $z_{t-1}$. The feedback from the output will allow the network to base its decisions on past history. Unrolling the RNN cell over time, through an operation called unfolding of the network, actually results in the same behavior as a feed forward type network (Fig. 2.6). We introduce for the purpose the concept of network unfolding, which translates, in essence, into an operation of transforming an RNN network into a feed-forward type network. Indeed, if one looks at the Fig. 2.6, one can easily see that the RNN neural network has become a feedforward neural network. Clearly a single recurrent neuron has very limited memory capacity, if you need something more complex you need to use more complex neurons such as LSTM depicted in Fig. 2.7.

Whereas a simple recurrent neuron manages only an additional output $h_t$ for the current state and an input $h_{t-1}$ for the previous state, as well as two different activation

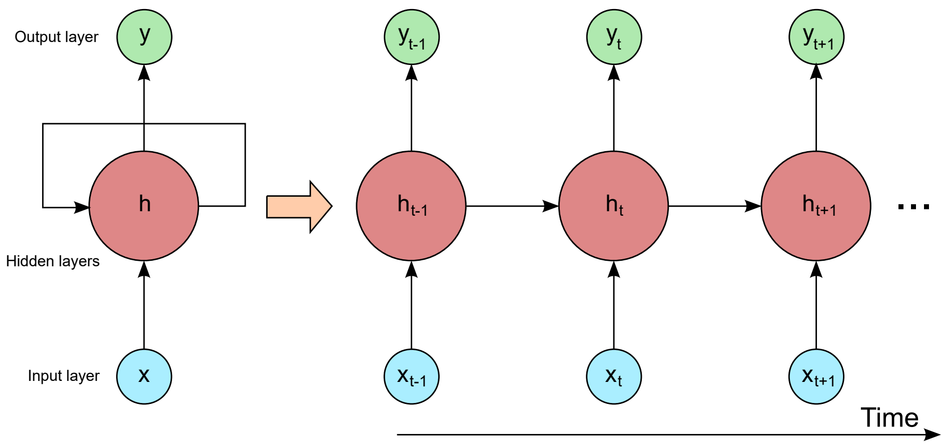


Figure 2.6: RNN cell on the left and its unrolling over time on the right. This process is called unfolding of the network and allows an RNN to be represented as a feed-forward network

functions for state $h$ and output $z$, the LSTM neuron has several gates internally that allow it to decide independently during the training phase what is worth storing or forgetting, whether and how to combine the input with the internal state, and whether and how to return the output. The forget gate decides whether the input information should be thrown away or retained. To it comes the information of the current input $x_t$ and the previous feedback output $h_{t-1}$. A sigmoidal activation function is applied to this information, which will return an output between 0.0 and 1.0. This output will be multiplied to the value of the previous state $c_{t-1}$. Thus if the sigmoidal returns a value close to 0.0, the previous state will tend to reset (be forgotten), while if it returns a value close to 1.0, the previous state will tend to remain the same (be stored). The input gate similarly decides whether the input values $x_t$ and $h_{t-1}$ can be processed together with the previous state $c_{t-1}$, or what remains of it after passing through the forget gate. The processing of the inputs and the current state, or at least the values that the forget and input gate let through, become the neuron's current state $c_t$. Finally, the output gate, similarly to the other two gates, always exploiting the input values $x_t$ and $h_{t-1}$ decides whether the current state $c_t$ can be presented at the output and

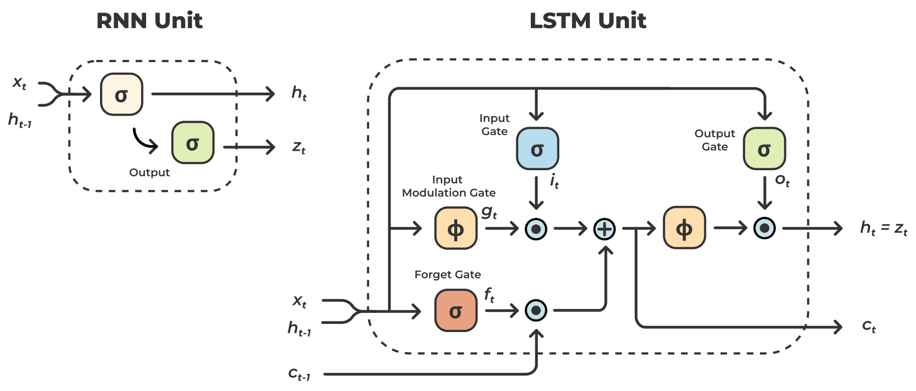


Figure 2.7: Comparison between an RNN Unit and an LSTM Unit, where the difference in structural complexity can be seen. Among the components observable in the LSTM are the gates, each with specific tasks, allowing it, for example, to decide which information to keep in memory and which to discard .

become the output $z_t$ that also corresponds to the next feedback input $h_t$. As can be guessed, therefore, the training of a recurrent neural network made of LSTM neurons turns out to be far more complex since the parameters involved are numerous.

### 2.2.3 Autoencoder

When someone wants to compress the information of a given input such as an image, it is possible, for example, to use combinations of convolutional and fully connected layers, but there are also ready-made and more sophisticated architectures such as Autoencoders (AE). The goal of these neural networks is not only to compress a piece of data with many dimensions (e.g., an image) into a space of latent variables, but it is also to return as output a reconstruction of the input based on the information acquired. This type of architecture is shown in Fig. 2.8 consists mainly of two components:

- *Encoder*: The part of the network that compresses the input into a space of

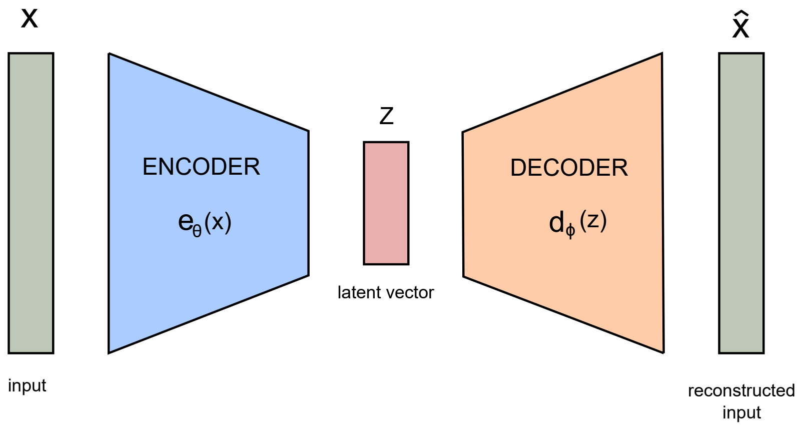


Figure 2.8: Schematic representation of an Autoencoder Architecture. The architecture consists of two components: Encoder and Decoder. The first allows the compression of an input with a certain size into a latent vector $z$ of smaller size. The second, on the other hand, starting from the latent vector $z$ reconstructs the initial input.

latent variables.

- *Decoder*: The part that deals with reconstructing the input based on previously collected information.

Using autoencoders will certainly result in a loss of the information of the supplied data, which means that they will not be able to perfectly reconstruct the input image. However, as the autoencoder is forced to reconstruct the input image as best as possible, it must learn to identify and represent its most significant features. During the model training phase with this type of architecture generally a loss function such as Mean Squared Error is used, in which the encoder input and the decoder output are compared. One problem with AE is that it has latent space that is not regularized, for example, there are some regions/clusters that have generative capacity and from which sensible images similar to those used in training can be reconstructed. But by randomly sampling you could fall back to points outside these regions and you would get a reconstruction that is meaningless. So with this architecture you do not have all the latent space with generative capacity; therefore it is mostly used for compression.

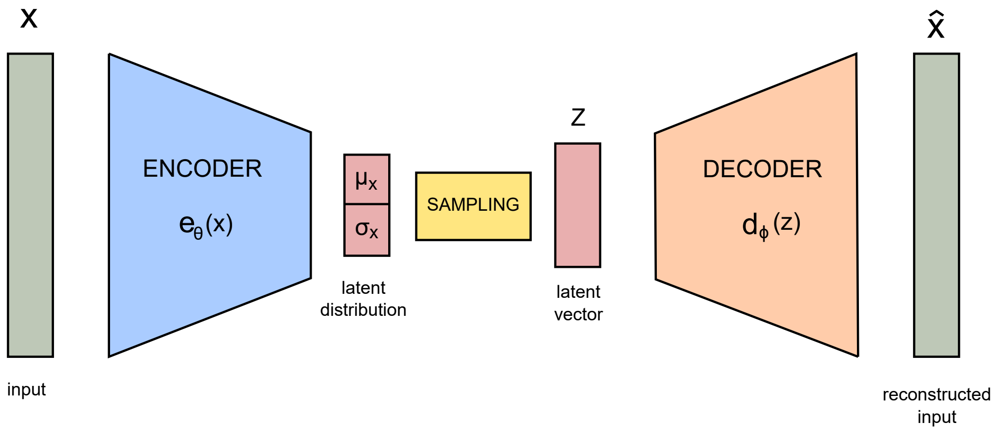


Figure 2.9: Schematic representation of a Variational Autoencoder Architecture. The architecture consists of three components: Encoder, Sampling, and Decoder. The first allows the compression of an input into mean and variance values for each latent variable. The second allows the sampling of a latent vector $z$ from the Gaussians obtained with the mean and variance values generated by the Encoder. The third, on the other hand, reconstructs the input from the latent vector $z$.

### 2.2.4 Variational Autoencoder

Variational Autoencoder (VAE) addresses the issue of non-regularized latent space in autoencoder and provides the generative capability to the entire space. The encoder in the AE offered directly latent vectors. Instead the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution. This constraint makes sure that the latent space is regularized. So by sampling any point in the latent space it is possible to get through the decoder something more or less sensible. So the structure of VAE is like the one in Fig. 2.9 and as we can see it no longer consists only of Encoder and Decoder, but also of Sampling. In this case the encoder doesn't return the latent vector $z$ directly, but a mean and standard deviation for each latent variable. The latent vector is then sampled from the normal distribution obtained from this mean and variance. The decoder part, on the other hand, is exactly

like the classic autoencoder part. In this case the goal of VAE is not only to reconstruct the input, but also to have a latent space normally distributed, so to do this a loss is used which is formed by a reconstruction loss (like that of VAE) and a similarity loss which is the Kullback–Leibler divergence [62] between the latent space distribution and standard gaussian (zero mean and unit variance).
Since the latent vector is obtained by randomly sampling from the generated latent space, there is a problem with backpropagation in the encoder, because we can't trace back errors due to this random sampling. To overcome this problem we usually use what is called a "reparametrization trick", which is to represent the latent vector $z$ as a function of the encoder output:

$$z = \mu_x + \sigma_x \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I) \tag{2.4}$$

## 2.3 Reinforcement Learning

As already mentioned in 2.1.4, when we talk about Reinforcement Learning (RL) we refer to those Machine Learning (ML) methods whose training is not done preliminarily, but the machine learns by a trial-and-error mechanism. The philosophy of Reinforcement Learning is very intuitive and closely resembles the techniques used to make a child learn something or those used for training animals. Indeed, the training process is based on the concept of reward. If an agent, i.e., the subject to be trained, performs a correct action, then it will receive a positive reward value; conversely, if the action taken turns out to be wrong it will receive a disincentive, a negative reward value. So, Reinforcement Learning involves the presence of an agent, who, first of all, due to its perception skills observes the dynamic environment around it, and, based on its knowledge, performs an action that changes its state and that of the external environment. Based on the obtained state, through a feedback mechanism the algorithm will tend to figure out which actions are best and refine the learning. The agent's goal is to learn a policy, that is, a behavior strategy that allows it to identify what are the best actions to perform so that the reward is maximized. Initially, the agent will perform actions that are the result of random trial and error, and slowly it will come to a learning sophisticated tactics that will enable it to make correct and

conscientious choices. Conceptually the idea is that if the agent discovers a way to get a higher reward then it will then tend to always perform it, at the same time the action it is performing may not be the absolute best possible. So if the agent will always try to execute the action that it knows will lead him to a positive reward it will adopt a type of policy that is called Greedy Policy. However, in certain situations, this is not the best solution, because it may bind the agent not to do other types of actions that would lead him to an even better reward. Therefore, a tradeoff between exploration and exploitation is used, adopting what is called an Epsilon Greedy Policy, that is, a generally small (or time-varying and tending to shrink) value is defined for an epsilon variable that defines the probability of choosing an exploration action rather than performing the one that is already known to lead to a given reward. In this way, during training the agent will always have the opportunity to discover new actions that may be better than those it already knows. Additionally, it should be taken into consideration that actions may have long-term consequences. Thus positive rewards may be somewhat delayed in being received. It may sometimes be better to sacrifice immediate better rewards in order to receive even better ones in the future. A Reinforcement Learning system can thus be summarized with the diagram in Fig. 2.10, which represents a continuous loop over time: What happens in the future depends on history, the agent deciding one action rather than another inevitably influences history. Thus a state is defined as: $S_t = f(H_t)$, where $H_t$ is the sequence of observations, actions and reward from time 0 to time $t$.
It is necessary, at this point, to distinguish the state of the environment from the state of the agent. The former is the agent internal representation and contains all the useful information that the agent uses to determine the action to be performed, i.e., the information needed by a Reinforcement Learning algorithm. The second, on the other hand, is not visible to the agent and contains the data with which the environment selects subsequent observations and rewards, thus contains information that is irrelevant to the agent.

After this introduction it is now possible to evaluate what makes Reinforcement Learning different from other Machine Learning paradigms:

- There is no supervisor, but solely a reward signal. So no one specifically tells

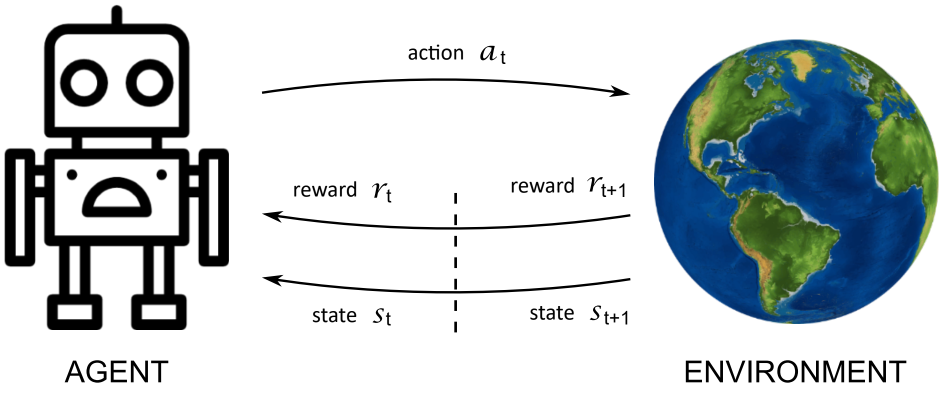


Figure 2.10: Schematic representation of a Markov Decision Process. It shows the interaction between the agent and its environment. At each time step $t$ the agent is in state $s_t$, in which it performs an action $a_t$ for which it receives a reward $r_{t+1}$ and which brings it to state $s_{t+1}$. In Reinforcement Learning this process is repeated until the agent has reached the terminal state at the end of an episode.

the agent what is the best action to take.

- Feedback is delayed and not instantaneous. The agent may realize that an action taken earlier that seemed to bring a low reward may actually turn out to be surprisingly good after some time.
- Sequential decision processing: sequences that last over time are used. In Supervised and Unsupervised a single step is sufficient, whereas in Reinforcement Learning it is difficult to evaluate on a single step.
- Agents' actions change the state of the environment and consequently the sequence of data they receive.

In the field of autonomous driving actually a simple Reinforcement Learning algorithm is not enough. Traditional techniques can only be applied to simple contexts and small problems. For something more complex and onerous there is what is called Deep Reinforcement Learning, which combines the theory of Reinforcement Learning with the capabilities of Deep Learning algorithms. In fact, throughout the project we will use only Deep Reinforcement Learning techniques.

In the next paragraphs, some components of Reinforcement Learning theory will be analyzed in a little more detail.

### 2.3.1 Markov Decision Process

Since the agent's state contains all the useful information in the story, it is possible to define it as a Markov state. Indeed, a state $s_t$ is a Markov state if and only if:

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, s_2, ..., s_t] \tag{2.5}$$

By knowing the current state, which in turn is a function of history, it is in some way possible to predict the future. In the case of a Markov State the current state, summarizing within it the past, is a sufficient statistic of the future. Considering the system at a time step $t$, the agent will perform the action $a_t$ to move from state $s_t$ to state $s_{t+1}$ and receive the reward $r_t$. This process can be defined as a Markov Decision Process (MDP), where $M =< S, A, P, r, \gamma >$ and in which $S$ is a set of Markov states, $A$ is a set of discrete or continuous actions, $P$ is the state transition probability $P(s_{t+1}|s_t, a_t)$, $r$ represents the reward function and $\gamma$ is the discount factor, with a value between 0.0 and 1.0 that modulates the importance of future rewards. The closer $\gamma$ is to 0.0 the more importance is given to immediate rewards over future rewards. So formally the goal of the agent is to find the best policy $\pi$ that maximize the expected return:

$$R_t = \sum_t^T r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t} r_T \tag{2.6}$$

where $T$ represent the time instant of the terminal state, which we now explain what is meant.

Generally in Reinforcement Learning, the agent has a certain goal that it must achieve with certain actions. An episode is defined as the set of the entire sequence of states, actions and rewards from time instant 0 until the time instant in which the agent either achieves the goal or decrees itself according to a metric that it can never achieve it. The final state of an episode is called the terminal state. That being said, we can now formally define policy as a mapping function that for each state $s_t \in S$ predicts an action $a_t \in A$, which is nothing more than the concrete definition of a behavior.

### 2.3.2 Value Functions

Almost all reinforcement learning algorithms are based on estimating value functions, that tell how good it is for the agent to be in a given state. The concept of "how good" is based on the future rewards that can be expected or expected return with respect to the policy that the agent will follow. Because it is the policy that defines the actions to be taken, and it is the actions chosen determine the reward that the agent will take. For MDPs, we can define formally as:

$$V_\pi(s) = \mathbb{E}_\pi(R_t | s_t = s) \tag{2.7}$$

which corresponds to a state-value function obtained with policy $\pi$.
Depending on the chosen policy it is then possible to have different state-value functions. If the policy considered is also the best $\pi^*$, it is possible to calculate the optimal value function $V_{\pi^*}$ in this way:

$$V_{\pi^*}(s) = \max_\pi V_\pi(s) \tag{2.8}$$

This value function corresponds to the one with the highest value compared to all other state-value functions obtainable with the other policies. Similarly, we define the value of taking action $a$ in state $s$ under a policy $\pi$, denoted $Q_\pi(s,a)$, as the expected return starting from $s$, taking the action $a$, and thereafter following policy $\pi$:

$$Q_\pi(s,a) = \mathbb{E}_\pi(R_t | s_t = s, a_t = a) \tag{2.9}$$

We call $Q_\pi(s,a)$ the action-value function for policy $\pi$. Also in this case it is possible to estimate the optimal action value function $Q_{\pi^*}$ following the optimal policy $\pi^*$:

$$Q_{\pi^*}(s,a) = \max_\pi Q_\pi(s,a) \tag{2.10}$$

### 2.3.3 Bellman Expectation Equation

Bellman was an applied mathematician who derived equations that help to solve an Markov Decision Process. Replacing 2.6 in the equation 2.7 it is possible to define the state value function $V_\pi(s)$ iteratively, using the value of the next state:

$$V_\pi(s_t) = \mathbb{E}_\pi(r_{t+1} + \gamma V_\pi(s_{t+1})) \tag{2.11}$$

In practice, Bellman showed that it is possible to define the value function for a given policy $\pi$ in terms of the value function of the next state. This is called Bellman Expectation Equation. Information about the value of the next states is transferred to the current state. The same concept can be applied to the action value function $Q_\pi(s,a)$:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi(r_{t+1} + \gamma Q_\pi(s_{t+1}|a_{t+1})) \tag{2.12}$$

### 2.3.4 Model-free and Model-based reinforcement learning

In Reinforcement Learning, the terms "model-based" and "model-free" refers strictly as to whether, whilst during learning or acting, the agent uses predictions of the environment response. A model-based algorithm, as it sounds, involves an agent trying to understand the environment and creating a model based on its interactions with it. In contrast, in model-free, the agent does not know the environment and tries to learn from the consequences of its actions. If the agent is able to predict the reward of an action before performing it, thus planning its action, the algorithm is model-based. Whereas if it has to actually perform the action to see what happens and learn from it, the algorithm is model-free.

### 2.3.5 Elementary Solution Methods

There are three classes of elementary methods for solving the reinforcement learning problem: Dynamic Programming, Monte Carlo method and Temporal-Difference Learning. All these methods solve the complete version of the problem, including delayed rewards:

- *Dynamic Programming (DP)*: It is a set of algorithms that can solve a problem in which you have a perfect model of the environment and in which an agent can take only discrete actions. DP essentially solves a "planning" problem rather than a more general RL problem. The main difference is that for a general RL problem, the environment can be very complex and dynamic, and its specifications are not initially known. This method is based on the concept of dividing the problem into subproblems and solving them, then combining the solutions of the subproblems. These algorithms break the process into two parts. Initially the policy $\pi$ using Bellman Expectation Equation is evaluated and then one acts greedy to this evaluated value function to find an optimal policy. The algorithm proceeds by iterating these two steps until it converges to Optimal Value Function and Optimal Policy.

- *Monte Carlo (MC)*: In the real world, it is difficult to have complete information about the environment in order to apply DP, and one possible solution is to use the MC method. The Monte Carlo method involves an agent learning from the environment by interacting with it and collecting samples. In this learning process, it is necessary for the agent to reach the terminal state. The values of each state are updated only according to the final reward and not according to the estimates of other neighboring states. Thus MC can only be applied to what are called episodic MDPs. The updated state-value formula is:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{2.13}$$

  where $\alpha$ is the learning rate.

- *Temporal Difference (TD)*: The Monte Carlo reinforcement learning algorithm overcomes the difficulty of state-value estimation caused by an unknown model. However, a disadvantage is that the state-value can only be updated after the whole episode. Instead, in TD it is also possible to update it to the next step, as in the Bellman Equation. The trick is that rather than attempting to calculate the total future reward, temporal difference learning just attempts to predict the combination of immediate reward and its own reward prediction at the next moment in time. Now when the next moment comes and brings fresh information with it, the new prediction is compared with the expected prediction. If these two predictions are different from each other, the Temporal Difference Learning algorithm will calculate how different the predictions are from each other and make use of this temporal difference to adjust the old prediction toward the new prediction. There are several TD methods, and the simplest is: TD(0), which has the following update formula for the state-value function:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.14}$$

### 2.3.6 Policy Gradient Method

So far we have seen methods that improve the policy implicitly, by optimizing the state value function or the action value function. Policy Gradient Methods instead target at modeling and optimizing the policy directly. As we have already mentioned, the goal of Reinforcement Learning is to maximize the "expected" reward when following a policy $\pi$. We define $\theta$ as the set of parameters of policy $\pi$. Thus we obtain that for a given trajectory $\tau$, maximizing the total reward $r(\tau)$ of that trajectory corresponds to:

$$J(\theta) = \mathbb{E}_\pi[r(\tau)] \tag{2.15}$$

Being able to find the parameters $\theta^*$ that maximize $J$ corresponds to solving the policy optimization problem. A standard approach to solving this maximization problem in Machine Learning literature is to use Gradient Ascent or Gradient Descent. Taking the case of gradient ascent as an example, sliding of the parameters is done according to the following update rule:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) = \theta_t + \alpha \nabla \mathbb{E}_\pi[r(\tau)] \tag{2.16}$$

Since integrals are always bad in a computational setting, we can reformulate the gradient in this way:

$$\begin{aligned}\nabla \mathbb{E}_{\pi}[r(\tau)] &= \nabla \int \pi(\tau) r(tau)\, d\tau \\ &= \int \nabla \pi(\tau) r(tau)\, d\tau \\ &= \int \pi(\tau) \nabla \log \pi(\tau) r(tau)\, d\tau \\ &= \mathbb{E}_{\pi}[r(\tau) \nabla \log \pi(\tau)]\end{aligned} \tag{2.17}$$

The policy gradient can be represented as an expectation. It means we can use sampling to approximate it. Also, we sample the value of $r$ but not differentiate it. It makes sense because the rewards do not directly depend on how we parameterize the model. But the trajectories $\tau$ are. Considering that $\pi_{\theta}(\tau)$ is defined as:

$$\pi_{\theta}(\tau) = p(s_1) \prod_{t=1}^{T} \pi_{\theta}(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2.18}$$

The log of $\pi_{\theta}(\tau)$ corresponds to:

$$\log \pi_{\theta}(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_{\theta}(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \tag{2.19}$$

Since the first and third components do not depend on $\theta$ we can remove them.
So the formula for updating the policy parameters $\theta$ then becomes:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \tag{2.20}$$

where:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \tag{2.21}$$

One advantage these methods have over those seen previously is that they depend only indirectly on state. While in the others, uncertain state information leads to making those approaches impossible to apply. In this case uncertainty can degrade policy performance, but policy techniques can still be applied. Thanks to gradient ascent or gradient descent, however, convergence to at least a local optimum is always guaranteed. An additional advantage of these methods is that the policy representation can be chosen so that it is meaningful to the task at hand and can incorporate domain knowledge. Clearly, policy gradients are not the solution to all problems but also have significant problems. For example, the value function methods are guaranteed to converge to a global maximum while policy gradients only converge to a local maximum and there may be many maxima in discrete problems. In addition, policy gradient methods are often quite challenging to apply, mainly because it is necessary to have considerable knowledge of the system you want to control in order to define reasonable policies.

### 2.3.7 Actor-Critic Method

Briefly summarizing the methods seen above, we can say that in policy-based RL the optimal policy is computed by manipulating directly the policy, and value-based function implicitly finds the optimal policy by discovering the optimal value function. Policy-based RL is effective in high dimensional and stochastic continuous action spaces, and learning stochastic policies. At the same time, value-based RL excels in sample efficiency and stability.

In Actor-Critic methods, these two approaches are combined. Specifically, the policy is exactly parameterized and optimized as in policy gradient methods, but simultaneously the state value function is estimated to reduce the variance of updates. In other words, Actor-Critic is a Temporal Difference version of Policy gradient.
This methods consists of two neural networks, the Actor and the Critic. The former decides what action should be taken by the agent based on the policy, while the latter defines how good the action taken is by calculating the value function. In this type of algorithm there is the concept of Advantage $A(s_t, a_t)$, which defines how good a state is compared to the expected state and consequently how much the action taken deviates from what should have been performed. Advantage is thus represented by the following expression:

$$A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.22}$$

Then substituting this into 2.21 instead of policy gives:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\end{aligned} \tag{2.23}$$

The update formulas of actor and critic parameters result as follows:

$$\begin{aligned}\theta_{t+1}^{\pi} &= \theta_t^{\pi} + \alpha_\pi A(s_t, a_t)\frac{\nabla_{\theta^\pi}\pi(a_t|s_t, \theta^\pi)}{\pi(a_t|s_t, \theta^\pi)} \\ &= \theta_t^{\pi} + \alpha_\pi A(s_t, a_t)\nabla_{\theta^\pi}\log \pi(a_t|s_t, \theta^\pi)\end{aligned} \tag{2.24}$$

$$\theta_{t+1}^{v} = \theta_t^{v} + \alpha_v A(s_t, a_t)\nabla_{\theta^v} V(s_t, \theta) \tag{2.25}$$

This corresponds to the policy update method used in the Advantage Actor Critic (A2C) algorithm. The key concept is that if the Advantage value is positive, it means that the action taken at that particular instant was good or even better than expected. This implies the policy to take this into consideration and to execute more frequently this type of action that was extremely positive. In case the action turns out to be negative it is simply ignored and different or opposite actions to the latter are attempted.

### 2.3.8 Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) is an algorithm developed by Google's DeepMind [63] and corresponds to the asynchronous version of A2C. The basic principle is the same, but the difference is that in A3C there is not a single agent optimizing the network, but there are multiple agents independent of each other acting in separate environments or in the same environment and simultaneously optimizing the same global network (Fig. 2.11).

At the beginning of every episode, each agent creates a local copy of the global network, then, during the episode, continues to update the parameters of its local network. Every $N$ steps, with $N$ variables, the agent sends an update of its parameters to the global network. In this algorithm because there are multiple agents involved at the same time encountering different situations the typical RL exploration process

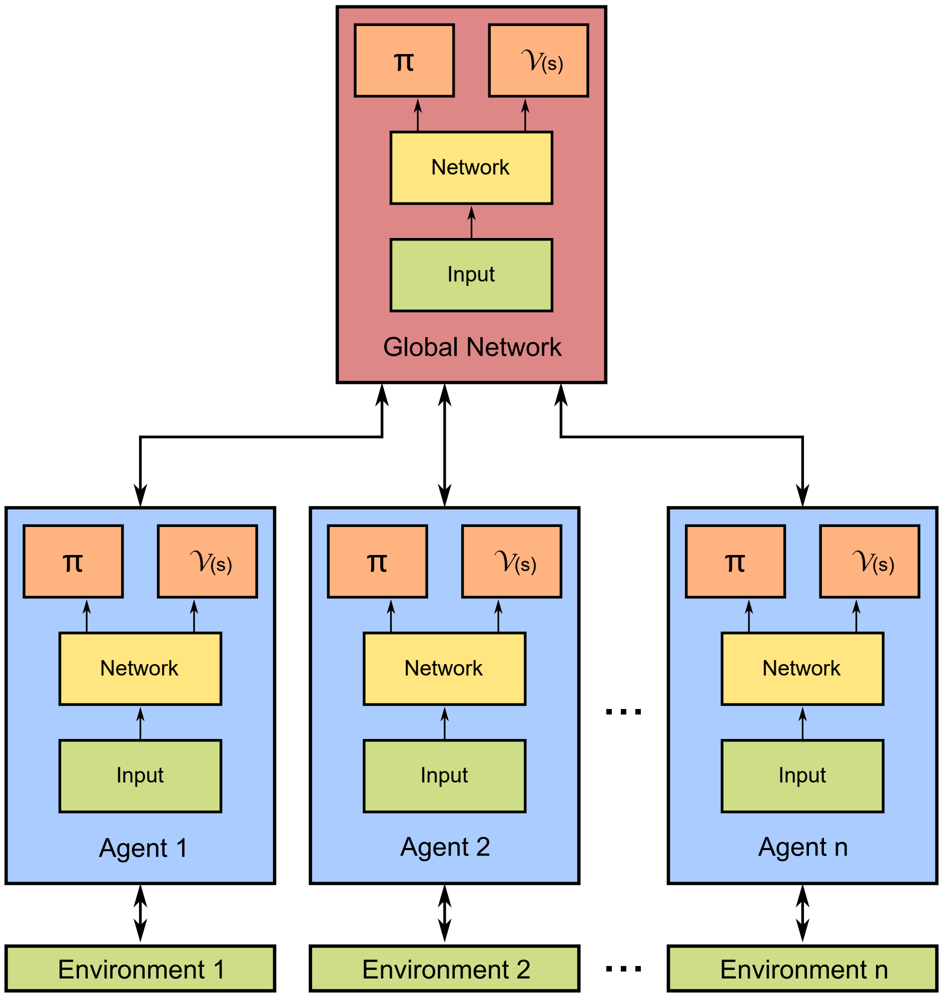


Figure 2.11: Scheme representing the architecture of a neural network based on Asynchronous Advantage Actor-Critic. Each agent has its own local network and interacts with its own environment, but all agents participate in policy and value function optimization of a global network.

is greatly speeded up and, in parallel, a good generalization capability is achieved. With this algorithm there is also the possibility of developing a Multi-Agent system, a system in which two or more agents share the same environment and interact or negotiate with each other and, at the same time, both participate in training the global network. This algorithm lends itself very well to the purposes of the research project and has been the most widely used algorithm, both Single-Agent and Multi-Agent versions.

### 2.3.9 Delayed Asynchronous Advantage Actor-Critic

Actually, a variant of the classical A3C was used in the project. Which was called Delayed Asynchronous Advantage Actor-Critic (D-A3C) by the authors and which they show in [64] and [65] that it performs better in autonomous driving tasks. The D-A3C algorithm differs from the classical algorithm simply by a particular in the updating of global network parameters. Indeed, in both algorithms (A3C and D-A3C) each agent starts the episode with a local copy of the latest version of the global network and contributes to the update of the global network parameters. In addition, in both at every step each agent updates the parameters of its local network. The difference that distinguishes these two algorithms is that in A3C after a predetermined amount of steps it sends an update to the global network based on its current parameters; on the other hand, in D-A3C, the update is sent to the global network only at the end of the episode, when the terminal state is reached. The name "Delayed" comes precisely from the fact that the update occurs in a delayed manner compared to the classical one.

## 2.4 Imitation Learning

Certainly Reinforcement Learning is one of the most interesting areas of machine learning, mainly because of its particularity of autonomously learning a policy using a trial-and-error mechanism based on interaction with an environment. The goal of RL is precisely to learn the best policy that maximizes the long-term cumulative rewards. To get good results with this kind of algorithm, however, it is necessary to

manually define the reward function. In certain contexts, it is not always easy to manually design a reward function that satisfies the desired behavior.
A feasible solution to this problem is Imitation Learning (IL). In this learning method, the agent does not use the trial-and-error mechanism and does not exploit a reward function to evaluate the best policy, but tries to learn the best policy by imitating the behavior of an expert, typically that of a human. The simplest form of Imitation Learning is Behavior Cloning (BC), which focuses on learning the expert's policy using supervised learning. The most famous example in the literature is ALVINN [66], an autonomous vehicle that had learned to obtain steering angle values directly from data obtained from sensors. This project dates back to 1989 and was the first application of Imitation Learning in the field of autonomous driving.
Another very interesting work is that of NVIDIA [47], in which, thanks to a dataset manually collected by themselves, they were able to train a neural network that was able to learn the task of lane keeping, controlling the steering angle in different contexts: urban road, parking lots, etc.
Clearly, this approach also has limitations, the first is definitely the need for a complete dataset, which can involve a lot of expensive work to obtain it. Differently in Reinforcement Learning there are some approaches, Model-Free ones that do not strictly need a dataset. As consequence, RL has good generalization capabilities, instead, imitation of a behavior limits the system to learn only what is included in the dataset and, therefore, other techniques are needed to achieve generalization in IL.
A final important consideration to complete the comparison between these two approaches and that is crucial in the field of autonomous driving is related to whether or not to use a simulator. Certainly for both it is strongly recommended to be used for all its benefits, but while in IL it is not strictly necessary, in RL it is strongly recommended. Both in the training phase to avoid accidents being RL based on the trial-and-error mechanism, and in the testing phase to speed up the debugging phase and to be aware of the vehicle behavior before testing it on a real vehicle. In the next chapter we will give a more specific overview of simulators generally used in RL, with a focus on the one used in this research project.

# Chapter 3

# Simulators

In this chapter we will clarify some aspects about a fundamental component of Reinforcement Learning: the simulator. Let's start with defining the term simulation as it's quite an abstract concept. Simulation is nothing but the imitation of a real behavior, environment, process or system in time. In other words, thanks to a simulator we can recreate a copy of reality within software. Clearly there are limitations and advantages. The main limitation is caused by the inherent characteristics of reality, namely that it changes over time and is infinitely complicated. Therefore it is impossible to recreate it faithfully, but only to generate an approximation that tends to be as true as possible. Nowadays, with current technologies there is a possibility in any case to obtain results that are very close to reality. Nonetheless, one must also take into account that in certain contexts such precision is not necessary and indeed it is more convenient to have a simplified reproduction of reality. For example, in Reinforcement Learning the simulator plays a key role, both in the training pipeline and during the testing phase. Indeed, Reinforcement Learning uses the trial-and-error approach to learning, and during this process it is important that it sees all kinds of situations to achieve an optimal policy. Autonomous driving is one of those contexts in which it would be impossible to train a model for an agent by means of RL without using a simulator, mainly because the vehicle would continually tend to have accidents or go off the road. Since RL is a learning mechanism that is basically complex, it is necessary that the environment from which features are extracted be both simple but meaningful. A simulator can be perfect for this purpose, because it is indeed a simplification of reality that predominantly contains the most important details. In addition to this a huge advantage of using a simulator in RL is in the training time, with good hardware it is possible to train a model in a few hours. If, on the other hand, we were to consider, for example, training an anthropomorphic robot to walk without a simulator, an exaggerated amount of time and constant supervision by a human would be required. The application fields in which RL is used to perform certain tasks are different. The main tests have been in the areas of gaming, robotics and autonomous driving.

In the world of scientific research, a fundamental concept is the reproducibility of an experiment. Without being able to recreate an experiment and compare it with one's own work, it is difficult to determine whether progress has been made and to evaluate it against the state of the art. In the survey that Nature [67] published in 2016 they showed the results of a short online questionnaire done by 1576 researchers specifically about reproducibility. The result was striking, because more than 70 percent of the researchers admitted that they have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments. In this regard, some groups of researchers have developed open source libraries or simulators to overcome this problem and have already become a reference. We elaborate in the next sections of this chapter on what these simulators are, dividing them between generic reinforcement learning simulators and those suitable for autonomous driving. The chapter will conclude with a presentation of those used within this research project.

## 3.1 Reinforcement Learning Simulators

In 2016, the OpenAI organization involved in artificial intelligence research developed an open source Python library [68] that allows users to create new environments or test their algorithm on one of the many already in the library. Basically they have provided the community with an effective way to compare RL algorithms by implementing a standard API to communicate between learning algorithms and environments. Thanks to this library in the field of RL, the problem of lack of standardization in papers has been removed. A researcher no longer has to deal with recreating an environment and then testing his algorithm in it, but can focus only on developing his algorithm and can test it using the environment he needs without knowing how it was implemented. Another key aspect is that since this library has taken hold there has been a benchmarking system right away in which it is extremely easy to compare one's algorithm with any other state-of-the-art algorithm. OpenAI Gym provides a diverse suite of environments that you can divide into different categories based on the purpose of your research and from the data you need. The division of environments that they themselves make separates them into these collections:

- *Classic control and toy text*: used primarily for complete small-scale tasks.
- *Algorithmic*: perform different types of calculations, from the simplest to the most complex.
- *Atari*: play classic Atari games such as: Space Invaders, Pac-Man, Breakout, and many others.
- *Board Game*: this category includes all those types of games where two or more participants compete against each other such as Go.
- *2D and 3D robots*: controlling and making a robot learn tasks. Here the Mujoco [69] physic engine is exploited to more accurately recreate the physics of reality in simulation.

In parallel with OpenAI other research groups have in turn developed other suites of environments for Reinforcement Learning. In particular, DeepMind has introduced some alternative tools and libraries:

- *AI Safety Gridworlds* [70]: which is a set of environments that allow the evaluation of various safety properties of intelligent agents.

- *DeepMind Control Suite* [71]: DeepMind's DM Control Suite resembles OpenAI's fifth collection of environments and also exploits the physics introduced by Mujoco [69].
- *DeepMind Lab* [72]: is a 3D customisable game-like platform tailored for agent-based AI research. It has the characteristic of simulating with a first-person point of view. It provides a variety of 3D navigation and puzzle-solving tasks.

Preceding these can be found The Arcade Learning Environment [73] a framework that allowed researchers to create AI agents for Atari 2600 games. In addition to these, there are other simulation environments related to autonomous driving and we will discuss them in detail in the next chapter.

## 3.2 Autonomous Driving Simulators

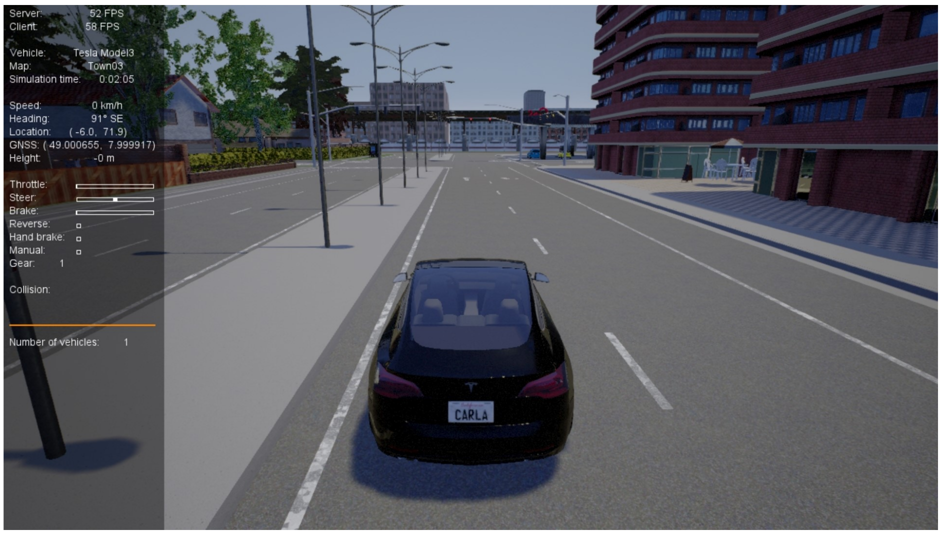
Driving is one of the most complicated activities when it comes to teaching an AI agent. In the field of autonomous driving, the use of a simulator is increasingly a common practice, both during the training and testing phases. This makes it possible to drastically reduce the time required for training and especially debugging algorithms. In addition, a simulator opens up the possibility of generating any possible situation with ease, which would be much more complex in the real world. Among developers of RL algorithms there are two currents of thought, those who do not use a simulator but train the agent simply by providing him with scalar data (such as vehicle speed, vehicle size, position of roadway lines, etc.) and those who use a simulator for both the training and testing parts. In the former case, complexity is drastically reduced, which greatly simplifies the learning process, but it exposes the system to a high risk of overfitting on the scalar information provided to it, making the algorithm less generalizable. A simulator then, if it is structured well allows you during the test phase (and potentially during the training phase as well) to preview the behavior of the vehicle, something you could not do using only the scalars. In order to have a proper simulation environment it is necessary that this takes into account the following key parts:

- *Real Physics and Dynamics*: both the vehicle itself and the other components and obstacles in the environment must be as realistic as possible.
- *Observations*: includes all those elements that are present in the environment and that the vehicle must take into account in order to drive correctly. For example, the shapes of objects, lines of the road, signs, etc.
- *Sensor feed*: the input of the algorithm in simulation must be equivalent to that used later on the real vehicle. So the perception part must be the same.

In this section we will mainly focus on autonomous driving simulators used to train RL agents. In particular, we examine the characteristics of two different families of simulators, Realistic Graphic Simulators that try to faithfully reproduce reality and Synthetic Simulators that reproduce it more schematically, dwelling only on certain features of the environment.

### 3.2.1 Realistic Graphic Simulators

That of realistic graphic simulators is a highly active area of research nowadays. Gaming, racing, and the advent of autonomous driving bring continuous incentives for ever better and more detailed simulators. The massive deployment of Deep Learning techniques for image reconstruction is facilitating this progress. In the context of video games, Grand Theft Auto V (GTA V) [74] is one of the most realistic and most widely used simulators for autonomous driving. Within it, pseudo-realistic image sequences and other data can be collected with special tools. However, the fact that it is based on an always static map turns out to be a limitation. Customization for this type of tool would be essential, because it would allow the creation of complete datasets and reproduce particular situations.
It is exactly for this purpose that the open-source CARLA (CAR Learning to Act) simulator [75] was born, developed specifically for the training, and validation of autonomous driving systems (Fig. 3.1). This simulator provides a countless amount of open digital assets (urban layout, buildings, vehicles) that are freely usable. It also



```
Server:           52 FPS
Client:           58 FPS

Vehicle:          Tesla Model3
Map:              Town03
Simulation time:  0:02:05

Speed:            0 km/h
Heading:          91° SE
Location:         ( -6.0,  71.9)
GNSS: ( 49.000655,  7.999917)
Height:           -0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:      1

Collision:

Number of vehicles:      1
```

Figure 3.1: Carla simulator.

provides a complete and flexible set of APIs that allows users to control all aspects related to the simulation. Everything is configurable: types of sensors, the behavior of other vehicles or pedestrians, weather, and much more. This type of simulator, although extremely engaging and interesting, still has limitations for some autonomous driving tasks. First of all, as we mentioned earlier, trying to reproduce reality is a very complicated task, and these simulators in some way must necessarily make an approximation. In addition, the introduced traffic agents have rule-based behavior, which deviates from human behavior and might be uncomfortable in a real vehicle. If the goal is to teach the agent to drive by Deep Reinforcement Learning or with Imitation Learning, as in our case, these two problems just mentioned are extremely limiting, because they incorrectly influence the policy that the agent learns. Finally, on these simulators, the focus is more on visual quality than on other aspects. For example, vehicle dynamics should be much more like the dynamics of a real self-driving car, with actuation delays and other factors influencing behavior. For these reasons, these types of simulators are not suitable for the purpose of this research.

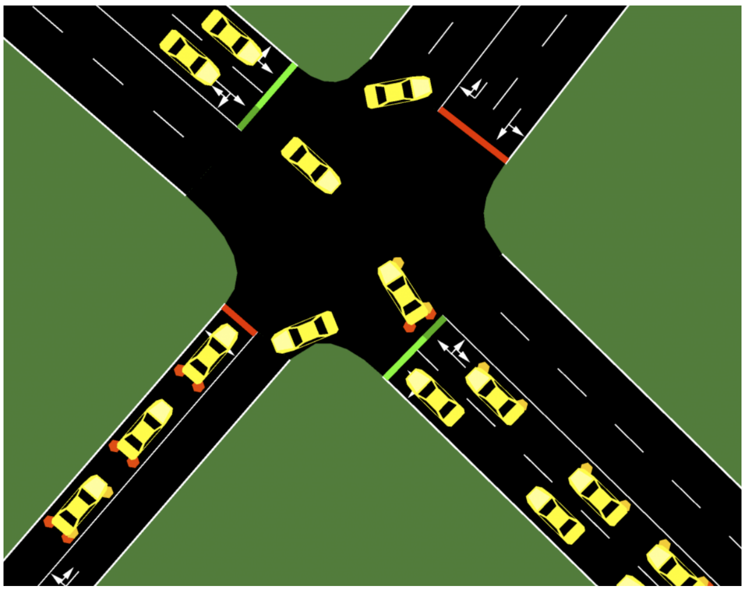


Figure 3.2: SUMO simulator.

### 3.2.2 Synthetic Simulators

Synthetic Simulators does not attempt to represent reality, but simply certain features. The details present in reality are often superfluous. A planning and control algorithm needs only some aspects of the environment to achieve correct and comfortable driving, not all the details. Typically, synthetic simulators consist of a bird's-eye view of the scene in which only the basic parts are schematically represented: roadways, obstacles, ego vehicle, traffic signals, etc. A snapshot of SUMO (Simulation of Urban MObility) [76], an open source, microscopic and continuous multi-modal traffic simulator, is shown in Fig. 3.2. As can be seen, the scene is simplified, all details of the external environment that do not affect driving have been removed, and only the basic ones have been retained. Agents are stylized and some traffic signs are represented simply by colored lines, such as stop sign in red, yield signal in yellow, etc.

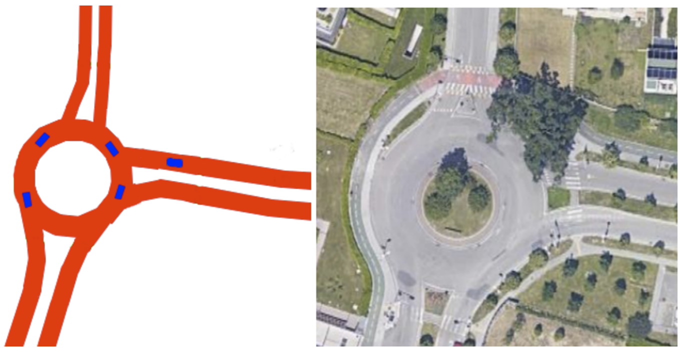


Figure 3.3: Synthetic representation of a roundabout in Parma and its real counterpart. In the simulated image, the red part corresponds to the navigable space and the blue rectangles are the agents.

## 3.3 Multi-Agent Traffic Simulator and HD Simulator

In this section we will discuss the two different simulators that were used in this research project. The first is a synthetic simulator that was implemented in [77] and also used in [64] and [65]. This simulator is based on Cairo [78], an open source 2D graphics library. The purpose for which it was developed is to synthetically represent real scenes, and especially roundabout, as in Fig. 3.3. The authors' goal was very similar to that of this project, indeed, they used RL as main approach and Asynchronous Advantage Actor-Critic (A3C), described in 2.3.8, as algorithm. Although in our case we use it to address the intersection problem, they instead focused on the traffic circle entry problem. The simulator was developed to be used in a multi-agent system, to give them the ability to negotiate and interact with each other. In this environment each agent was able to perceive a surrounding area of $50 \times 50$ meters from which it was possible to derive 3 different channels that could then be used as input to the neural network: Navigable Space, Path and Obstacles. This same information is then readily available on a real vehicle through perception and localization algorithms.

For the second part of the project in which we developed a Deep Reinforcement Learning Planner capable of driving safely and comfortably in an urban environment, it was necessary to implement a new, more comprehensive simulator based on HD

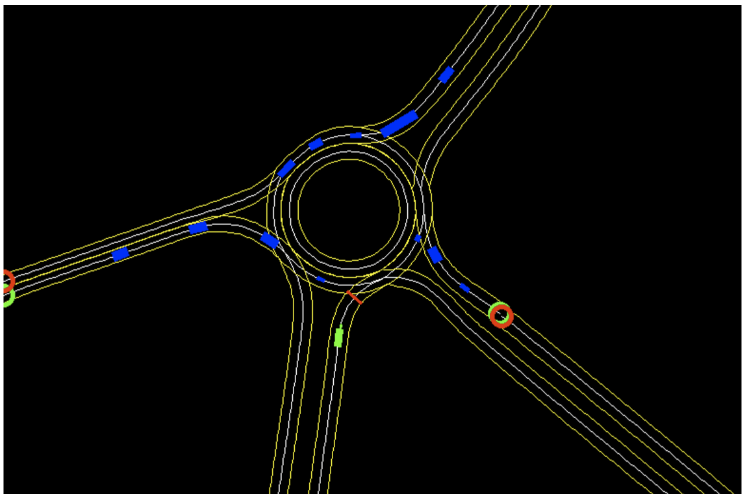


Figure 3.4: Top-view of a roundabout depicted on a synthetic simulator based on HD maps. The yellow lines correspond to the road lanes. The blue rectangles are the traffic agents and the green rectangle is the ego agent. The red line is the stop-line that sees the green ego agent.

maps [40]. This new simulator allowed us to greatly speed up the process of creating new scenarios for training because they can be directly extrapolated from the HD maps. Fig. 3.4 shows the view of a complete scenario, from which a reduced view of the agent and consequently the input channels for the neural network can be obtained at each time instant, as we will explain later in 5.1. An additional advantage that the introduction of HD maps provided was the possibility of having more "real" information about the scenario, such as, for example, speed limits, exact stopline locations, roadway dimensions, and much more. All very useful data to create a more complete and accurate reward function, which is necessary to improve the behavior of RL agents.

# Chapter 4

# Intersection Handling using Deep Reinforcement Learning

A previous paper, [64] explored the capabilities of a Deep Reinforcement Learning algorithm in dealing with the entry of an autonomous vehicle into a traffic circle. In the paper, they used a variant of the Asynchronous Advantage Actor-Critic (A3C) called Delayed-A3C, which was described in 2.3.9 . The neural network they implemented was capable of predicting the probability of entering a roundabout among three different cases: permitted, not permitted and caution; it was also capable of defining a discrete action that was reflected in the acceleration value used to perform the maneuver. Based on this work, we tried to obtain a more complex model in which an agent, once trained, was able to deal with intersections. Since in [64] a limitation had been found due to the choice of a discrete action, we decided not to predict a probability and then later estimate the acceleration, but the model is able to directly estimate the actions to be performed by the vehicle, acceleration and steering angle. In this way, the system is able at any instant to modulate its behavior directly based on the current observed state. So, we propose a Multi-Agent System, using a continuous, model free Deep Reinforcement Learning algorithm that allows acceleration and steering angle to be estimated every 100 milliseconds, in order to cross intersection scenarios. In this work we focused on the type of intersections where traffic lights are not present, because with them is simpler and solvable even with only a few rules. For this reason we decided to focus on intersections that have only traffic signs and where agents simply have to learn how to handle precedences and drive safely. In the final part of the chapter we will also show how this system achieved better performance than classical rule-based methods, especially under heavy traffic conditions. Finally, also in this part of the project we tested this model on real scenarios taken from inD Dataset (Intersection Drone Dataset) [79] and showed how it is able to generalize even with real recorded traffic data and on scenarios never seen during training.

## 4.1 Intersection Handling Problem

Deep Reinforcement Learning is a paradigm that fits very well for certain uses, such as solving Atari games ([80], [63]), robot control ([81], [82], [83]) and others. In contrast, in other contexts, such as autonomous driving, it is far from trivial to be able to achieve good results with this method of learning, both because of the mutability of the environment and the complexity of the agent itself. In the specific case of intersection management and more generally for evaluating potential maneuvers, a typical rule-based method that is widely used is the time-to-collision (TTC) algorithm [84]. Time-to-collision provides a valuable behavioral safety measure in a self-driving vehicle. By knowing whether collisions are imminent and when they might occur and with what object, an autonomous system can better plan safe maneuvers in its environment or possibly perform emergency evasive maneuvers if necessary. To estimate the collision time between two dynamic agents, their predicted paths are estimated in this algorithm and it is evaluated whether possible collision points exist (Fig 4.1). If a collision point exists then the time to collision corresponds to the amount of time before the collision. The main limitation of this algorithm is that it relies on an accurate understanding of the space that will be occupied by agents in the future to obtain estimates of the time to collision. However, this approach is extremely complex because it relies on the accuracy of the geometries of the agents and the accuracy of the predicted trajectories (position, heading, and velocity). Since it is difficult to obtain accurate or exact values for both, it follows that the collision time must necessarily

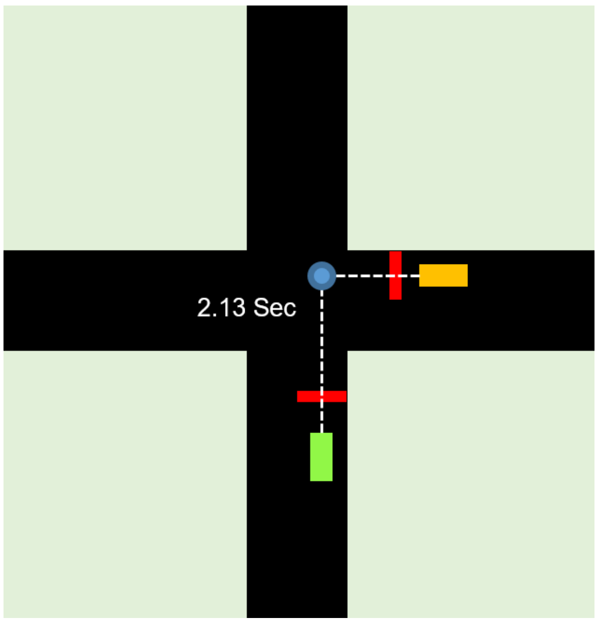


Figure 4.1: Time-to-collision concept representation. The green rectangle is the ego agent and the orange rectangle is the traffic agent. The blue dot represents the collision point and those dashed in white the trajectories of the two agents. The numerical value in seconds represents an estimated time-to-collision for the two agents considering that they will move with constant speed.

be treated as an approximation. In addition in these estimates it is necessary to define constraints that may not correspond to real situations, e.g., the velocities of the two agents may vary over time from those assumed to define the collision point, furthermore, the intentions of the other agents may also be different from those predicted.

It is precisely because of all these limitations caused by hard coded road rules that a strand of research has developed that attempts to address the problem of intersection management using neural networks. [85] have proposed a scalable alternative trained and tested on multi-agent environments in which road rules emerge as optimal solutions to the traffic flow maximization problem. This is also complemented by the work [86] in which they address intersections as a reinforcement learning problem

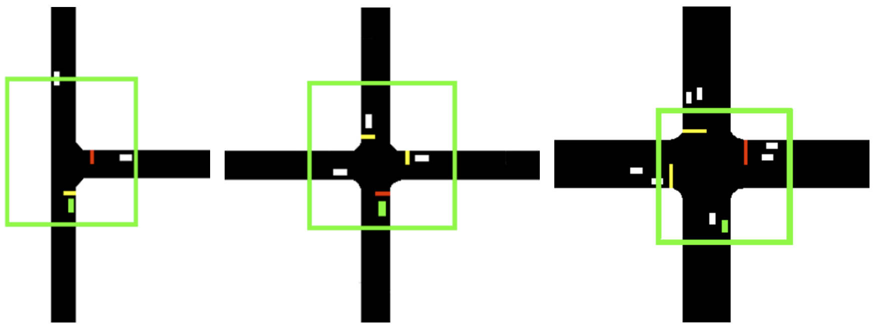
using two different Deep Q Networks (DQNs) ([80], [87]) on a variety of intersection scenarios. In these papers, however, although are interesting approaches, they only succeed in obtaining policies that do not generalize well. Indeed, as can be seen from the results and as they themselves point out, albeit rarely, they sometimes lead to collisions. Moreover, it is also necessary to take into account that in these works the traffic agents involved follow control actions dictated by the Intelligent Driving Model (IDM) algorithm. In this way there is in no interaction or negotiation between the main agent and the traffic agents. Differently in reality this negotiation phase is natural when human drivers approach the intersection. So, the agent will be trained with behaviors different from human ones and then considering a possible testing phase on a real vehicle a solution would have to be found to bridge this gap between simulated and real data.
In parallel to this work, to minimize uncertainty delays and collisions, [88] proposed the: decentralized coordination learning of autonomous intersection management (DCL-AIM) to optimize control policy. The movement and actions of each individual agent are modeled as multi-agent Markov decision processes (MAMDPs) and the system is trained using multi-agent reinforcement learning.

In this work, we decided to use a multi-agent approach, in which each agent is able to negotiate with other vehicles and evaluate and adapt its behavior every 100 milliseconds. Each agent involved in the scenario contributes during the training phase and its actions influence those of the others, and vice versa. The neural network that we trained using Deep Reinforcement Learning allows us to estimate acceleration and steering angle with which the agent must cross the intersection safely and in a reasonable time. The choice to use a multi-agent system in which the vehicles involved learn to interact, in addition to the reasons mentioned above, is also dictated by the fact that we do not want to always assign the lowest priority to the agent over the traffic ones, as for example is the case in [86] and [89]. Such a system could lead to even very high delays and would in no way be usable in a congested or otherwise complex traffic situation. In our system, due to the negotiation between agents, each individual vehicle is able to understand all priorities, including the right of way. That is, that the moment two vehicles coming from two different branches of the intersection and having the same traffic sign (or no traffic sign) would collide in the intersection then the car coming from the right has priority and the one on the left must stop to let it pass. If, on the other hand, the precedences are defined by road signs then the agents follow the order of entry specified by those. It is important to note that the choice of these regulations with which the agents are trained is due to the fact that the final goal is to test the system on a real vehicle, and to do so, we found it convenient to use the regulations of the country in which we conducted the research.

## 4.2 Environment Definition

Before analyzing the architecture of the neural network used in this part of the project, let us take a look at the environment that was chosen for the training and testing phase. As was already pointed out in the introduction of this chapter, we decided to consider only road intersection scenarios. In particular, we chose intersections that have only traffic signs and not traffic lights. Indeed, there are already many works in the literature showing that traffic management at an intersection that presents traffic lights is a task that has already been extensively studied and with already excellent results ([90], [91]). Therefore, in this paper we decided to leave out this relatively simpler case to focus on a more generic and more complex case.

For the training phase, we used the 3 scenarios in Fig. 4.2, which have different levels of difficulty. The simplest one, is the one depicted in Fig. 4.2a where there are only three possible directions that can be traveled; the second, the medium difficulty one (Fig. 4.2b has four possible directions that can be traveled; the third, the most complex one (Fig. 4.2c), is like the previous one, but with two entry and exit roadways for each direction. Despite the presence of multiple lanes in the last type of intersection, for simplicity, we do not consider the lane change maneuver. If a vehicle in training attempts the lane change maneuver, even though it might be a legitimate maneuver in reality, in this case it results as if it has exited its path and thus receives a negative reward. The three scenarios in Fig. 4.2 correspond to a synthetic representation of real intersections and are derived from the first simulator described in



(a) Easy Scenario (b) Medium Scenario (c) Hard Scenario

Figure 4.2: Synthetic representations of three intersections used for training agents to execute the crossing maneuver. Each of these scenarios has a higher level of difficulty. Within them, the green rectangles correspond to the ego agent and the white rectangles to the traffic agents. The red lines correspond to stop signs and the yellow lines correspond to yield signs. Each agent can only see a $50 \times 50$ meters area that is represented by the green square and corresponds to the ego agent's view.

the section 3.3. During the training phase, only one agent can be born per lane during each episode. Thus, the maximum number of agents involved in the intersection is equal to the number of lanes in the selected scenario, 3 in the easiest, 4 in the medium-level, and 8 in the difficult one.

In order for the system to learn to generalize as much as possible, each of the agents is assigned a new path and traffic sign each episode, which then are not fixed in the scenario, but also change with each episode. In this way, every agent explores as many state configurations as possible during training.

However, in the scenario in Fig. 4.2c the two side-by-side lanes in each direction have the same traffic signs, and this allows us not to include inconsistencies in training and not to have to change the configurations of some traffic signs during the episode.

During an episode, each agent is not aware of the complete scenario, but perceives only an area of $50 \times 50$ meters. From this the system then derives four semantic images (Fig. 4.3) that correspond to:

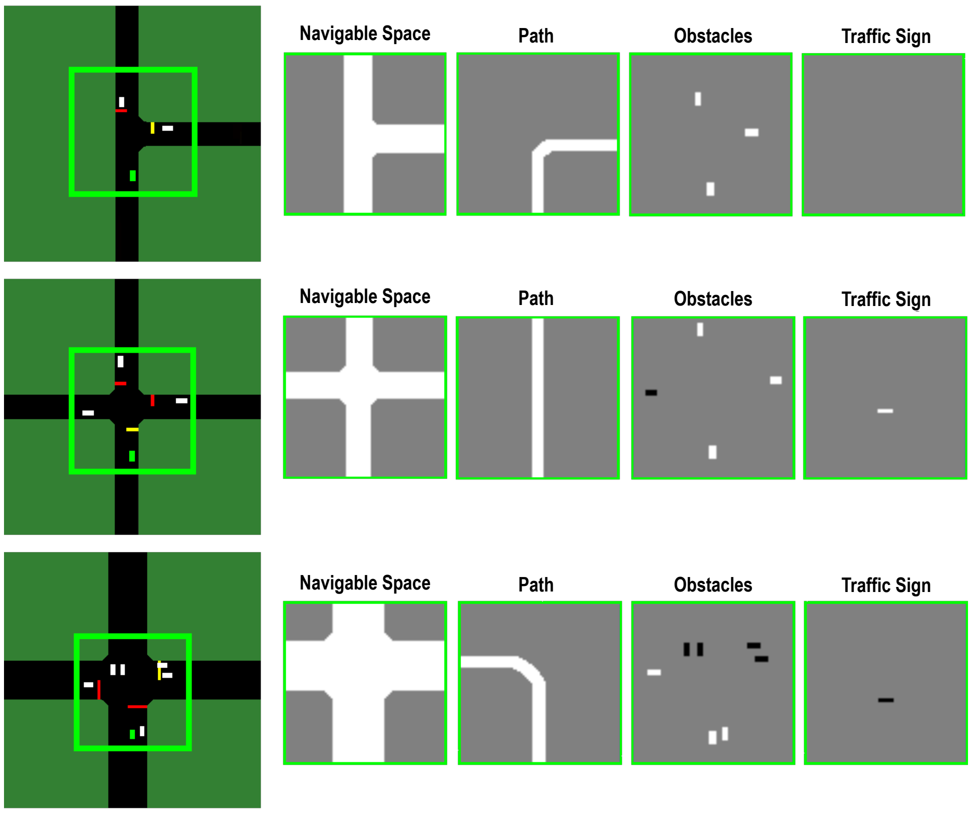


Figure 4.3: Channels derived by agents from their view in the three different intersections: Navigable Space, Path, Obstacles, Traffic Sign. The four channels correspond only to the content in the green rectangle, which is the $50 \times 50$ meters area perceived by the ego agent. The channels are grayscaled to allow the agent to distinguish stop signs (black) and yield signs (white) in the Traffic Signs channel and also, to distinguish agents to be given precedence (black) from those that need not be given precedence (white) in the Obstacles channel.

- *Navigable Space*: which represents the whole space in which the agents can move.
- *Path*: which the agent must travel through during the episode and which is randomly determined at the beginning of the episode.
- *Obstacles*: which provides information about both the ego agent's location and that of any other vehicles in its surrounding view.
- *Traffic Sign*: which identifies how the agent should approach the intersection, if nothing is present the agent can go because it has the right of way, otherwise it may have a stop line or yield line for which it must negotiate with the other traffic agents present.

Each of these four images corresponds to an Input channel of the neural network and has a size of $84 \times 84$ pixels. Given that the system leads to control actions, having images with such a small size is limiting. A vehicle should be able to see an area large enough to move safely. It follows that this size is sufficient for the scenarios under consideration and for contained agent speeds, but with larger intersections or higher speeds it would be necessary to enlarge the size of these bird's-eye views. However, it must be kept in mind that enlarging the input inevitably involves longer execution times and a change in the neural network's architecture. The work done below and described in chapter 6 aims in this direction. As shown in Fig. 4.3, we opted to use grayscale rather than black-and-white images for the four channels. This choice allowed us to compress more information in the obstacle bird's-eye view and the traffic sign one. Indeed, in order to train the agents to behave differently according to the traffic sign they encounter in their path before the intersection, these can take different colors:

- *No sign*: the ego agent is traveling the main road, so it has precedence over all other traffic agents except those who like it have no signs on their roadway and are coming from the right side.
- *Yield sign*: in the traffic sign channel it is represented with the color white and correspondingly with a yellow segment in the simulator.

- *Stop sign*: in the traffic sign channel it is represented with the color black and correspondingly with a red segment in the simulator.

The simulator represents a bird's-eye view that takes into account the whole scenario and the agents involved. In Fig. 4.3, the four channels do not correspond to the whole scene, but only to the area depicted in the green box that highlights the view of the agent, the green one, at that particular instant. Since the simulator is equivalent to the complete scenario, it contains all the stoplines in each direction. In contrast, in the corresponding traffic sign channel of the agent there is only that of the agent under consideration. To give the agent information about which traffic signs other agents will encounter we use the obstacle channel. This is because if we used the traffic sign one, the system might misinterpret them and stop even if there are actually no obstacles in the other lane. The obstacle channel then gives us both the position of the other agents in the agent's surrounding view and their priority level relative to the ego agent. So, the ego agent can perceive agents in two different ways in the obstacle channel:

- *White agents*: which correspond to those over whom it has priority and therefore do not influence its actions (also includes the ego agent himself).
- *Black agents*: which are those to whom it has to give precedence, which have a higher priority than itself and therefore influence its behavior.

In the training phase this prior knowledge is embedded in the obstacle channel because it is based on traffic signs and is necessary for learning. In addition, in testing on the real vehicle, this prior knowledge contained in the traffic sign and obstacle channels can be easily recreated using perception algorithms and high-definition maps. An example of how this type of system can be developed in a real autonomous vehicle is shown in [65]. As a final note it is important to point out that there is a control to avoid unpleasant situations of traffic deadlock, i.e., situation where there are for example four vehicles coming from four different directions and they all have the same traffic sign. In this situation all agents would have the same traffic sign and a vehicle to their right and, therefore, no one would move.

## 4.3 Training Considerations

In this work, the goal is to make sure that each agent is able to cross the intersection safely by using acceleration and steering angle values obtained with a neural network every 100 milliseconds as input for the control. For this reason, we have chosen the kinematic bicycle model for the vehicle dynamic. This is the standard model that is used for capturing vehicle motion under normal driving conditions. In practice, the bicycle model considers the front wheels of a vehicle as one and equivalently for the rear wheels as well. This assumption is possible because effectively the wheels are connected on the same axle and therefore perform the same motions. Thus instead of having to handle 4 wheels and 2 steering angles, we only need to consider 2 wheels and 1 steering angle. In addition to this assumption, the bicycle model also assumes that the vehicle is moving in a 2D plane and that there is no lateral or longitudinal slippage of the vehicle. Clearly, these are assumptions that must be taken into account when testing the system on a real vehicle. The goal of this type of model is to generate a set of equations that can fully describe the model at any point in time, i.e., calculate the $x$,$y$ pose and $\theta$ heading. The input of the model are vehicle speed and steering angle. Since acceleration is the derivative of velocity and the output of our neural network is just acceleration and steering angle, we can use it as input to the bicycle model. The steering angle values that are assigned to the agents are always between $[-0.2, +0.2]$ and the acceleration values between $[-3.0\frac{m}{s^2}, +3.0\frac{m}{s^2}]$. To make the system more realistic during training, we generate the vehicles with an initial speed between $[3.0\frac{m}{s}, 6.0\frac{m}{s}$ and impose to them a target speed between $[7.0\frac{m}{s}, 10.0\frac{m}{s}]$, which is the speed we would like them to reach, but without exceeding it, as if it were a speed limit. To increase the generalization and the number of cases seen in training, we introduced a delay for spawning new vehicles, which varies between 0.0 and a scenario dependent maximum value:

- *Easy scenario*: $[0.0, 30.0]$ seconds
- *Medium scenario*: $[0.0, 50.0]$ seconds
- *Hard scenario*: $[0.0, 100.0]$ seconds

This ensures greater mutability in the density of traffic that is about to cross the intersection. Thus, in summary, each agent spawns into the scenario with a given delay and initial speed and then negotiates with the other agents to decide what actions to take until the episode is concluded. Each agent can end the episode in one of the following four states:

- *Goal Reach*: that is, when the agent has crossed the intersection without crashing, following its path and respecting traffic signs.
- *Crash*: the officer collided with another vehicle.
- *Out of Path*: the agent did not respect the path set for him/her at the beginning of the episode.
- *Time over*: exceeded the time limit to reach the objective. A time limit has been built into the system in which the agent must be able to cross the intersection and finish the episode correctly.

In the section 4.5 we will see how the final state affects the Reward of our algorithm.

## 4.4 Algorithm and Neural Network Architecture

The system is trained using Delayed-A3C (D-A3C) which was explained in the section 2.3.9 In this work, we trained a Multi-Agent version of D-A3C in which all agents in the scenario simultaneously contribute to parameter optimization.

The architecture of the neural network is as shown in Fig. 4.4 and consists of two sub-modules, the red one to handle acceleration (*acc*) and the blue one for steering angle (*sa*). The neural network input consists of both a set of visual channels and scalar parameters. Indeed, both sub-modules simultaneously receive the same 16 input images that correspond to the sequence of the last 4 frames of the 4 channels seen by the agent (navigable space, path, obstacles, traffic sign). Each of these images has a size of $84 \times 84$ pixels. In addition to these visual inputs, each sub-module receives the same scalar parameters as input at each step, representing the agent's current speed

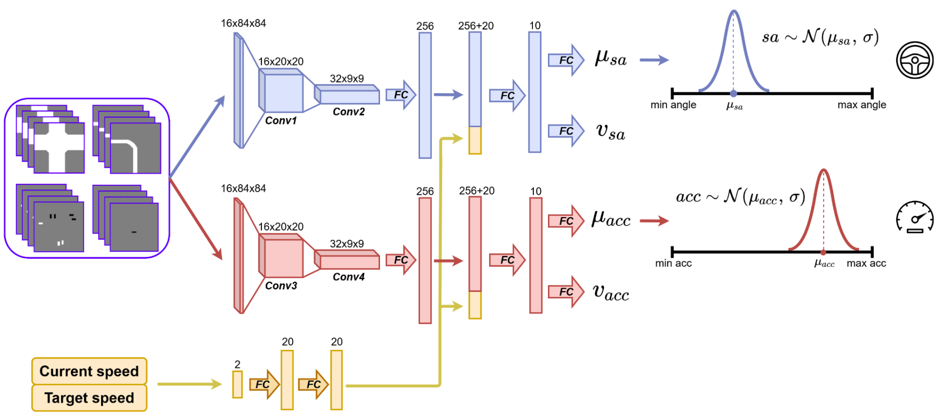


Figure 4.4: Neural network architecture used for the intersection handling task. The network consists of two sub-modules. The blue one to derive the steering angle and the red one for acceleration. Each receives the same 4 visual inputs, each consisting of the last 4 frames seen by the agent, and two scalar parameters: current speed and target speed. The neural network produces two different outputs corresponding to the means ($\mu_{sa}$,$\mu_{acc}$) of two different Gaussian distribution from which the acceleration and the steering angle performed by the agent are sampled using a standard deviation $\sigma$ that decreases linearly from 0.65 to 0.05 during the training phase.

and the target speed to which it should aspire. In order to ensure exploration, the actions performed by the agent are sampled by two Gaussian distribution centered on the output of the two sub-modules ($\mu_{sa}$,$\mu_{acc}$). So the final outputs of acceleration and steering angle that are passed to the implementation component are: $sa \sim \mathcal{N}(\mu_{sa}, \sigma)$ and $acc \sim \mathcal{N}(\mu_{acc}, \sigma)$, where $\sigma$ is a tunable parameter and it decreases linearly from 0.65 to 0.05 during the training phase.

Simultaneously with the estimation of $\mu_{sa}$ and $\mu_{acc}$, the Critic part of the algorithm produces the corresponding state-value estimations ($v_{sa}$, $v_{acc}$) using two different reward functions: $R_{acc,t}$ and $R_{sa,t}$. The state-value functions can be written as: $v_{sa}(s_t;\theta^{v_{sa}}) = \mathbb{E}(R_{sa,t}|s_t)$ , $v_{acc}(s_t;\theta^{v_{acc}}) = \mathbb{E}(R_{acc,t}|s_t)$. In this case, the policy update described in

Equation 2.24 can be defined as follows:

$$\begin{aligned}\theta_{t+1}^{\mu_{sa}} &= \theta_t^{\mu_{sa}} + \alpha A_{sa,t}\frac{\nabla\pi(a_t|s_t,\theta_t^{\mu_{sa}})}{\pi(a_t|s_t,\theta_t^{\mu_{sa}})}\\ &= \theta_t^{\mu_{sa}} + \alpha A_{sa,t}\frac{\nabla\mathcal{N}(sa,\mu_{sa}(\theta_t^{\mu_{sa}}))}{\mathcal{N}(sa,\mu_{sa}(\theta_t^{\mu_{sa}}))}\\ &= \theta_t^{\mu_{sa}} + \alpha A_{sa,t}\frac{sa-\mu_{sa}}{\sigma^2}\nabla\mu_{sa}(\theta_t^{\mu_{sa}})\end{aligned} \tag{4.1}$$

and the Advantage $A_{sa,t}$:

$$A_{sa,t} = R_{sa,t} + v_{sa}(s_{t+1};\theta^{v_{sa}}) - v_{sa}(s_t;\theta^{v_{sa}}) \tag{4.2}$$

The same equations can be written for the acceleration output, replacing $\mu_{sa}$ with $\mu_{acc}$ and *sa* with *acc*.

## 4.5 Reward Shaping

As we mentioned in the previous section, we decided to define two different reward functions, one for the steering angle $R_{sa}$ and one for the acceleration $R_{acc}$. In this way it is possible to evaluate acceleration and steering angle separately. Since we have decided not to allow lane changing, a possible departure of the vehicle from the road depends exclusively on the steering angle. By the same principle, we decided that an accident with another agent depends entirely on an error in acceleration. Each of the two rewards is defined by two components:

$$R_{sa,t} = r_{localization} + r_{terminal} \tag{4.3}$$

$$R_{acc,t} = r_{speed} + r_{terminal} \tag{4.4}$$

The first component $r_{localization}$ of $R_{sa}$ is a penalization given to the agent when its position ($x$,$y$) and its heading ($h_a$) differs from the center lane and the heading ($h_p$) of the path respectively. The greater the difference between the two headings, the greater the penalty the agent will receive.
This factor can be defined as:

$$r_{localization} = \phi\cos(h_a - h_p) + \psi d \tag{4.5}$$

where $\phi$ and $\psi$ are constants to which we assigned the fixed value 0.05 and $d$ is the distance between the position of the agent and the center of the lane. The first component of $R_{acc,t}$, on the other hand, is $r_{speed}$ which incentivizes the agent to reach a certain $target\ speed$. The closer the agent is able to approach this speed, the greater the value of this component and thus of the overall reward.
It is defined as:

$$r_{speed} = \varepsilon \frac{current\ speed}{target\ speed} \tag{4.6}$$

where $\varepsilon$ is a constant set to 0.005.

Both rewards then are affected by the value of the terminal state the agent is in at the end of the episode:

- *Goal Reach*: It means that the agent was able to achieve the goal on time and by traveling exactly the path it was assigned. Therefore, $r_{terminal}$ will have value $+1.0$ both for $R_{sa,t}$ and $R_{acc,t}$.

- *Crash*: Since the system is multi-agent we are in this situation if two agents have collided. Both are on their path at the time of impact, otherwise we would fall into the *Out of Path* case, so the penalty occurs only for $R_{acc}$. The agent who had precedence and therefore saw the other agent as white in the obstacle channel will have an $r_{terminal}$ equal to $-0.5$. The one who is more at fault in the accident and saw the other agent as black in the obstacle channel will receive an $r_{terminal}$ equal to $-1.0$.

- *Out of Path*: The episode ends when the agent goes off its path and we assume that it is due only to an inaccurate estimation of the steering angle output. For this reason $r_{terminal}$ will be 0.0 for $R_{acc,t}$ and $-1.0$ for $R_{sa,t}$.

- *Time over*: To prevent the agent from standing still indefinitely, we set a time limit in which the agent must be able to reach the goal. Since this happens if the vehicle is moving at too low speed, we only penalize acceleration. So we will have that $r_{terminal}$ takes the value 0.0 for $R_{sa}$ and $-1.0$ for $R_{acc}$.

## 4.6 Experiments

In this part of the project, we focused on teaching an agent to deal with intersections with different traffic conditions and traffic sign configuration. It proves difficult, however, to be able to compare our method with others. Indeed, there are few works that address and attempt to solve this issue with some method similar to ours. In [86] and [89] for example, they train an agent to find the appropriate time to cross the intersection without crashing. However, there are many differences between our system and theirs, and it is difficult to make a comparison. The agent in our case learns to negotiate and learns rules of behavior (traffic signs and priority to the right rule), in their work, on the other hand, the agent always has the lowest priority in the intersection and only has to learn how to cross it, without caring about other traffic agents and without having to negotiate with them. Furthermore, in papers [86] and [89] the type of training is single-agent, instead in our case it is multi-agent and tends to look much more like human behavior.
Since there are all these differences and even they themselves point out that the system still did not lead to consistently optimal results, we opted to make the comparison with the Time to Collision (TTC) algorithm. In addition to this comparison, we also focused on other types of tests, each aimed at demonstrating the effectiveness of our system. Initially, we analyzed its behavior, assessing whether it was able to move correctly along the path and safely. Then we demonstrated that the agent had learned to negotiate and comply with the right of way rule. Finally, we explored its generalization capabilities both on scenarios different from those of training and, also, with real traffic, i.e., where the behavior of the other agents involved is of human drivers.

### 4.6.1 System Testing

The first test in chronological order was the one in which we evaluate whether effectively the agents learned to drive safely and along the route assigned to them at the beginning of the episode. In this test we mainly evaluate the agent's percentage of *Reach*, *Crash*, *Out of Path* and *Time Over* and its average speed in finishing the episode. To make the experiment more varied and more consistent, we tested the agent on all 3 scenarios in Fig. 4.2 and, every episode, we modified the traffic signs to evaluate different behaviors. For each scenario we ran 3000 episodes analyzing the agent's behavior with different traffic conditions. During the training phase the spawn ranges for agents was fixed for each scenario $[0, 30]$ seconds for the easy one (Fig. 4.2a), $[0, 50]$ seconds for the medium one (Fig. 4.2b) and $[0, 100]$ seconds for the hard one (Fig. 4.2c). Instead, during this test we extracted the spawn delay value between a variable range: $[delay_{min}, delay_{max}]$. The lower the delay value, the more heavier the traffic turns out to be and it becomes more complicated for the agent to cross it. We initially started on all scenarios with the delay range $[0, 10]$ seconds and then increased the $delay_{max}$ by 10 with each successive test phase until we reached the maximum value used in training. We then performed 3 test phases for the easy scenario, 5 for the medium scenario, and 10 for the hard scenario.

Table 4.1 shows the percentages obtained through this experiment and divided according to the traffic sign the agent faced.
It can be seen that the agent finds it slightly more complex to cross the intersection in the situation where the traffic sign in the agent's lane is the stop sign. It is also interesting to see how the system was able to modulate its speed according to the difficulty of the scenario and also according to the traffic sign it encounters. From the percentages in the table, it is then also possible to see that the system never went out of path and was practically always able to end the episode in a reasonable time. Such a result leads us to state that the agent has learned to move correctly along its assigned path, safely and without unnecessary delays. The following video[1] shows the behavior of the agents in the three training scenarios with different traffic signs.

### 4.6.2 Comparison with TTC method

The next test was the comparison with the TTC algorithm, where we also wanted to demonstrate the efficiency of our system from the point of view of the time with which it is able to cross the intersection and conclude the episode correctly. For this test we had to create an ad hoc situation for the TTC; that is, where the task of the

[1]`https://www.youtube.com/watch?v=x28qRJXiQfo&ab_channel=AlessandroPaoloCapasso`

| | **Easy Scenario** | | |
|---|---|---|---|
| | No Sign | Yield Sign | Stop Sign |
| Goal Reach (%) | 0.998 | 0.995 | 0.992 |
| Crash (%) | 0.002 | 0.005 | 0.008 |
| Out of Path (%) | 0.0 | 0.0 | 0.0 |
| Time over (%) | 0.0 | 0.0 | 0.0 |
| Average Speed ($\frac{m}{s}$) | 8.533 | 8.280 | 8.105 |
| | **Medium Scenario** | | |
| | No Sign | Yield Sign | Stop Sign |
| Goal Reach (%) | 0.995 | 0.991 | 0.992 |
| Crash (%) | 0.005 | 0.009 | 0.008 |
| Out of Path (%) | 0.0 | 0.0 | 0.0 |
| Time over (%) | 0.0 | 0.0 | 0.0 |
| Average Speed ($\frac{m}{s}$) | 8.394 | 7.939 | 7.446 |
| | **Hard Scenario** | | |
| | No Sign | Yield Sign | Stop Sign |
| Goal Reach (%) | 0.997 | 0.991 | 0.972 |
| Crash (%) | 0.003 | 0.009 | 0.012 |
| Out of Path (%) | 0.0 | 0.0 | 0.0 |
| Time over (%) | 0.0 | 0.0 | 0.016 |
| Average Speed ($\frac{m}{s}$) | 8.224 | 7.365 | 5.855 |

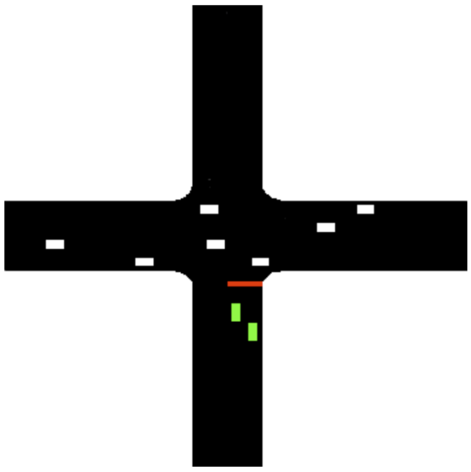
Table 4.1: Results obtained in the training scenarios (Fig. 4.2) with the three different traffic signs.

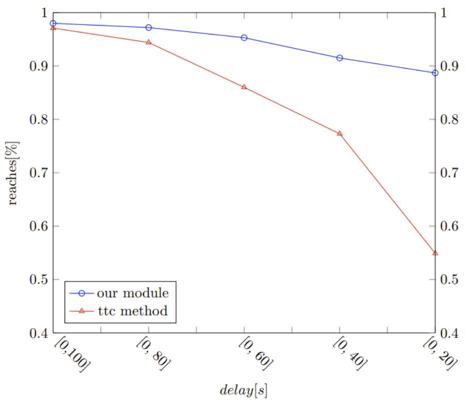
TTC is exclusively to decide the instant of time when the agent should cross the intersection. For this reason, we chose to compare the two methods on the hard scenario (Fig. 4.2c) and for both the traffic sign the agent encounters is the stop sign. The traffic agents involved remain in the center lane and move with a speed defined by the Intelligent Driver Model (IDM); moreover, their spawn points are only in the left and right branches with respect to the one where the ego agent spawns. The agent's behavior when applying the TTC algorithm is to decelerate to the stopline, then wait to cross the intersection until all calculated collision times with other vehicles exceed a certain threshold. If this happens, the vehicle moves with an acceleration of $3.0\frac{m}{s^2}$. For the estimation of collision times with other agents, it is assumed that there is a possible collision point in the intersection and that the velocities of the agents involved are constant. The threshold was set to obtain the best possible results with TTC and 0% accidents. A time limit was set for both algorithms within which they must necessarily conclude the episode to avoid failure. The tests were performed considering 3 traffic levels: low, medium and high, which correspond respectively to having a maximum of 4,8 or 12 agents simultaneously active in the scenario. The agents' spawn delay is also variable in this test, initially being in the range $[0,20]$ seconds then increases the $delay_{max}$ by 20 for every successive new stage. Both algorithms work well with medium and low traffic level, indeed, with each of the two a percentage of successfully concluded episodes above 99% was obtained. The difference is noticeable with a high level of traffic, as can be seen from the Fig. 4.5, the success rate in the case of our RL-based algorithm decreases, but still remains very high. For the TTC, on the other hand, the percentage decreases quite a bit because a good portion of the episodes end with the agent stopped at the stop sign and fails to find an opportunity to enter and exceeds the time limit set to end the episode. The difference between the two methods becomes more and more evident as the traffic level increases, that is, when the maximum agent spawn delay is less.

### 4.6.3 Test the Right of Way Rule

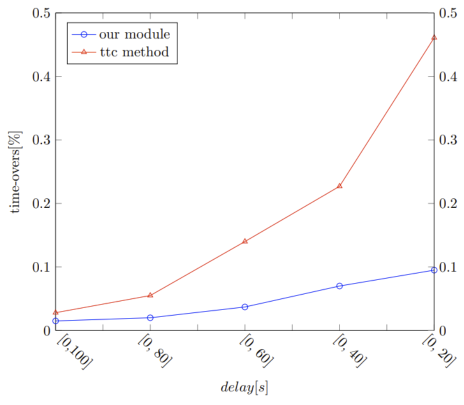
So far, we have shown that the agent is able to deal with the intersection correctly, but we have not yet evaluated its negotiation skills and whether it has actually learned the right of way rule.

This third test focuses precisely on analyzing the vehicle's behavior when faced with choices due to the presence of other obstacles approaching the intersection. To make sure that there are always other vehicles arriving at the intersection at the same time, we set the initial and target speeds the same for all agents involved. The test was performed on all three scenarios in Fig. 4.2, respectively with 3,4 and 8 vehicles


(a) Test scenario



(b) Reaches



(c) Time-overs

Figure 4.5: 4.5a represent the test scenario used to compare the performances of our module with those obtained with the time-to-collision method. The tested agents are represented by the green rectangles, while the white rectangles correspond to the traffic agents moving using the Intelligent Driver Model (IDM) algorithm to define their speed. 4.5b and 4.5c show the comparison between the percentage of successfully completed episodes and time-overs respectively. In the graphs, the blue curves correspond to the results of our module and the red curves correspond to the results of the time-to-collision method.

simultaneously active (one per lane) as in training. With this configuration we are able to assess whether precedence was respected by evaluating the terminal state of each agent at the end of the episode. For each type of scenario, 9000 episodes were performed. In the Table 4.2 we can see the percentage of *No Infraction* and *Infraction* that were obtained. An episode was considered to have *No Infraction* if all agents complied with the precedence rules, but if at least one vehicle did not comply it was evaluated as *Infraction*. The results show that indeed our agent is able to negotiate and respect precedence and the right of way rule in most cases. The success rate drops a bit in the hard scenario, but this is mostly due to the fact that this scenario imposes wider spaces that allow agents to risk the maneuver more.

| | **Easy Scenario** | **Medium Scenario** | **Hard Scenario** |
|---|---|---|---|
| No Infraction (%) | 0.985 | 0.990 | 0.863 |
| Infraction (%) | 0.015 | 0.010 | 0.137 |

Table 4.2: Percentages of episodes ended following the right of way rule by all the agents involved in the episode (*No infraction*) and with at least an infraction (*Infraction*).

### 4.6.4 Test on Real Data

Since the overall goal of the project is to obtain a system that works on a real vehicle, as the last test of this first work we analyzed the behavior of the trained agent in a real traffic situation. Specifically, we used 33 sequences of inD Dataset (Intersection Drone Dataset) [79] divided over 4 scenarios, shown in Fig. 4.6 for a total of 10 hours of recorded data. The dataset includes pedestrians, bicyclists, cars, trucks and buses that passed through the intersection during the recordings of a drone equipped with cameras. A total of 11500 road users were tracked. Because only vehicles moving along the roadway are present in our training dataset, we removed pedestrians, bicyclists, and vehicles that remain stationary. In this test, our agent, represented in green in Fig. 4.6, always enters the intersection from the same lane and then can randomly exit from one of the other lanes. Because the traffic is real and pre-recorded it is

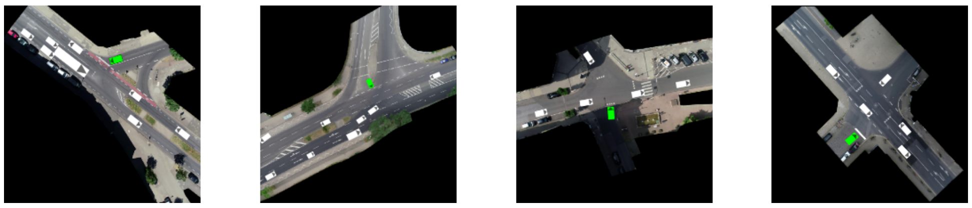


Figure 4.6: Four real intersections contained in the inD dataset [79] used for testing our agents (green rectangles) using the recorded traffic data (white rectangles).

not aware of our agent's presence. Therefore, to avoid unintentional collisions, traffic agents that would spawn in the same lane as the agent were removed and the ego agent was assigned the lowest priority in the scenario.
Having made all these considerations, the dataset was reduced from 11500 to 7386 users populating the traffic, while the executed episodes are 2702. In order to visualize our agent behavior, we reproduced the 4 intersections with CAIRO graphic library [78], based on the trajectories of the dynamic agents. For each episode, we saved data related to the RL agent (position, speed and heading) in order to project them on the real scenarios (Fig. 4.6) using the code provided by [79] on their github page[2]. Doing an analysis we found that the percentage of correctly concluded episodes is over 99%, with 0% time-over and 0% off-road cases. In this video[3] we show the agent's behavior on the scenarios. Watching the video carefully, it can be seen that in some cases the vehicle entry seems a bit risky, especially when a vehicle comes over the intersection after it has already entered. This fact is justified, our agent has been trained to negotiate with other vehicles and in this case it cannot do so because the other agents do not perceive its presence. In a similar situation that occurs during training, the vehicle behind our agent would slow down, making the entry seem much less dangerous. Having made these considerations, we can say that the results obtained are promising, especially considering that the agent is not trained on these scenarios and is not used to seeing the behavior of real traffic. However, by doing some tests

[2]`https://github.com/ika-rwth-aachen/drone-dataset-tools`

[3]`https://www.youtube.com/watch?v=SnKUk2k9YCg&ab_channel=AlessandroPaoloCapasso`

directly on a real vehicle we found that the driving style is not feasible and comfortable, despite being able to cross the intersection correctly. So before proceeding with further testing in the real world, we focused on solving this issue, as we will see in the following chapters.

# Chapter 5

# Deep Reinforcement Learning Planner

In the previous chapter, we demonstrated how it is possible, at least in simulation, to use Deep Reinforcement Learning to enable an agent to manage and navigate an intersection by negotiating with other vehicles and observing precedence. In particular, we demonstrated how the vehicle learned to follow the path and to observe the priority to the right rule. From the very first moment we started testing this model in the real world, we realized that the quality of driving was not comfortable, although in simulation we were able to achieve good behavior. So we decided to take a step back and develop a model-free Deep Reinforcement Learning Planner that was capable of drive comfortably and safe in a real urban environment.
This chapter describes the neural network that was implemented and all the strategies adopted to achieve better driving comparable to that of a human driver. We trained the model predicting continuous actions related to the acceleration and steering angle, and testing it on board of a real self-driving car on an entire urban area of the city of Parma (Fig. 5.1).
With this work, we developed a system capable of generalizing even over those areas of the map not used in the training phase. In addition, we were also able, with a tiny neural network, to reproduce the dynamics of the real autonomous vehicle in simula-

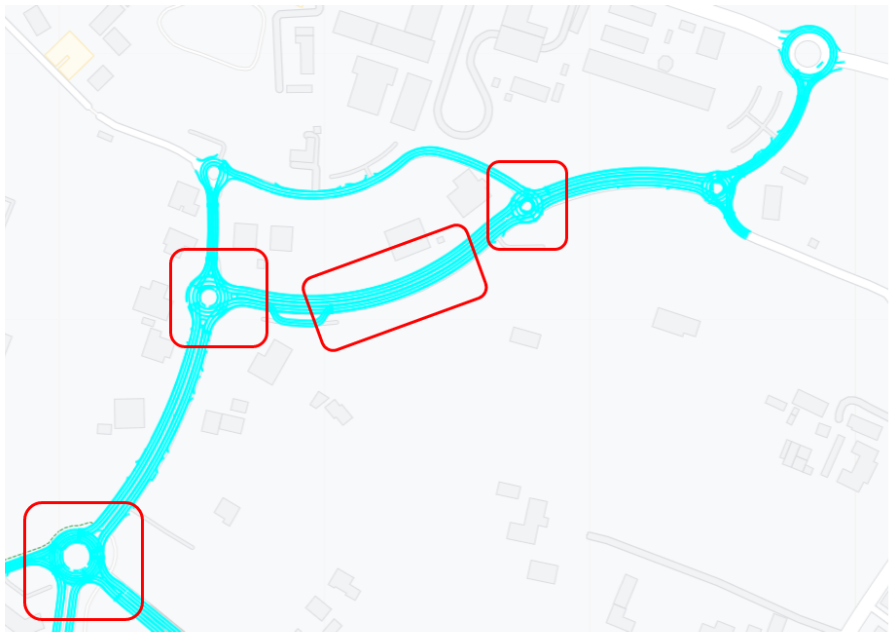


Figure 5.1: Mapped area of a neighborhood of Parma. The four red rectangles contain the scenarios used for training agents. The whole light blue area was used for testing.

tion, thus bridging the gap between simulation and reality to a large extent.
Finally, we also demonstrate how pre-training by Imitation Learning enabled us to drastically reduce overall training time.

## 5.1 Environment Definition

For this part of the project and subsequent tests, we used a new simulator based on HD maps as explained in the chapter 3.3. In Fig. 5.1, the light blue area represents one of the neighborhood areas in Parma where autonomous driving tests are allowed. The area is in a new neighborhood with little traffic and lends itself well to this kind of testing. From this map it is then possible to derive an arbitrary amount of scenarios useful in the training phase. Specifically in our case we used the scenarios contained in the red rectangles. An example scenario is illustrated in Fig. 5.2.

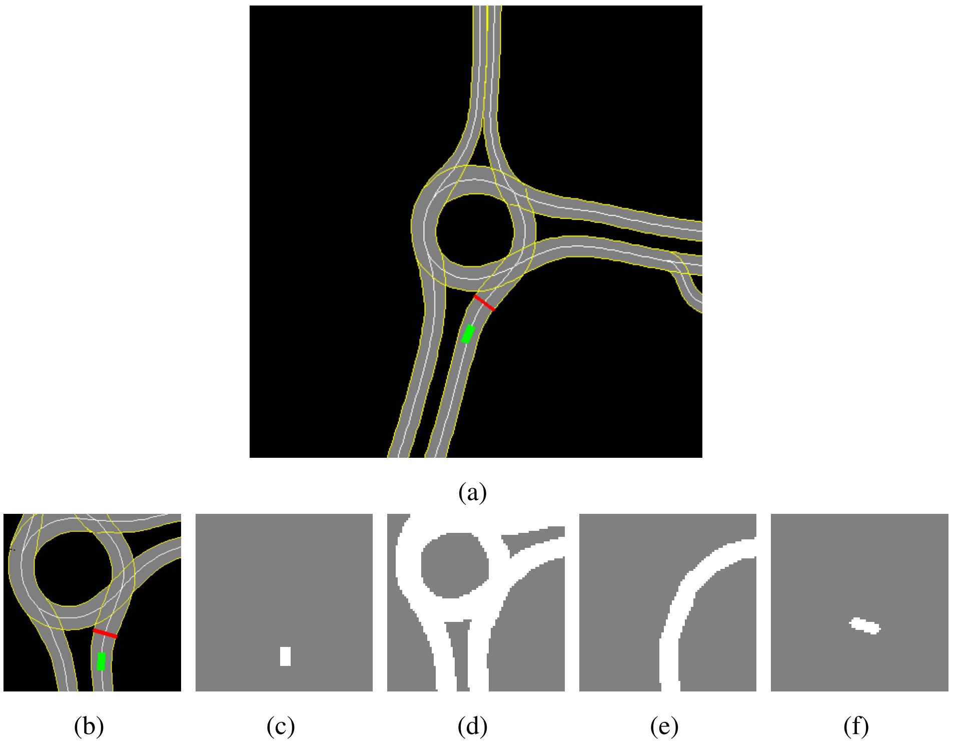


Figure 5.2: Synthetic representation of training scenario (Fig. 5.2a) included in the mapped area of Fig. 5.1. Fig. 5.2b represents the $50 \times 50$ meters surrounding perceived by the agent, that is split in four channels: obstacles (Fig. 5.2c), navigable space (Fig. 5.2d), path (Fig. 5.2e) and stop line (Fig. 5.2f).

The first Fig. 5.2a shows an entire scenario representing a roundabout. The green rectangle corresponds to the agent and the red line the stopline that lies in its path. However, as we have also seen in the previous chapter 4.2, an agent can only see a limited area around him, which for this part of the work we set at $50 \times 50$ meters. More specifically, he sees 25.0 $m$ left and right, 40.0 $m$ ahead and 10.0 $m$ behind. Fig. 5.2b represents exactly this surrounding view of the agent, that is, the portion of the world it sees when it is in the position of Fig. 5.2a. To facilitate training, we decided not to use 5.2b directly as input for the neural network, but split it into 4 different grayscale channels, so that each of them contains specific information:

- *Obstacles*(Fig. 5.2c): in this stage of the project where we are analyzing the capability to drive comfortably, it contains only the ego agent, so as not to further complicate the task with the presence of other static or dynamic obstacles.
- *Navigable Space* (Fig. 5.2d): representing the entire area of the road on which the agent is allowed to move.
- *Path* (Fig. 5.2e): representing the path that the agent has to follow. It connects his spawn point with the end point of the episode and is randomly assigned to it at the beginning of the episode.
- *Stop Line* (Fig. 5.2f): which contains all the points at which the agent must stop or slow down to give precedence. In the dataset on which we trained the agent for this part of the project it is only present for entering roundabout.

Since the simulator is based on HD maps, during the training phase, there are also additional advantages, because from these maps can be derived other information about the surrounding environment: location, width and number of lanes, road speed limits, and much more. However, in this part of the project we only exploit the topological part of the maps to create the four channels and road speed limits to define the target speed of the vehicle, but in the future some other information could be used as a scalar parameter of the network.

## 5.2 Neural Network and Training Settings

Since this part of the project is a natural evolution of the one described in chapter 4, the neural network we implemented is also based on the previous one, but with some difference. The basic algorithm we use is still the same, namely D-A3C (explained in 3.3), and the new neural network is the one shown in Fig. 5.3. Since the main goal is to achieve smooth and safe driving style, the architecture is focused on obtaining as output actions that are as correct as possible and consistent with the previous ones, so that there is no abrupt braking or unpleasant sensation when people are on board

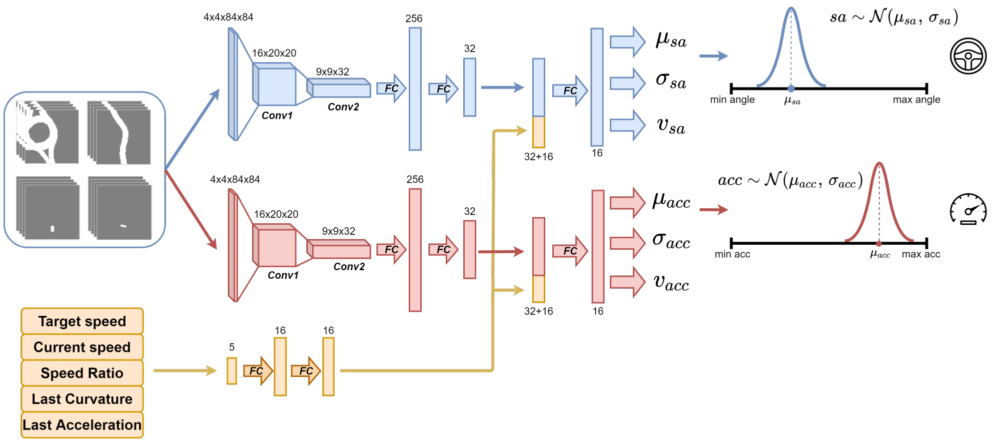


Figure 5.3: Neural network representing the evolution of the one seen in Fig. 4.4 . The main differences are the number of scalar inputs the two sub-modules receive and the fact that in this case standard deviations ($\sigma_{sa}$ and $\sigma_{acc}$) are also predicted.

of the vehicle.

The dynamic vehicle model we consider is the kinematic bicycle model, so again the output of the neural network are set points of acceleration and steering angle every 100 milliseconds. The architecture is then divided into two sub-modules: the first one able to define the steering angle *sa* (blue one) and the second one for the acceleration *acc* (red one). Because the system is trained using Reinforcement Learning it is necessary to ensure exploration for the system to be able to find the optimal policy, for this reason the acceleration and steering angle set points do not correspond directly to the model output but are sampled from two Gaussian distributions obtained through the neural network: $sa \sim \mathcal{N}(\mu_{sa}, \sigma)$ and $acc \sim \mathcal{N}(\mu_{acc}, \sigma)$. The two standard deviations $\sigma_{sa}$ and $\sigma_{acc}$ are also predicted and modulated by the neural network along with the two values $mu_{sa}$ and $mu_{acc}$. This choice leads us to have throughout training an estimate of the model's uncertainty on an instant-by-instant basis.

Furthermore, the network produces the corresponding state-value estimations ($v_{sa}$ and $v_{acc}$) using two different reward functions $R_{sa,t}$ and $R_{acc,t}$ related to the acceleration and steering angle respectively, exactly as is the case in the previous work 4.4.

Instead, on the input side, both sub-modules receive $84 \times 84$ pixels images that correspond to the four channels (Navigable Space, Path, Obstacles and Stop Line) described in the previous paragraph. To give the agent an idea of what its past history looks like, they actually receive 4 images for each channel, related to the last 4 time steps, for a total of 16 visual inputs. Together with this, the network receives the same 5 scalar parameters including the target speed (the road speed limit), the current speed of the agent, the ratio between the current speed and the target speed, and the last actions related to the steering angle and the acceleration. The parameters of the neural network model were trained only on the scenarios contained in the red rectangles in Fig 5.1. In each scenario we deliberately decided not to include any obstacles or other road users, because the goal of this part of the project was to evaluate the agent's learning capabilities in an environment that was as simple as possible. For this reason, since we did not have agents training at the same time, we decided to create multiple instances of each scenario, on which the agents act independently of each other. Every agent moves using values in the range $[-2.0\frac{m}{s}, +2.0\frac{m}{s}]$ for acceleration and $[-0.2, +0.2]$ for steering angle. In order to make the system more generalizable for future real-vehicle testing, we decided not to set an initial velocity that was always the same, but to assign each agent an initial random velocity chosen from the range $[0.0, 8.0]$ with which it spawns in each episode. The maximum initial speed was set at $8.0\frac{m}{s}$ because the road speed limits used in the urban area from which the training scenarios are taken vary between $4.0\frac{m}{s}$ and $8.3\frac{m}{s}$, and therefore we did not want to exceed this value. Once spawn, the agent will have to drive comfortably by following the route and obeying the speed limit until the terminal state of the episode is reached.

Finally, since there are no obstacles inside the training scenarios, episodes can finish in one of the following terminal states:

- *Off-road*: when the agent goes off the route it was assigned at the beginning of the episode due to a wrong decision on the action to be taken.

- *Time-over*: when the time limit to end the episode is reached. For the system to learn to drive properly, it must also be able to arrive at its destination within a reasonable time. The main cause for an agent to happen in this terminal state is to choose extremely cautious driving with very low acceleration values.

- *Goal reached*: when the agent is able to reach the terminal position of his route without going off the road and within a reasonable time.

## 5.3 Reward Shaping

The ability to stay in the center lane, to reach but not exceed the target speed, and not to brake or accelerate abruptly are all characteristics that must be considered when defining the reward components. Indeed, if an agent wants to obtain a policy that is able to drive smoothly both in simulation and in the real world environment, reward shaping is essential to achieve the desired behavior. The main problem we found in the experiments performed in the previous chapter 4.6, was related to the extreme variability with which the agent chose consecutive actions, which led to rapidly changes between positive and negative accelerations and thus to very abrupt braking and acceleration. To obtain better results from this point of view, we modified the rewards, one for acceleration and the other for steering angle, redefining them as follows:

$$R_{acc,t} = r_{speed} + r_{acc\_indecision} + r_{terminal} \tag{5.1}$$

$$R_{sa,t} = r_{localization} + r_{sa\_indecision} + r_{terminal} \tag{5.2}$$

Let us now analyze the components of both rewards individually, dwelling on their significance and how they affect the actions taken by the agent. The $r_{speed}$ component allows us to quantify how close the current speed is to the desired speed. So it depends on the value of the ratio between the vehicle's current speed and the target speed, and can be defined like this:

$$r_{speed} = \begin{cases} sr \cdot \zeta & \textbf{if } sr < 1.0, \\ (1.0 - sr) \cdot \zeta & \textbf{otherwise}, \end{cases} \tag{5.3}$$

where $\zeta$ is a constant set to 0.009.
This factor encourages the agent to reach but not exceed the target speed defined by

the route speed limit in that section of road.

$r_{localization}$ is an element that penalizes the reward of the steering angle when its heading $h_a$ and position $(x,y)$ differs from those of the HD maps. It is defined as:

$$r_{localization} = \phi \cdot (h_a - h_p) + \chi \cdot d \tag{5.4}$$

where $\phi$ and $\chi$ are constants set to 0.05, $h_p$ is the heading of the road, and finally $d$ is the lateral distance between the position of the agent and the center of the lane. In addition to these components that were already present in previous work, another component was added in both $R_{sa,t}$ and $R_{acc,t}$ with the task of penalizing in case two consecutive actions differ too much from each other. That is, if the modulus of their difference is less than a certain threshold. Specifically, the difference between two consecutive accelerations is calculated as $\Delta_{acc} = |acc(t) - acc(t-1)|$ and $r_{acc\_indecision}$ is defined as:

$$r_{acc\_indecision} = \psi \cdot \min(0.0, \delta_{acc} - \Delta_{acc}) \tag{5.5}$$

where $\psi$ is a constant set to 0.1 and $\delta_{acc}$ is set to 0.5 $\frac{m}{s^2}$.

Instead, the difference between two consecutive predictions of the steering angle is calculated as $\Delta_{sa} = |sa(t) - sa(t-1)|$, such that $r_{sa\_indecision}$ is defined as:

$$r_{sa\_indecision} = \lambda \cdot \min(0.0, \delta_{sa} - \Delta_{sa}) \tag{5.6}$$

where $\lambda$ is a constant set to 0.01 and $\delta_{sa}$ is set to 0.05. Finally, the last term of both rewards depends on the agent's terminal state, so it takes on value 0.0 at every step of the episode except the last one. Thus, the values it can take depend on the terminal states defined earlier:

- *Off-road*: The agent did not maintain the path assigned to it. Since this type of error is mainly due to wrong curvature we decided to penalize only the steering angle reward, so $r_{terminal}$ takes value $-1.0$ in $R_{sa,t}$ and 0.0 in $R_{acc,t}$.

- *Time-over*: The time to end the episode is over before the agent has reached the target position. This is mainly due to an overly cautious acceleration predictions of the agent; for this reason $r_{terminal}$ assumes the value of $-1.0$ for $R_{acc,t}$ and 0.0 for $R_{sa,t}$.

- *Goal reached*: the agent reached the goal position without leaving the roadway and in a reasonable time, so $r_{terminal}$ is set to $+1.0$ for both rewards.

## 5.4 Deep Model

Since in this project we also wanted to test the system on a real vehicle, we had to deal with all those problems that make a simulated system different from a real system. Primarily, the discrepancy between a real and simulated data, caused by the difficulty of reproducing a real situation in a simulator. To overcome this problem we used a synthetic simulator (Fig. 3.4) in order to simplify the input of the neural network and to reduce the gap between simulated and real world data. Indeed, the input we have chosen for our network is grayscale and consists of 4 channels each focused on a certain feature of the environment (Obstacles, Navigable Space, Path, Stop Line). In this way we are able to synthesize reality, eliminating all details that are irrelevant to the agent. The 4 images we pass as input to the neural network can be easily reproduced by perception and localization algorithms and by HD Maps embedded on the real self-driving car. An additional factor that must be taken into consideration when using a simulator is that each real system has its own characteristics that make it unique. For example, when considering two human beings, even if they belong to the same species, they are different from each other, reacting in different times and ways to stimuli. Just as with humans, for which the example is self-evident, the same is true for other systems. Taking generic systems into account in simulation can then result in differences with a specific system in reality. In our case, a vehicle has its own peculiarities that depend on engine capabilities, wheel size, aerodynamic characteristics, and countless other parameters. Some are easily configured as parameters in the simulation and it is simple to take them into account, such as the size of the vehicle, but for others it is more complex. Just think about reaction times, that is, how each simulated agent performs a target action compared to how the autonomous car would behave executing that command. Indeed, a target action to be performed by an agent at time instant $t$, in simulation has immediate effects at that precise moment, but this does not happen on a real vehicle. Every real vehicle executes this target action with a certain dynamics that involves a delay in execution $(t+\delta)$. For this reason, wanting to test a model trained in simulation also in reality, it is necessary to overcome this problem by introducing in a delay also in simulation.

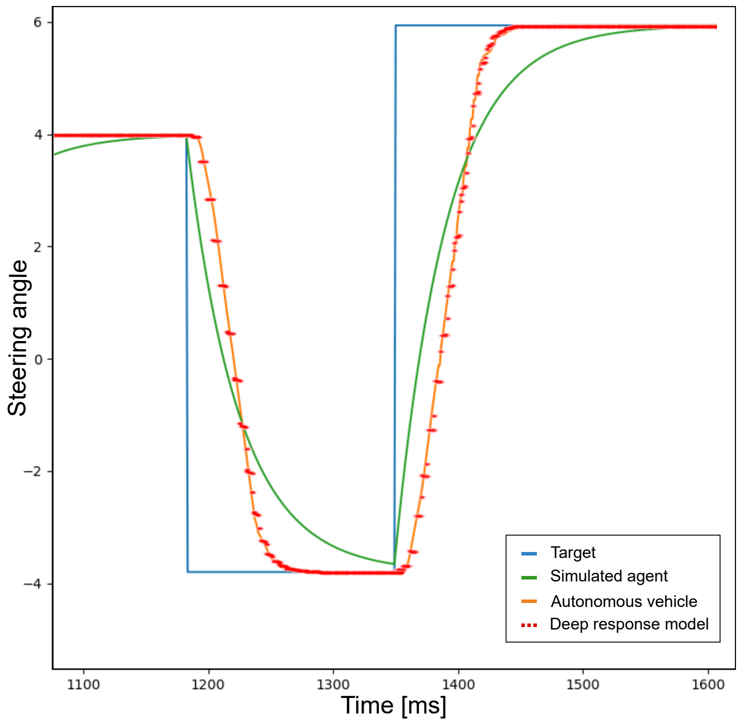


Figure 5.4: Comparison between simulated agents (green curve), real vehicle (orange curve) and *deep response* model (red dotted curve) behaviors executing target actions (blue curve).

Fig. 5.4 shows the different curves that represent the actual behavior of the models we are analyzing when there is a change in steering angle. The blue curve corresponds to the ideal, instantaneous response time that occurs in simulation. Whereas, the orange curve shows the behavior of the autonomous vehicle while performing the same steering action. The purpose of our research focused on making the behavior of the agents in simulation as close as possible to the real behavior (orange).
We initially trained agents by adding a low-pass filter to the target action predicted by

the neural network. In the Fig. 5.4, the green curve represents the response time of an agent in simulation with the introduction of the low-pass filter. However, the difference in the response time between simulated and real vehicle still remains relevant. The problem persists even with the introduction of the low-pass filter, the set points predicted by the neural network remain non-feasible commands and do not take into account some factors such as the inertia of the system, the delay of the actuators and other non-idealities. Therefore, the solution we found was the introduction of a model consisting in a small neural network composed by 3 fully connected layers (which we will call *deep response*). This model is trained using a dataset collected directly with the vehicle we will then test on. The input of this neural network corresponds to the commands given to the vehicle directly by the driver, i.e., accelerator pressure and steering wheel movement. The output, on the other hand, corresponds to the throttle, brake and curvature of the vehicle that can be measured with GPS, odometry or other techniques. Then applying this model to the actions generated with the other neural network yields in simulation a behavior like that represented by the red dotted curve in Fig 5.4. As can be seen, the curve overlaps almost perfectly with the orange curve that corresponds to that of the real vehicle. In this way we were able to report as truthfully as possible the dynamics of the autonomous car. Given the absence of obstacles and traffic vehicles in the training scenarios, the described problem was more evident for the actuation of the steering angle, but the same idea has been applied to the acceleration output.

## 5.5 Experiments

To validate our Deep Reinforcement Learning Planner, experiments were conducted both in simulation and in a real environment. Considering a modular architecture 1.3.2, our system aspires to replace that module that deals with planning and control. Once data is obtained from the sensors, it is processed and the 4 images that serve as input to the neural network are generated, exactly as in simulation. Initially, we conducted prolonged tests to evaluate the effectiveness and impact of the *deep response* model on the system. The experiment consists of comparing two policies, one in which it is included and the other in which it is not used. The next step was to verify that the vehicle actually followed the assigned path and that its speed approached the speed limit defined by the HD maps without exceeding it. In the last part of this work we focused on optimizing the system and showed that pre-training the neural network with Imitation Learning dramatically reduces the overall training time.

### 5.5.1 Test on Real Data

In this first experiment we tested the behavior of the real vehicle over the whole mapped area shown in Fig. 5.1 with two different policies:

- *Policy 1*: trained without the *deep response* model, but adding a delay in simulation with a low-pass filter.
- *Policy 2*: trained with the *deep response* model, applied to the output of the neural network in Fig. 5.5.

We initially performed the tests in simulation and then also on the real vehicle. In the simulated environment, the agent was able to drive in a way that was apparently comfortable and correct, succeeding 100% of the time with both policies. However, the difference was noticeable when we switched to testing on the real vehicle. *Policy 1* was unable to handle the dynamics of the vehicle, in which the execution of the predicted actions occurs with completely different timing than in the simulation. Indeed, the model, at each step predicted an action that should have brought it to a certain state, but actually brought it to a different state. This uncertainty had negative effects on the behavior, making the driving style uncomfortable on board of the self-driving car. In addition to this, the reliability of the system was also undermined because of this behavior, sometimes leading to human intervention to prevent the car from running off-road. In contrast, *Policy 2* is capable of predicting actions correctly because it is based on a model learned directly from the vehicle itself. The resulting behavior is comfortable and comparable to that of a human driver. Moreover, using *Policy 2* it has never been necessary to intervene to correct its driving except to avoid accidents with other vehicles. However, we do not consider these cases as failures because both

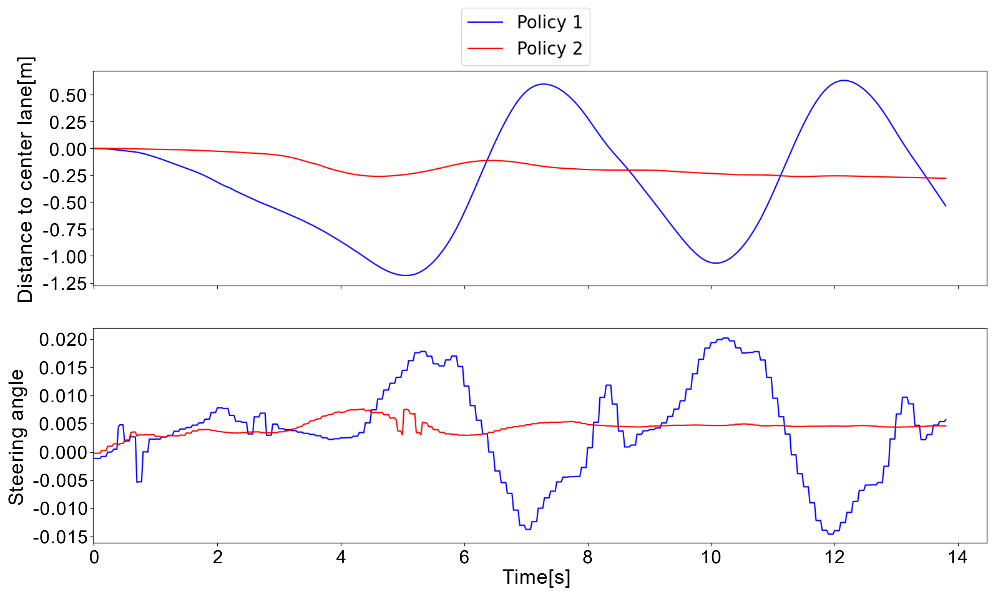


Figure 5.5: The distance to the center lane and the steering angle output predicted by *Policy 1* (blue curves) and *Policy 2* (red curves) on a short time window of the real world test on board the self-driving car.

policies were trained in obstacle-free scenarios. To better understand the difference between *Policy 1* and *Policy 2* we plot (Fig. 5.5) the steering angles predicted by the neural network and the distance to the center lane in a short time window of the real world tests. The behavior described earlier is perfectly reflected in this graph. The vehicle using *Policy 1* (blue curves) often deviates from center lane and its steering angle constantly oscillates, never finding a way to stay parallel to the center lane. The vehicle using *Policy 2* (red curves), on the other hand, has a much more stable and less noisy behavior as evidenced by the almost straight curves.
This test prove that the *deep response* module is essential for the deployment of the policy on board of the real self-driving car. An example of the real world test performed using the *Policy 2* is showed in video[4]

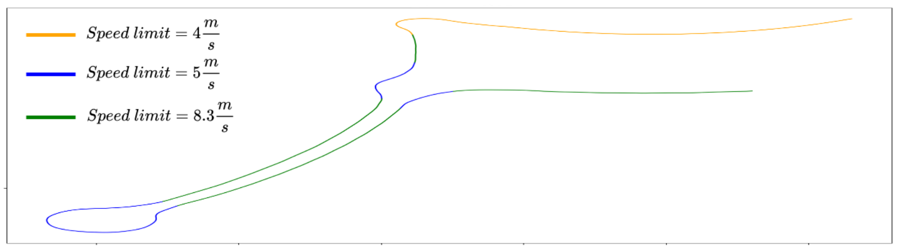
[4]`https://drive.google.com/file/d/1ZCI2qqNPY2CsE4U1xKfEqGzNintDxf9Q/view`

### 5.5.2 Speed Limits

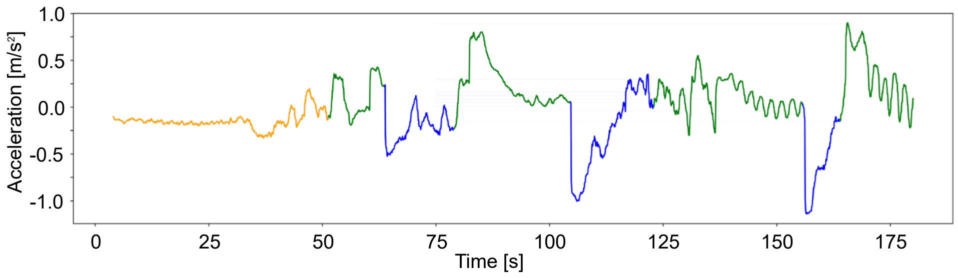
The second test we performed was aimed at demonstrating that the vehicle was able to follow its assigned path and adapt its speed to the speed limits imposed by different types of road. Fig. 5.6 depicts the behavior of a trained vehicle with a Policy 2 driving autonomously in a real urban area. In particular, the Fig. 5.6a represents a portion of the map (Fig. 5.1) with the speed limits of $4\frac{m}{s}$, $5\frac{m}{s}$ and $8.3\frac{m}{s}$ , corresponding to orange, blue and green curve colors respectively. In order to show a more detailed example of the longitudinal behavior of the real self-driving car, we plot the accelerations predicted by the network (Fig. 5.6b) and the vehicle speeds (Fig. 5.6c) obtained driving along the route of Fig. 5.6a.
From the speed graph Fig. 5.6c it is possible to see how the agent never exceeds the speed limit imposed by the different sections of the road and extrapolated from HD Maps. When it has to face a straight road it manages to increase its speed quickly, and when it encounters a roundabout or a section with a lower speed limit it manages to slow down just as fast. On the other hand, looking at the trend of the acceleration curve Fig. 5.6b, it can be seen that it appears very noisy. Actually for the difference between two consecutive predicted values never exceeds the threshold $\delta_{acc}$ used in the reward function in Equation 5.5, resulting in a smooth and comfortable longitudinal behavior perceived on-board vehicle.

### 5.5.3 Imitation Learning Pre-training

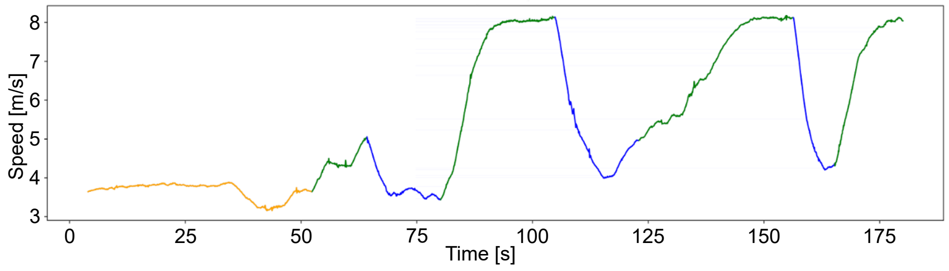
In this part of the project, the system is single-agent and there is only one agent on each scenario, but training takes place on multiple instances of different scenarios simultaneously. This already makes training long and onerous. If we then consider that the ultimate goal we are striving for is to achieve a multi-agent system we easily come to the conclusion that the training time could become prohibitive. To overcome this limitation of RL that is the need for millions of episodes to reach the optimal solution, we decided to perform a pre-training using Imitation Learning. Generally, when a neural network is trained by IL, the architecture that is used is very large and complex. Since our idea is to use the imitation approach only for pre-training and


$Speed\ limit = 4\frac{m}{s}$
$Speed\ limit = 5\frac{m}{s}$
$Speed\ limit = 8.3\frac{m}{s}$

(a) Path with different speed limits



(b) Acceleration



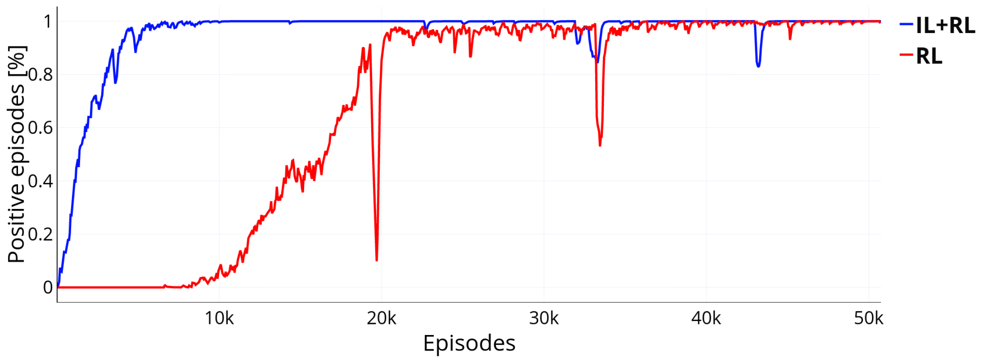
(c) Speed

Figure 5.6: Fig. 5.6a represents the path performed with the self-driving car in which colors identify the different speed limits faced in this area. Fig. 5.6b and Fig. 5.6c are respectively the trend of acceleration and speed during the real test.

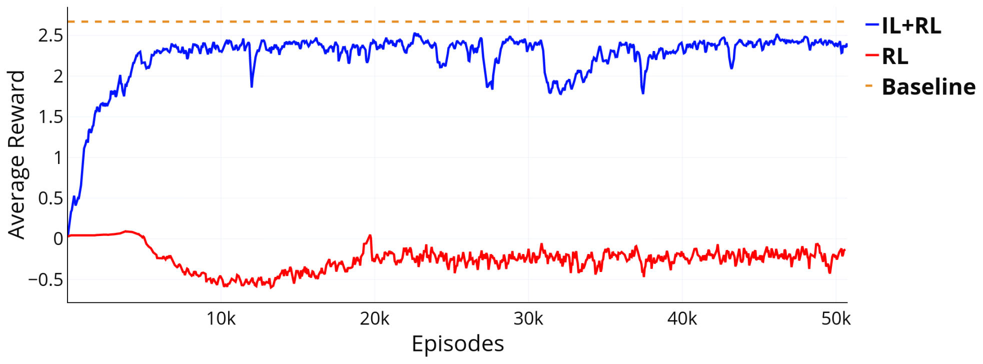
then continue the training with RL, we decided to use the same neural network Fig. 5.3 that is small instead, about one million parameters. The most obvious advantages obtained with this choice are increased robustness and good generalization capability. Because IL is a supervised approach the first problem we faced was to create a dataset with which to train the model. To create a complete dataset using data directly obtained from human drivers would have been overly burdensome and hardly feasible. Therefore, we decided to take advantage of our simulator to create a dataset with agents moving following rule-based approaches. In particular, for the curvature we use the tracking algorithm called Pure Pursuit [92], where the agent's purpose is to move following specific waypoints. Instead, we use the Intelligent Driver Model (IDM) [93] to control the longitudinal acceleration of the agent.

During the dataset creation, each agent moves to one of 4 scenarios (Fig 5.1) and every 100 ms saves the data needed to derive the scalar parameters and an image representing the agent's view at that instant. Instead, the output is given by the Pure Pursuit algorithm and the Intelligent Driver Model. During the IL training phase we do not estimate the values of the standard deviations ($\sigma_{acc}$, $\sigma_{sa}$) and neither the value functions ($v_{acc}$, $v_{sa}$), but only the tuple ($\mu_{acc}$, $\mu_{sa}$) that corresponds to the two lateral and longitudinal controls that are returned as output by the neural network. These features, together with the *deep response* module, are learned during the IL + RL training phase. A comparison of two different trainings is shown in Fig. 5.7. The first *Pure RL* that does not involve any kind of pre-training and starts directly with *Pure RL* (red curve); the second *IL + RL*, on the other hand, is still training using RL, but has been pre-training using IL (blue curve).

Fig. 5.7a shows how both approaches are able to achieve a good success rate, but *IL + RL* training requires fewer episodes than the *Pure RL* and the trend is also more stable. However, the most interesting result is obtained in the graph depicted in Fig. 5.7b, where it is evaluated how the average reward value grows during training. Looking at a time window of 50000 episodes, *IL + RL* policy manages to reach an optimal solution in a few episodes (blue curve), *Pure RL* approach, on the other hand, would require many more episodes to reach a comparable reward. In this case an optimal solution is represented by the dashed orange curve. This baseline represents the av-



(a) Positive episodes



(b) Average reward

Figure 5.7: Comparison between *Pure RL* training (red curve) and using a pre-training through Imitation Learning, *IL + RL* (blue curve). In Fig. 5.7b the orange dashed line represents the average rewards obtained in 50000 episodes using simulated agents that follow deterministic rules as those used for collecting the dataset for IL pre-training.

erage rewards obtained using simulated agents that perform 50000 episodes in the 4 scenarios of Fig. 5.1. The simulated agents follow deterministic rules as those used for collecting the dataset for IL pre-training, which therefore use Pure Pursuit for curvature and IDM for longitudinal acceleration. Since we are not only interested in the agent being able to reach the goal, but also in achieving a good policy; pre-training with Imitation Learning saved us quite a bit of time, since the time it takes for pre-training is significantly less than it takes *Pure RL* to achieve an optimal solution. Thus we proved that pre-training with IL drastically reduces the time needed for training. This gap between the two approaches may be more evident training the system performing more complex maneuvers in which agents interactions could be required.

# Chapter 6

# World Models for Autonomous Driving

Before proceeding with the work, we analyzed the limitations and issues related to the Deep Reinforcement Learning planner explained in the previous chapter.
First and foremost is the lack of obstacles. An autonomous vehicle must necessarily be able to drive in traffic situations and be able to negotiate and interact with other road users. For this reason, the inclusion of obstacles was one of the priorities evaluated in the further work.
In addition to this, we also realized that the size of the visual inputs we have used so far ($84 \times 84$ pixels) was very limiting. Indeed, wanting to drive at high speeds would require the vehicle to be able to see a larger area to comply with safety regulations. Currently it is limited to seeing only up to 40 meters ahead. Furthermore, considering that the agent's surrounding view is $50 \times 50$ meters it can be inferred that the resolution is about 60.0 *cm* per pixel. This implies a very high error range that could lead to accidents, since with this resolution the system must necessarily make approximations on the position and speed of obstacles. One type of situation that highlights this problem is when the agent should stop behind another vehicle, such as before entering a roundabout or at a traffic light. In these situations, the distances between two vehicles might be even less than 60.0 *cm*.

In addition to the lack of obstacles and the small size of the input bird's-eye views, we also found limitations related to the type of architecture we chose. Indeed, although it apparently seems to be a lightweight network, wanting to train it in a multi-agent system, on multiple scenarios simultaneously, with larger visual inputs trying to make the agent learn different maneuvers, we find that it actually turns out to have a very onerous computational load and extremely long training times. This is also caused by the fact that the neural network turns out to be compact, not decomposable into several simple models that can be trained separately. Moreover, in a compact system such as the one seen so far, in case of malfunctions or in case the agent learns unwanted behaviors, the process of debugging the system is really difficult.
Because of all the limitations just listed and present in the previous architecture, we decided in this concluding work to test new ones, choosing mainly from decomposable ones, that is, where it was possible to separate the global neural network into several simple models and train them separately.
After several experiments we chose to use the architecture proposed in the paper World Models [94]. This paper was very important in the field of reinforcement learning because it introduced a novel machine learning algorithm that solved a previously unsolved continuous action space, pixel observation space control problem. The architecture is divided into 3 separately trainable components: VAE, MDNRNN and a final control part. In this chapter we see in detail the different components and how we were able to adapt them to our system. Finally, we will see the results of some tests performed once the various components have been trained.
The purpose of this last part of the work was to evaluate how this architecture could be introduced in autonomous driving, and then evaluate the results and see if it could be a viable solution for the future. The authors had only tested this architecture in "game" environments, such as Car-Racing-v0 and ViSDoom. So this last part is simply a preliminary work where we demonstrated with some results that this architecture can be adapted to autonomous driving.

## 6.1 World Models

Before proceeding with the explanation of the main components of the World Model architecture [94] it is necessary to show the key and innovative concepts it proposes, especially because it has met with enormous success and had a major impact in the community of researchers in the field of Reinforcement Learning. First of all, their philosophy is to take cues from the human brain system, not only for the structure of the neural network, but also because of its way of learning from experience, interacting with the environment and consequently learning to make decisions by predicting the future. Now we take a closer look at these concepts in detail. A human, every day, is constantly interacting with the world around him and receiving millions of information from it. Our senses help us take in this information and our brains transform it into abstract representations. These representations cover spatial and temporal aspects and help us navigate and interact in our world. Every human being uses these representations to create his or her own model of the world that helps him or her unconsciously and significantly in daily life. Many actions we perform in the day are done automatically and instinctively, without any need to think or plan them. This means that our model of the world allows us to have small, constant predictions about the future of the sensory data we observe and enables us to react instinctively and adapt our "motor actions" without the need to plan based on the predictive model. In the paper [94], their main goal was to have an agent capable to learn a model of the environment, just as a human does. In this way, thanks to the learned model and through a control part, the agent is able to move and interact with the environment. In addition, they have also shown that their agent is also able to learn by training itself in the model of the environment it has created. As we have already mentioned, the result they obtained is a model that was able to train an agent to play and gain high scores in the OpenAi Gym environments. Since their system is based on learning a model by which the agent becomes able to predict future actions, clearly this architecture falls into the Reinforcement Learning category of model-based algorithms. Therefore, with the choice of this architecture, we have abandoned the model-free approach of the planner described in the previous chapter 5 to explore this different

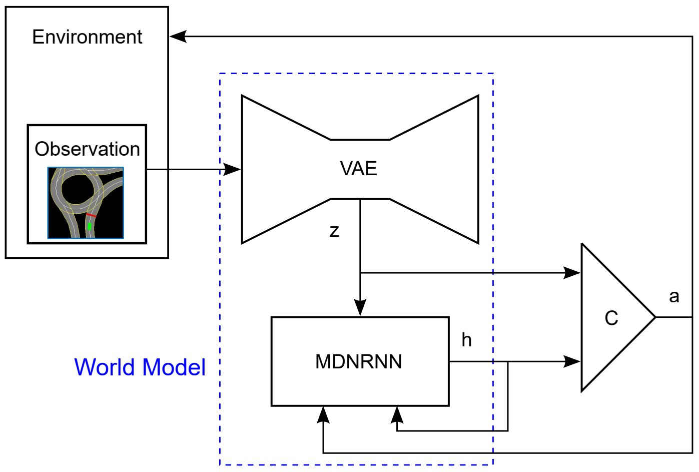


Figure 6.1: Architecture presented in the paper World Models, consisting mainly of three separately trainable components: Variational Autoencoder, MDNRNN and control.

type of algorithms. The architecture they proposed is the one in Fig. 6.1 and is decomposable into three sub-models: VAE-Model, MDNRNN-Model and Control-Model. The first two are the ones that allow the creation of the "World Model", while the control part deals with the decision-making process of the actions to be performed. Let us now take a closer look at the three sub-models and how they have been revisited in our system.

### 6.1.1 VAE-Model

The VAE-Model consists of a Variational Autoencoder, explained in section 2.2.4. Its task is basically to compress into a latent space $z$ what the agent sees at each instant of time $t$, that is, to reduce the high dimensional data of input into a lower dimensional space. Indeed, through this process, it can provide a lower dimensional space at the controlling part, in which it is easier to make decisions about the actions to be performed.
For training this model we used the same encoder and decoder structure used in the

paper [94], with the only difference that we added some Convolutional Layers (explained in 2.2), because compared to their work we chose to use a larger input size. Their input was $64 \times 64$ pixel images, in our case instead we use 4 images (Obstacles, Navigable Space, Path and Stop Line) obtained from the different channels as also seen in the previous chapter 5.1. In this work we chose to use images that have a size of $256 \times 256$ pixels, so an agent can have a wider overview of its surroundings, compared to the $50 \times 50$ meters it saw in the architectures of previous works. In addition, this change also results in higher resolution in the input images, greatly reducing the distance in meters corresponding to one pixel. A dataset of black-and-white images acquired in the 4 scenarios contained in the red rectangles in Fig. 5.1 was created for training the VAE-Model. As reconstruction loss we use the so called BCEWithLogitLoss, which combines a Sigmoid layer with the classical BCELoss in a single operation. This version turns out to be more numerically stable than using the two components separately. The complete loss is then the sum of the BCEWithLogitLoss and the similarity loss, for which we used the KL Divergence [62].
In this system, however, in the different four images, the amount of white pixels is completely different. The most obvious case can be seen by looking at the two channels of Navigable Space and Stop Line, in the former there are a good number of white pixels and in the latter there are only a few. Without any expedient and weighing the pixels in the same way on all 4 images, we found that the system was unable to reconstruct neither the obstacles nor the stoplines, which were the channels with the least concentration of white pixels. Thus, we decided to calculate a different weight to associate with the white pixels for each channel. Initially, all the white pixels of that channel in the entire dataset are added up and then divided by the number of total pixels.

$$channel\_pixels = \frac{white\_pixels}{total\_pixels} \tag{6.1}$$

The obtained value *channel_pixels* is then applied in the formula:

$$channel\_weight_i = \frac{1.0}{\log(c + channel\_pixels_i)} \tag{6.2}$$

which allows us to have the final weight of each channel *i*. This formula is also used in [95] and has a parameter *c* that must be calibrated in the training phase and usually depends on the composition of the dataset. In section 6.2 we will show some results obtained in the testing phase using this trained model.

### 6.1.2 MDNRNN-Model

The second element from which the World Model architecture is composed is the MDNRNN-Model, which, as can be guessed from its name, owns as its key component an RNN, more specifically an LSTM, explained in chapter 2.2.2. This element represents the memory of the system and contains a state within it that compresses the information of what happens in time. Keeping track of what happens in the input space would be too complex and onerous, so only the lower dimensional latent space $z_t$ learnt by the VAE-Model is kept track of. This process allows the internal state of the LSTM memory to accumulate knowledge and be able to predict how the environment changes based on the action the agent takes. In this way, the agent will be able to perform better actions because it can predict the consequences.
The MDNRNN-Model has two main components, the LSTM we just described, and a Gaussian Mixture Model, which together form a Mixture Density Network (MDN), from which the name "MDNRNN". A Mixed Density Network combines a neural network that can represent arbitrary non-linear functions, with a mixture model that can model arbitrary conditional distributions. This second component takes as input the deterministic $z_t$ value obtained with the RNN and returns a corresponding probability density function $p(z_t)$. Since most complex environments have a stochastic nature, this probability function $p(z_t)$ captures the stochasticity better than a deterministic prediction of $z_t$. The purpose of the MDNRNN-Model is to learn to predict the probability distribution of the next latent vector $z'_{t+1}$, based on current and past information. So as seen from the Fig. 6.2, at each time instant $t$, the MDNRNN-Model takes as input the latent space $z_t$ produced by the VAE-Model encoder, the action $a_t$ to be performed and the hidden state $h_t$ of the previous step, and returns the probability distribution of the next latent vector $z'_{t+1}$ and the hidden state $h_{t+1}$, which is useful for the next time instant. The output of this element, that is the predicted next environment latent representation $z'_{t+1}$ will actually not be used by the control part, just like the output decoded by the VAE-Model. Indeed, the C-Model makes use of an

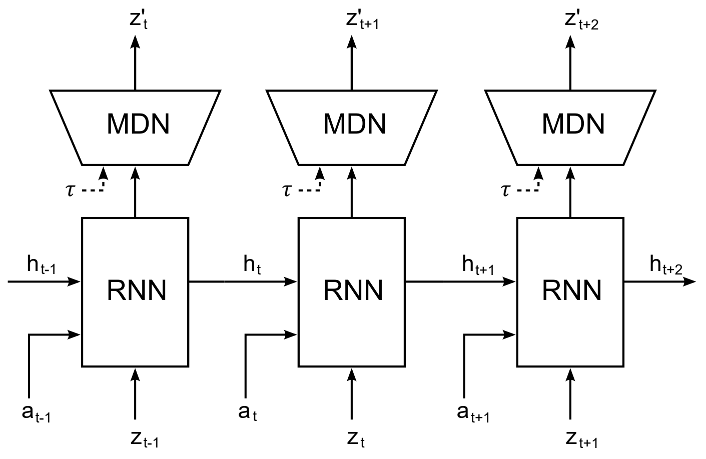


Figure 6.2: MDNRNN-Model architecture consisting of an RNN and a Mixture Density Network that predicts the probability distribution of the next latent vector.

internal representation learnt by the memory, that is the hidden state $h_t$ of the LSTM. This internal representation is a compressed representation that is useful to predict the future. For the training of the MDNRNN it is assumed that the VAE-Model has already been trained, in fact during the calculation of the latent $z_t$ space the encoder pre-trained in the VAE is used, but its parameters are locked, to keep the two models separate and have a more decomposable architecture.
Before introducing the changes we have made in our system on this component, it is necessary to make an additional premise. The "World Model" system is trained by Reinforcement Learning and as we will see later it predicts a reward at each step which is compared with the original reward contained in the training dataset. Since ours is a preliminary work with the purpose of evaluating and introducing this architecture in the autonomous driving domain we decided, as a first approach to use Imitation Learning, aware also of the results obtained in the paragraph 5.5.3. This choice mainly affects the control part, but also has feedback on this part of MDNRNN, especially in the composition of the loss function. In the original paper [94], the loss function was composed of 3 elements:

- *GMMLoss*: which compares the latent space computed by the VAE at time $t+$

1, with the latent space predicted by the MDNRNN at time $t$ and thus matching that at time $t+1$.

- *MSELoss*: applied between the dataset reward and the predicted reward.
- *BCELoss*: applied to evaluate the terminal state of the dataset and that predicted.

Having chosen an approach using Imitation Learning the our loss, instead, consists only of the GMMLoss. The action that takes as input the MDNRNN clearly cannot be the one obtained from the control part, because it has not yet been trained. They are therefore obtained from a dataset, which we will elaborate on in the next control section.

### 6.1.3 Control-Model

The final component of the architecture is the Control-Model and it has two tasks: deciding what actions to execute, and learning to decide what is the best action to execute. In the original architecture [94] this component is very simple because it can be represented by a single-layer linear model that maps the latent vector $z_t$ and the hidden state $h_t$ directly to an action $a_t$:

$$a_t = W_c[z_t, h_t] + b_c \tag{6.3}$$

where $W_c$ and $b_c$ are the parameters to be trained.
The combination of the compressed representation of the current environment ($z_t$) and the future environment ($h_t$), provided by the previous two models, are exactly what the control part needs to decide what action to perform. In the paper [94], the controller learns the parameters of this linear function that maximize the expected reward of our agent. This is an optimization problem and the algorithm that is used to find these parameters is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [96].
In our case, that we use Imitation Learning and not Reinforcement Learning, we still have to learn a policy, but not through a reward metric. Because IL is a supervised

approach, the agent must necessarily be trained based on a dataset, and the loss that is used is MSELoss, which compares the estimated acceleration and steering angle actions with those in the dataset. Since creating a complete dataset directly from a real vehicle was a very onerous and complex task, we generated it by exploiting the simulator described in section 3.3. The dataset is equivalent to the one in the test described in paragraph 5.5.3, that is, with agents moving with rule-based methods and traditional algorithms (IDM [93] and Pure Pursuit [92]). The only difference is that in that test obstacles were not present and, instead, in this dataset, rules were generated to introduce them and make them coexist in the scenario as in a real situation. This

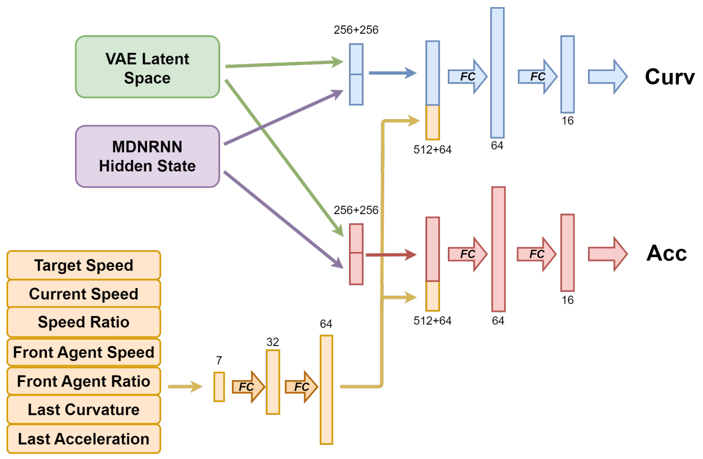


Figure 6.3: Architecture for the control part of the version of World Model that we adapted to autonomous driving. It is a very simple neural network that takes as input the latent space generated by the VAE, the hidden state of the MDNRNN, and a series of scalars to produce as output the acceleration and curvature set points to be passed to the actuation module.

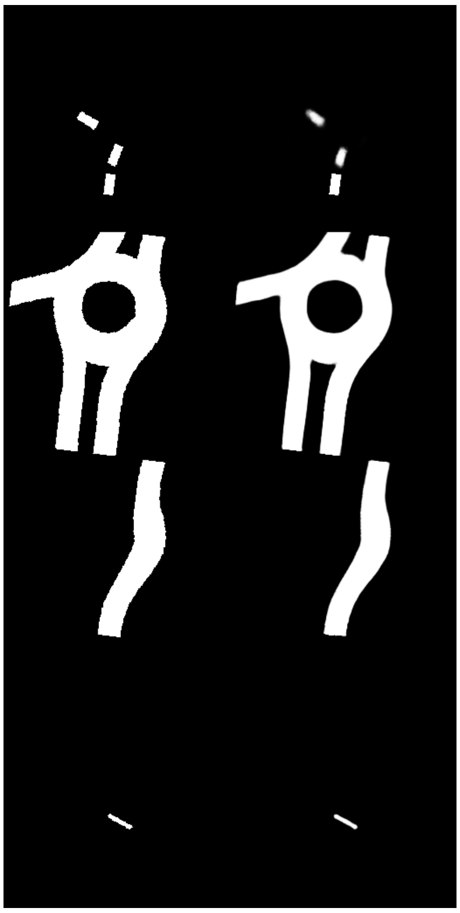
dataset was created so that it could be used for training all three components. In Fig. 6.3 it is possible to see the architecture of the neural network of the control part. In addition to the latent space generated by the VAE-Model and the hidden state generated by the MDNRNN-Model, we also added 7 scalar parameters as input. The 5 parameters already used in the neural network seen in the previous work 5.2 to which two parameters related to the speed of any vehicle present in front of the agent are added. During the training of the control part both the parameters of the VAE-Model and the MDNRNN-Model are locked so that they are not changed.

## 6.2 Experiments

One of the reasons we chose to use the World Models architecture rather than others was that it was decomposable, in which each model can be trained and tested individually. In this chapter we will look at the tests performed to verify the capabilities of the individual trained models. For the first two components, we were also able to do extensive testing on different types of scenarios, but for the control part, in this preliminary stage of the work, we focused only on straight roads. A unique dataset equivalent to the one described in the section 6.1.3 was created for all tests, but clearly different from the one used during the training phase.

### 6.2.1 VAE-Model Tests

The VAE-Model is a key component of the architecture, because since it is the one that generates the latent space that is then used by the other two models, if the latent space is not well descriptive of the input, the end result will be poor. Therefore, of the VAE-Model we are mainly interested in optimizing its capability to generate a latent space that is descriptive of the input, that is, the Encoder. Being compressed information, it is in no way possible to verify its accuracy based directly on the values contained in the latent space itself. Therefore, the Decoder component serves exactly that purpose, because it produces an output that is nothing more than the reconstructed Input. Therefore, to verify whether indeed the VAE is able to reconstruct the input well, we tested it on some sequences.
We report in Fig. 6.4 some images showing on the left the four input channels and on the right their respective reconstructions. We decided to include in this paper only two significant images as examples. In the first one (Fig. 6.4a ) the agent is able to correctly reconstruct a roundabout, obstacles and also the stop line. In the second one (Fig. 6.4b), on the other hand, it is noteworthy that he is able to correctly distinguish


(a) Roundabout

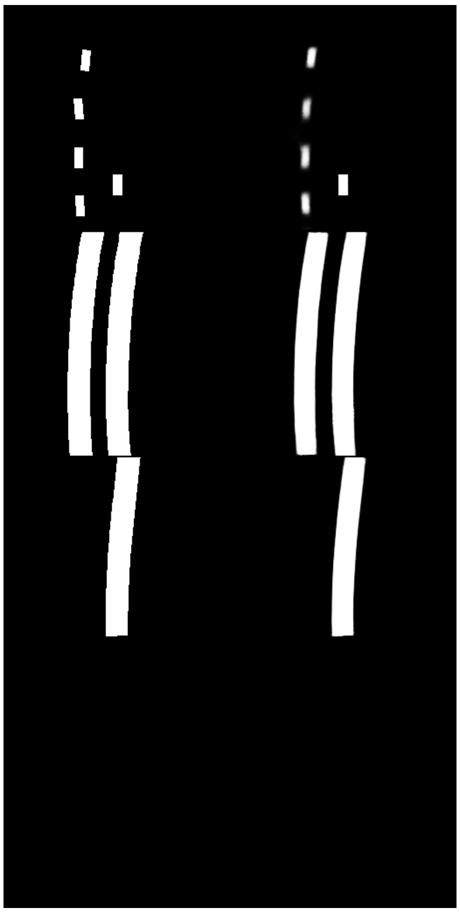

(b) Traffic

Figure 6.4: Results of VAE-Model Tests. 6.4a represents a correct reconstruction of a roundabout, 6.4b, instead, represents a situation with many traffic agents that the system can distinguish.

and reconstruct all agents on the scene, even if they are close to each other. To better show the results, a folder accessible from this link[5] was created that contains animated gifs representing the reconstruction of entire sequences, in which it is easier to appreciate the model's capability. Once we ascertained that the VAE-Model had achieved the desired results we moved on to training the next component.

### 6.2.2 MDNRNN-Model Tests

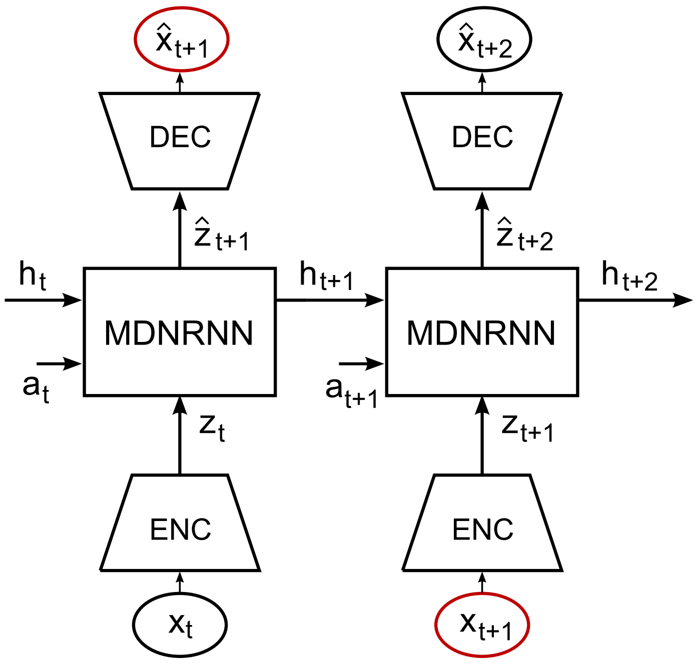


Figure 6.5: Baseline test architecture for MDNRNN-Model. In this test, the latent space of the next step predicted by the MDNRNN is decoded and compared with the input of the next step. That is, the two items circled in red.

In the MDNRNN-Model, it is important to evaluate whether the hidden state that is stored and then used in the control part is descriptive enough of the past history to

[5] `https://drive.google.com/drive/folders/1_3bW-VyN3ZxetLuFNkNJiOfBxKXxsDDy`

be able to generate correct and consistent future latent spaces. Being able to obtain a good hidden state puts the agent in a position to determine the action to be performed now by imagining what might happen in the future. Therefore, we performed two different types of tests. The first test, which we call *baseline test*, allows us to visually compare how much the latent space predicted by the MDNRNN for time $t+1$ differs from the actual input at time $t+1$. Basically, the reconstruction made with the VAE-Model decoder of the predicted latent space for the next step is compared with the input image of the next step, i.e., the two elements circled in red in Fig. 6.5. Fig 6.6 shows some example images of the results, in which the two elements to be compared can be seen superimposed. In particular, an erosion filter was applied to the original input to obtain only the contour (white), instead the grey pixels represent the reconstruction made with the decoder of the latent space predicted by the MDNRNN. They have been superimposed in order to better assess how much the reconstruction differs from the original.

It can be seen from the reconstructions that the agent has figured out how other agents are moving at the scene, because for example if it sees an agent coming from the opposite lane it can predict its downward movement. Similarly, it has learned that the navigable space also slopes downward if the agent is moving. Again, to better appreciate the results, a folder accessible from this link[6] was created, containing a number of gifs representing sequences of 70 consecutive steps.

For the trained MDNRNN model, we also performed an additional test in which we wanted to understand whether the generated latent spaces were substantial enough to allow it to use them as the latent spaces of the next step. Basically, the agent stopped at a certain step and began to dream about its future for a certain amount of subsequent steps. What we have therefore done is to use the latent space predicted by the MDNRNN-Model at the previous step as the input latent space for the MDNRNN at the next step. As seen in Fig. 6.7. Clearly, the expected result of this test is that there is a worsening going forward with the dream steps. And this is exactly what occurs in most cases. Some results, however, are still interesting. For example, the one shown

[6]`https://drive.google.com/drive/folders/1M2SX7xWiFhFvc_S3shpO9PZOa8ts5oeR`

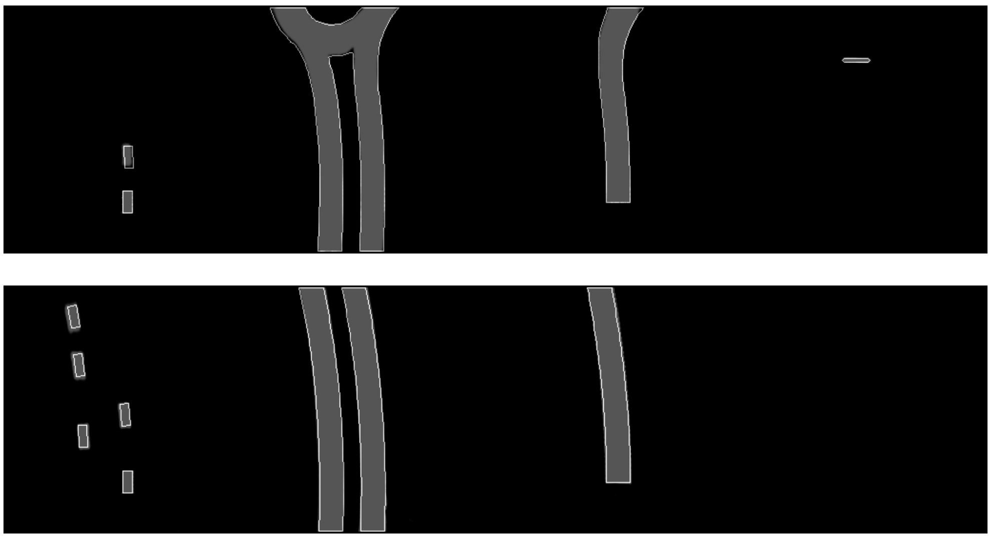


Figure 6.6: Two images representing examples of results obtained with the baseline test using trained MDNRN-Model. The white contour represents the input of the next step to which an erosion filter was applied to superimpose it on the gray areas representing the reconstruction made by the decoder of the latent space predicted by the MDNRNN

in Fig. 6.8, where the top image shows the initial step of a sequence 70 steps long and the bottom image the final step. The agent was imagined to be moving along the road and arriving at the stop line of a roundabout. Again it is difficult to show the results with single images, so we created a folder accessible from the link[7], in which there are gifs showing all the steps of the dream. In each gif you can see that actually the first 10 steps are always performed as in the baseline test, because before starting to dream the agent needs to accumulate information from the past history in the hidden state in order to then predict the future.

[7]`https://drive.google.com/drive/folders/1oKFfiYFG1_BIktDzLssNuw0ttfizAc_B`

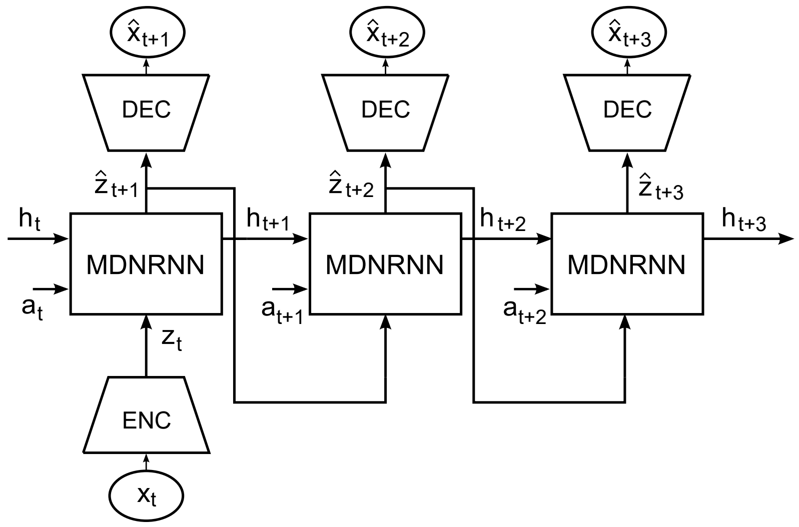


Figure 6.7: In this test, the agent is allowed to imagine the future for 70 steps. In fact, after a few initial steps in which the agent accumulates memory in the hidden state, the latent space decoded by an input is no longer used but the latent space predicted by the MDNRNN at the previous step.

### 6.2.3 Control-Model Tests

The test performed on the trained control model corresponds to a test on the whole architecture. Indeed, this model takes as input the result of the previous two components($z_t$ and $h_t$), as we saw in section 6.1.3. In this preliminary work, the agent was trained only to drive on a straight road. So the tests were also carried out in the same situation. To test it we used the simulator described in 3.3 and in which the agent drives in the middle of random traffic generated by other vehicles of various types. An example of its behavior is in the video[8]. In this video we have the agent (green) always driving along the same roadway of an almost straight road. The traffic is represented by the blue rectangles, which, according to their size, represent different types of vehicles. The results obtained are extremely satisfactory, indeed, the vehicle learned to

[8]https://drive.google.com/file/d/1LL9o7Kf_uflLC0kbmT_ttAqj-eBY8y9h/view?usp=share_link

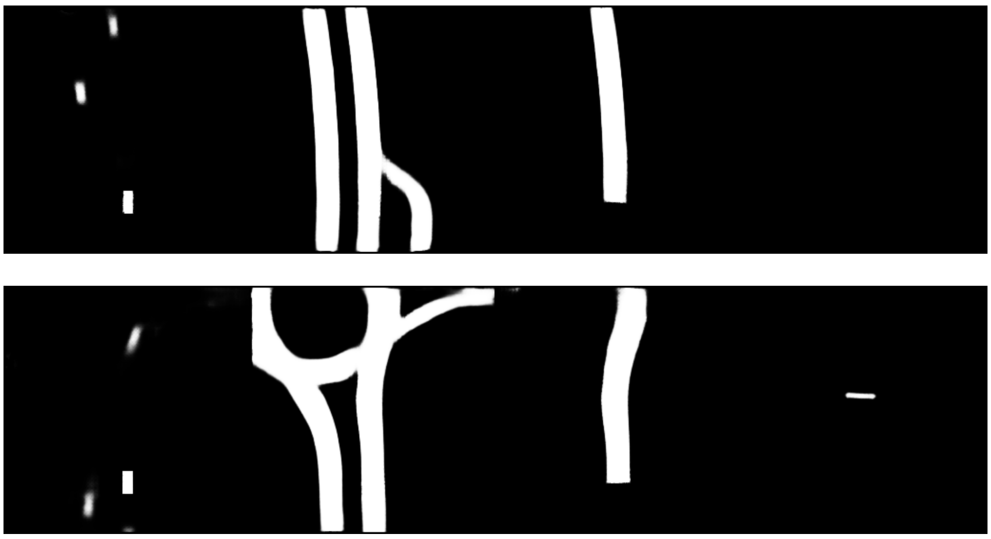


Figure 6.8: These two images represent the first and last steps of a dream test performed with the architecture in Fig. 6.7. In this case, the agent imagined itself driving along the road until it reach the stop line of a roundabout.

complete each episode without crashing into other vehicles or running off the road. In addition, it can also be seen that it is able to move correctly in traffic, slowing down when necessary and accelerating when the road is clear from traffic. For example, an astonishing result is that of the episode beginning at about 1.50 minute of the video, in which the ego agent manages to overtake a traffic vehicle who has stopped at the side of the road. However, these are only preliminary results; the system would then need to be tested directly on the real vehicle and trained on other scenarios, such as roundabout and intersections.

# Conclusion

The main purpose of this research work has been the exploration and development of Deep Reinforcement Learning and Imitation Learning algorithms in the field of autonomous driving and particularly in an urban environments. Indeed, the final goal of the project, which had its foundations in this work, is the development of a system based on a Deep Reinforcement Learning algorithm that can be used in a real vehicle and be able to drive safely and comfortably even in traffic conditions.
We started with the development of a neural network that was able to predict the correct lateral and logitudinal continuous control values that would allow an agent to safely cross an intersection, respecting the precedence rules dictated by traffic signs and also the right of way rule. The algorithm we proposed is D-A3C, explained in 2.3.9, that is able to obtain these output values based only on synthetic images reproducing navigable space, path, obstacles and traffic signs, and the agent's current speed and target speed values. Since our purpose was not to estimate when was the exact time to cross the intersection, but to make the agent intelligent and capable of negotiating and interacting with other vehicles, we decided to develop a multi-agent system, in which each individual agent participates in the optimization of the overall policy. Through testing, we proved that the agent learned to cross the proposed intersections by following its path and respecting basic driving rules, and it was also able to generalize the acquired behavior knowledge to different scenarios where there was a real traffic.
However, trying to test the proposed system on a real vehicle we found the driving style was not comfortable due to the acceleration values varying drastically very

quickly, leading the vehicle to make abrupt braking and starts. Because the goal of the project is to develop a system that also works in the real world, in this second part of the work we were concerned with the implementation of a Deep Reinforcement Learning planner capable of safe and comfortable driving that would be applicable on a real vehicle.
Since the focus was on driving style, we decided for this part to train the agent in an obstacle-free environment. The main problem we faced was to find a way to bridge the differences between simulation and reality, especially the delays introduced by vehicle dynamics. For this purpose, we implemented a small neural network that we called *deep response* model, which is trained directly on the real vehicle and allows us to learn what delays there are between the choice of an action and its implementation on a real vehicle. In this way, by embedding this module in simulation we were able to obtain agent behavior similar to the real thing even during training. Also for this part of the work the algorithm used was D-A3C in which the neural network is able to predict both the means and the standard deviations of two Gaussian distributions used to sample the acceleration and steering angle values. Tests conducted in a neighborhood area of Parma yielded satisfactory results, showing that the real vehicle was able to drive safely and comfortably even in those areas on which it had not been trained.
However, despite the good results, this work still has limitations, primarily the lack of static and dynamic obstacles. For a complete system usable in a daily urban context, it must necessarily be able to interact with other vehicles and adapt to their behaviors. In addition, the architecture used is compact and would be extremely onerous if a multi-agent system were to be trained. To overcome these limitations in the last part of this work, a new architecture proposed in the paper World Model [94] was tested, which is decomposable into 3 separately trainable models. The final purpose of the last part of the work was exactly to evaluate whether this architecture was also suitable for use in autonomous driving, and whether, consequently it could become the solution for our project. This new architecture has some differences from the one used in previous works, first of all, the first of the three models is a Variational Autoencoder which allowed us to use larger images as input, no longer just $84 \times 84$

pixels in size but also 256 $\times$ 256 pixels. This advantage is evident both in the quality of the actions, because a higher pixel-meter resolution can be achieved, and also because the agent is able to see at a greater distance and thus can move at higher speeds as well. An additional difference is that this system is no longer model-free like the previous one, but is model based, and is therefore able to exploit the model that is created to obtain predictions about the future that allow it to make better choices about what actions to take in the present. However, this last part of the project is just a preliminary work to evaluate this new architecture in the context of autonomous driving, and for simplicity we decided to use Imitation Learning as approach.

In any case, the behavior observed in simulation on a straight road with obstacles seems to be an excellent starting point for the development of a new complete planner. Since this new decomposable architecture scales very well with a complex system such as autonomous driving, it may allow us to test a multi-agent system. Future developments will tend precisely in this direction in which all agents present will contribute to the training of a global model, negotiating and interacting with each other. Before that, however, it will surely be necessary to proceed with training and testing of the control model to make it adequate to deal not only with straight roads but also with other types of more complex maneuvers: such as roundabout entry and intersection handling. Finally, it will also be crucial to evaluate how this system performs using Reinforcement Learning as main approach and not Imitation Learning as was done in this last part of the work. Certainly with Reinforcement Learning there will be an opportunity to introduce that generalization capability that is difficult to achieve from a supervised approach such as Imitation Learning.

# Bibliography

[1] Pawan Deshpande. Road safety and accident prevention in india: a review. *International Journal of Research in Advanced Engineering and Technology*, 5:64–68, 2014.

[2] Keigo Akimoto, Fuminori Sano, and Junichiro Oda. Impacts of ride and car-sharing associated with fully autonomous cars on global energy consumptions and carbon dioxide emissions. *Technological Forecasting and Social Change*, 174:121311, 2022.

[3] Daisuke Wakabayashi. Self-driving uber car kills pedestrian in arizona, where robots roam, 2018.

[4] E.D. Dickmanns and A. Zapp. Autonomous high speed road vehicle guidance by computer vision1. *IFAC Proceedings Volumes*, 20(5, Part 4):221–226, 1987.

[5] E.D. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, and J. Schiehlen. The seeing passenger car 'vamors-p'. In *Proceedings of the Intelligent Vehicles '94 Symposium*, pages 68–73, 1994.

[6] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer. Toward autonomous driving: the cmu navlab. i. perception. *IEEE Expert*, 6(4):31–42, 1991.

[7] Dean A Pomerleau. Knowledge-based training of artificial neural networks for autonomous robot driving. In *Robot learning*, pages 19–43. Springer, 1993.

[8] Alberto Broggi, Massimo Bertozzi, Alessandra Fascioli, C Guarino Lo Bianco, and Aurelio Piazzi. The argo autonomous vehicle's vision and control systems. *International Journal of Intelligent Control and Systems*, 3(4):409–441, 1999.

[9] R Behringer. The darpa grand challenge-autonomous ground vehicles in the desert. *IFAC Proceedings Volumes*, 37(8):904–909, 2004.

[10] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[11] Chris Urmson, J Andrew Bagnell, Christopher Baker, Martial Hebert, Alonzo Kelly, Raj Rajkumar, Paul E Rybski, Sebastian Scherer, Reid Simmons, Sanjiv Singh, et al. Tartan racing: A multi-modal approach to the darpa urban challenge. 2007.

[12] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Michele Buzzoni, Elena Cardarelli, Stefano Cattani, Pietro Cerri, Alessandro Coati, Stefano Debattisti, Andrea Falzoni, et al. Viac: An out of ordinary experiment. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 175–180, 2011.

[13] Alberto Broggi, Pietro Cerri, Mirko Felisa, Maria Chiara Laghi, Luca Mazzei, and Pier Paolo Porta. The vislab intercontinental autonomous challenge: an extensive test for a platoon of intelligent vehicles. *International Journal of Vehicle Autonomous Systems*, 10(3):147–164, 2012.

[14] Massimo Bertozzi, Alberto Broggi, Alessandro Coati, and Rean Isabella Fedriga. A 13,000 km intercontinental trip with driverless vehicles: The viac experiment. *IEEE Intelligent Transportation Systems Magazine*, 5(1):28–41, 2013.

[15] Massimo Bertozzi, Alberto Broggi, Alessandra Fascioli, and Stefano Nichele. Stereo vision-based vehicle detection. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No. 00TH8511)*, pages 39–44, 2000.

[16] Massimo Bertozzi, Emanuele Binelli, Alberto Broggi, and MD Rose. Stereo vision-based approaches for pedestrian detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*, pages 16–16, 2005.

[17] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. Stereo r-cnn based 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7644–7652, 2019.

[18] Juergen Dickmann, Jens Klappstein, Markus Hahn, Nils Appenrodt, Hans-Ludwig Bloecher, Klaudius Werber, and Alfons Sailer. Automotive radar the key technology for autonomous driving: From detection and ranging to environmental understanding. In *2016 IEEE Radar Conference (RadarConf)*, pages 1–6, 2016.

[19] Shunqiao Sun, Athina P Petropulu, and H Vincent Poor. Mimo radar for advanced driver-assistance systems and autonomous driving: Advantages and challenges. *IEEE Signal Processing Magazine*, 37(4):98–117, 2020.

[20] Divas Karimanzira, Helge Renkewitz, David Shea, and Jan Albiez. Object detection in sonar images. *Electronics*, 9(7):1180, 2020.

[21] You Li and Javier Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37(4):50–61, 2020.

[22] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A Chapman, Dongpu Cao, and Jonathan Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3412–3432, 2020.

[23] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 3266–3273, 2018.

[24] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. Sensors and sensor fusion in autonomous vehicles. In *2018 26th Telecommunications Forum (TELFOR)*, pages 420–425, 2018.

[25] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6):2140, 2021.

[26] Xiangmo Zhao, Pengpeng Sun, Zhigang Xu, Haigen Min, and Hongkai Yu. Fusion of 3d lidar and camera data for object detection in autonomous vehicle applications. *IEEE Sensors Journal*, 20(9):4901–4913, 2020.

[27] Angelo Nikko Catapang and Manuel Ramos. Obstacle detection using a 2d lidar system for an autonomous vehicle. In *2016 6th IEEE International conference on control system, computing and engineering (ICCSCE)*, pages 441–445, 2016.

[28] Arturo De la Escalera, J Ma Armingol, and Mario Mata. Traffic sign recognition and analysis for intelligent vehicles. *Image and vision computing*, 21(3):247–258, 2003.

[29] N Deepika and VV Sajith Variyar. Obstacle classification and detection for vision based navigation for autonomous driving. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2092–2097, 2017.

[30] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4231, 2018.

[31] Heng Wang, Bin Wang, Bingbing Liu, Xiaoli Meng, and Guanghong Yang. Pedestrian recognition and tracking using 3d lidar for autonomous vehicle. *Robotics and Autonomous Systems*, 88:71–78, 2017.

[32] Guilherme V Raffo, Guilherme K Gomes, Julio E Normey-Rico, Christian R Kelber, and Leandro B Becker. A predictive controller for autonomous vehicle path tracking. *IEEE transactions on intelligent transportation systems*, 10(1):92–102, 2009.

[33] Raul de Queiroz Mendes, Eduardo Godinho Ribeiro, Nicolas dos Santos Rosa, and Valdir Grassi Jr. On deep learning techniques to boost monocular depth estimation for autonomous navigation. *Robotics and Autonomous Systems*, 136:103701, 2021.

[34] V Harisankar, Variyar VV Sajith, and KP Soman. Unsupervised depth estimation from monocular images for autonomous vehicles. In *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, pages 904–909, 2020.

[35] Sampo Kuutti, Saber Fallah, Konstantinos Katsaros, Mehrdad Dianati, Francis Mccullough, and Alexandros Mouzakitis. A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. *IEEE Internet of Things Journal*, 5(2):829–846, 2018.

[36] Jesse Levinson and Sebastian Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *2010 IEEE international conference on robotics and automation*, pages 4372–4378, 2010.

[37] Brent Schwarz. Mapping the world in 3d. *Nature Photonics*, 4(7):429–430, 2010.

[38] Talha Takleh Omar Takleh, Nordin Abu Bakar, Shuzlina Abdul Rahman, Raseeda Hamzah, and ZA Aziz. A brief survey on slam methods in autonomous vehicle. *International Journal of Engineering & Technology*, 7(4):38–43, 2018.

[39] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.

[40] Sven Bauer, Yasamin Alkhorshid, and Gerd Wanielik. Using high-definition maps for precise urban vehicle localization. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 492–497, 2016.

[41] Laurene Claussmann, Marc Revilloud, Dominique Gruyer, and Sébastien Glaser. A review of motion planning for highway autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 21(5):1826–1848, 2019.

[42] Wenda Xu, Jia Pan, Junqing Wei, and John M Dolan. Motion planning under uncertainty for on-road autonomous driving. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2507–2512, 2014.

[43] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 1094–1099, 2015.

[44] Zhiqing Huang, Ji Zhang, Rui Tian, and Yanxin Zhang. End-to-end autonomous driving decision based on deep reinforcement learning. In *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, pages 658–662, 2019.

[45] Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.

[46] Thomas van Orden and Arnoud Visser. End-to-end imitation learning for autonomous vehicle steering on a single-camera stream. In *International Conference on Intelligent Autonomous Systems*, pages 212–224, 2021.

[47] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[48] Ürün Dogan, Johann Edelbrunner, and Ioannis Iossifidis. Autonomous driving: A comparison of machine learning techniques by means of the prediction of lane change behavior. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 1837–1843, 2011.

[49] Ajay Agrawal, Joshua Gans, and Avi Goldfarb. What to expect from artificial intelligence, 2017.

[50] Andrés Serna and Beatriz Marcotegui. Detection, segmentation and classification of 3d urban objects using mathematical morphology and supervised learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:243–255, 2014.

[51] Karsten Behrendt, Libor Novak, and Rami Botros. A deep learning approach to traffic lights: Detection, tracking, and classification. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1370–1377, 2017.

[52] Vsevolod Nikulin, Albert Podusenko, Ivan Tanev, and Katsunori Shimohara. Regression-based supervised learning of autosteering of a road car featuring a delayed steering response. *International Journal of Data Science and Analytics*, 7(2):149–163, 2019.

[53] Brook W Abegaz. Asdvc-a self-driving vehicle controller using unsupervised machine learning. In *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*, pages 1–6, 2020.

[54] Mehdi Rezagholiradeh and Md Akmal Haidar. Reg-gan: Semi-supervised learning based on generative adversarial networks for regression. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2806–2810, 2018.

[55] Alex Teichman and Sebastian Thrun. Tracking-based semi-supervised learning. *The International Journal of Robotics Research*, 31(7):804–818, 2012.

[56] Jiachen Li, Haiming Gang, Hengbo Ma, Masayoshi Tomizuka, and Chiho Choi. Important object identification with semi-supervised learning for autonomous driving. *arXiv preprint arXiv:2203.02634*, 2022.

[57] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[58] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[59] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[60] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[61] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[62] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[63] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[64] Alessandro Paolo Capasso, Giulio Bacchiani, and Daniele Molinari. Intelligent roundabout insertion using deep reinforcement learning. *arXiv preprint arXiv:2001.00786*, 2020.

[65] Alessandro Paolo Capasso, Giulio Bacchiani, and Alberto Broggi. From simulation to real world maneuver execution using deep reinforcement learning. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1570–1575, 2020.

[66] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[67] Monya Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604), 2016.

[68] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[69] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[70] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

[71] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[72] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[73] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[74] Mark Martinez, Chawin Sitawarin, Kevin Finch, Lennart Meincke, Alex Yablonski, and Alain Kornhauser. Beyond grand theft auto v for training, testing and enhancing deep learning in self driving cars. *arXiv preprint arXiv:1712.01397*, 2017.

[75] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16, 2017.

[76] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011.

[77] Giulio Bacchiani, Daniele Molinari, and Marco Patander. Microscopic traffic simulation by cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1903.01365*, 2019.

[78] Keith Packard and Carl Worth. Cairo graphics library. 2003-2020.

[79] Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1929–1934, 2020.

[80] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[81] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[82] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338, 2016.

[83] Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin, and Birgitta Dresp-Langley. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics*, 10(1):22, 2021.

[84] David N Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976.

[85] Avik Pal, Jonah Philion, Yuan-Hong Liao, and Sanja Fidler. Emergent road rules in multi-agent driving environments. *arXiv preprint arXiv:2011.10753*, 2020.

[86] David Isele and Akansel Cosgun. To go or not to go: a case for q-learning at unsignalized intersections. 2017.

[87] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[88] Yuanyuan Wu, Haipeng Chen, and Feng Zhu. Dcl-aim: Decentralized coordination learning of autonomous intersection management for connected and automated vehicles. *Transportation Research Part C: Emerging Technologies*, 103:246–260, 2019.

[89] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039, 2018.

[90] Carlos Gershenson and David A Rosenblueth. Self-organizing traffic lights at multiple-street intersections. *Complexity*, 17(4):23–39, 2012.

[91] LA Prashanth and Shalabh Bhatnagar. Reinforcement learning with average cost for adaptive control of traffic lights at intersections. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1640–1645, 2011.

[92] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[93] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[94] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[95] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[96] M Willjuice Iruthayarajan and S Baskar. Covariance matrix adaptation evolution strategy based design of centralized pid controller. *Expert systems with Applications*, 37(8):5775–5781, 2010.